

GALERKIN CFD SOLVERS FOR USE IN A
MULTI-DISCIPLINARY SUITE FOR MODELING
ADVANCED FLIGHT VEHICLES

By

NICHOLAS J MOFFITT

Bachelor of Science in Aerospace Engineering
Oklahoma State University
Stillwater, Oklahoma
2002

Master of Science in Mechanical Engineering
Oklahoma State University
Stillwater, Oklahoma
2004

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May 2013

GALERKIN CFD SOLVERS FOR USE IN A
MULTI-DISCIPLINARY SUITE FOR MODELING
ADVANCED FLIGHT VEHICLES

Dissertation Approved:

Dr. Andrew S. Arena

Dissertation Adviser

Dr. Frank W. Chambers

Dr. J. Keith Good

Dr. A. J. Johannes

ACKNOWLEDGEMENTS

I am very grateful to have many family, friends, faculty, staff, and students to thank for all of their help throughout this work:

I should start with those who affected the work directly. My work is the continuation of Tim Cowan's dissertation. Tim, thank you leaving a solid base for development and many opportunities for advancement.

Charles O'Neill gave me the initial training in using the CASE lab software. Charles, thank you for all your help, guidance, support and friendship. I will always remember our late night discussions that mixed physics, economics, and religion.

I had the opportunity to work with several Master's students, who assisted my endeavors during their time in the CASE lab: Cody Pinkerman, Anthony Hassett, Colin Brown, Matt Sukraw, and Robert Fischer. Thank you to all of you for all hard work, dedicated hours, and respect while working with the codes that I was developing. Thank you for not only being co-workers but also friends.

Dr. Gupta, NASA, and AES provided the culmination of many small projects into the body of work in this document. I want to thank Dr. Gupta, NASA, and AES for the opportunity and funding that accompanied the task.

Dr. Arena, thank you for advising me in my endeavors in teaching, designing, and research. Thank you for bringing all of these people and projects into my life; I would not be the same person or engineer without these experiences. Times were often rocky, but I am ever grateful for the freedom to choose my own direction and experience that I gained along the way.

Thank you to my committee for the encouragement and great devotion to reading this document. I cannot thank you enough for your time and guidance.

The most influential people were not even involved directly in this work. My family and friends kept me fighting through the hard times and kept my spirits up when there did not seem to be a light at the end of the tunnel:

Mom and Dad, thank you for all of your love and support, emotionally and financially. I cannot thank you enough for everything that you have given me. I definitely would not have stuck out the last several years without your supporting words.

Heather and Tod, thank you for also being supporting and giving me a place to go when I left Stillwater. We have all been struggling to finish our degrees, and I am so happy that all three of us finished this last stretch together, as a family.

Uncle Tom, thank you for providing me with a laptop when I could not afford to replace my computer. The laptop gave me the opportunity to visit family and friends while still working on this research.

Thank you to my many friends, who have supported me over the many years. Thank you for distracting me with games and talks as breaks in the research; but most of all, thank you for helping me keep on task when my research required it. I hope we can all catch up on lost time: (alphabetically) Deric & Kati Babcock, Mike Beavers, Megan Busby, Carol Challenger, Max and Jean Cobb, Joe Conner, Alex Christie, Mark Davey, Josh and Anna Edwards, Shawn Fleming, Thomas Hayes, Shannon Honer, Fred Keating, James Kidd, Ryan Oliver & China, Heather Orr, Shannon Robinson, Gentry Shelton, John and Sarah Terrell, Cindy Washington, and many others.

Thank you for all of the people at Countryside and BSF (St Louis) for your encouraging words and support. And thank you to all of my co-workers at Boeing who have kept me going during these last four months: Andrew, Andy, Josh, Brian, Matthew, and Mark.

Finally, I have to thank God for the love and patience, for all of the people listed on this page, and for all of the opportunities and financial support that came along when it was most needed.

Name: NICHOLAS J MOFFITT

Date of Degree: MAY 2013

Title of Study: GALERKIN CFD SOLVERS FOR USE IN A MULTI-DISCIPLINARY SUITE FOR MODELING ADVANCED FLIGHT VEHICLES

Major Field: MECHANICAL AND AEROSPACE ENGINEERING

Abstract:

This work extends existing Galerkin CFD solvers for use in a multi-disciplinary suite. The suite is proposed as a means of modeling advanced flight vehicles, which exhibit strong coupling between aerodynamics, structural dynamics, controls, rigid body motion, propulsion, and heat transfer. Such applications include aeroelastics, aeroacoustics, stability and control, and other highly coupled applications. The suite uses NASA-STARS for modeling structural dynamics and heat transfer. Aerodynamics, propulsion, and rigid body dynamics are modeled in one of the five CFD solvers below.

Euler2D and Euler3D are Galerkin CFD solvers created at OSU by Cowan (2003). These solvers are capable of modeling compressible inviscid aerodynamics with modal elastics and rigid body motion. This work reorganized these solvers to improve efficiency during editing and at run time. Simple and efficient propulsion models were added, including rocket, turbojet, and scramjet engines.

Viscous terms were added to the previous solvers to create NS2D and NS3D. The viscous contributions were demonstrated in the inertial and non-inertial frames. Variable viscosity (Sutherland's equation) and heat transfer boundary conditions were added to both solvers but not verified in this work. Two turbulence models were implemented in NS2D and NS3D: Spalart-Allmarus (SA) model of Deck, et al. (2002) and Menter's SST model (1994). A rotation correction term (Shur, et al., 2000) was added to the production of turbulence. Local time stepping and artificial dissipation were adapted to each model.

CFDsol is a Taylor-Galerkin solver with an SA turbulence model. This work improved the time accuracy, far field stability, viscous terms, Sutherland's equation, and SA model with NS3D as a guideline and added the propulsion models from Euler3D to CFDsol.

Simple geometries were demonstrated to utilize current meshing and processing capabilities. Air-breathing hypersonic flight vehicles (AHFVs) represent the ultimate application of the suite. The current models are accurate at low supersonic speed and reasonable for engineering approximation at hypersonic speeds. Improvements to extend the models fully into the hypersonic regime are given in the Recommendations section.

TABLE OF CONTENTS

<u>Contents</u>	<u>Page</u>
Table of Figures	xii
Table of Tables	xxvi
List of Symbols	xxvii
Chapter 1: Introduction	1
1.1 State-of-the-Art	9
1.2 Comparison of Turbulence Models	17
Chapter 2: Problem Statement	28
2.1 Emphasis and Objectives	28
2.1.1 Work with NASA-CFDsol (NASA Contract)	28
2.1.2 Work with In-House Codes	30
2.1.3 Objectives and Milestones	32
2.1.4 Additional Work	37
Chapter 3: Theory	39
3.1 Euler Equations	39
3.2 Navier-Stokes Equations	42
3.3 Thermodynamic Relationships	45
3.4 Turbulence Modeling Theory	51
3.4.1 Reynolds-Average Navier-Stokes (RANS)	52

<u>Contents</u>	<u>Page</u>
3.4.2 Favre-Averaged Navier-Stokes (FANS)	64
3.4.3 Spalart-Allmarus Model (Inertial)	70
3.4.4 Menter's SST Model (Inertial)	75
3.4.5 Non-Inertial RC Correction to Turbulence Models	80
3.5 Non-Dimensional Equations	84
3.6 Boundary Conditions	90
3.7 Rigid Body Model	97
3.8 Structural / Controls Model	102
Chapter 4: Development	107
4.1 In-House Codes	107
4.1.1 Shape Function	107
4.1.2 Element Jacobian	109
4.1.3 Element Gradients	110
4.1.4 Consistent and Lumped Mass Matrices	112
4.1.5 Gauss Quadrature	113
4.1.6 Galerkin Formulation	115
4.1.7 Gauss's Theorem	117
4.1.8 Unsteady Term	119
4.1.9 Element Fluxes Terms	119
4.1.10 Boundary Flux Terms	120
4.1.10.1 Riemann Invariants	121
4.1.10.2 Viscous Boundary Fluxes	129
4.1.10.3 Turbulent Fluxes	130
4.1.11 Boundary Conditions	132

<u>Contents</u>	<u>Page</u>
4.1.11.1 Far Field	133
4.1.11.2 Inviscid Wall	133
4.1.11.3 Viscous Wall	136
4.1.11.4 Symmetry Plane	138
4.1.11.5 Rocket Exhaust	142
4.1.11.6 Turbojet Engine Planes	145
4.1.12 Artificial Dissipation	153
4.1.13 Predictor-Corrector and Temporal Discretization	156
4.1.14 Residual and Boundary Conditions	159
4.2 CFDsol	160
4.2.1 Taylor Formulation	161
4.2.2 Galerkin Formulation	162
4.2.2.1 Stiffness Matrix	163
4.2.2.2 Pressure Vector	164
4.2.3 Boundary Conditions	165
4.2.3.1 Far Field	165
4.2.3.2 Inviscid Wall	170
4.2.3.3 Viscous Wall	171
4.2.3.4 Symmetry Plane	171
4.2.3.5 Rocket Exhaust	172
4.2.4 Stability	174
4.2.5 Boundary Integrals	176
4.2.6 Artificial Dissipation	180
4.3 Source Terms	186

<u>Contents</u>	<u>Page</u>
4.3.1 Non-Inertial Terms	187
4.3.2 Turbulent Source Terms	189
4.3.3 Quasi-Combustion Terms	190
4.4 Integration of Momentum	194
4.5 Local Time Stepping	202
4.5.1 Inviscid Local Time Step	203
4.5.2 Inviscid vs. Viscous Stability	204
4.5.3 Numerical Error in Derivatives	205
4.5.4 Viscous Local Time Step	207
Chapter 5: Implementation	221
5.1 In-House Codes	221
5.1.1 General Changes	222
5.1.2 Upgrade to Rigid Body Dynamics Model	224
5.1.3 Creation of NS2D/3D	225
5.1.4 Proper Tracking of Total Energy / Pressure	231
5.1.5 Zero Dissipation Length	231
5.1.6 Sutherland's	232
5.1.7 Acoustic Output Files	233
5.2 NASA-CFDsol	234
5.2.1 Proper Tracking of Total Energy / Pressure	234
5.2.2 Non-Inertial Frame and Rigid Body Dynamics Model	234
5.2.3 Structural Dynamics Model	237
5.2.4 Viscous Terms	238
5.2.5 Sutherland's Equation	239

<u>Contents</u>	<u>Page</u>
5.2.6 Zero Dissipation Length – Instability	240
5.3 Propulsion Models	253
5.4 Turbulence Modeling	256
5.5 Memory	272
5.6 Run Times	273
5.7 Support Software	276
Chapter 6: Demonstration of In-House Codes	289
6.1 Inviscid Aerodynamics	289
6.1.1 Subsonic	289
6.1.2 Transonic	308
6.1.3 Supersonic	323
6.1.4 Time-Accurate	343
6.2 Propulsion Modeling	349
6.3 Viscous Aerodynamics	389
6.3.1 Laminar	390
6.3.2 Turbulent	421
Chapter 7: Comparison with NASA-CFDsol	436
7.1 Inviscid Aerodynamics	436
7.1.1 Subsonic	437
7.1.2 Transonic	450
7.1.3 Supersonic	465
7.1.4 Time-Accurate (Acoustic)	469
7.2 Propulsion Modeling	479
7.3 Viscous Aerodynamics	504

<u>Contents</u>	<u>Page</u>
7.3.1 Laminar	504
7.3.2 Turbulent	515
Chapter 8: Conclusions and Recommendations	520
8.1 Conclusions	520
8.1.1 Objectives	520
8.1.2 Evaluation	523
8.1.3 Precaution against Pitfalls	532
8.1.4 Standards and Good Practices	534
8.1.5 Converting between Solvers	539
8.2 Recommendations	542
References	552
Appendices	566
Appendix A: Stokes' Hypothesis	566
Appendix B: Analytical Integrals	570
Appendix C: Euler2D File Formats	584
Appendix D: NS2D File Formats	637
Appendix E: Euler3D File Formats	663
Appendix F: NS3D File Formats	718
Appendix G: 2D Acoustic Outputs	749
Appendix H: 3D Acoustic Outputs	752
Appendix I: Entropy-Based Artificial Dissipation	759

TABLE OF FIGURES

<u>Figure</u>	<u>Page</u>
Figure 1.1: Specific Impulse vs. Mach Number (taken from Curran, 1996).	4
Figure 2.1: Flow Diagram (Lower Order System Model).	26
Figure 2.2: Flow Diagram (Higher Order System Model).	27
Figure 2.3: Flow Diagram (NASA-CFDsol).	28
Figure 2.4: Flow Diagram (In-House OSU Codes).	31
Figure 3.1: Relative Velocity at Boundary Node of a Translating / Spinning Domain.	100
Figure 3.2: Solid Wall Normals with Elastic Motion.	103
Figure 4.1: Linear Shape Function and Property Distribution.	108
Figure 4.2: Local Coordinates and Shape Functions for Linear Triangular Element.	109
Figure 4.3: Discontinuity at the Boundary Element.	122
Figure 4.4: Behavior of Characteristic Waves in Different Flow Regimes.	125
Figure 4.5: 2D Characteristics: Acoustic, Subsonic, Sonic, & Supersonic.	125
Figure 4.6: Normals along Trailing Edge of an Airfoil.	135
Figure 4.7: Traveling Waves within Rocket Combustion Chamber.	143
Figure 4.8: Progressive Growth of Pressure within Rocket Combustion Chamber.	144
Figure 4.9: Rocket Engine – Pressure and Mach Number Distributions.	145
Figure 4.10: Phase Plane for Inlet Mass Flow Rate and Static Pressure.	147
Figure 4.11: Oscillating Mass Flow Rate and Boundary Pressure (Subsonic).	148

<u>Figure</u>	<u>Page</u>
Figure 4.12: Oscillating Mass Flow Rate and Boundary Pressure (Supersonic).	149
Figure 4.13: Engine Inlets under Various Conditions.	150
Figure 4.14: Properties along Turbojet Inflow Boundary over 15k Ramp-up Steps.	154
Figure 4.15: Fluxes along Turbojet Inflow Boundary over 15k Ramp-up Steps.	154
Figure 4.16: Effective Density Change and Number of Iterations to Zero.	158
Figure 4.17: Pressure Distribution around NACA 0012 Airfoil (Mach 0.5, 5o, CFDsol).	166
Figure 4.18: Velocity Distribution around NACA 0012 Airfoil (Mach 0.5, 5o, CFDsol).	166
Figure 4.19: Coefficient of Pressure from CFDsol and Potential Flow Solution.	167
Figure 4.20: Domain Pressure through NACA 0012 Solution (Mach 0.5, 5o, CFDsol).	168
Figure 4.21: Density and Velocity from NACA 0012 (Mach 0.5, CFDsol).	168
Figure 4.22: Velocity Distribution for NACA 0012 in Rectangular Domain (Mach 0.5).	169
Figure 4.23: Surface and Domain Pressure for NACA 0012 (Improved).	170
Figure 4.24: Static Properties at Rocket Boundary (CFDsol, Euler3D).	173
Figure 4.25: Development of Static Pressure Downstream of Rocket Inflow Plane.	174
Figure 4.26: Velocity around NACA 0012 in “Wind Tunnel” (CFDsol).	175
Figure 4.27: Oscillations in Pressure Distribution from CFDsol.	176
Figure 4.28: Unsteady Solution Accuracy vs. Time Step and Dissipation.	181
Figure 4.29: Inaccurate Motion of Wagner Vortex.	184
Figure 4.30: Corrected Motion of Wagner Vortex.	186
Figure 4.31: Cylinder Translating at 45-deg and CD vs. Re.	188
Figure 4.32: Oscillations in Velocity Profile around Spinning Cylinder.	188
Figure 4.33: Circumferential Velocity Profile around Spinning Cylinder.	189
Figure 4.34: Afterburner Properties Simulated with a “Step” Generation.	193
Figure 4.35: Afterburner Properties Simulated with a “Cosine” Generation.	193

<u>Figure</u>	<u>Page</u>
Figure 4.36: Body and Reaction Forces	194
Figure 4.37: Control Volume around Body.	195
Figure 4.38: Stable CFL Values vs. Element Reynolds Number Re_x .	204
Figure 5.1: Organization of Nodes in case.g2d and case.g3d.	228
Figure 5.2: Organization of Boundary Elements in case.g2d and case.g3d.	228
Figure 5.3: Laminar Velocity Profiles from NS2D with $diss = 1.0$ and 0.0 .	231
Figure 5.4: Effective Dissipation.	232
Figure 5.5: Pressure over NACA 0012 Airfoil (CFDsol, Inviscid, Non-Inertial).	236
Figure 5.6: Empty Domain under Non-Inertial Translation and Rotation.	237
Figure 5.7: Pressure over NACA 0012 Airfoil (CFDsol, Inviscid, Transpiration)	239
Figure 5.8: Cross-Flow Velocity Vectors (Looking Down on Plate, Sym on Top & Btm).	241
Figure 5.9: Contributions and Cross-Flow Velocity at Single Node.	243
Figure 5.10: Inviscid Contributions to Crossflow Velocity at Single Node.	243
Figure 5.11: Large Vertical Oscillation near Bottom of Boundary Layer.	244
Figure 5.12: Divergence in Pressure on Surface of Inviscid Ellipse ($niter \times \tau = 958$).	247
Figure 5.13: Oscillations in Density along Wall of Ellipse ($Re = 4000$).	248
Figure 5.14: Oscillations in Velocity along Wall of Ellipse ($Re = 4000$).	248
Figure 5.15: Oscillations in Pressure Distribution around Inviscid Cylinder.	249
Figure 5.16: Time Step Study for Circular Cylinder ($Re = 41.0$).	250
Figure 5.17: Time Step Study for Circular Cylinder ($Re = 26.0$).	251
Figure 5.18: Six Snapshots of Turbulent Advection in a Straight Domain.	262
Figure 5.19: Turbulent Bubble Initial Conditions and Advected Downstream.	263
Figure 5.20: Slices of Turbulent Bubble being Advected Downstream.	263
Figure 5.21: Slices of Turbulent Bubble with Different Levels of Viscous Diffusion.	264

<u>Figure</u>	<u>Page</u>
Figure 5.22: Advected Bubble at Various Levels of Molecular & Artificial Dissipation.	266
Figure 5.23: Slices of Turbulent Bubble Solution from NS2D-SA (Various Re).	266
Figure 5.24: Turbulent Bubble Initial Conditions and Advected Downstream.	267
Figure 5.25: Slices of Turbulent Bubble being Advected Downstream.	267
Figure 5.26: Slices of Turbulent Bubble with Different Levels of Viscous Diffusion.	268
Figure 5.27: Advected Bubble at Various Levels of Molecular & Artificial Dissipation.	269
Figure 5.28: Slices of Turbulent Bubble from Complete SA Model (Various Re).	269
Figure 5.29: Comparison of Run Times for Euler2D and NS2D.	274
Figure 5.30: Comparison of Run Times for Euler3D and NS3D.	275
Figure 5.31: Comparison of Run Times in 2D Inertial and Non-Inertial Frames.	276
Figure 5.32: Comparison of Run Times in 3D Inertial and Non-Inertial Frames.	277
Figure 5.33: Remeshed Solutions for Supersonic Wedge.	281
Figure 5.34: Remeshed Solutions for Supersonic Cones.	281
Figure 6.1: NACA 0012 Airfoil Mesh, Elliptical Far Field.	291
Figure 6.2: Pressure, Density, Mach, and Entropy Distrib. for NACA 0012 (Mach 0.3).	291
Figure 6.3: Pressure, Density, Mach, and Entropy Distrib. for NACA 0012 (Mach 0.3).	292
Figure 6.4: Coefficient of Pressure from Euler2D Compared to Smith-Hess Solution.	293
Figure 6.5: Coefficient of Pressure from Euler3D Compared to Smith-Hess Solution.	293
Figure 6.6: Coefficient of Pressure through Euler3D Solution Domain.	294
Figure 6.7: Distribution of Density and Velocity from Euler3D Solution.	295
Figure 6.8: Mesh for NACA 0012 Airfoil (Mach 0.502, 1.77-deg AOA).	296
Figure 6.9: Pressure and Mach around NACA 0012 (Mach 0.502, 1.77-deg AOA).	297
Figure 6.10: Surface Pressure over NACA 0012 Airfoil (Mach 0.502, 1.77-deg AOA).	297
Figure 6.11: Mesh for RAE 2822 Airfoil (Mach 0.6, 2.57-deg AOA).	298

<u>Figure</u>	<u>Page</u>
Figure 6.12: Pressure and Mach around RAE 2822 Airfoil (Mach 0.6, 2.57-deg AOA).	299
Figure 6.13: Surface Pressure over RAE 2822 Airfoil (Mach 0.6, 2.57-deg AOA).	300
Figure 6.14: Pressure, Mach Number, and Entropy Distrib. near Cat's Eye (Mach 0.3).	300
Figure 6.15: Mesh for 6:1 Ellipse (Mach 0.3).	301
Figure 6.16: Pressure and Mach Distributions around Ellipse (Mach 0.3).	302
Figure 6.17: Surface Pressure over Ellipse (Mach 0.3).	302
Figure 6.18: Mesh for Circular Cylinder (Mach 0.3).	303
Figure 6.19: Pressure, Mach Number, and Entropy Distrib. near Cylinder (Mach 0.3).	304
Figure 6.20: Surface Pressure Distributions over Circular Cylinder (Mach 0.3).	305
Figure 6.21: Velocity Distributions over Surface of Circular Cylinder (Mach 0.3).	305
Figure 6.22: Mesh for Sphere (Mach 0.3).	306
Figure 6.23: Pressure and Mach around Sphere (Mach 0.3).	307
Figure 6.24: Surface Pressure over Sphere (Mach 0.3).	308
Figure 6.25: Pressure and Mach around NACA 0012 Airfoil (Mach 0.835, -0.13-deg).	309
Figure 6.26: Mesh for NACA 0012 Airfoil (Mach 0.835, -0.13-deg AOA).	310
Figure 6.27: Surface Pressure over NACA 0012 Airfoil (Mach 0.835, -0.13-deg AOA).	311
Figure 6.28: Pressure and Mach around RAE 2822 Airfoil (Mach 0.73, 2.8-deg AOA).	312
Figure 6.29: Mesh for RAE 2822 Airfoil (Mach 0.73, 2.8-deg AOA).	313
Figure 6.30: Surface Pressure over RAE 2822 Airfoil (Mach 0.73, 2.8-deg AOA).	314
Figure 6.31: Cast 7 Airfoil Mesh (Mach 0.765, 2.52-deg AOA).	316
Figure 6.32: Pressure and Mach Distrib. around CAST 7 (Mach 0.765, 2.52-deg AOA).	317
Figure 6.33: Surface Pressure Distrib. over CAST 7 (Mach 0.765, 2.52-deg AOA).	317
Figure 6.34: Cast 7 Airfoil Mesh (Mach 0.785, 2.52-deg AOA).	318
Figure 6.35: Pressure and Mach Distributions around CAST 7 (Mach 0.785, 2.52-deg).	319

<u>Figure</u>	<u>Page</u>
Figure 6.36: Surface Pressure Distribution over CAST 7 Airfoil (Mach 0.785, 2.52-deg).	319
Figure 6.37: Pressure and Mach around NASA 10% Supercritical (Mach 0.79, 2-deg).	320
Figure 6.38: Mesh for NASA 10% Supercritical Airfoil (Mach 0.79, 2-deg AOA).	321
Figure 6.39: Surface Pressure over NASA 10% Supercritical Airfoil (Mach 0.79, 2-deg).	322
Figure 6.40: Pressure Distribution over NASA 10% Supercritical Airfoil (Mach 0.7).	323
Figure 6.41: Pressure Distribution over NASA 10% Supercritical Airfoil (Mach 0.7)	323
Figure 6.42: Shock Tube Geometry	324
Figure 6.43: Pressure Slices through Shock Tube at Regular Intervals.	326
Figure 6.44: Density Slices through Shock Tube at Regular Intervals	327
Figure 6.45: Velocity Slices through Shock Tube at Regular Intervals.	328
Figure 6.46: Mesh and Pressure for Compression Corner at $M = 10$.	329
Figure 6.47: Pressure along Wall of Compression Corner, $M = 10$.	330
Figure 6.48: Pressure Profiles Downstream of Compression Corner versus Spacing.	331
Figure 6.49: Pressure Profiles Downstream of Compression Corner ($M = 2$).	332
Figure 6.50: Number of Elements versus Compression Angle and Mach Number.	334
Figure 6.51: Mesh and Density for Expansion Corner at $M = 5$.	335
Figure 6.52: Pressure along Wall of Expansion Corner, $M = 3$.	335
Figure 6.53: Number of Elements versus Expansion Angle and Mach Number.	336
Figure 6.54: Double-Wedge Airfoil Geometry (to scale).	337
Figure 6.55: Pressure and Mach Distributions for Double-Wedge Airfoil (Euler2D).	338
Figure 6.56: Pressure and Mach Distributions for Double-Wedge Airfoil (Euler3D).	339
Figure 6.57: Double-Wedge Airfoil Mesh (Euler3D).	339
Figure 6.58: Pressure, Density, Mach, and Entropy Distrib. for Double-Wedge Airfoil.	341
Figure 6.59: Mach Distrib. for Double-Wedge Airfoil at Mach 5 and 10 (Euler3D).	342

<u>Figure</u>	<u>Page</u>
Figure 6.60: Convergence of Energy Residual vs. CFL.	343
Figure 6.61: Wagner Solution from Euler2D Compared to Jones' Approximation.	345
Figure 6.62: Wagner Solution from Euler3D Compared to Jones' Approximation.	346
Figure 6.63: Snapshots of the Unsteady Velocity Distribution for Wagner Solution.	346
Figure 6.64: Snapshots of the Unsteady Pressure Distrib. for Wagner Solution.	347
Figure 6.65: Wagner Pressure Slices Up and Downstream of Airfoil.	348
Figure 6.66: LE and TE Wave Position as a Function of Time.	349
Figure 6.67: Subsonic (Mach 0.4) Constant Heat Generation.	354
Figure 6.68: Subsonic (Mach 0.4) Cosine Heat Generation.	355
Figure 6.69: Subsonic (Mach 0.4) Constant Mass Generation.	356
Figure 6.70: Subsonic (Mach 0.4) Cosine Mass Generation.	357
Figure 6.71: Subsonic (Mach 0.4) Constant Mass and Heat Generation.	358
Figure 6.72: Subsonic (Mach 0.4) Cosine Mass and Heat Generation.	359
Figure 6.73: Supersonic (Mach 2.0) Constant Heat Generation.	360
Figure 6.74: Supersonic (Mach 2.0) Cosine Heat Generation.	361
Figure 6.75: Supersonic (Mach 2.0) Constant Heat Generation.	362
Figure 6.76: Supersonic (Mach 2.0) Cosine Heat Generation.	363
Figure 6.77: Supersonic (Mach 2.0) Constant Mass Generation.	364
Figure 6.78: Supersonic (Mach 2.0) Cosine Mass Generation.	365
Figure 6.79: Supersonic (Mach 2.0) Constant Mass and Heat Generation.	366
Figure 6.80: Supersonic (Mach 2.0) Cosine Mass and Heat Generation.	367
Figure 6.81: Supersonic (Mach 7.0) Constant Heat Generation.	368
Figure 6.82: Supersonic (Mach 7.0) Cosine Heat Generation.	369
Figure 6.83: Supersonic (Mach 7.0) Constant Heat Generation.	370

<u>Figure</u>	<u>Page</u>
Figure 6.84: Supersonic (Mach 7.0) Cosine Heat Generation	371
Figure 6.85: Flow around Generic Hypersonic Vehicle and Exit of Combustor.	373
Figure 6.86: GHV Geometry and Mesh.	375
Figure 6.87: Local Mach Number around GHV at Mach 11 and 0o AOA.	375
Figure 6.88: Variation of Force and Moment Coefficients with Nozzle Length.	376
Figure 6.89: Variation of Force and Moment Coefficients with Lower Lip Length.	376
Figure 6.90: Variation of Force and Moment Coeff. with Mach Number with Stability.	377
Figure 6.91: Variation of Force and Moment Coeff. with Angle of Attack with Stability.	377
Figure 6.92: Variation Force and Moment Coeff. with Pitching Rate with Stability.	378
Figure 6.93: Local Mach along Centerline and Wall Surface of Rocket Nozzle.	379
Figure 6.94: Rocket Nozzle Mesh.	380
Figure 6.95: Local Mach Number within Rocket Nozzle: $p_t/p_{inf} = 1.67, 1.78, \text{ and } 2.$	381
Figure 6.96: Pressure, Mach, and Internal Energy for Subsonic Inlet at Mach 0.6.	383
Figure 6.97: Local Mach Number at Various Freestream Velocities.	384
Figure 6.98: Local Mach Number at 0 and 4-deg Angle-of-Attack.	385
Figure 6.99: Supersonic Normal Shock Inlet at Mach 1.5 and Mach 0.7.	385
Figure 6.100: Mesh for Supersonic Oblique Shock Inlet.	386
Figure 6.101: Local Mach Number for Oblique Shock Inlet at Two Mass Flow Rates.	387
Figure 6.102: Mesh for Coupled Turbojet.	388
Figure 6.103: Local Mach and Pressure around Coupled Turbojet.	388
Figure 6.104: Two Coupled Engines in one Domain.	389
Figure 6.105: Laminar Velocity Profile and Shear Stress for Coarse Meshes (NS2D).	391
Figure 6.106: Laminar Velocity Profile and Shear Stress for Fine Meshes (NS2D).	392
Figure 6.107: Suggested Meshing Scheme for Laminar Flows.	393

<u>Figure</u>	<u>Page</u>
Figure 6.108: Boundary Layer Profile Properties for Mesh 3.	394
Figure 6.109: Laminar Velocity Profile and Shear Stress for Coarse Meshes (NS3D).	395
Figure 6.110: Laminar Velocity Profile and Shear Stress for Fine Meshes (NS3D).	396
Figure 6.111: Boundary Layer Profile Properties for Mesh 3.	397
Figure 6.112: Effective Reynolds Number vs. Artificial Dissipation Scalar in NS3D.	398
Figure 6.113: Oscillating Wake behind Ellipse (Entropy, Rec = 4000, diss = 0.6).	399
Figure 6.114: Mesh for Ellipse (Rec = 4000).	400
Figure 6.115: Surface Pressure at Various Times over Ellipse (Rec = 4000, diss = 0.6).	400
Figure 6.116: Lift and Drag Histories for Ellipse (Rec = 4000, diss = 0.6).	401
Figure 6.117: Velocity Distribution near Circular Cylinder at Re = 1.54 (NS2D).	402
Figure 6.118: Velocity Distribution near Circular Cylinder at Re = 9.6 (NS2D).	402
Figure 6.119: Velocity Distribution near Circular Cylinder at Re = 26 (NS2D).	403
Figure 6.120: Cylinder at Re = 41: Velocity (top) and Entropy (bottom) (NS2D).	403
Figure 6.121: Oscillating Wake Behind Circular Cylinder (Entropy, Re = 105) (NS2D).	404
Figure 6.122: Oscillating Wake Behind Circular Cylinder (Entropy, Re = 200) (NS2D).	404
Figure 6.123: Entropy Distribution Downstream of Solid Fence.	407
Figure 6.124: Density Fluctuations Downstream of Solid Fence.	408
Figure 6.125: Entropy Showing Vortices Downstream of Solid Fence.	408
Figure 6.126: Entropy Distribution Downstream of Porous Fence.	409
Figure 6.127: Density Fluctuations Downstream of Porous Fence.	410
Figure 6.128: Entropy Showing Vortices Produced by Holes in Porous Fence.	410
Figure 6.129: Velocity Distribution around Equally Sized Holes	411
Figure 6.130: Velocity Distribution around Incremented Holes	411
Figure 6.131: Instantaneous Velocity Profile One and Two Fence Heights Downstream.	412

<u>Figure</u>	<u>Page</u>
Figure 6.132: Square Cavity Simulation (NS2D).	414
Figure 6.133: Mesh for Top of Cavities (L:D = 1:1, 1:2, and 4:1).	415
Figure 6.134: Entropy and Velocity in Square Cavity (L:D = 1:1).	415
Figure 6.135: Entropy and Velocity in Shallow Cavity (L:D = 4:1).	416
Figure 6.136: Entropy and Velocity in Deep Cavity (L:D = 1:2).	416
Figure 6.137: Pressure Frequency Spectra for 17 Locations around Square Cavity.	417
Figure 6.138: Pressure Frequency Spectra for 17 Locations around Shallow Cavity.	418
Figure 6.139: Lipped Cavity (Entropy, NS2D).	419
Figure 6.140: Acoustic Waves above Lipped Cavity (Density, NS2D).	420
Figure 6.141: Vorticity in Lipped Cavity Flow.	420
Figure 6.142: Grid Convergence of SA Velocity and Eddy Viscosity Profiles at $x = 7$.	423
Figure 6.143: Convergence of SA Skin Friction with Near-Wall Spacing.	423
Figure 6.144: Grid Convergence of Profile Properties with SA Turbulence Model.	424
Figure 6.145: Geometry for Rumsey's Flat Plate Grids (Mesh 4).	426
Figure 6.146: Skin Friction along Rumsey Plate for 5 Meshes in Family	427
Figure 6.147: Convergence of Velocity and Eddy Viscosity Profiles at $x = 0.97$.	428
Figure 6.148: Convergence of Velocity and Eddy Viscosity Profiles at $x = 1.83$.	428
Figure 6.149: Maximum Eddy Viscosity as a Function of Distance along Plate.	429
Figure 6.150: Rumsey's Coarsest Grid for Bump Plate Case.	430
Figure 6.151: Pressure Distribution over Bump Plate for Four NS2D Solutions.	430
Figure 6.152: Skin Friction along Bump Plate for Four NS2D Solutions.	431
Figure 6.155: Pressure, Velocity, and Eddy Viscosity Distribution around Bump Plate.	432
Figure 6.156: Velocity Vectors around Bump Plate.	432
Figure 6.153: Velocity and Eddy Viscosity Profiles at Top of Bump ($x = 0.75$).	433

<u>Figure</u>	<u>Page</u>
Figure 6.154: Maximum Eddy Viscosity along Bump Plate for Four NS2D Solutions.	433
Figure 6.157: Orthogonal Elements.	434
Figure 7.1: NACA 0012 Airfoil Mesh (Mach 0.3, 2-deg AOA).	438
Figure 7.2: Pressure Distribution around NACA 0012 Airfoil at Mach 0.3, 2-deg AOA.	438
Figure 7.3: Pressure Distribution around NACA 0012 Airfoil at Mach 0.3, 2-deg AOA.	439
Figure 7.4: Mesh for NACA 0012 Airfoil (Mach 0.502, 1.77-deg AOA).	440
Figure 7.5: Pressure and Mach Distributions around NACA 0012 Airfoil (Mach 0.502).	440
Figure 7.6: Pressure Distribution over NACA 0012 Airfoil at Mach 0.502, 1.77-deg.	441
Figure 7.7: Mesh for RAE 2822 Airfoil (Mach 0.6, 2.57-deg AOA).	442
Figure 7.8: Pressure Distribution around RAE 2822 Airfoil (Mach 0.6).	442
Figure 7.9: Mach Distribution around RAE 2822 Airfoil (Mach 0.6).	443
Figure 7.10: Pressure Distribution over RAE 2822 Airfoil (Mach 0.6, 2.57-deg AOA).	443
Figure 7.11: Pressure and Velocity around Inviscid Ellipse (Mach 0.3).	444
Figure 7.12: Ellipse Mesh.	445
Figure 7.13: Pressure over Inviscid Ellipses (Mach 0.3).	445
Figure 7.14: Inviscid Cylinder Mesh.	446
Figure 7.15: Pressure and Velocity Distrib. around Inviscid Cylinder.	447
Figure 7.16: Pressure Distribution over Cylinder.	448
Figure 7.17: Pressure and Velocity Distrib. along Sym Plane around Inviscid Sphere.	449
Figure 7.18: Inviscid Sphere Mesh.	450
Figure 7.19: Pressure Distribution over Sphere.	450
Figure 7.20: Mesh for NACA 0012 Airfoil (Mach 0.835, -0.13-deg AOA).	452
Figure 7.21: Pressure and Mach Distrib. around NACA 0012 Airfoil (Mach 0.835).	453
Figure 7.22: Pressure Distribution over NACA 0012 Airfoil (Mach 0.835, -0.13-deg).	453

<u>Figure</u>	<u>Page</u>
Figure 7.23: Pressure and Mach Distributions around RAE 2822 Airfoil (Mach 0.73).	454
Figure 7.24: Mesh for RAE 2822 Airfoil (Mach 0.73, 2.8-deg AOA).	455
Figure 7.25: Pressure Distribution over RAE 2822 Airfoil (Mach 0.73, 2.8-deg AOA).	456
Figure 7.26: Mesh for CAST 7 Airfoil (Mach 0.765, 2.52-deg AOA).	457
Figure 7.27: Pressure and Mach Distributions around CAST 7 Airfoil (Mach 0.765).	458
Figure 7.28: Pressure Distribution over CAST 7 Airfoil (Mach 0.765, 2.52-deg AOA).	458
Figure 7.29: Pressure Distribution over CAST 7 Airfoil (Mach 0.785, 2.52-deg AOA).	459
Figure 7.30: Mesh for CAST 7 Airfoil (Mach 0.785, 2.52-deg AOA).	460
Figure 7.31: Pressure and Mach Distributions around CAST 7 Airfoil (Mach 0.785).	460
Figure 7.32: Mesh for NASA 10% Thick Supercritical Airfoil (Mach 0.79, 2-deg AOA).	462
Figure 7.33: Pressure Distribution around NASA 10% Supercritical Airfoil (CFDsol).	463
Figure 7.34: Mach Distribution around NASA 10% Supercritical Airfoil (CFDsol).	463
Figure 7.35: Pressure Distribution around NASA 10% Supercritical Airfoil (Euler2D).	464
Figure 7.36: Mach Distribution around NASA 10% Supercritical Airfoil (Euler2D).	464
Figure 7.37: Pressure Distribution over NASA 10% Supercritical Airfoil (Mach 0.79).	465
Figure 7.38: Double-Wedge Airfoil Mesh (Mach 2, 2-deg AOA).	466
Figure 7.39: Flow around a Supersonic Double-Wedge Airfoil at Mach 2.	466
Figure 7.40: Pressure and Mach around Double Wedge Airfoil at Mach 2 and 2-deg.	467
Figure 7.41: Four Flow Regions around Double-Wedge Airfoil.	468
Figure 7.42: Convergence Rate of Energy Residual vs. tau.	468
Figure 7.43: Wagner Solution for NACA 0012 Airfoil (Mach 0.3, 2o AOA).	470
Figure 7.44: Snapshots of Unsteady Pressure Distribution for Wagner Solution.	470
Figure 7.45: Velocity Distribution around Pitching Airfoil (Non-Inertial).	471
Figure 7.46: Entropy Distribution around Pitching Airfoil (Non-Inertial).	472

<u>Figure</u>	<u>Page</u>
Figure 7.47: Lift and Drag History for Pitching Airfoil (Non-Inertial).	472
Figure 7.48: Moment History for Pitching Airfoil (Non-Inertial).	473
Figure 7.49: Velocity & Entropy around Plunging Airfoil (Non-Inertial).	474
Figure 7.50: Lift and Drag History for Pitching Airfoil (Non-Inertial).	475
Figure 7.51: Moment History for Pitching Airfoil (Non-Inertial).	475
Figure 7.52: Generalized Force for Pitching Mode (Transpiration).	476
Figure 7.53: Velocity (top) & Entropy (bottom) around Pitching Airfoil (Transp.).	477
Figure 7.54: Velocity Distribution around Plunging Airfoil (Transpiration).	478
Figure 7.55: Entropy Distribution around Plunging Airfoil (Transpiration).	478
Figure 7.56: Generalized Force for Plunging Mode (Transpiration).	479
Figure 7.57: Subsonic (Mach 0.4) Constant Heat Generation.	483
Figure 7.58: Subsonic (Mach 0.4) Cosine Heat Generation.	484
Figure 7.59: Subsonic (Mach 0.4) Constant Mass Generation.	485
Figure 7.60: Subsonic (Mach 0.4) Cosine Mass Generation.	486
Figure 7.61: Subsonic (Mach 0.4) Constant Mass and Heat Generation.	487
Figure 7.62: Subsonic (Mach 0.4) Cosine Mass and Heat Generation.	488
Figure 7.63: Supersonic (Mach 2.0) Constant Heat Generation.	490
Figure 7.64: Supersonic (Mach 2.0) Cosine Heat Generation.	491
Figure 7.65: Supersonic (Mach 2.0) Constant Heat Generation.	492
Figure 7.66: Supersonic (Mach 2.0) Cosine Heat Generation.	493
Figure 7.67: Supersonic (Mach 2.0) Constant Mass Generation.	494
Figure 7.68: Supersonic (Mach 2.0) Cosine Mass Generation.	495
Figure 7.69: Supersonic (Mach 2.0) Constant Mass and Heat Generation.	496
Figure 7.70: Supersonic (Mach 2.0) Cosine Mass and Heat Generation.	497

<u>Figure</u>	<u>Page</u>
Figure 7.71: Supersonic (Mach 7.0) Constant Heat Generation.	499
Figure 7.72: Supersonic (Mach 7.0) Cosine Heat Generation.	500
Figure 7.73: Supersonic (Mach 7.0) Constant Heat Generation.	501
Figure 7.74: Supersonic (Mach 7.0) Cosine Heat Generation.	502
Figure 7.75: Local Mach Number within Rocket Nozzle at $p_t = 1.67$ $p_{inf} = 0.84$ p_{design} .	503
Figure 7.76: Local Mach Number along Centerline and Wall Surface of Rocket Nozzle.	504
Figure 7.77: Moderate Mesh for Laminar Boundary Layer ($Re = 3600$).	506
Figure 7.78: Laminar Boundary Layer Results from CFDsol.	506
Figure 7.79: Course and Fine Meshes and Dim'less Velocity Profiles.	507
Figure 7.80: Effective Reynolds Number vs. Artificial Dissipation Scalar.	509
Figure 7.81: Pressure and Velocity around Viscous Ellipse ($Re = 4000$).	510
Figure 7.82: Ellipse Mesh.	511
Figure 7.83: Pressure over Inviscid & Viscous Ellipse (Mach 0.3).	511
Figure 7.84: Viscous Cylinder Mesh ($ReD = 9.6$).	512
Figure 7.85: Pressure and Velocity around Viscous Cylinder ($ReD = 9.6$).	513
Figure 7.86: Pressure and Velocity around Viscous Cylinder ($ReD = 41$).	514
Figure 7.87: Viscous Cylinder Mesh ($ReD = 41$).	515
Figure 7.88: Velocity and Eddy Viscosity for Turbulent Section.	516
Figure 7.89: Eddy Viscosity Profiles at Quarter Locations across Turbulent Section.	516
Figure 7.90: Velocity and Eddy Viscosity for Trans. Boundary Layer.	518
Figure 7.91: Velocity and Eddy Viscosity Profiles along a Flat Plate Boundary Layer.	519
Figure 7.92: Growth of Turbulence (Eddy Viscosity) along Length of Plate.	519

TABLE OF TABLES

<u>Table</u>	<u>Page</u>
Table 1.1: State-of-the-Art in Viscous Finite Element Solvers.	18
Table 4.1: Gauss Quadrature – Triangle Elements.	114
Table 4.2: Gauss Quadrature – Tetrahedra Elements.	114
Table 6.1: Number of Elements (N) versus Compression Angle and Mach Number.	333
Table 6.2: Number of Elements (N) versus Expansion Angle and Mach Number.	336
Table 6.3: Results from Euler2D Compared with Shock-Expansion Theory.	340
Table 6.4: Results from Euler3D Compared with Shock-Expansion Theory.	340
Table 6.5: Comparison of Mach 5 Solutions to SE Theory.	342
Table 6.6: Comparison of Mach 10 Solutions to SE Theory.	342
Table 6.7: Comparison of GHV Solution to SE Theory.	373
Table 6.8: Profiles Tested for SA Grid Convergence.	422
Table 7.1: Comparison to Theory for Double-Wedge Airfoil (Mach 2, 2-deg AOA).	467

LIST OF SYMBOLS

a	Acoustic speed ($=\sqrt{\gamma p/\rho}$)
$a_{o,i}$	Acceleration (vector) of non-inertial frame
$a_{t,i}$	Local mesh acceleration (non-inertial frame)
A_{be}	Area of boundary element (3D domain)
A_e	Area of domain element (2D domain)
\mathbf{A}_e	Inverse Jacobian matrix for element
A_i	Rows of inverse Jacobian matrix
\mathbf{A}^*	Riemann invariant matrix
\hat{b}	Unit binormal vector for boundary element ($=\hat{n}\times\hat{t}$)
B_{ij}	Transformation of the plane; rotation about center of frame
c_p	Specific heat under constant pressure
c_v	Specific heat under constant volume
\mathbf{C}	Damping matrix
d	Wall distance
	Number of dimensions in domain and governing equations
e	Internal energy ($=c_v T$)
$E, \rho E$	Total energy ($=\rho e + \frac{1}{2}\rho u_i u_i + \rho K$)
$E_r, \rho E_r$	Total relative energy ($=\rho e + \frac{1}{2}\rho(u_i u_i - V_t^2) + \rho K$)
F	Uninstalled thrust (turbojet engine)
\vec{F}	Rigid body forces
F_1, F_2	Interpolation functions (SST models)
\mathbf{f}^*	Pressure contribution vector, adapted
$\mathbf{f}_{1,e}, \mathbf{f}_{2,e}$	Pressure contribution vector
\vec{F}_c	Generalized forces on control surfaces
\vec{F}_d	Rigid body forces and moments ($=\vec{F}, \vec{M}$)
F_i	External forces on body
\mathbf{F}_i	Inviscid flux vector
$\mathbf{F}_{v,i}$	Viscous flux vector
\mathbf{f}_σ	Viscous boundary integral vector

\vec{g}	Gravity vector (body frame)
h	Enthalpy ($= c_p T = e + p/\rho$)
$H, \rho H$	Total enthalpy ($= \rho h + \frac{1}{2} \rho u_i u_i + \rho K$)
$H_r, \rho H_r$	Total relative enthalpy ($= \rho h + \frac{1}{2} \rho (u_i u_i - V_t^2) + \rho K$)
$\mathbf{I}, [\mathbf{I}]$	Identity matrix ($= 1$, if $i = j$; $= 0$, otherwise)
\mathbf{I}_m	Mass moment of inertial (inertial) matrix
\mathbf{J}_e	Element Jacobian matrix
$ J_e $	Element Jacobian ($= d! \Omega_e = l_e, 2A_e, 6V_e$)
k	Thermal conductivity
\mathbf{K}	Stiffness matrix
\mathbf{K}, \mathbf{K}_e	Element inviscid stiffness matrices, global and elemental
\mathbf{K}^*	Element inviscid stiffness matrices, adapted
$K, \rho K$	Turbulent kinetic energy ($= \frac{1}{2} \overline{\rho u'_i u'_i} = \frac{1}{2} \overline{\rho u''_i u''_i}$)
K_{amb}	Ambient (freestream) turbulent kinetic energy
\mathbf{K}_σ	Viscous element contributions matrix
l_{be}	Length of boundary edge (2D domain)
m	Mass of vehicle
\dot{m}	Mass flow rate, inlet (turbojet engine)
\dot{m}_f	Mass flow rate of fuel (turbojet engine)
$\dot{m}H$	Enthalpy flow rate (turbojet engine)
M	Mach number ($= \sqrt{u_i u_i} / a$)
\vec{M}	Rigid body moments
\mathbf{M}	Mass matrix ($= m \mathbf{I}$)
\mathbf{M}_C	Consistent mass matrices (global)
$\mathbf{M}_{c,e}, \mathbf{M}_{c,be}$	Consistent mass matrices (element and boundary element)
\mathbf{M}_L	Lumped mass matrix
M_T	Turbulent Mach number ($= \sqrt{2K} / a$)
\hat{n}, \hat{n}_i	Unit normal vector for boundary element
p	Static pressure
p_b	Static pressure along boundary
P_j	Momentum vector
p_t	Total pressure
Pr	Prandtl number ($= c_p \mu / k = 0.71$ for air)
Pr _T	Turbulent Prandtl number ($= c_p \mu_T / k_T = 0.9$ for air)
\dot{q}'''	Heat generation (quasi-combustion)
q''_b	Heat flux through boundary
q''_i	Heat flux
Q_i	Turbulent transport of heat ($= \overline{\rho u'_i h'} = \overline{\rho u''_i h''}$)

s	Entropy
S	Sutherland's coefficient (= 110.4 K = 198.72 °R for air)
	Strain tensor, magnitude (= $\sqrt{2S_{ij}S_{ij}}$)
S_{ij}	Strain tensor
\mathbf{S}	Source terms, integrated and combined
\mathbf{S}_C	Source term (combustion)
\mathbf{S}_{NI}	Source term (non-inertial)
\mathbf{S}_T	Source term (turbulence)
t	Time
\hat{t}	Unit tangent vector for boundary element
T	Static temperature
	Period of integration for time-averaging
T_b	Static temperature along boundary
\mathbf{T}_e	Inverse Jacobian dotted with velocity vector (= $u_{r,i} A_i / J_e $)
T_T	Total temperature
u_i	Velocity vector
$u_{r,i}$	Relative velocity vector (= $u_i - V_{t,i}$)
\mathbf{U}	Unknowns vector
$\mathbf{U}_e, \mathbf{U}_{be}$	Unknowns vector for element and boundary element
$\vec{V}_o, V_{o,i}$	Translational velocity (vector) of non-inertial frame (body fixed)
\vec{V}_b	Velocity (vector) of boundary
\vec{V}_d	Rigid body unknowns vector (= $\vec{V}_o, \vec{\omega}$)
V_e	Volume of domain element (3D domain)
\vec{V}_{nt}	Velocity vector in normal-tangential frame (= V_n, V_t, V_b)
$V_{t,i}$	Local mesh velocity (non-inertial frame)
x_i	Coordinates in global (inertial) frame
$x_{o,i}$	Center of non-inertial rotation (inertial) frame
$x_{b,i}$	Coordinates relative to center of rotation (non-inertial) frame (= $B_{ij}^T (x_i - x_{o,i})$)
\vec{x}_d	Rigid body unknowns vector (= $\vec{x}_o, \vec{\theta}$)
\vec{x}_n	Elastic unknowns vector (= $\vec{\delta}_n, \vec{V}_n$)
γ	Ratio of specific heats (= c_p / c_v)
Γ_{be}	Boundary element (= $\pm 1, l_{be}, A_{be}$)
δ_c	Deflection of control surface(s)
δ_{ij}	Kronecker delta (= 1 for $i = j$; = 0, otherwise)
$\Delta t, \Delta t_e, \Delta t_n$	Time step (global, elemental, nodal)
$\varepsilon, \rho\varepsilon$	Dissipation of turbulent kinetic energy
ε_{ijk}	Permutation tensor (= $(i - j)(j - k)(k - i) / 2$)

Φ_1, Φ_2	Interpolation variables (SST model)
Φ_e, Φ_{be}	Shape function for element and boundary element
$\vec{\theta}$	Euler angles as a vector (= ϕ, θ, ψ)
$[\theta]$	Transformation between global and normal-tangential frames
κ	von Karman constant, law of the wall
λ	Second viscosity ($\lambda \geq -\frac{2}{3}\mu$)
	Local flow characteristics (Riemann problem)
μ	Viscosity
μ_T	Eddy viscosity (effective viscosity of turbulence)
\hat{v}	Turbulent transport variable; SA variable
ξ_i	Coordinates in local (element) frame
$\Pi, \rho\Pi$	Production of turbulent kinetic energy
ρ	Density
$\dot{\rho}$	Mass generation (quasi-combustion)
ρ_b	Density along boundary
$\sigma, \sigma_k, \sigma_\omega$	Turbulent Prandtl number(s) for diffusion of \hat{v} , K , and ω
τ_{ij}	Viscous stress tensor
$T_{ij}, \rho T_{ij}$	Reynolds stress tensor (= $\overline{\rho u'_i u'_j} = \overline{\rho u''_i u''_j}$)
ω	Rate of dissipation of turbulent kinetic energy; SST variable
ω_{amb}	Ambient (freestream) rate of dissipation of turbulent kinetic energy
$\vec{\omega}, \omega_i$	Rotational rate (vector) of non-inertial frame
Ω	Vorticity tensor, magnitude (= $\sqrt{2\Omega_{ij}\Omega_{ij}}$)
	Rotational rate matrix (= $\vec{\omega} \times \mathbf{I}$)
Ω_e	Element domain (= I_e, A_e, V_e)
Ω_{ij}	Vorticity tensor
∇	Gradient vector
∇_{nt}	Gradient vector in normal-tangential frame

CHAPTER I

INTRODUCTION

Air-breathing hypersonic flight vehicles (AHFVs) represent the next generation of high speed military, civilian, and research aircraft. AHFVs have been proposed as military fighters, bombers, and transports to quickly move tactical personnel and equipment anywhere in the world in a matter of hours. Scramjet technology can be used to efficiently accelerate cruise missiles and piloted aircraft to speeds well beyond current designs. The commercial market also looks to AHFVs as the next generation of transportation for passengers and cargo. Cheng (1993) suggests that scramjet technology of the future will propel aircraft to the other side of the globe in a matter of hours. For NASA, AHFVs offer a promising alternative for the next generation of Highly Reliable Reusable Launch Systems (HRRLS). Since the catastrophic loss of the Space Shuttle Columbia in 2003, NASA and government officials have been searching for the next safe alternative to return astronauts and researchers to orbit, the moon, and even Mars.

AHFVs are highly complicated systems, where each subsystem interacts with all of the other subsystems. The science of hypersonic flows and scramjets, let alone complete AHFV systems, is slowly coming of age. Experimentalists are beginning to obtain repeatable results

and take measurements that can be used to validate analytical and numerical models (Bertin, 2006). Flight tests are also being conducted successfully; but the cost, not to mention time, to plan a hypersonic or scramjet-driven flight test is very great. Very few flights have been conducted; and of those tests, little data has been produced to track those flights. Even more difficult to the researcher “on the outside”, the results from most scramjet ground tests and flight tests go unpublished for national security and commercial design rights. No company or country wants to show their hand to anyone else before they have a finished product.

With our limited ability to adequately represent hypersonic flow experimentally, the challenge of hypersonic CFD predictions becomes even more difficult because substantial experimental data for a variety of flows and flight conditions are **not** available. (Bertin and Cummings, 2006, emphasis added)

AHFVs can be broken down into six major subsystems: Aerodynamics, elastics, heat transfer, propulsion, flight dynamics, and controls. For transition from subsonic to supersonic, the propulsion system is broken apart into a turbojet and a scramjet engine, increasing the complexity. Given all of these sub-systems and their interaction, it is a wonder that the hypersonic / scramjet community has reached the level of understanding and sophistication demonstrated by vehicles, such as the X-30 National Aerospace Plane concept and the X-43A unmanned test vehicle.

AHFVs see very high temperatures created by combustion, skin friction, and very high speeds. Most scramjets are designed to decelerate the flow to static temperatures of 1000 to 2000K before the flow enters the combustion chamber (Ferri, 1964). The temperature increases further through the combustion chamber. Temperatures greater than 2000K have

been measured at the sharp leading edge of the vehicle, when using passive cooling (Cheng, 1993). The straight passage ahead of the combustor (called the “isolator”) produces a shock-train. The complicated shock-shock interactions in this region can create very high local heating rates on the cowl lip, much higher than predicted by theory (Cheng, 1993).

Scramjet propulsion has come a long way in recent years, due to the use of CFD, computational combustion models, and experiments to verify those numerical models (Curran, 1996). The knowledge gained through computational means has allowed scramjets to evolve into a viable means of propulsion. Wind tunnel testing is complex due to the extreme speeds, heat, and loading, but successful wind tunnel tests have been conducted on scramjets. A handful of flight tests have been conducted under power, but a flight test vehicle adds more complexity from the coupling of flight-dynamic-propulsion modes.

According to Ferri (1964), a scramjet can operate with a fixed geometry inlet over a wide range of flight speeds. The properties at the entrance of the burner must vary with freestream Mach number to obtain reasonable flight performance. Scramjets operate efficiently at high speeds (Mach 4 or greater). At low supersonic speeds, the flow slows down to subsonic speeds in the combustion chamber, and the engine is referred to as a ramjet. Ramjets are designed using different principles, but advances have been made to use scramjets with subsonic combustion. One of these concepts uses *thermal choking* to accelerate the heated flow using a “nozzling” effect (Curran, 1996). Figure 1.1 shows the limitations of scramjets and ramjets, and the application of turbojets at subsonic speeds. Dual-mode engines are proposed that switch from subsonic turbojet propulsion to scramjet propulsion at supersonic speeds. Little effort has been made to model the transition between modes because of complexity.

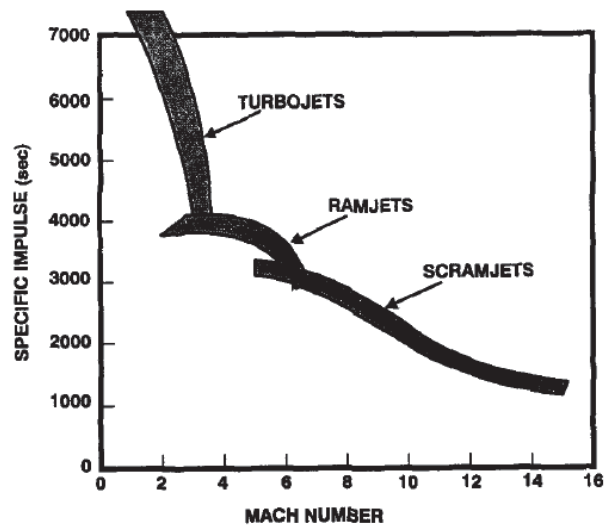


Figure 1.1: Specific Impulse vs. Mach Number (taken from Curran, 1996).

AHFVs exhibit many nonlinear characteristics: High skin temperatures are used to preheat the fuel (usually hydrogen) before being sprayed into the combustor (Ferri, 1964).

Preheating results in higher engine efficiency, greater thrust, and reduced skin temperatures, but now the temperature of the vehicle directly affects the propulsion system. Oxygen in the air dissociates at temperatures from 2000-4000K, and nitrogen dissociates between 4000-9000K. At 9000K, the oxygen and nitrogen molecules begin to ionize (Bertin, 2006). At elevated temperatures, radiation is also seen within the flow field.

Many researchers say that viscous effects also play an important part in the performance of AHFVs. Shock-shock interactions on the cowl lip and in the pre-combustion “isolator” are compounded by boundary layer interaction (Cheng, 1993). Such interactions have also been shown to produce locally severe heating on dummy ramjets mounted on the X-15 (Bertin, 2006). Shock-shock and shock-boundary layer interactions can only be predicted using time-accurate viscous analysis. Euler solutions have been coupled with boundary layer models to model steady solutions for simple geometries, and the Parabolized Navier-Stokes (PNS)

equations have been used to model more complex geometries. Time-accurate solutions are generated by reintroducing the upstream influences. The full Navier-Stokes equations are required to achieve the greatest space-time accuracy (Cheng, 1993). Hypersonic flow fields have been successfully modeled using continuum approximations for air ($\gamma = 1.4$) and combustion, but the full Burnett equations are necessary to ensure accuracy in rarefied flow fields at high altitudes.

Hypersonic aerodynamics can be simplified somewhat, with reasonable accuracy. The performance of wave-riders designed using viscous (boundary layer displacement) methods varied little from those designed with only inviscid methods (Cheng, 1993). But viscous methods with heat conduction are required to obtain the highest levels of time-accuracy. Air chemistry, outside of the combustion chamber, has little effect on the overall aerodynamic performance of the vehicle. Air chemistry is used to refine the local pressure and temperature distributions, elongated by shocks and rarefactions. At high altitudes (80-120km), the mean free path of the air particles becomes as large as the surface of the vehicle (Cheng, 1993). The no-slip condition cannot be strictly applied. Instead, the Knudsen number (a measure of the mean free path) can be used to create a *slip velocity boundary condition* instead of using the full Burnett equations.

The factors that limit the use of CFD in the early design phase are, how quickly the grids can be generated and how quickly ‘sufficiently’ accurate CFD simulations can be provided. For complex shapes, 10-100 CFD solutions with marginal fidelity but a very fast turn around time, would allow CFD to be part of this stage. (Papadopoulos, et al., 1999)

Advanced Engineering Solutions (AES) and NASA were attempting to develop a multi-disciplinary, physics based model for AHFVs. Several existing commercial and in-house codes were investigated for their ability to model AHFVs or their subsystems. AES approached researchers at Oklahoma State University (OSU) to investigate several in-house and NASA codes for their validity to the AHFV model. The available codes were found to have the ability to model one or more of the subsystems, but no code was found to have the ability to model the entire AHFV system. Stewart (2002) found that off-the-shelf CFD, structural FEA, and combustion models could be coupled by a user to model a scramjet system. Stewart found that repetitive gridding of deformed structures required the most time. NASA asked AES to combine several subsystem models together into a suite capable of the time-accurate dynamics of the AHFV system. AES desired to build a multi-disciplinary suite out of in-house codes NASA-STARS, OSU-Euler3D, and NASA-CFDsol. In-house codes were chosen because the codes can be tailored to improve run time, reduce overhead and data transfer, and simplify the integration of modules into the suite; and, Euler3D has the unique capability of modeling deformed structures without remeshing the solution. The suite of codes AES/MDA is broken down into NASA-STARS, an elastic deformation and heat conduction model; OSU-Euler3D, an inviscid aerodynamics model with incorporated flight dynamics and elastics models; and NASA-CFDsol, a viscous aerodynamics code. External user-defined modules will be integrated into the suite at a later time to act as controls and combustion models.

AES has asked OSU to update Euler3D and CFDsol in order to fulfill the aerodynamic, flight dynamic, and propulsion needs of the suite. Both solvers were outfitted with propulsion models for turbojets, rockets, and scramjets. The boundary conditions in CFDsol were

adapted to handle all of the possible flight regimes. Turbulence models, elastic boundary conditions, and non-inertial motion were also implemented in CFDsol.

Early in this project, the CFD solvers were demonstrated on aero-acoustic problems. This demonstration became a basis to investigate other needs throughout NASA and the surrounding industries. Through talking with researchers at NASA and aerospace contacts around Oklahoma, a need was identified for a comprehensive design tool for aero-acoustic-elastic analysis, controls, and simulation of tightly coupled aircraft systems. To give OSU the full capabilities of this work, NS3D was created by expanding Euler3D to include viscous, heat transfer, and turbulence models.

Examples in the area of aero-acoustics include, but are not limited to: The NASA SOFIA, a high-altitude telescope platform installed in the tail section of a Boeing 747SP; engine exhaust noise; bomb bay door design; open-air sensor packages; and instrumentation platforms. Panaras (1990) found that point vortices could be placed in an Euler solution to model vortex-acoustic interactions. The Euler solver advected the vortices downstream, but without some form of dissipation, artificial or natural, the solver could not generate or dissipate vorticity in the flow. Viscous dissipation allows for vortices to be generated from separated regions and dissipate naturally in the downstream flow. NS2D (Moffitt, 2004) was used to check the validity of the CFD solvers to model cavity flows.

NASA Aries design team desired a multidisciplinary tool to model aero-acoustic-elastic and their interaction with the vehicle motion throughout its launch and separation. Estimation of payload and crew vibrations is the primary purpose here. The noise from flaps, landing gear, gaps, and ledges is best predicted using aero-acoustic-elastic analysis. Gai (2000) showed

that transonic shocks are often felt further upstream than predicted by Euler solutions because of the “thickening” effects of the boundary layer. The boundary layer allows acoustic responses to travel upstream in wholly or locally supersonic flows. Lee (1990) found acoustic waves traveling from shock to trailing edge via the mean flow and similar responses traveling upstream through the boundary layer. A viscous solver is desirable for many aeroelastic cases especially at transonic speeds. Aeroelastics have already been proven in Euler3D through transpiration, but the concepts of viscous transpiration do not readily extend to viscous fluid dynamics. Transpiration theory is discussed in Chapter 3 along with possible research directions for pursuing viscous aeroelastics.

The multidisciplinary suite can be used in various aircraft designs. Conceptual designs can be tested for validity, and then trade studies can be conducted where analytical methods are not available. The initial sizing of vehicles can be tested for overall performance and stability. Structural loads can be estimated using steady or unsteady aerodynamic conditions. Engines can be integrated with inlet and nozzle designs to estimate their overall thrust and efficiency. A similar application is demonstrated on a simple hypersonic cross-section using the codes designed in this work.

Unmanned aerial vehicles (UAVs) are no longer limited by the presence of a pilot. These designs can be tailored to a particular mission through structural, aerodynamic, or controls design. Babcock (2004) and O’Neill (2004) showed with Euler3D could be used to produce linearized aircraft models for complex designs. The linearized models can be used to design controller and navigation systems. The guidance and control systems can then be tested in a high-fidelity environment to simulate the “real world” before being installed on the aircraft, and hybrid controllers can be tested in transition to ensure vehicle stability and orientation.

Aircraft modification companies can simulate proposed design changes in a high fidelity aero-acoustic-elastic environment before costly modifications are made on a vehicle.

Radomes, panels, and protruding sensors can be tested for changes in stall, stability, and power performance to minimize flight testing expenses.

The non-inertial frame can be used to model wind turbines, which require viscous calculations to capture power production accurately. The structural efficiency can also be tested.

Some turbine blades are designed to turn at a constant speed and stall at a certain wind speed so that the structure and electrical generator are not overloaded. Others turbines are designed to generate power according to their speed. Other non-inertial solvers utilize constant speeds without acceleration, so these solvers cannot model startup or active controlling.

1.1 State-of-the-Art

This section gives a comparison between Euler3D and CFDsol with the current state-of-the-art in finite element methods. The chapter is broken down into subsections that describe popular methods, numerical stability, and other features of the finite element method in fluid dynamics. The chapter culminates with a survey of four decades of viscous finite element research and a comparison of the state-of-the-art with the proposed research codes.

1.1.1 Discretization Methods

The finite element discretizations in fluid dynamics resemble those used in structural and heat conduction FEA. Actually, the fluids methods get their roots from the structural and heat transfer methods developed years before finite element research moved into fluids.

Baker (1983) wrote a book for those who had previous experience in structural finite

elements and wanted to expand to solving fluid dynamics problems. Zienkiewicz (2000) also gives a good summary of each method.

Galerkin (G): In the Galerkin (or Bubnov-Galerkin) method, the governing equations are scaled by a weighting function. Traditionally, the Galerkin weighting function is chosen to be the same as the shape function used to distribute properties throughout the domain. The scaled equations are then integrated over the entire domain. The property shape function is defined so that the shape function is non-zero on a given element, and zero everywhere else. Galerkin methods of various orders and distributions have been demonstrated in the literature. Variants of the method have used different weighting functions for different equations or different shape functions to represent the distribution of properties throughout the domain.

Discontinuous-Galerkin (D-G): Discontinuous-Galerkin is derived in a similar form to the traditional Galerkin method, except each element is allowed to be discontinuous from its neighbors. In other words, the solution is truly piecewise. The elements are tied together using inter-element fluxes along their boundaries. Therefore, discontinuous elements are attractive for higher-order methods and parallel processing because the method is solved element-by-element with little interaction between the elements.

Petrov-Galerkin (PG): Petrov-Galerkin is a variant of the Galerkin method, where the weighting function is skewed along the flow direction. The traditional Galerkin method creates equations that are similar to central differencing in finite difference applications, whereas Petrov-Galerkin resembles upwind-differencing in the analogy. The “upwind” calculations give the Galerkin method more stability.

Taylor-Galerkin (TG): Taylor-Galerkin is very similar to traditional Galerkin; so much, that a first-order time approximation results in the same system of equations for the two methods. In the Taylor-Galerkin method, the governing equations are discretized in time using a Taylor expansion. The time-expanded equation is discretized in space using the Galerkin method. (This method is demonstrated in the discussion of the CFDsol formulation.) Higher order methods can create terms with mixed temporal and spatial derivatives. Variations of the method discretize the spatial terms first and create a Taylor expansion in a later step, but the basic concept is the same for all Taylor-Galerkin solvers.

Characteristic-Based-Split (CBS) Algorithm: The CBS algorithm is useful to solve the incompressible Euler or Navier-Stokes equations by dividing the momentum update into a two steps, like a predictor-corrector. The momentum equation is used to update the velocities (without pressure terms) and then corrected using pressure, in a way that satisfies the continuity equation. The Galerkin method is used to discretize the governing equations for the predictor step. Compressible variants of the CBS algorithm have been created using artificial compressibility (AC) terms. The artificial compressibility changes the way the pressure correction is formulated and how the governing equations interact with density.

Penalty Methods (PM): The penalty method is useful for problems where pressure is not a dominant factor or where pressure is loosely tied to the velocity profile (incompressible flows). The pressure term is replaced by a penalty function that links pressure to the divergence of velocity. The penalty function is then inserted into the governing equations (usually Galerkin). The result is a system of momentum equations that are only dependent on velocity. The pressure is calculated using the penalty function. The penalty scalar, used in the penalty function, must be determined *a priori* through experimentation. Once the penalty

scalar is determined, the system has been reduced in order by eliminating pressure from the unknowns. Heinrich (1981) suggests that the method cannot be explicit stepped due to instabilities.

Least-Squares Method (LSM): Least-squares discretizations are a minimization of a least-squares function that represents the residual norms of the differential equation. The boundary conditions are often incorporated into the least-squares function. Great care must be taken when choosing the norms by which the residuals are measured. Improper norms result in an ill-posed problem. Some supporters of the least-squares method say that the method has better stability than the Galerkin method and its variants. For example, de Sampaio (2005) points out terms in the derivation of the least-squares method that closely resemble dissipative terms, like those found in SUPG. With these terms, many least-squared methods do not require as much artificial dissipation, if any at all.

Hybrid Finite Elements: Traditional methods calculate the gradient of properties using the element shape function so that the resulting gradient is one order less than the shape function. The viscous stresses and heat fluxes are calculated from the gradients, so the distribution of viscous fluxes is one order lower than the shape function. The hybrid method also distributes the viscous quantities using the property shape functions, which increases the viscous distribution by one order. The viscous terms are calculated using supplemental Galerkin equations constructed using their mathematical definition (i.e., viscous stress of a Newtonian fluid and Newton's law of cooling). The resulting viscous contributions are much smoother within the domain but suffer from increased noise near the boundaries of the computational domain.

Other Methods: There are many other discretization methods that have been adapted from finite difference and finite volume: Marker-and-cell (MAC) is a finite difference technique that has been expanded to the finite element method and renamed the Generalized and Simplified Marker-and-Cell (**GSMAC**) method. Tanahashi (1990) uses edge based velocity information and cell-centered total energy on quadrilateral elements using the GSMAC method. Luo (1998) and Arminijon (1999) combine the finite volume and finite element techniques on the same set of governing equations in what they both call a Mixed Finite Volume-Finite Element (**MFVFE**) method. The method applies finite volume principles to portions of the governing equations and finite element weighting to the other portions. First-order finite volume methods can be thought of as a subset of the Galerkin method, where the weighting function is equal to unity everywhere. The generalized minimal residual (**GMRES**) method is an iterative algorithm that can be applied to any of the discretization methods above. GMRES has roots in finite difference and is outlined in nonlinear form (Bristeau, 1990) for use in finite element methods.

1.1.2 Numerical Dissipation Methods

Every discretization of the governing equations of fluid flow, whether finite difference, volume, or element, suffers moderate to serious instabilities that result in “wiggles” in the discretized solution. These perturbations from the desired solution are easily minimized using an artificial, or numerical, dissipation. Some methods promise “inherent stability” without using a numerical dissipater. These methods contain a form of numerical dissipation within the discretization method, such as the two examples below that allow the system of equations to have enough stability to maintain its own solution. The resulting solutions from these methods still contain small errors due to the “inherent” dissipaters.

Streamline Upwind Petrov-Galerkin (SUPG): SUPG is a numerical stabilizer, whereas Petrov-Galerkin, where SUPG derives its name, is a discretization method. The difference being: All of the equations in a Petrov-Galerkin discretization are skewed upwind, whereas only the momentum equations are skewed by SUPG. SUPG creates a numerical dissipation term that resembles natural dissipation and then scales that term automatically to achieve natural curvature in the solution. Supporters suggest that the solution retains its complete accuracy, but opponents show that the solution is still skewed and has “slight” inaccuracies due to the skewed distribution.

Two-Level Finite Element Method (TLFEM): TLFEM uses element bubble functions, considered state-of-the-art by some, to create a dissipative component very similar to SUPG. The bubble function is evaluated using a second solution on a refined (subdivided) mesh. Traditional bubble functions require an *a priori* scalar that is often difficult to calculate, but TLFEM eliminates the scalar by calculating its stabilization from the two solution levels.

1.1.3 Implicit / Explicit Derivation

The spatial discretization can be written in full spectrum of manners that generate fully implicit, fully explicit, and partially implicit derivations. Traditionally, implicit (**Imp**) derivations are absolutely stable, but the governing equations for fluid flow (Euler or Navier-Stokes) are non-linear partial differential equations that cannot be inverted when written in implicit form. Two approaches are possible with implicit derivations: Fixed-point iteration of the implicit equations, or linearization (**L**) of the governing equations. Linearization rewrites the advection term using previous and subsequent velocities so that the governing equations can be written in matrix form.

Explicit (**Exp**) derivations suffer from greater numerical stability issues, including those written in implicit form, but explicit solvers are often more accurate once their stability has been achieved. Partially implicit solvers (**I / E**) try to combine the stability of implicit solvers and accuracy of explicit solvers. One very famous partially implicit derivation is the Crank-Nicolson (**C.N.**) method, which calculates half of the spatial contributions in terms of the previous step and half in terms of the next, making the derivation 50-50 implicit-explicit. Partially implicit scheme suffer numerical stability limits and slight inaccuracies at all advancement step sizes.

1.1.4 Comparison to State-of-the-Art

Table 1.1, on the following page, shows 34 viscous finite element applications. This list is not exhaustive of the viscous FEM in the literature, but rather the list is intended to be a comparison of the development of viscous FEM through time. The list was created from a survey of viscous literature, leaving out incompressible streamline-vorticity and velocity-vorticity formulations, which are meant to “model” the full Navier-Stokes equations instead of actually solving the equations directly. Book references were avoided where possible, since most book authors compile many different methods into their books. The books that are included represent one main derivation and give slight variations to that derivation. Journal articles were selected that represent the initial derivation of the solver. Articles referring back to a derivation were considered applications of the original solver and were left off the list. Finally, mathematical literature seeking to investigate stability or error estimation is not listed in Table 1.1 unless the article showed enough validation to prove that the derivation was intended to be used in an application instead of as an academic exercise.

CFDsol (Gupta, 2007) has been included in this list for comparison. (Euler3D, being an inviscid code, has been left off of the list.) Many of the codes listed in Table 1.1 were generated from the “big names” in the finite element fluids community: Zienkiewicz, Taylor, Oden, Peraire, Baker, Agarwal, Brooks, Hughes, Glowinski, and Periaux. (Many listed as subsequent authors.) These names appear more frequently in finite element mathematics, development, and applications literature, and many of these authors have written books on finite elements in fluids.

The applications listed in Table 1.1 are dominated by implicit (or partially implicit) Galerkin solvers with first order discretizations in space and time. Most of the solvers are two-dimensional and use triangles or quadrilateral to break up the solution domain. Seven of the solvers are three-dimensional, of which, four discretize their domains using tetrahedral. The list is split evenly between incompressible and compressible solvers, but incompressible applications are found in far greater frequency in the literature as a whole. Eight of the solvers, the majority shown, explicitly define SUPG as their method of dissipation. (Over half of the codes listed in Table 1.1 did not explicitly state a form of numerical dissipation. Even though none is stated, some inherent or additional dissipation must have been incorporated into these solvers. Computational fluids researchers are often guarded when discussing their numerical stability.)

CFDsol, and the proposed NS3D, are both Galerkin methods that are first order in space and time of the compressible Navier-Stokes equations. (CFDsol is a Taylor-Galerkin method by derivation, but its first order nature in time results in the same equations as the first order finite difference equations in Euler3D and NS3D.) Both derivations are partially implicit: CFDsol uses a Crank-Nicolson approach, whereas NS3D is derived using a fully implicit

method and stepped using an explicit predictor-corrector algorithm (Cowan, 2003). Euler3D is derived in fully nonlinear form, while CFDsol takes advantage of a “linearization” of the advection terms. Both solvers use unstructured tetrahedral elements to breakup their solution domains and have some form of artificial dissipation. Artificial dissipation was selected by both Gupta and Cowan to be a direct and simple method for enhancing numerical stability. SUPG is popular, but the method indirectly hides the numerical dissipation from the user, and the amount of solution “skewing” is less easily manipulated. Derivations for Euler3D, CFDsol, and NS3D are shown in later chapters.

1.2 Comparison of Turbulence Models

Hundreds of turbulence models are available in the literature. Each model is more applicable to certain flows and less accurate for others. Some models lend themselves to the structured geometries and finite difference techniques in which most models were developed. Other models are independent of technique or geometry but suffer from a lack of accuracy for all cases. Some cases have complicated algebraic or differential equations that would require more computational power, compared to other models that are over-simplified, computationally efficient, and do not provide a sufficient model for any specific flow field.

With these concerns in mind, several points are set aside that are required and/or desirable for implementation in the current work:

- Unstructured Mesh: Minimize references to geometry and orientation / direction.
- Compressible Model: Approaches incompressible model as Mach goes to zero.
- Complete Model: Governing equations are closed with other differential or algebraic equations, not left open for interpretation.
- No Adjustable Factors: Minimal user inputs other than freestream conditions.
- Accurate in All Regimes: Laminar, turbulent; subsonic, supersonic, even hypersonic.
- Handles Complex Flows: Boundary layer, wake, shear layer, jet, and separated flows.

Table 1.1: State-of-the-Art in Viscous Finite Element Solvers.

Date	Author	Method	In/Comp	Imp/Exp	Temporal	Spatial	Un/Struct	Element	Dissipation
1971	Tay	LSM	Incomp			1st	Struct	Tri	
1979	Bercovier	G, PM	Incomp			1st	Struct	Quad	
1981	Heinrich	G, PM	Incomp	Imp		1st, 2nd	Struct	Quad	
1982	Brooks	PG	Incomp	Imp	1st, P-C	1st	Struct	Quad	SUPG
1984	Mohr	G	Incomp	I / E		2nd	Struct	Tri	
1990	Bristeau	G, GMRES	Comp	Imp	1st / 2nd	1st	Unstruct	Tri	
1990	Jiang	LSM	Incomp			1st	Struct	Quad	
1990	Tanahashi	GSMAC	Incomp	Exp	1st or P-C	1st		Quad	
1991	Fernandez	G	Comp	Imp	1st or P-C	1st	Struct	Tri	Upwinding
1992	Marcum	G	Comp	Exp	RK or L-W	1st	Unstruct	Tetra	2nd & 4th RK; 2nd L-W
1992	Tworzydło	L, TG	Comp	I / E	2nd	1st	Struct	Quad	A.D.
1994	Kallinderis	G	Incomp	I / E		1st	Struct	Quad, Tri (sub)	4th-order Smoothing
1994	Soulaimani	LSM, GMRES	Comp	Imp	2nd	1st, 2nd (V)	Unstruct	Tri	SUPG
1995	Barsoum	PG		Imp	1st or P-C	1st		Quad	
1996	Mittal	G, GMRES	Comp		2nd		Struct	Quad	SUPG
1997	Masud	LSM	Comp	Imp	1st or P-C	1st, H.O.	Unstruct	Tri	SUPG
1998	Bonhaus	G, GMRES		Imp	1st or P-C	1st, H.O.		Tri	SUPG
1998	Luo	GMRES, MFVFE	Comp	Imp	1st	1st	Unstruct	Tetra	
1998	Massarotti	CBS	Incomp	I / E	1st	1st	Unstruct	Tri	
1999	Arminjon	MFVFE	Comp	Exp	2 step P-C	1st	Unstruct	Tri	
1999	Baumann	D-G	Comp	Imp	1st	H.O.	Struct	Brick	
1999	Whiting	LSM		Imp	2nd or P-MC	1st, H.O.		Tri	SUPG
2000	Jakobsen	PG						Quad	SUPG
2000	Kashiyama	PG	Incomp	I / E	1st	1st, 0th (p)	Unstruct	Brick	SUPG
2000	N'dri	G	Incomp	Imp	1st	1st	Struct	Tri	
2000	Wood	TG	Comp	Exp	1st	1st	Unstruct	Tri	
2002	Burbeau	D-G			RK	H.O.	Unstruct	Tri	
2004	Dolejsi	D-G	Comp	I / E	1st	1st	Unstruct	Tri	
2005	de Sampaio	LSM	Incomp	I / E	1st	1st	Unstruct	Tri	SUPG-like
2006	Nithiarasu	CBS	Inc / AC	Exp	1st	1st	Unstruct	Tetra	A.D.
2006	Pontaza	LSM	Comp	I / E	1st	1st	Struct	Brick	
2007	Gupta	L, TG	Comp	I / E	1st, C.N.	1st	Unstruct	Tetra	A.D.
2008	Cheng	G	Incomp	Imp	Freq Dom	2nd	Struct	Brick	
2008	Nesliturk	G	Incomp	I / E	1st	1st	Struct	Tri	TLFEM

Key:

G = Galerkin	Incomp = Incompressible N-S
TG = Taylor-Galerkin	Comp = Compressible N-S
PG = Petrov-Galerkin	AC = Artificial Compressibility
D-G = Discontinuous Galerkin	
LSM = Least-Squared Method	1st = 1st-order
GSMAC = Generalized Simplified Marker-and-Cell	2nd = 2nd-order
MFVFE = Mixed Finite Volume & Finite Element	C.N. = Crank-Nicolson
GMRES = Generalized Minimal Residual	P-C = Predictor-Corrector
CBS = Characteristic-Based-Split Algorithm	L-W = Lax-Wendroff
PM = Penalty Method	P-MC = Predictor-Multicorrector
L = Linearized Governing Equations	Freq Dom = Frequency Domain
	RK = Runge-Kutta
	H.O. = Higher-order
SUPG = Streamline-Upwind Petrov-Galerkin	Tri = Triangles
TLFEM = Two-Level Finite Element Method	Quad = Quadrilaterals
A.D. = Artificial Dissipation	Tetra = Tetrahedra
	Brick = Brick (3D quads)
Imp = Implicit	
Exp = Explicit	
I / E = Partially Implicit	

Mixing Length Models. Mixing length models specify a length scale for the flow, called a mixing length. Sometimes the mixing length varies with distance from the wall. Generally,

the mixing length is calculated empirically or used in analytical models. Mixing lengths are not transferrable from one flow region to another. For instance, the mixing length for a boundary layer is different than that used for a wake, or a jet, or a duct flow. Each mixing length is tuned to give a particular production/dissipation rate, momentum exchange, or other experimental quantity. Mixing length models are highly useful for developing analytical solutions to flow fields. The resulting analytical distribution that falls out from using the mixing length is used to quantify the flow.

Algebraic Models. Models like Cebecchi-Smith, Baldwin-Lomax, and Johnson-King use algebraic equations to specify the distribution of eddy viscosity or mixing length for a wide variety of situations. Cebecchi-Smith and Baldwin-Lomax are two-layer, composite functions that model the mixing length near the wall using van Driest's damping function and outer regions with a different mixing length function. Johnson-King (1985), often referred to as the 1/2-Equation model, is a two-layer, composite function with a van Driest damping function near the wall and an ordinary differential equation for the maximum shear stress for the remaining field. These models are not as accurate as their differential counterparts, especially in representing the advection and diffusion of turbulence along the direction of the flow; but, robust algebraic models, such as Johnson-King, have been much more accurate than very complicated, modern models in transonic applications. These models suffer from complicated calculations and IF statements.

One-Equation Models. The most simple differential equation models use a single differential equation to represent the transport of turbulent kinetic energy or a similar quantity through the flow. The model contains advection, diffusion, and source terms to represent such transport. Prandtl developed a k -model that closed the k -equation with a turbulence length.

The turbulent length was not directly a mixing length, but the empirical application of two lengths in the model is very similar. Baldwin-Barth (1990) created a differential equation representing the variation of *turbulence Reynolds number* in a field. The differential equation is supported by three functions and seven coefficients. Baldwin-Barth is a complete model with no adjustable coefficients.

Spalart-Allmaras (SA) Model. The Spalart-Allmaras (1992) model is another one-equation model that uses a differential equation in eddy viscosity to represent the transport of turbulence. The differential equation is supported by ten functions and eight coefficients that have been tuned for attached, external aerodynamic flows. The model is very reasonable for fuselages, nacelles, and wings but falls short on separated, internal, and radial jet flows. The model contains advection, diffusion, and source terms along with a unique dissipation term for “model stability”. The source terms utilize both the strain in the flow and distance to the wall, similar to a mixing length. Spalart-Allmaras is a very fast, complete model developed for aerodynamic drag estimation.

k - ϵ Models. The k - ϵ models use two differential equations to represent the transport of turbulent kinetic energy and its dissipation. These two variables are then combined to calculate the eddy viscosity on the field. The Standard k - ϵ Model (Jones and Launder, 1972) is the most simplified form. Yakhot and Orszag (1986) added small-eddy corrective functions using renormalization group theory, giving their model the name RNG k - ϵ . Other modelers, such as K.Y. Chien (1982), modified the ϵ -equation with empirical damping functions in the near-wall (low Reynolds number) regions. k - ϵ models are robust transport

models, but exhibit poor performance in severe and adverse pressure gradients and strong streamline curvatures.

k- ω Models. Kolmogorov posed the first *k- ω* model long before it could be realized in computational applications. Since Kolmogorov, *k- ω* modelers have not reached a consensus on what ω officially represents. Wilcox (1988, 2002) has brought much publicity to the *k- ω* model and its success. *k- ω* models are applicable to strong and adverse pressure gradients including separated flows and can be used to model transitional boundary layers, although the models tend to be highly sensitive to freestream conditions.

Mentor SST Model. Mentor (1992, 1994) developed a Shear-Stress Transport model, which combines *k- ω* and *k- ϵ* , has become widely used and respected for many wall bounded flows. The combination makes the SST model less susceptible to freestream conditions and more applicable to separated flow.

Other Two-Equation Models. Other two-equation models are also available; most coupling the turbulent kinetic energy with a second differential variable. Rotta developed a pair of equations in *k* and the integral length scale *l*. Zeierman and Wolfshtein present a *k-k τ* model, where the integral time scale τ that represents the time scale its turbulent kinetic energy. Speziale, Abid, and Anderson separate the variables by reformulating the *k- ϵ* model into a *k- τ* model with the transport of the integral time scale τ . Each of these two-equation models couples *k* with a measure of the rate that turbulent kinetic energy dissipates. These models have demonstrated their accuracy but never reached the popularity of *k- ϵ* and *k- ω* models.

Reynolds Stress Transport (RST) Models. The most complex and computationally expensive turbulence models avoid the Boussinesq approximation by directly modeling the anisotropic flow field with differential equations in each Reynolds stress. The differential equations represent the advection, diffusion, and production of Reynolds stresses in their transport throughout field. RST models create a unique problem in closing their many complex equations. Launder, Reece, and Rodi (1975) pose a pressure-strain model using purely kinematic considerations. Speziale, Sarkar, and Gatski (1991) present a similar but nonlinear model for pressure-strain, known as SSG in its incompressible form. Each model requires an additional closure equation, usually the ε - or ω -equation. The Launder-Reece-Rodi model uses the ε -equation; Wilcox and Rubesin (1980) use the ω -equation for closure. RST models replace the isotropic k -equation in two-equation models with differential equations representing the Reynolds stress tensor, adding five differential equations. RST models become more computationally expensive because of the additional equations, but the directional modeling of the Reynolds stresses makes RST models more accurate for anisotropic flow fields, like separated boundary layers, vortical wakes, and other highly-complex classes of flows.

Large Eddy Simulations (LES). Kolmogorov proposed that the large scales of turbulent flows are dependent on geometry while the smaller scales are almost universal. Large eddy simulations resolve the “large eddies” with a filtered Navier-Stokes (N-S) equation and appropriate mesh sizing. “Small eddies” are resolved through a subgrid-scale model. LES models use the Boussinesq approximation to close the filtered N-S equations, closely resembling RANS. The differences arise in the “large eddy” filter used on the N-S equations and “small eddy” subgrid models. Smagorinsky (1963) uses a simple volume-based strain model to calculate the subgrid-scale (SGS) viscosity. Nicoud and Ducros (1999) propose a

wall-adapting local eddy-viscosity (WALE) model that uses a nonlinear strain tensor function to calculate the eddy viscosity. LES models are very applicable to modeling the physics of flows while their applications to engineering problems are often brought into question because LES models require very tight meshes near walls in order to resolve the flow. LES models are most useful for smooth visualizations of flow fields.

Direct Numerical Simulation (DNS). Direct Numerical Simulation models turbulent fluid dynamics with highly accurate Navier-Stokes codes. DNS requires that the full range of turbulent scale be resolved in both time and space. Such refinement requires the spatial mesh and temporal discretization are smaller than their respective Kolmogorov scales. These requirements restrict DNS to a subset of applications where high-speed computing is used to solve specific problems. DNS is generally used in conjunction with experimental data to gain a better understanding of specific flow fields or comparing to other models.

Direction of Current Research. This research will concentrate on the Spalart-Allmaras (SA) and Menter's SST models. Completed compressible models can be found for each. These models work well on unstructured meshes without the need for adjustable factors. The SST model uses composite functions to interpolate between the $k-\varepsilon$ and $k-\omega$ models, but such interpolation brings the best of both models together. Both models require knowledge of the geometry through distance to the nearest wall. The Spalart-Allmaras model is suggested for aerodynamics of attached, external flows without radial jet exhausts. The SST model is applicable to separated and internal flows in a variety of fluids, regimes, and complexities. These two models represent a wide range of applications with years of experience in the literature while minimizing computational requirements and coding complexity.

CHAPTER II

PROBLEM STATEMENT

Advanced Engineering Solutions (AES) was contracted by NASA to develop a model for AHFVs and other advanced aerospace applications. AES has asked OSU to assist in this effort by developing our inviscid solver Euler3D and a NASA viscous solver CFDsol to be incorporated into a suite of codes called AES/MDA. The suite will operate in two ways: The suite will act as a lower order model that can be used to aid in initial design and trade studies. The suite can also operate as a high-fidelity, coupled system of modules, where each module represents one subsystem of AHFVs or other aerospace systems.

In the lower order system model, illustrated as a flow diagram in Figure 2.1, the flow solver generates a steady flow solution using *a priori* combustion, rocket, and turbojet models. The temperature along the solid surface of the body is passed to the conduction model, where a steady state temperature distribution is generated throughout the vehicle. The elastic properties in the structure are varied according to this temperature distribution, and then elastic mode shapes and system matrices are passed back to the CFD solver. The mode shapes and rigid body dynamics are used in unsteady or deformed steady computations. An external controls routine is optional.

In the higher order model, illustrated in Figure 2.2, the process begins similar to the low order model, where the steady flow and heat transfer solutions create the initial conditions. These conditions will then be perturbed using a completely coupled system. The CFD model passes boundary temperatures to the heat conduction routine, which passes back boundary heat fluxes. (These two conditions have been selected to help couple the two solvers into a seamless computational domain, where the boundary flux accommodates the flow solution the best and the boundary temperature simplifies the conduction solution.) The elastics module uses conduction temperature profile and CFD pressures to calculate deflections and velocities at all points on the solid surface, which are passed back to the CFD solver. The CFD solver and propulsion models interact by passing the most recent boundary properties back and forth. Flight dynamics and controls will interact with the CFD solver in the same manner as described for the low order model. The higher-order methods can be mixed with lower-order methods to create the desired solution fidelity to match a given problem. Ideally, the CFD solvers be used exchanged smoothly for inviscid or viscous flow solutions.

The suite is setup in a modular nature so that the various components can be replaced in the future to update any subsystem model or to utilize any user specified model. AES and NASA desire physics-based models with various levels of fidelity to be used in conceptual sizing, design optimization, and accurate full-system performance estimation. Much of the current system is already in place and in use. Later sections discuss the necessary steps to incorporate the current flow models into the desired system models.

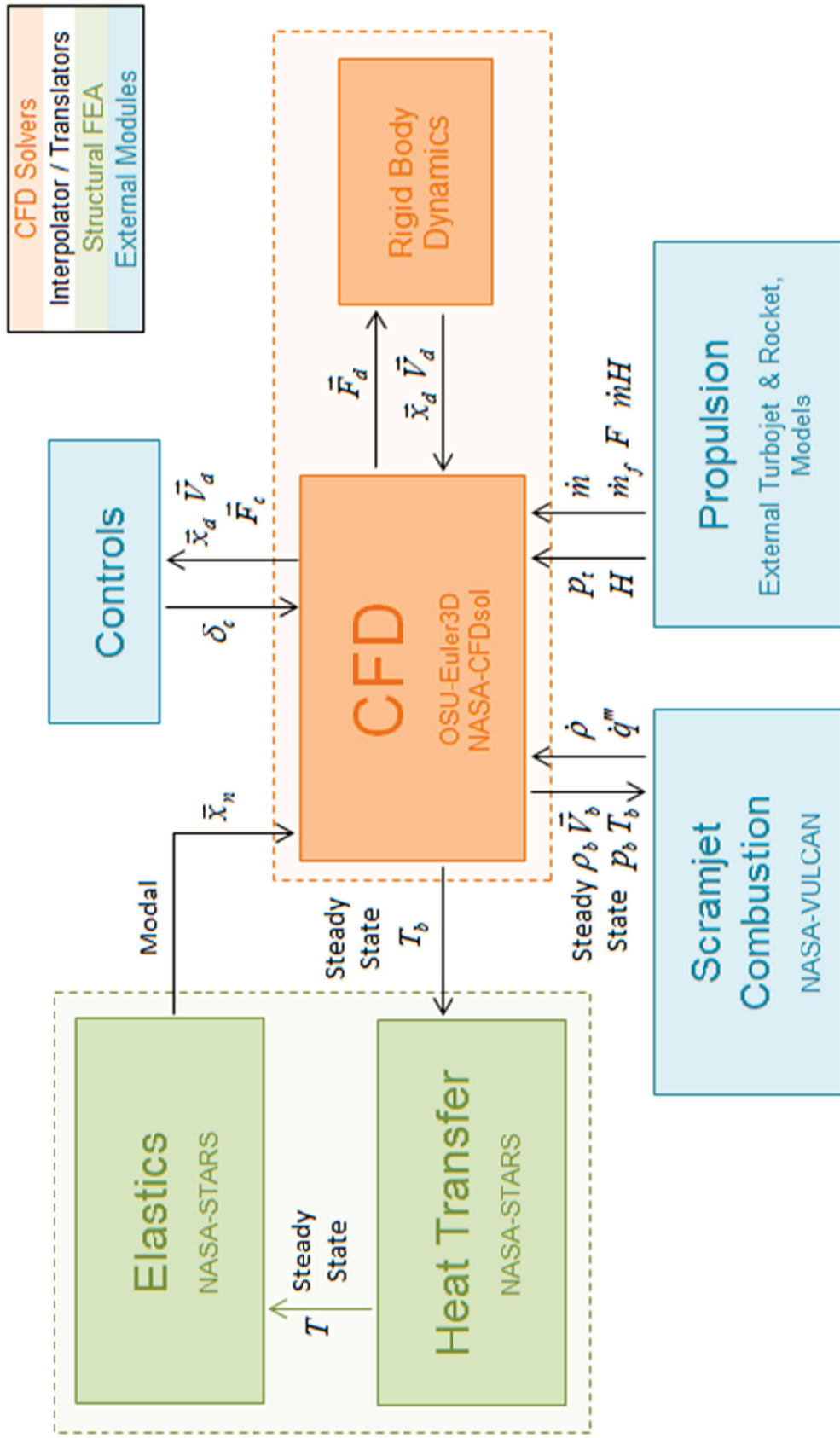


Figure 2.1: Flow Diagram (Lower Order System Model).

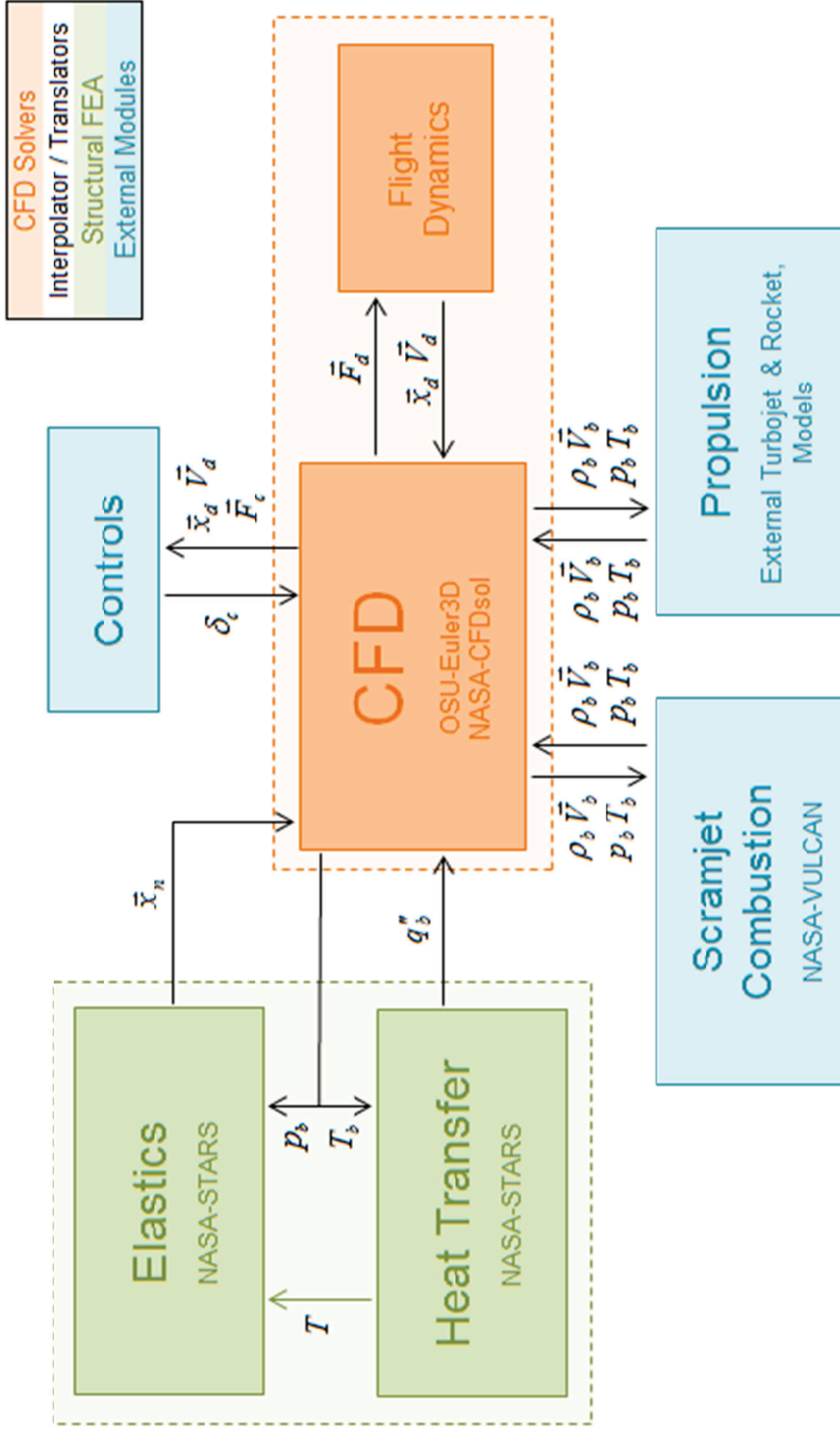


Figure 2.2: Flow Diagram (Higher Order System Model).

2.1 Emphasis and Objectives

This section outlines the work done on CFDsol, Euler2D/3D, and NS2D/3D and shows flow charts for how the final work fits into the MDA environment described above. Finally, a description of objectives and milestones is given for the project.

2.1.1 Work with NASA-CFDsol (NASA Contract)

The NASA contract was broken down into three areas of focus: (1) Prepare individual routines for (2) integration in a multi-disciplinary environment and then (3) demonstrate the capabilities of that environment. The emphasis of the contract was placed on information passed to CFDsol and how that data was utilized within CFDsol. Although no work was done in hypersonics, the models were developed to be expanded fully into the hypersonic regime at a later time. The end results of the contract are illustrated in Figure 2.3.

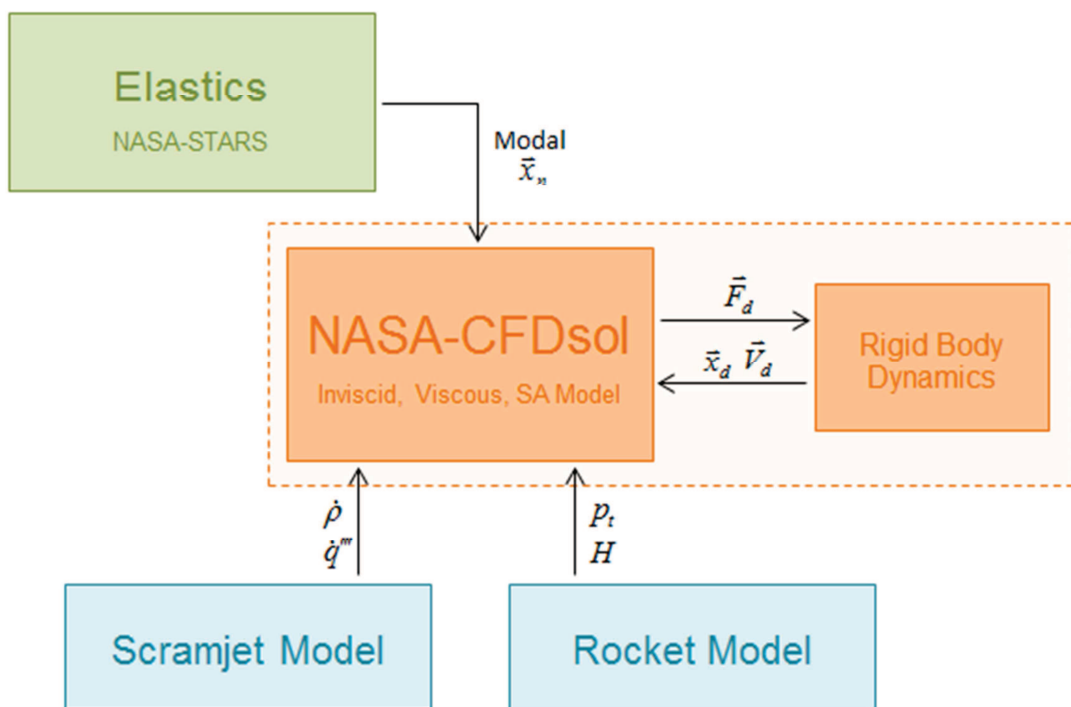


Figure 2.3: Flow Diagram (NASA-CFDsol).

Several features had to be improved in CFDsol before any new features could be added. The far field outflow was found to be unstable at subsonic speeds, and the energy lost by applying the wall and symmetry boundary conditions needed to be reconciled. Time accuracy of the solution was hampered by the update equation and artificial dissipation. Small inconsistencies needed to be removed from the implementation of the viscous terms and Sutherland's equation. Finally, the Spalart-Allmaras (SA) turbulence model need to be reformulated in compressible form and adapted to be applicable to generic flow fields.

Six modules were added to CFDsol: Non-inertial and quasi-combustion source terms; rocket boundary condition; inviscid wall transpiration; and, modal elastics and rigid body models. The deliverables of the NASA contract were confined to the new features of CFDsol. These models were developed in Euler2D and NS2D to minimize time and testing. CFDsol has a very different nature and dynamic than the OSU codes, so the routines were expanded into Euler3D and NS3D before being moved to CFDsol. This process allowed the routines to be proven in three-dimensions before interacting with CFDsol. When the routines were added to CFDsol, the data structures and names were changed to fit the standards already in place in CFDsol, and new arrays were created to fit the new features.

The capabilities of CFDsol were then demonstrated on several steady and transient, inviscid and viscous cases across the subsonic, transonic, and supersonic regimes. Specific cases were selected to demonstrate the new features of CFDsol, particularly the new propulsion models, elastic deformations, and non-inertial frame. Euler3D and NS3D were used to produce one-to-one comparisons on the same mesh. When time constraints limited run time, Euler2D and NS2D were used to produce quick results for comparison.

2.1.2 Work with In-House Codes

Prior to this work, Euler2D and Euler3D were developed by Cowan (2003). Moffitt (2004) expanded Euler2D to include viscous terms, creating NS2D. Moffitt only tested NS2D in the inertial frame; Sukraw (2008) tested the non-inertial frame. NS2D was used to investigate methods for implementing viscous transpiration, but these efforts turned up fruitless. Before any further work was done with the two-dimensional codes, Euler2D was updated with more recent efforts in Euler3D, and NS2D was recreated from Euler2D using methods that were more efficient in development, storage, and run time.

Several development efforts were made in Euler2D and NS2D prior to work on the contract. Per suggestions from O'Neill (2011), higher-order integration techniques were developed in Euler2D, including more points for Gauss quadrature and analytical integrals. Brown (2009) compared these techniques, but artificial dissipation hampered his ability to fairly compare the methods. The quasi-combustion terms, rocket exhaust, and engine inlets were developed and tested in Euler2D along with their support software. Heat transfer boundary conditions were implemented in NS2D without testing. The adiabatic wall is the default condition and has been used throughout this work. Because of the recent interest in acoustics, acoustic output files were added to Euler2D and Euler3D to track property histories.

During the contract, Euler2D and NS2D were used to develop new techniques, which were pushed to Euler3D and NS3D for testing before moving to CFDsol: The quasi-combustion terms and rocket boundary condition were expanded to Euler3D. The rigid body dynamics model in Euler3D was also expanded to include a full inertial matrix. The viscous terms

from NS2D were added to Euler3D to create NS3D. Finally, an SA turbulence model was developed in NS2D, pushed to NS3D and then used to adapt the SA model in CFDsol.

After the contract, work continued in Euler2D/3D and NS2D/3D. The turbojet exhaust was coupled with the inlet boundary in Euler2D. The turbojet boundaries were then expanded into Euler3D. All propulsion routines were added to NS2D and NS3D. An SST model was developed in NS2D. NS3D was then recreated using Euler3D, using the relationship between Euler2D and NS2D as a guide. Finally, testing of viscous non-inertial and turbulence modeling was completed in NS2D and NS3D.

The final coupling of Euler3D and NS3D with external models is shown in Figure 2.4:

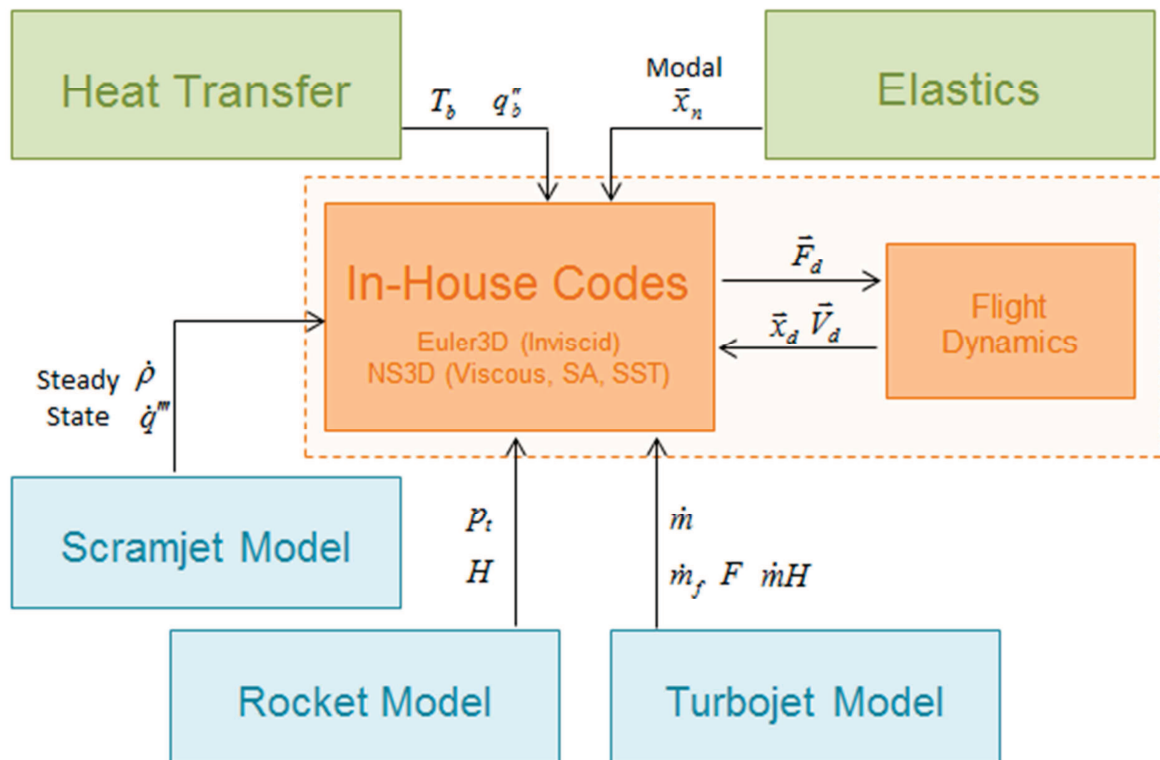


Figure 2.4: Flow Diagram (In-House OSU Codes).

2.1.3 Objectives and Milestones

The goals for this research are summarized in the five objectives below. The first three objectives are broken down further into milestones. The first objective represents the development of concepts, formulation of the method, and initial testing. The second objective represents the expansion of the method to three-dimensions. The fourth objective demonstrates the codes developed in the first two objectives. The third and fifth objectives encompass the NASA contract: The transfer of routines to CFDsol and their demonstration.

- Objective 1: Enhancement of Euler2D and NS2D
 - Implement Quasi-Combustion Terms
 - Implement Rocket and Engine Models
 - Implement Viscous Terms
 - Implement Turbulence Models
- Objective 2: Enhancement of Euler3D and NS3D
 - Implement Quasi-Combustion Terms
 - Implement Rocket and Engine Models
 - Implement Viscous Terms
 - Implement Turbulence Models
- Objective 3: Enhancement of NASA-CFDsol
 - Implement Quasi-Combustion Terms
 - Implement Rocket Model
 - Implement Inviscid Transpiration
 - Implement Non-Inertial Source Terms
 - Implement Modal Elastics and Rigid Body Dynamics
 - Improve Viscous Implementation
 - Improve Turbulence Models
- Objective 4: Demonstrate Four In-House Codes
- Objective 5: Comparison of Post-Contract CFDsol with In-House Codes

The subsequent sections describe for each objective and its milestones:

2.1.3.1 Objective 1: Enhancement of Euler2D and NS2D

Euler2D and NS2D were used as a development and testing platform for all methods investigated in this work. The major areas of development are summed up in four milestones:

Implement Quasi-Combustion Terms. Heat and mass is added to the flow in a scramjet, ramjet, or afterburner. This additional heat produces the thrust necessary to maintain flight conditions in a simulated vehicle. To avoid using a full combustion model, mass and heat source terms, called *quasi-combustion* terms, were added to the governing equations.

Galerkin's method was applied to the quasi-combustion terms in a manner that is flexible to the generation of combustion rates, whether by analytical, experimental, or numerical means. These terms were tested in Euler2D and then transferred to NS2D.

Implement Rocket and Engine Models. In previous simulations of flight vehicles, such as a missile, FA-18, and F-22, the thrust to maintain steady flight was created using an "imaginary string" that pulled the vehicle along at the desired conditions. The inlet and exhaust of engines were modeled using far field properties because no other boundary condition was available. Inflow and outflow planes were created to model rocket and turbojet engines. The rocket exhaust is modeled by specifying properties similar to a combustion chamber and allowing those properties to expand naturally through a nozzle. The turbojet is modeled using two boundary planes: The upstream plane represents the conditions being pulled into the compressor, and the downstream plane represents the turbine exhaust. The inlet and nozzle are modeled in the domain, but the complexity of the turbomachinery is avoided. Force and moment calculation was updated for momentum exchange through the surfaces. These boundaries were tested in Euler2D and transferred to NS2D.

Implement Viscous Terms. Viscous stresses and heat fluxes were added to Euler2D so that all of the features of Euler2D were also present in NS2D. The terms were developed using Moffitt's method (2004) but implemented more efficiently. Viscous terms were also added to the momentum exchange used to calculate forces and moments.

Implement Turbulence Models. A turbulence model is necessary to efficiently model flows at higher Reynolds numbers. A simple, efficient, and effective turbulence model was desired. The Spalart-Allmaras (SA) and Menter's SST models are well documented in the literature and broadly used throughout research and industry. These two models were implemented along with correction terms for non-inertial rotation. Artificial dissipation was added to each model for stability while retaining as much accuracy as possible.

2.1.3.2 Objective 2: Enhancement of Euler3D and NS3D

The methods developed in Euler2D and NS2D were expanded to the third dimension and implemented in Euler3D and NS3D. The major areas are summed up in four milestones:

Implement Quasi-Combustion Terms. The quasi-combustion terms were expanded to a three-dimensional Galerkin integral and implemented in Euler3D. These terms were developed and tested using the techniques and cases used with Euler2D. The quasi-combustion terms were added to NS3D during its creation.

Implement Rocket and Engine Models. The rocket and engine boundary conditions were expanded to three-dimensional boundary integrals and implemented in Euler3D. These terms were developed and tested using the techniques and cases used with Euler2D. The rocket and engine boundary conditions were added to NS3D during its creation.

Implement Viscous Terms. Viscous stresses and heat fluxes were added to Euler3D so that all of the features of Euler3D were also present in NS3D. The conversion of Euler2D to NS2D was used as a pattern for the creation of arrays, controls, and subroutines. Viscous terms were also added to the momentum exchange used to calculate forces and moments.

Implement Turbulence Models. The SA and SST models implemented in NS2D are expanded to three-dimensional formulations. The rotation correction and artificial dissipation models were likewise expanded to include the new dimension.

2.1.3.3 Objective 3: Enhancement of NASA-CFDsol

The methods proven in Euler2D/3D and NS2D/3D were transferred to CFDsol. Adaptations were made as necessary to integrate into the new data structures and algorithms in CFDsol.

The major areas are summed up in eight milestones:

Implement Quasi-Combustion Terms. The development of quasi-combustion source terms is the same following the formulations of Cowan (2003) and Gupta (2007), so the routines implemented in Euler3D to produce the quasi-combustion terms were transferred to CFDsol. The input file formats and data structures were kept as much as possible.

Implement Rocket Model. CFDsol does not use boundary integrals for inviscid fluxes. Instead, the boundary conditions are applied explicitly between updates of the governing equations. The methods applied in Euler3D were adapted to this implementation. The input file formats and data structures were kept as much as possible.

Implement Inviscid Transpiration. Transpiration is an adaptation of inviscid flow tangency. The inviscid wall boundary condition in CFDsol was adapted to include the transpired

normals and boundary velocities. The normal and boundary velocities were also updated every iteration in order to prepare for integration with the structural motion.

Implement Non-Inertial Source Terms. The development of non-inertial source terms is the same following the formulations of Cowan (2003) and Gupta (2007), so the code used in Euler3D to produce the non-inertial terms was transferred to CFDsol. The terms were recalculated every iteration in order to prepare for integration with the rigid body motion.

Implement Modal Elastics and Rigid Body Dynamics Models. The modal elastics and rigid body dynamics models are accomplished using similar techniques, data structures, and routines. These routines and data structures were transferred directly to CFDsol.

Improve Viscous Implementation. Viscous terms were already implemented in CFDsol using boundary integrals similar to the method proven by Moffitt (2004). NS3D and its development were compared with CFDsol to ensure that the viscous stresses, heat fluxes, and Sutherland's equation were implemented in CFDsol in the most accurate manner.

Improve Turbulence Models Implementation. An incompressible SA turbulence model was already implemented in CFDsol. The model could not be used on generic applications because of its trip transition model, lack of compressibility, and errors in implementation. The transition model was removed, and a compressible formulation was implemented in the correct manner, similar to that seen in NS3D. (Direct transfer was not used for this code.)

2.1.3.4 Objective 4: Demonstrate Four In-House Codes

The four OSU in-house codes were demonstrated on subsonic, transonic, supersonic, and hypersonic flows with inviscid, laminar, and RANS models. These demonstrations were

compared to experimental data, where possible, to make a validation effort. When this was not possible, analytical and numerical comparisons were used to make a verification effort. The propulsion models are also demonstrated on various simple geometries to illustrate the fundamental physics that are present in the models, even in their simplification.

2.1.3.5 Objective 5: Comparison of Post-Contract CFDsol with In-House Codes

Some of the cases used in the previous objective were used to demonstrate the capabilities of CFDsol. Verifications and validations were made where possible, but all cases were compared to one of the four in-house codes. Euler3D and NS3D were used to make comparisons on the same mesh. Euler2D and NS2D were used when time constraints did not allow.

2.1.4 Additional Work

Time was also spent in three other areas: Higher-order integration, heat transfer, and acoustic histories. These three areas are fully implemented but need to be tested before extended use.

Higher-Order Integration. Cowan (2003) developed Euler2D with one- and three-point Gauss quadrature. Cowan's experiments showed that additional Gauss points did not improve the accuracy of calculations. O'Neill (2011) worked in higher-order elements and suggested that higher-order integration improved the accuracy and allowed for larger elements. This work expanded Cowan's work to a fourth Gauss point, which should represent the highest order necessary for the inviscid flux terms, and analytical integration.

Brown (2009) tested subsonic and supersonic convergence using this method. Brown found that the current artificial dissipation models cause too many inaccuracies to properly evaluate

the order of integration. Brown's study should be revisited with inviscid walls and viscous dissipation, a new artificial dissipation model, or a different geometry.

Heat Transfer Boundary Conditions. Heat transfer boundary conditions were implemented in NS2D and NS3D. The boundary conditions are read from a file, per node for temperature (enthalpy) conditions and per boundary element for normal heat fluxes. The adiabatic wall is the default condition and has been used throughout this work. The heat conduction terms and boundary conditions need to be verified and validated before future use.

Acoustic Output Files. Because of the recent interest in acoustics, acoustic files were added to all four OSU codes. These files can be used to track the history of properties at one or more locations on the domain. An FFT routine was also prepared for reading, parsing, and decomposing the acoustics files. The acoustic output in Euler2D and NS2D has been used to produce profiles and point studies during this work, but the methods used to interpolate data from Euler3D and NS3D domains still needs to be tested.

CHAPTER III

THEORY

This chapter outlines the theory used in this research. The chapter begins by discussing the Euler and Navier-Stokes equations in their inertial and non-inertial formulations. These equations represent the basis for all five solvers used in this work. The governing equations are incomplete without thermodynamic relationships and the equation of state. The Navier-Stokes equations are then Reynolds-averaged to demonstrate the application of turbulence models. Two families of turbulence models are discussed along with the specific flavor that is implemented in this work. All of the governing equations and their properties are then written in dimensionless form. The governing equations can only be used to solve problems once their boundaries have been defined. Finally, the chapter closes with the discussion of rigid body and elastic modeling.

3.1 Euler Equations

The Euler equations represent the transport of continuity, momentum, and energy through an inviscid continuum. The Euler equations are developed using a first-order Taylor series expansion of fluxes across an infinitesimal element in a continuum flow. The Euler equations should be seen as the *simplest* representation of a compressible, inviscid continuum.

Any discretization of the Euler equations will also be limited to continuum flows. The Euler equations are the basis of any compressible CFD solver, including Euler2D/3D (Cowan, 2003) and CFDsol (Gupta, 2007).

3.1.1 Inertial Formulation

The Euler equations can be written in many forms. The Euler equations in the inertial frame are written here in conservative form and compact notation:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x_i} = 0 \quad (3.1)$$

where

$$\mathbf{U} = \begin{Bmatrix} \rho \\ \rho u_j \\ \rho E \end{Bmatrix} \quad \mathbf{F}_i = \begin{Bmatrix} \rho u_i \\ \rho u_i u_j + p \delta_{ij} \\ (\rho E + p) u_i \end{Bmatrix} \quad (3.2)$$

3.1.2 Non-Inertial Formulation

Cowan derives the Euler equations in the non-inertial frame. The equations are derived in a relative frame of motion, where the relative velocities $u_{r,i}$ is related to the velocity in the stationary frame u_i :

$$u_i = B_{ij} (u_{r,j} + V_{t,j}) \quad (3.3)$$

where B_{ij} is the rotation of the frame, which will be discussed later for the rigid body model.

$V_{t,i}$ is the mesh velocity in the non-inertial frame under translation $V_{o,i}$ and rotation ω_j . The translational velocity is marked “(I)” because it is transformed from the inertial frame.

$$V_{t,i} = V_{o,i} + \varepsilon_{ijk} \omega_j x_{b,k} \quad V_{o,i} = B_{ji} V_{o,j}^{(I)} \quad (3.4)$$

The total relative energy ρE_r in the moving frame is related to the internal energy ρe using Eq. 3.5. Similarly, the total energy in the stationary frame ρE is related to the internal energy using Eq. 3.6. Eq. 3.7 falls out from these two equations being linked through the internal energy and Eq. 3.3 being substituted for the velocity in the stationary frame:

$$\rho E_r = \rho e + \frac{1}{2} \rho (u_{r,i} u_{r,i} - V_{t,i} V_{t,i}) \quad (3.5)$$

$$\rho E = \rho e + \frac{1}{2} \rho u_i u_i \quad (3.6)$$

$$\rho E_r = \rho E - \rho u_i V_{t,i} \quad (3.7)$$

Cowan reposes the Euler equations in the non-inertial frame using the relative velocity and total relative energy in the moving frame:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x_{b,i}} = \mathbf{S} \quad (3.8)$$

where

$$\mathbf{U} = \begin{Bmatrix} \rho \\ \rho u_{r,j} \\ \rho E_r \end{Bmatrix} \quad \mathbf{F}_i = \begin{Bmatrix} \rho u_{r,i} \\ \rho u_{r,i} u_{r,j} + p \delta_{ij} \\ u_{r,i} (\rho E_r + p) \end{Bmatrix} \quad \mathbf{S} = -\rho \begin{Bmatrix} 0 \\ a_{t,j} + \varepsilon_{jkl} \omega_k u_{r,l} \\ a_{t,k} \cdot (V_{t,k} + u_{r,k}) \end{Bmatrix} \quad (3.9)$$

$$a_{t,i} = a_{o,i} + \varepsilon_{ijk} \omega_j \varepsilon_{klm} \omega_l x_{b,m} + \varepsilon_{ijk} \dot{\omega}_j x_{b,k} + \varepsilon_{ijk} \omega_j u_{r,k} \quad a_{o,i} = B_{ji} a_{o,j}^{(I)} \quad (3.10)$$

where \mathbf{S} is the non-inertial source term and $a_{t,j}$ is the relative acceleration (marked “(I)”) aligned with the inertial frame. Coordinates in the global frame x_i are related to the coordinates in the rotated frame $x_{b,i}$ relative to the center of rotation $x_{o,i}$:

$$x_i = x_{o,i} + B_{ij} x_{b,j} \quad (3.11)$$

Notice that the gradient in Eq. 3.8 has been written in terms of the relative mesh coordinates $x_{b,i}$, which remains constant even under translation or rotation.

3.2 Navier-Stokes Equations

The Navier-Stokes represent the transport of continuity, momentum, and energy through a viscous continuum. The Navier-Stokes equations are also developed using a first-order Taylor series expansion of fluxes across an infinitesimal element in a continuum flow. The Navier-Stokes equations should be seen as the *simplest* representation of a compressible, viscous continuum. Any discretization of the Navier-Stokes equations will also be limited to continuum flows. The Navier-Stokes equations are the basis for NS2D (Moffitt, 2004) and CFDsol (Gupta, 2007).

3.2.1 Inertial Formulation

The Navier-Stokes equations add viscous stresses to momentum transport and viscous dissipation with heat conduction to the energy equation. The Navier-Stokes equations in the inertial frame are written here in conservative form and compact notation:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x_i} = \frac{\partial \mathbf{F}_{v,i}}{\partial x_i} \quad (3.12)$$

where

$$\mathbf{U} = \begin{Bmatrix} \rho \\ \rho u_j \\ \rho E \end{Bmatrix} \quad \mathbf{F}_i = \begin{Bmatrix} \rho u_i \\ \rho u_i u_j + p \delta_{ij} \\ (\rho E + p) u_i \end{Bmatrix} \quad \mathbf{F}_{v,i} = \begin{Bmatrix} 0 \\ \tau_{ij}^{(I)} \\ \tau_{ij}^{(I)} u_j - q_i'' \end{Bmatrix} \quad (3.13)$$

The left side of the governing equations is the same as the Euler equations. The viscous stresses and heat fluxes in the viscous fluxes $\mathbf{F}_{v,i}$ are represented using Newtonian fluids:

$$\tau_{ij}^{(I)} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) + \lambda \frac{\partial u_k}{\partial x_k} \delta_{ij} \quad (3.14)$$

$$q_i'' = -k \frac{\partial T}{\partial x_i} \quad (3.15)$$

where μ is the kinetic viscosity, λ is the second viscosity coefficient, and k is the coefficient of thermal conductivity. The superscript “(I)” denotes the stress tensor in the inertial frame.

3.2.2 Non-Inertial Formulation

The Navier-Stokes equations can be posed in the non-inertial frame using Cowan’s method.

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x_{b,i}} = \frac{\partial \mathbf{F}_{v,i}}{\partial x_{b,i}} + \mathbf{S} \quad (3.16)$$

where the unknowns \mathbf{U} , inviscid fluxes $\mathbf{F}_{v,i}$, and non-inertial source term \mathbf{S} are calculated using Eq. 3.9. The viscous fluxes are calculated:

$$\mathbf{F}_{v,i} = \left\{ \begin{array}{c} 0 \\ \tau_{ij} \\ \tau_{ij} u_{r,j} - q_i'' \end{array} \right\} \quad (3.17)$$

The energy flux is represented by the product of relative velocity and momentum. The energy flux is shown in the inviscid $u_{r,i} p$ and viscous stress $\tau_{ij} u_{r,j}$ flux terms. The relative velocity is used because the domain element moves with the fluid, so translation and rotation do not perform work on the element. Instead, energy flux is created by local changes in the flow field. Vanyo (1993) suggests these equations to solve fluid fields in all frames. The viscous stress tensor τ_{ij} can be defined either in terms of relative velocities or total velocity. To prove this, we start with Eq. 3.14 (written in matrix-vector form, Eq. 3.18) and transform the gradients between frames using Eq. 3.11 (where $\nabla_b = [\partial \bar{x} / \partial \bar{x}_b] \nabla = \mathbf{B}^T \nabla = \mathbf{A} \nabla$):

$$\boldsymbol{\tau}^{(I)} = \mu \left(\nabla \bar{\boldsymbol{v}}^T + (\nabla \bar{\boldsymbol{v}}^T)^T \right) + \lambda \nabla^T \bar{\boldsymbol{v}} \mathbf{I} \quad (3.18)$$

$$\boldsymbol{\tau}^{(I)} = \mu \left(\mathbf{B} \nabla_b \bar{\boldsymbol{v}}^T + (\mathbf{B} \nabla_b \bar{\boldsymbol{v}}^T)^T \right) + \lambda (\mathbf{B} \nabla_b)^T \bar{\boldsymbol{v}} \mathbf{I}$$

Eqs. 3.3 and 3.4 are used to transform the velocity into the non-inertial frame:

$$\begin{aligned} \boldsymbol{\tau}^{(I)} &= \mu \left(\mathbf{B} \nabla_b (\mathbf{B} \bar{\boldsymbol{v}}_r)^T + (\mathbf{B} \nabla_b (\mathbf{B} \bar{\boldsymbol{v}}_r)^T)^T \right) + \lambda (\mathbf{B} \nabla_b)^T (\mathbf{B} \bar{\boldsymbol{v}}_r) \mathbf{I} \\ &+ \mu \left(\mathbf{B} \nabla_b (\bar{\boldsymbol{v}}_o)^T + (\mathbf{B} \nabla_b (\bar{\boldsymbol{v}}_o)^T)^T \right) + \lambda (\mathbf{B} \nabla_b)^T (\bar{\boldsymbol{v}}_o) \mathbf{I} \\ &+ \mu \left(\mathbf{B} \nabla_b (\mathbf{B} \Omega \bar{\boldsymbol{x}}_b)^T + (\mathbf{B} \nabla_b (\mathbf{B} \Omega \bar{\boldsymbol{x}}_b)^T)^T \right) + \lambda (\mathbf{B} \nabla_b)^T (\mathbf{B} \Omega \bar{\boldsymbol{x}}_b) \mathbf{I} \end{aligned}$$

The matrix transforms are resolved (acknowledging that $\mathbf{B}^T = \mathbf{B}^{-1} = \mathbf{A}$):

$$\begin{aligned} \boldsymbol{\tau}^{(I)} &= \mu \mathbf{B} \left(\nabla_b \bar{\boldsymbol{v}}_r^T + (\nabla_b \bar{\boldsymbol{v}}_r^T)^T \right) \mathbf{A} + \lambda \nabla_b^T \mathbf{A} \mathbf{B} \bar{\boldsymbol{v}}_r \mathbf{I} \\ &+ \mu \left(\mathbf{B} \nabla_b \bar{\boldsymbol{v}}_o^T + (\nabla_b \bar{\boldsymbol{v}}_o^T)^T \right) \mathbf{A} + \lambda \nabla_b^T \mathbf{A} \bar{\boldsymbol{v}}_o \mathbf{I} \\ &+ \mu \mathbf{B} \left(\nabla_b \bar{\boldsymbol{x}}_b^T \Omega^T + \Omega (\nabla_b \bar{\boldsymbol{x}}_b^T)^T \right) \mathbf{A} + \lambda \nabla_b^T \mathbf{A} \mathbf{B} \Omega \bar{\boldsymbol{x}}_b \mathbf{I} \end{aligned}$$


Finally, the stress tensor is transformed back into the non-inertial frame:

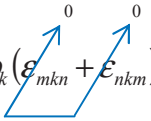
$$\begin{aligned} \boldsymbol{\tau} &= \mathbf{A} \boldsymbol{\tau}^{(I)} \mathbf{B} = \mu \left(\nabla_b \bar{\boldsymbol{v}}_r^T + (\nabla_b \bar{\boldsymbol{v}}_r^T)^T \right) + \lambda \nabla_b^T \bar{\boldsymbol{v}}_r \mathbf{I} \\ &+ \mu \left(\nabla_b (\mathbf{A} \bar{\boldsymbol{v}}_o)^T + (\nabla_b (\mathbf{A} \bar{\boldsymbol{v}}_o)^T)^T \right) + \lambda \nabla_b^T \mathbf{A} \bar{\boldsymbol{v}}_o \mathbf{I} \\ &+ \mu \left(\nabla_b \bar{\boldsymbol{x}}_b^T \Omega^T + \Omega (\nabla_b \bar{\boldsymbol{x}}_b^T)^T \right) + \lambda \nabla_b^T \Omega \bar{\boldsymbol{x}}_b \mathbf{I} \end{aligned}$$

which can also be written in indicial notation:

$$\begin{aligned} \tau_{mn} &= A_{mi} \tau_{ij} B_{jn} = \mu \left(\frac{\partial u_{r,m}}{\partial x_{b,n}} + \frac{\partial u_{r,n}}{\partial x_{b,m}} \right) + \lambda \frac{\partial u_{r,p}}{\partial x_{b,p}} \delta_{mn} + \mu \left(\frac{\partial A_{mi} \overset{0}{V}_{o,i}}{\partial x_{b,n}} + \frac{\partial A_{nj} \overset{0}{V}_{o,j}}{\partial x_{b,m}} \right) + \lambda \frac{\partial A_{pk} \overset{0}{V}_{o,k}}{\partial x_{b,p}} \delta_{mn} \\ &+ \mu \left(\frac{\partial}{\partial x_{b,n}} (\varepsilon_{mkl} \omega_k x_{b,l}) + \frac{\partial}{\partial x_{b,m}} (\varepsilon_{nkl} \omega_k x_{b,l}) \right) + \lambda \frac{\partial}{\partial x_{b,k}} (\varepsilon_{klp} \omega_l x_{b,p}) \delta_{mn} \end{aligned}$$

The entire domain is translating $V_{o,i}$ and rotating ω_j at the same rate so the gradient of these terms are identically zero. The translational contributions have been identified as zero in Eq. 3.18. The rotational components are simplified in Eqs. 3.19 and 3.20.

$$\frac{\partial}{\partial x_{b,k}} (\varepsilon_{klp} \omega_l x_{b,p}) = \varepsilon_{klp} \omega_l \frac{\partial x_{b,p}}{\partial x_{b,k}} = \varepsilon_{klp} \omega_l \delta_{kp} = \varepsilon_{klk} \omega_l = 0 \quad (3.19)$$


$$\frac{\partial}{\partial x_{b,n}} (\varepsilon_{mkl} \omega_k x_{b,l}) + \frac{\partial}{\partial x_{b,m}} (\varepsilon_{nkl} \omega_k x_{b,l}) = \omega_k (\varepsilon_{mkl} \delta_{nl} + \varepsilon_{nkl} \delta_{ml}) = \omega_k (\varepsilon_{mkn} + \varepsilon_{nkm}) = 0 \quad (3.20)$$


The non-inertial stress tensor can be calculated through relative velocities alone (Eq. 3.21):

$$\tau_{mn} = A_{mi} \tau_{ij} B_{jn} = \mu \left(\frac{\partial u_{r,m}}{\partial x_{b,n}} + \frac{\partial u_{r,n}}{\partial x_{b,m}} \right) + \lambda \frac{\partial u_{r,p}}{\partial x_{b,p}} \delta_{mn} \quad (3.21)$$

3.3 Thermodynamic Relationships

The governing equations are not enough to represent the fluid dynamics. Thermodynamic relationships are required to relate different properties. Specifically, an equation of state is required to relate the pressure to other properties of the fluid. Variable property relationships are considered where necessary, but constant relationships are used wherever possible.

Sutherland's equation is given for variable viscosity. These relationships are discussed along with their effects in simplifying previous equations. Finally, isentropic relationships are summarized for use in boundary conditions and other relationships.

3.3.1 Thermodynamics

The energy in the flow can be viewed in several forms: Internal energy, enthalpy, temperature, kinetic energy, or total energy. Internal energy e is a quantity that is lumped together to support continuum analysis. Internal energy is the kinetic energy measured at the mole-

cular level, including that created by molecule translation, rotation, and vibration and even the chemical bonds holding a molecule together. The quantity on the right side of Eq. 3.22 appears so often in thermodynamic equations that the quantity *enthalpy* h was created out of mathematical convenience. Enthalpy represents the combination of internal energy and *flow work*, or the work done by pressure on a finite control volume in the flow.

$$h = e + \frac{p}{\rho} \quad (3.22)$$

Temperature is another measure of the molecular kinetic energy. Temperature is more common than the previous measures of energy. Most people are more familiar with the temperature of a room than the internal energy or enthalpy of the air in that room. The internal energy and enthalpy can be related to the temperature using specific heats. The ratio of these specific heats γ can be used to relate enthalpy directly to the internal energy:

$$e = c_v T \quad h = c_p T \quad h = \gamma e \quad (3.23)$$

Kinetic energy, here, is the measure of the energy in the mean flow of the fluid: $u_i u_i / 2$. The kinetic energy is added to the internal energy in Eq. 3.6 to calculate the total energy. A similar quantity can be created to represent the total enthalpy ρH in the flow (Eq. 3.24). The definition of enthalpy (Eq. 3.22) can be substituted into this relationship, creating a relationship between total enthalpy, total energy, and flow work that resembles Eq. 3.22:

$$\rho H = \rho h + \frac{1}{2} \rho u_i u_i = \rho e + p + \frac{1}{2} \rho u_i u_i = \rho E + p \quad (3.24)$$

The internal energy was also combined with the kinetic energy in the non-inertial frame to calculate the total relative energy of the fluid (Eq. 3.5). A similar quantity can be created to represent the total relative enthalpy ρH_r in the flow (Eq. 25). The definition of enthalpy can

be substituted into this relationship to create a relationship between total relative enthalpy, total relative energy, and flow work:

$$\rho H_r = \rho h + \frac{1}{2} \rho (u_{r,i} u_{r,i} - V_{t,i} V_{t,i}) = \rho e + p + \frac{1}{2} \rho (u_{r,i} u_{r,i} - V_{t,i} V_{t,i}) = \rho E_r + p \quad (3.25)$$

3.3.2 Ideal Gas

Moffitt (2004) performed an order of magnitude analysis on the van der Waals equation and demonstrated that the ideal gas equation is sufficiently accurate for all continuum flows considered by this research. The ideal gas equation relates the pressure, density, and some form of energy. Traditionally, the temperature is used in the ideal gas equation, but the solvers considered here do not track the temperature of the flow. The ideal gas equation has been written in Eq. 3.26 in terms of internal energy and enthalpy:

$$p = \rho RT = \frac{R}{c_v} \rho e = \frac{R}{c_p} \rho h \quad (3.26)$$

The total enthalpy is tracked in the OSU codes, and the total energy is tracked in CFDsol. Eqs. 3.6 and 3.24 are substituted into Eq. 3.26 to formulate the ideal gas equation in terms of total energy and total enthalpy:

$$p = \frac{R}{c_v} \left(\rho E - \frac{1}{2} \rho u_i u_i \right) = \frac{R}{c_p} \left(\rho H - \frac{1}{2} \rho u_i u_i \right) \quad (3.27)$$

For the total relative energy and total relative enthalpy, in the non-inertial frame, Eqs. 3.5 and 3.25 are substituted into Eq. 3.26:

$$p = \frac{R}{c_v} \left(\rho E_r - \frac{1}{2} \rho (u_{r,i} u_{r,i} - V_{t,i} V_{t,i}) \right) = \frac{R}{c_p} \left(\rho H_r - \frac{1}{2} \rho (u_{r,i} u_{r,i} - V_{t,i} V_{t,i}) \right) \quad (3.28)$$

These equations can be simplified further after addressing whether c_p and c_v should be treated as constant or calculated as a function of the other thermodynamic properties.

3.3.3 Constant vs Variable Properties

Moffitt (2004) considered the variation of viscosity μ , specific heat c_p , ratio of specific heats γ and Prandtl number Pr over a full range of flight altitudes in Earth and Martian atmosphere at speeds up to Mach 3.5. Moffitt found that the specific heat, ratio of specific heats, and Prandtl number varied by less than 10% and only higher speeds at sea level. Viscosity could experience changes up to 200% and requires further modeling.

All of the equations presented thus far have not assumed constant or variable properties. If constant properties are assumed, several relationships can be simplified. To accomplish this, the two specific heats can be written in terms of the ratio of specific heats (Mattingly, 1996):

$$\frac{c_v}{R} = \frac{1}{\gamma-1} \quad \frac{c_p}{R} = \frac{\gamma}{\gamma-1} \quad (3.29)$$

Eq. 3.26 can be used to calculate the enthalpy using the pressure and density. Eq. 3.29 is substituted the relationship to eliminate the need for tracking specific heats:

$$h = \frac{c_p}{R} \frac{p}{\rho} = \frac{\gamma}{\gamma-1} \frac{p}{\rho} \quad (3.30)$$

The ideal gas equations in Eqs. 3.27 and 3.28 can also be simplified using Eq. 3.29:

$$p = (\gamma-1) \left(\rho E - \frac{1}{2} \rho u_i u_i \right) = \frac{\gamma-1}{\gamma} \left(\rho H - \frac{1}{2} \rho u_i u_i \right) \quad (3.31)$$

$$p = (\gamma-1) \left(\rho E_r - \frac{1}{2} \rho (u_{r,i} u_{r,i} - V_{t,i} V_{t,i}) \right) = \frac{\gamma-1}{\gamma} \left(\rho H_r - \frac{1}{2} \rho (u_{r,i} u_{r,i} - V_{t,i} V_{t,i}) \right) \quad (3.32)$$

The Prandtl number Pr , which is considered constant for this research, can be used to relate the viscosity, thermal conductivity, and specific heat. The Prandtl number will later be used to remove dimensions from heat conduction term:

$$Pr = \frac{c_p \mu}{k} \quad (3.33)$$

The variation of kinetic viscosity will be addressed in the next section. The second coefficient of viscosity is addressed here. Most CFD codes are developed around a constant relationship between first and second viscosities, where the former is $^{-2/3}$ times the later. This quantity is often referred to as *Stokes' hypothesis*. Moffitt (2004) reviews Stokes' hypothesis and its origination. Stokes developed Eq. 3.34 as a basis for maintaining positive viscous dissipation, which is required to be physically accurate. Stokes' hypothesis only creates a lower limit on second viscosity. CFDsol assumes the lower limit in all cases. NS2D and NS3D are setup to maintain the lower limit while allowing the user to increase λ as necessary. Additional discussion of Stokes' hypothesis can be found in Appendix A.

$$\lambda \geq -\frac{2}{3} \mu \quad (3.34)$$

3.3.4 Sutherland's Equation

The viscosity μ is calculated from the freestream properties and Reynolds number. The local viscosity can be calculated as a function of temperature (enthalpy) using Sutherland's law (Eq. 3.35). Sutherland's equation is traditionally written in terms of a reference condition and reference viscosity. Here, the reference conditions have been taken from the freestream conditions. The temperatures and coefficient S can be scaled by the specific heats c_p to repose Sutherland's equation in terms of enthalpy instead of temperature:

$$\frac{\mu}{\mu_\infty} = \left(\frac{T}{T_\infty}\right)^{\frac{3}{2}} \frac{T_\infty + S}{T + S} = \left(\frac{h}{h_\infty}\right)^{\frac{3}{2}} \frac{h_\infty + c_p S}{h + c_p S} \quad (3.35)$$

The thermal conductivity k also distributed using Sutherland's equation through a constant Prandtl number (Eq. 3.33). The local and freestream enthalpy h is calculated using Eq. 3.30.

3.3.5 Isentropic Relationships

Assuming an ideal gas with constant specific heats, the total properties of a fluid can be defined from its static properties. The total properties are created by decelerating a fluid to zero velocity through an isentropic process. For instance, a fluid with an enthalpy h and velocity u_i decelerated through an isentropic process to a zero velocity has an enthalpy H , otherwise known as its total enthalpy. The enthalpy at these two conditions is linked through conservation of energy (Eq. 3.25). The velocity can be written in terms of the local Mach M :

$$M^2 = \frac{u_{r,i} u_{r,i}}{a^2} \quad a^2 = \gamma \frac{\partial p}{\partial \rho} = \gamma \mathcal{R}T = \gamma \frac{p}{\rho} \quad (3.36)$$

$$\frac{H_r}{h} = 1 + \frac{u_{r,i} u_{r,i} - V_{t,i} V_{t,i}}{2h} = 1 + \frac{\gamma \mathcal{R}T}{2c_p T} \left(M^2 - \frac{V_{t,i} V_{t,i}}{a^2} \right) = 1 + \frac{\gamma - 1}{2} \left(M^2 - \frac{V_{t,i} V_{t,i}}{a^2} \right) \quad (3.37)$$

Since the total enthalpy H is also the static enthalpy when the flow has been decelerated isentropically to zero, Eq. 3.23 can be used to calculate its temperature, or *total temperature*.

The ratio of total to static enthalpy is used to calculate the ratio of total to static temperature:

$$\frac{H_r}{h} = \frac{c_p T_t}{c_p T} = \frac{T_t}{T} \quad (3.38)$$

For an isentropic gas with constant properties, the ratio of any two temperatures can be used to calculate the corresponding pressure ratio. If the temperature ratio is total to static, then the pressure ratio will also be total to static:

$$\frac{p_t}{p} = \left(\frac{T_t}{T} \right)^{\gamma/\gamma-1} \quad (3.39)$$

Viscous dissipation of energy produces entropy. In the presence of viscous dissipation (boundary layers, vortical flows, etc.), the isentropic relationships are no longer valid. Eqs. 3.36 through 3.39 are used to model rocket properties upstream of the nozzle. Such flow is upstream of any viscous dissipation so the isentropic relationships are again applicable.

3.4 Turbulence Modeling Theory

Several methods exist for capturing turbulence that is present in high Reynolds number flows. Direct Numerical Simulation (DNS) uses the Navier-Stokes equations, as presented above, and very fine meshes to simulate turbulence in the flow. DNS is the most accurate method but also the most expensive, requiring spatial and temporal discretizations on the order of the Kolmogorov scales.

Large-Eddy Simulation (LES) reduces some of the expense by filtering out the smallest eddies and capturing the larger eddies within the filtered Navier-Stokes equations. LES requires very fine near-wall meshes, which makes the method too expensive for most applications, but LES makes up for its expense through accurate results well-preserved flow physical. LES is best for highly separated and vortical flows.

Reynolds-Averaged Navier-Stokes (RANS) is the next step in efficiency and accuracy.

Some RANS models only require the mean properties be appropriately modeled in the near-wall and high gradient regions in order to achieve accuracy. RANS tracks the bulk properties of turbulence in the flow, which are more mathematical than physical in nature. RANS models more turbulent in the flow than either of the previous methods, which puts all of the

weight on the shoulders of the model developer. The accuracy of a RANS model is often tuned on a series of simple or specific cases, on which the developer hopes to use his/her model. The generality to other flow regimes is lost in the tuning. Further, U-RANS (or Unsteady-RANS) is the extension of RANS models to unsteady flows. U-RANS is thought to be the least accurate of the above methods because RANS models are almost never tuned for unsteady problems. The models that are investigated here are posted in U-RANS form but only steady tests are incorporated into their verification.

3.4.1 Reynolds-Average Navier-Stokes (RANS)

Reynolds-averaging is a form of time-averaging similar to that used in many experimental techniques. The properties are averaged at a particular point of interest, assuming that if the period of the average is much larger than the period of the turbulence then the mean properties will be constant. If the flow is unsteady, the period T of the mean properties (denoted by overbar, \bar{p}) must be much larger than the turbulent fluctuations (denoted by prime, p').

Consider the velocity vector u_i as a decomposition of mean and fluctuating terms (Eq. 3.40). The velocity can be averaged over a period T to calculate the mean velocity. The fluctuating component is identically zero when averaged over the period T (Pope, 2000):

$$u_i = \bar{u}_i + u'_i \quad (3.40)$$

$$\bar{u}_i = \frac{1}{T} \int_t^{t+T} u_i dt \quad \frac{1}{T} \int_t^{t+T} u'_i dt = 0 \quad (3.41)$$

These principles are applied when the Navier-Stokes equations are time-averaged, creating the Reynolds-averaged Navier-Stokes (RANS) equation. Properties are grouped together through their multiplication into conservative properties, fluxes, or other constructions.

These terms are also time-averaged. Consider the time-average of $u_i u_j$ as an example. The mean velocity components (first term) are retained through the time-average. For the mean-fluctuation components (second and third terms), the constant mean-component can be factored out of integral, leaving the fluctuations, whose integrals are identically zero. The final term pairs two fluctuations, which by the definitions in Eq. 3.41 cannot be mathematically eliminated. These and similar terms are preserved in the RANS equations.

$$\frac{1}{T} \int_t^{t+T} u_i u_j dt = \frac{1}{T} \int_t^{t+T} \bar{u}_i \bar{u}_j dt + \frac{\bar{u}_i}{T} \int_t^{t+T} u_j' dt + \frac{\bar{u}_j}{T} \int_t^{t+T} u_i' dt + \frac{1}{T} \int_t^{t+T} u_i' u_j' dt = \bar{u}_i \bar{u}_j + \overline{u_i' u_j'} \quad (3.42)$$

Finally, the Navier-Stokes equations are partial differential equations relating the temporal and spatial derivatives to represent the physics of fluids. These derivatives must also be time-averaged. Eq. 3.43 and 3.44 illustrate the method for Reynolds-averaging derivatives.

$$\frac{1}{T} \int_t^{t+T} \frac{\partial p}{\partial x_i} dt = \frac{\partial}{\partial x_i} \left(\frac{1}{T} \int_t^{t+T} p dt \right) = \frac{\partial \bar{p}}{\partial x_i} \quad (3.43)$$

$$\frac{1}{T} \int_t^{t+T} \frac{\partial p}{\partial t} dt = \frac{\partial}{\partial t} \left(\frac{1}{T} \int_t^{t+T} p dt \right) = \frac{\partial \bar{p}}{\partial t} \quad (3.44)$$

We will start by Reynolds-averaging the incompressible Navier-Stokes equations, which will be used to develop and discuss RANS turbulence modeling. When compressibility is added, the RANS equations become very complicated. Favre-averaging (or mass-averaging) will be used in place of traditional Reynolds-averaging. Favre-averaging allows the CFD developer to implement RANS turbulence models into a compressible solver. Compressible corrections will be discussed. Then the SA and SST models will be presented in multiple forms.

3.4.1.1 Incompressible RANS

Turbulence can be conceptualized as properties that are continuously perturbed about a mean value. For example, the pressure can be represented: $p = \bar{p} + p'$. Using this concept the incompressible momentum equation (inertial) becomes (Pope, 2000):

$$\frac{\partial}{\partial t} (\bar{u}_j + u'_j) + \frac{\partial}{\partial x_i} ((\bar{u}_i + u'_i)(\bar{u}_j + u'_j)) = \frac{1}{\rho} \frac{\partial}{\partial x_i} (-(\bar{p} + p')\delta_{ij} + \bar{\tau}_{ij} + \tau'_{ij}) \quad (3.45)$$

where the viscous stresses are split into a component calculated using the mean velocities $\bar{\tau}_{ij}$ and fluctuating velocities τ'_{ij} . (Strictly speaking, $\bar{\tau}_{ij}$ is the time-average of the molecular stress tensor, and τ'_{ij} represents all of the terms that fall-out during time-averaging.)

$$\bar{\tau}_{ij} = \frac{\mu}{\text{Re}} \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) \quad \tau'_{ij} = \frac{\mu}{\text{Re}} \left(\frac{\partial u'_i}{\partial x_j} + \frac{\partial u'_j}{\partial x_i} \right) \quad (3.46)$$

The incompressible momentum equation is time-averaged:

$$\begin{aligned} \frac{1}{T} \int_t^{t+T} \frac{\partial}{\partial t} (\bar{u}_j + u'_j) dt + \frac{1}{T} \int_t^{t+T} \frac{\partial}{\partial x_i} ((\bar{u}_i + u'_i)(\bar{u}_j + u'_j)) dt \\ = \frac{1}{T} \int_t^{t+T} \frac{1}{\rho} \frac{\partial}{\partial x_i} (-(\bar{p} + p')\delta_{ij} + \bar{\tau}_{ij} + \tau'_{ij}) dt \end{aligned} \quad (3.47)$$

$$\frac{\partial \bar{u}_j}{\partial t} + \frac{\partial}{\partial x_i} (\bar{u}_i \bar{u}_j + \overline{u'_i u'_j}) = \frac{1}{\rho} \frac{\partial}{\partial x_i} (-\bar{p} \delta_{ij} + \bar{\tau}_{ij}) \quad (3.48)$$

The additional term $\overline{u'_i u'_j}$ is called the *Reynolds stress tensor*: $T_{ij} = -\overline{u'_i u'_j}$. The Reynolds stress is derived from the advection term but more closely resembles the viscous stresses, so the Reynolds stress is grouped with the total stress tensor:

$$\frac{\partial \bar{u}_j}{\partial t} + \frac{\partial (\bar{u}_i \bar{u}_j)}{\partial x_i} = \frac{1}{\rho} \frac{\partial}{\partial x_i} (-\bar{p} \delta_{ij} + \bar{\tau}_{ij} + \rho \Gamma_{ij}) \quad (3.49)$$

In this form, the Reynolds-averaged momentum equations looks very similar to the original momentum equation. The Reynolds stress tensor will be *closed* (modeled) later.

Now we seek to Reynolds-average all of the essential equations presented in the previous sections. In this way, we will Reynolds-average the Navier-Stokes equations in their non-inertial formulation and their support relationships. (The inertial equations take on a similar form, where the mesh velocity $V_{t,i}$ and rotation vector ω_j are identically zero, and the relative subscript “ r ” is dropped as a reference.) Before applying Reynolds-averaging to the governing equations, we first focus on the thermodynamic relationships, which will simplify the later process. The energy, enthalpy, and temperature are related through Eqs. 3.5, 3.7, 3.22 through 3.25, 3.30, 3.32, and 3.36 through 3.38, which are time-averaged here:

$$\rho \bar{E}_r = \rho \bar{e} + \frac{1}{2} \rho (\overline{\bar{u}_{r,i} \bar{u}_{r,i}} + \overline{\bar{u}'_{r,i} \bar{u}'_{r,i}} - V_{t,i} V_{t,i}) \quad (3.50)$$

$$\rho \bar{E}_r = \rho \bar{E} - \rho \bar{u}_i V_{t,i} \quad (3.51)$$

$$\bar{h} = \bar{e} + \frac{\bar{p}}{\rho} \quad (3.52)$$

$$\bar{e} = c_v \bar{T} \quad \bar{h} = c_p \bar{T} \quad \bar{h} = \gamma \bar{e} \quad (3.53)$$

$$\rho \bar{H}_r = \rho \bar{h} + \frac{1}{2} \rho (\overline{\bar{u}_{r,i} \bar{u}_{r,i}} + \overline{\bar{u}'_{r,i} \bar{u}'_{r,i}} - V_{t,i} V_{t,i}) = \rho \bar{E}_r + \bar{p} \quad (3.54)$$

$$\bar{h} = \frac{\gamma}{\gamma - 1} \frac{\bar{p}}{\rho} \quad (3.55)$$

$$\bar{p} = (\gamma - 1) \left(\rho \bar{E}_r - \frac{1}{2} \rho (\overline{\bar{u}_{r,i} \bar{u}_{r,i}} + \overline{\bar{u}'_{r,i} \bar{u}'_{r,i}} - V_{t,i} V_{t,i}) \right) \quad (3.56)$$

$$\bar{p} = \frac{\gamma-1}{\gamma} \left(\rho \bar{H}_r - \frac{1}{2} \rho (\bar{u}_{r,i} \bar{u}_{r,i} + \overline{u'_{r,i} u'_{r,i}} - V_{t,i} V_{t,i}) \right) \quad (3.57)$$

$$M^2 = \frac{\bar{u}_{r,i} \bar{u}_{r,i}}{\bar{a}^2} \quad \bar{a}^2 = \gamma \mathcal{R} \bar{T} = \gamma \frac{\bar{p}}{\rho} \quad (3.58)$$

$$\frac{\bar{T}_t}{\bar{T}} = \frac{\bar{H}_r}{\bar{h}} = 1 + \frac{\bar{u}_{r,i} \bar{u}_{r,i} + \overline{u'_{r,i} u'_{r,i}} - V_{t,i} V_{t,i}}{2\bar{h}} = 1 + \frac{\gamma-1}{2} \left(M^2 + \frac{\overline{u'_{r,i} u'_{r,i}} - V_{t,i} V_{t,i}}{a^2} \right) \quad (3.59)$$

The trace of the Reynolds stress tensor appears with the mean kinetic energy (Eqs. 3.50, 3.54, 3.56, and 3.57). This term is also called the *turbulent kinetic energy*: $K = \frac{1}{2} \overline{u'_{r,i} u'_{r,i}}$. The turbulent kinetic energy also appears in Eq. 3.59 in a Mach number form, which will be referred to as the *turbulent Mach number*: $M_T^2 = \overline{u'_{r,i} u'_{r,i}} / a^2 = 2K / a^2$. These two relationships are substituted back into the previous equations:

$$\rho \bar{E}_r = \rho \bar{e} + \frac{1}{2} \rho (\bar{u}_{r,i} \bar{u}_{r,i} - V_{t,i} V_{t,i}) + \rho K \quad (3.60)$$

$$\rho \bar{H}_r = \rho \bar{h} + \frac{1}{2} \rho (\bar{u}_{r,i} \bar{u}_{r,i} - V_{t,i} V_{t,i}) + \rho K = \rho \bar{E}_r + \bar{p} \quad (3.61)$$

$$\bar{p} = (\gamma-1) \left(\rho \bar{E}_r - \frac{1}{2} \rho (\bar{u}_{r,i} \bar{u}_{r,i} - V_{t,i} V_{t,i}) - \rho K \right) \quad (3.62)$$

$$\bar{p} = \frac{\gamma-1}{\gamma} \left(\rho \bar{H}_r - \frac{1}{2} \rho (\bar{u}_{r,i} \bar{u}_{r,i} - V_{t,i} V_{t,i}) - \rho K \right) \quad (3.63)$$

$$\frac{\bar{T}_t}{\bar{T}} = \frac{\bar{H}_r}{\bar{h}} = 1 + \frac{\gamma-1}{2} \left(M^2 + M_T^2 - \frac{V_{t,i} V_{t,i}}{a^2} \right) \quad (3.64)$$

Now we return to the governing equations. The non-inertial formulation of the Navier-Stokes equations is time-averaged:

$$\frac{\partial \bar{\mathbf{U}}}{\partial t} + \frac{\partial \bar{\mathbf{F}}_i}{\partial x_{b,i}} = \frac{\partial \bar{\mathbf{F}}_{v,i}}{\partial x_{b,i}} + \bar{\mathbf{S}} \quad (3.65)$$

where

$$\bar{\mathbf{U}} = \begin{Bmatrix} \rho \\ \rho \bar{u}_{r,j} \\ \rho \bar{E}_r \end{Bmatrix} \quad \bar{\mathbf{F}}_i = \begin{Bmatrix} \rho \bar{u}_{r,i} \\ \rho \bar{u}_{r,i} \bar{u}_{r,j} + \rho \overline{u'_{r,i} u'_{r,j}} + \bar{p} \delta_{ij} \\ \bar{u}_{r,i} \rho \bar{H}_r + \bar{u}_{r,j} \rho \overline{u'_{r,i} u'_{r,j}} + \rho \overline{u'_{r,i} h'} + \rho \overline{u'_{r,i} \frac{1}{2} u'_{r,j} u'_{r,j}} \end{Bmatrix} \quad (3.66)$$

$$\bar{\mathbf{F}}_{v,i} = \begin{Bmatrix} 0 \\ \bar{\tau}_{ij} \\ \bar{\tau}_{ij} \bar{u}_{r,j} + \overline{\tau'_{ij} u'_{r,j}} - \bar{q}''_i \end{Bmatrix} \quad \bar{\mathbf{S}} = -\rho \begin{Bmatrix} 0 \\ \bar{a}_{t,j} + \varepsilon_{jkl} \omega_k \bar{u}_{r,l} \\ \bar{a}_{t,k} \cdot (V_{t,k} + \bar{u}_{r,k}) + a_{t,k} \overline{u'_{r,k}} \end{Bmatrix} \quad (3.67)$$

$$\bar{u}_i = B_{ij} (\bar{u}_{r,j} + V_{t,j}) \quad (3.68)$$

$$\bar{a}_{t,i} = a_{o,i} + \varepsilon_{ijk} \omega_j \varepsilon_{klm} \omega_l x_{b,m} + \varepsilon_{ijk} \dot{\omega}_j \cdot x_{b,k} + \varepsilon_{ijk} \omega_j \bar{u}_{r,k} \quad (3.70)$$

$$\overline{a_{t,i} u'_{r,i}} = \varepsilon_{ijk} \omega_j \overline{u'_{r,i} u'_{r,k}} = 0 \quad (3.71)$$

Notice that the continuity equation does not contain any Reynolds-terms, while the energy equation contains five such terms. (All three Reynolds-terms in the inviscid energy flux are derived by expanding H_r using Eq. 3.25 before Reynolds-averaging. Then terms are collected back together using Eq. 3.61.) Also notice that the energy source term has a Reynolds-term that is created by Reynolds –averaging the mesh acceleration $a_{t,k}$ dotted with the relative velocity $u'_{r,k}$, but that term is identically zero (Eq. 3.71).

The Reynolds-stress and turbulent kinetic energy have already been defined. Three more Reynolds-terms are needed to complete the representation, which will be defined in the next section. For now the three terms are defined as the *turbulent transport of heat* $\rho \overline{u'_{r,i} h'}$, *molecular diffusion* $\overline{\tau'_{ij} u'_{r,j}}$, and *turbulent transport of turbulent kinetic energy* $\rho \overline{u'_{r,i} \frac{1}{2} u'_{r,j} u'_{r,j}}$.

3.4.1.2 Closure Terms

The Reynolds stress tensor, turbulent transport of heat, molecular diffusion, and turbulent transport of turbulent kinetic energy are modeled using closure models, either as algebraic or

differential equations. For the simple models used in this work, the Reynolds stress tensor T_{ij} is modeled using Boussinesq's approximation:

$$\rho T_{ij} \approx \mu_T \left(\frac{\partial \bar{u}_{r,i}}{\partial x_j} + \frac{\partial \bar{u}_{r,j}}{\partial x_i} \right) - \frac{2}{3} \rho K \delta_{ij} = \frac{\mu_T}{\mu} \bar{\tau}_{ij} - \frac{2}{3} \rho K \delta_{ij} \quad (3.72)$$

where μ_T is the eddy viscosity. The turbulent kinetic energy is included so that the identity is maintained¹:

$$\rho T_{ii} \approx 2\mu_T \frac{\partial \bar{u}_{r,i}^2}{\partial x_i} - \frac{2}{3} \rho K \delta_{ii} = -\frac{2}{3} \rho K (3) = -2\rho K \quad (3.73)$$

The molecular diffusion $\overline{u'_{r,j} \tau'_{ij}}$ and turbulent kinetic energy transport $\overline{u'_{r,i} \frac{1}{2} u'_{r,j} u'_{r,j}}$ are approximated using the gradient of turbulent kinetic energy (Wilcox, 2002):

$$\overline{u'_{r,j} \tau'_{ij}} - \rho \overline{u'_{r,i} \frac{1}{2} u'_{r,j} u'_{r,j}} \approx (\mu + \sigma_k \mu_T) \frac{\partial K}{\partial x_i} \quad (3.74)$$

where σ_k is the correlation of the turbulent transport vector and kinetic energy gradients, and may change depending on the gradient being used. Generally, these quantities are assumed to be perfectly correlated for all three directions, so σ_k is constant for the model and only used to scale the calculation.

The turbulent heat transport Q_i is constructed using Reynolds analogy for momentum and heat transfer, seen in the molecular Prandtl and turbulent Prandtl numbers (Kays, et al, 2005).

From Eq. 3.15, 3.23, and 3.33, the molecular conduction is calculated:

¹ The three in parentheses (3) arises by taking the trace in three dimensions, since turbulence is truly a three-dimensional process. In other words, in three-dimensions, $\delta_{ii} = 3$.

$$\bar{q}_i'' = -k \frac{\partial \bar{T}}{\partial x_j} = -\frac{c_p \mu}{Pr} \frac{\partial \bar{T}}{\partial x_j} = -\frac{\mu}{Pr} \frac{\partial \bar{h}}{\partial x_j} \quad (3.75)$$

through analogy

$$Q_i = \rho \overline{u_i' h'} \approx -\frac{\mu_T}{Pr_T} \frac{\partial \bar{h}}{\partial x_i} \quad (3.76)$$

where Pr_T is the *turbulent Prandtl number* (= 0.9, for air).

When the Reynolds -terms are substituted back into Eqs. 3.65 through 3.67, the non-inertial RANS equations are rearranged placing Reynolds-terms with the viscous flux terms:

$$\frac{\partial \bar{U}}{\partial t} + \frac{\partial \bar{\mathbf{F}}_i^*}{\partial x_{b,i}} = \frac{\partial \bar{\mathbf{F}}_{v,i}^*}{\partial x_{b,i}} + \bar{\mathbf{S}} \quad (3.77)$$

where

$$\bar{\mathbf{F}}_i^* = \begin{Bmatrix} \rho \bar{u}_{r,i} \\ \rho \bar{u}_{r,i} \bar{u}_{r,j} + \bar{p} \delta_{ij} \\ \bar{u}_{r,i} \rho \bar{H}_r \end{Bmatrix} \quad \bar{\mathbf{S}} = -\rho \begin{Bmatrix} 0 \\ \bar{a}_{t,j} + \varepsilon_{jkl} \omega_k \bar{u}_{r,l} \\ \bar{a}_{t,k} \cdot (V_{t,k} + \bar{u}_{r,k}) \end{Bmatrix} \quad (3.78)$$

$$\bar{\mathbf{F}}_{v,i}^* = \begin{Bmatrix} 0 \\ \bar{\tau}_{ij} + \rho \Gamma_{ij} \\ \bar{u}_{r,j} (\bar{\tau}_{ij} + \rho \Gamma_{ij}) - \bar{q}_i'' - Q_i + (\mu + \sigma_k \mu_T) \nabla_i K \end{Bmatrix} \quad (3.79)$$

These equations contain mean flow properties, eddy viscosity μ_T , and turbulent kinetic energy K . Simplified models (mixing-length, algebraic, and one-equation model) have no way to calculate K and neglect the terms containing any form of turbulent kinetic energy.

3.4.1.3 Turbulent Kinetic Energy Equation

Turbulent kinetic energy is defined as:

$$K = \frac{1}{2} \overline{u'_j u'_j} \quad (3.80)$$

A governing equation can be created by time-averaging the dot product of velocity vector fluctuation with the momentum equation (as a vector) (Wilcox, 2002):

$$\overline{u'_j \left[\frac{\partial}{\partial t} (\bar{u}_j + u'_j) + \frac{\partial}{\partial x_i} ((\bar{u}_i + u'_i)(\bar{u}_j + u'_j)) \right]} = \overline{u'_j \left[\frac{1}{\rho} \frac{\partial}{\partial x_i} (-(\bar{p} + p')\delta_{ij} + \bar{\tau}_{ij} + \tau'_{ij}) \right]} \quad (3.81)$$

After time-averaging and manipulation, the equation becomes:

$$\begin{aligned} \frac{\partial \left(\frac{1}{2} \overline{u'_j u'_j} \right)}{\partial t} + \bar{u}_i \frac{\partial \left(\frac{1}{2} \overline{u'_j u'_j} \right)}{\partial x_i} &= -\overline{u'_i u'_j} \frac{\partial \bar{u}_j}{\partial x_i} - \frac{1}{\rho} \overline{\tau'_{ij}} \frac{\partial \bar{u}_j}{\partial x_i} + \frac{1}{\rho} \overline{p'} \frac{\partial \bar{u}_i}{\partial x_i} \\ &\quad - \frac{1}{\rho} \frac{\partial}{\partial x_i} \left(\overline{u'_i p'} + \rho \overline{u'_i \frac{1}{2} u'_j u'_j} - \overline{\tau'_{ij} u'_j} \right) \end{aligned} \quad (3.82)$$

The second term on the right side is called the *dissipation of turbulent kinetic energy*:

$$\rho \varepsilon = \overline{\tau'_{ij} \frac{\partial u'_j}{\partial x_i}} \quad (3.83)$$

The *pressure diffusion* term $\overline{u'_i p'}$ and *pressure dilatation* $\overline{p'(\nabla \cdot \mathbf{u}')}$ are often neglected for incompressible flows. Substituting in T_{ij} , K , ε , and the closure approximations discussed above (Eq. 3.74), the equation becomes:

$$\frac{\partial K}{\partial t} + \bar{u}_i \frac{\partial K}{\partial x_i} = T_{ij} \frac{\partial \bar{u}_j}{\partial x_i} - \varepsilon + \frac{1}{\rho} \frac{\partial}{\partial x_i} \left((\mu + \sigma_k \mu_T) \frac{\partial K}{\partial x_i} \right) \quad (3.84)$$

3.4.1.4 Turbulent Dissipation Equation

The previous section defined the *turbulent dissipation* (Eq. 3.83). An approximation for the turbulent dissipation rate can be constructed:

$$\rho \varepsilon = \overline{\tau'_{ij} \frac{\partial u'_j}{\partial x_i}} = \overline{\mu \left(\frac{\partial u'_i}{\partial x_j} + \frac{\partial u'_j}{\partial x_i} \right) \frac{\partial u'_j}{\partial x_i}} \approx \overline{\mu \frac{\partial u'_j}{\partial x_i} \frac{\partial u'_j}{\partial x_i}} \quad (3.85)$$

A governing equation can be created by time-averaging viscosity times the dot product of the gradient of velocity fluctuation with the gradient of the momentum equation (as a vector):

$$\begin{aligned} \overline{\mu \frac{\partial u'_j}{\partial x_i} \frac{\partial}{\partial x_i} \left[\frac{\partial}{\partial t} (\bar{u}_j + u'_j) + \frac{\partial}{\partial x_i} ((\bar{u}_i + u'_i)(\bar{u}_j + u'_j)) \right]} \\ = \overline{\mu \frac{\partial u'_j}{\partial x_i} \frac{\partial}{\partial x_i} \left[\frac{1}{\rho} \frac{\partial}{\partial x_i} (-(\bar{p} + p')\delta_{ij} + \bar{\tau}_{ij} + \tau'_{ij}) \right]} \end{aligned} \quad (3.86)$$

After time-averaging and much manipulation, the equation becomes (Shih, 1995):

$$\begin{aligned} \rho \frac{\partial \varepsilon}{\partial t} + \rho \bar{u}_i \frac{\partial \varepsilon}{\partial x_i} = \frac{\partial}{\partial x_i} \left(\mu \frac{\partial \varepsilon}{\partial x_i} - \overline{\mu u'_i \frac{\partial u'_j}{\partial x_k} \frac{\partial u'_j}{\partial x_k}} - 2\mu \overline{\frac{\partial u'_i}{\partial x_j} \frac{\partial p'}{\partial x_j}} \right) \\ - 2\mu \frac{\partial \bar{u}_j}{\partial x_i} \left(\overline{\frac{\partial u'_i}{\partial x_k} \frac{\partial u'_j}{\partial x_k}} + \overline{\frac{\partial u'_k}{\partial x_i} \frac{\partial u'_k}{\partial x_j}} \right) - 2\mu \overline{u'_k \frac{\partial u'_i}{\partial x_j} \frac{\partial^2 \bar{u}_i}{\partial x_j \partial x_k}} \\ - 2\mu \overline{\frac{\partial u'_i}{\partial x_j} \frac{\partial u'_i}{\partial x_k} \frac{\partial u'_k}{\partial x_j}} - 2 \frac{\mu^2}{\rho} \overline{\frac{\partial^2 u'_i}{\partial x_j \partial x_k} \frac{\partial^2 u'_i}{\partial x_j \partial x_k}} \end{aligned} \quad (3.87)$$

This equation is very complicated and requires many closure approximations. Modelers usually choose a more empirical approach to developing a dissipation equation:

$$\frac{\partial \varepsilon}{\partial t} + \bar{u}_i \frac{\partial \varepsilon}{\partial x_i} = C_{\varepsilon 1} \frac{\varepsilon}{K} T_{ij} \frac{\partial \bar{u}_j}{\partial x_i} + \frac{1}{\rho} \frac{\partial}{\partial x_i} \left((\mu + \sigma_{\varepsilon} \mu_T) \frac{\partial \varepsilon}{\partial x_i} \right) - C_{\varepsilon 2} \frac{\varepsilon^2}{K} \quad (3.88)$$

3.4.1.5 ω -Equation

Some two equation turbulent models use the kinetic energy equation along with the transport of a property ω , which model designers describe using different attributes. Wilcox (2002) simply suggests that ω is proportional to the ratio of dissipation and kinetic energy:

$$\omega = \frac{1}{C_\mu} \frac{\varepsilon}{K} \quad \text{or} \quad \varepsilon = C_\mu \omega K \quad (3.89)$$

The ω -transport equation is generally developed by postulating an equation from the most common processes and then generating closure coefficients that accurately match basic flows. The ω -equation can also be created by starting with the ε -equation and the relationship above and replacing the terms with those of K and ω . The chain rule can be applied to the above relationship, assuming that C_μ is constant:

$$\frac{\partial \varepsilon}{\partial t} = C_\mu K \frac{\partial \omega}{\partial t} + C_\mu \omega \frac{\partial K}{\partial t} \quad \frac{\partial \varepsilon}{\partial x_i} = C_\mu K \frac{\partial \omega}{\partial x_i} + C_\mu \omega \frac{\partial K}{\partial x_i} \quad (3.90)$$

Substituting these relationships into the empirical form of the ε -equation:

$$\begin{aligned} K \left(\frac{\partial \omega}{\partial t} + \bar{u}_i \frac{\partial \omega}{\partial x_i} \right) + \omega \left(\frac{\partial K}{\partial t} + \bar{u}_i \frac{\partial K}{\partial x_i} \right) &= C_{\varepsilon 1} \omega T_{ij} \frac{\partial \bar{u}_j}{\partial x_i} - C_{\varepsilon 2} C_\mu \omega^2 K \\ &+ \frac{1}{\rho} \frac{\partial}{\partial x_i} \left((\mu + \sigma_\varepsilon \mu_T) \left(K \frac{\partial \omega}{\partial x_i} + \omega \frac{\partial K}{\partial x_i} \right) \right) \end{aligned} \quad (3.91)$$

Subtracting the K -equation from the previous equation and dividing by K :

$$\begin{aligned} \frac{\partial \omega}{\partial t} + \bar{u}_i \frac{\partial \omega}{\partial x_i} &= (C_{\varepsilon 1} - 1) \frac{\omega}{K} T_{ij} \frac{\partial \bar{u}_j}{\partial x_i} + (1 - C_{\varepsilon 2}) C_\mu \omega^2 \\ &+ \frac{1}{\rho} \frac{\partial}{\partial x_i} \left((\mu + \sigma_\varepsilon \mu_T) \frac{\partial \omega}{\partial x_i} \right) + \frac{2}{\rho K} (\mu + \sigma_\varepsilon \mu_T) \frac{\partial \omega}{\partial x_i} \frac{\partial K}{\partial x_i} \end{aligned} \quad (3.92)$$

The last term is referred to as *cross-diffusion*, where the derivatives of K and ω are both used in one term. The cross-diffusion term demonstrates a major difference between the k - ε and k - ω models. Many modelers have attempted to improve their k - ω models by including the cross-diffusion term, but the physics of the turbulence do not support such a term (Wilcox, 2002). Eliminating this term and combining the coefficients, the ω -equation can be written:

$$\frac{\partial \omega}{\partial t} + \bar{u}_i \frac{\partial \omega}{\partial x_i} = \alpha \frac{\omega}{K} \Gamma_{ij} \frac{\partial \bar{u}_j}{\partial x_i} + \beta \omega^2 + \frac{1}{\rho} \frac{\partial}{\partial x_i} \left((\mu + \sigma_\omega \mu_T) \frac{\partial \omega}{\partial x_i} \right) \quad (3.93)$$

3.4.1.6 Compressible RANS

Reynolds-averaging the compressible Navier-Stokes equations is much more complex. The density fluctuation must also be taken into account now: $\rho = \bar{\rho} + \rho'$. To illustrate the added complexity, the compressible momentum equation (inertial) is time-averaged:

$$\frac{\partial}{\partial t} ((\bar{\rho} + \rho')(\bar{u}_j + u'_j)) + \frac{\partial}{\partial x_i} ((\bar{\rho} + \rho')(\bar{u}_i + u'_i)(\bar{u}_j + u'_j)) = \frac{\partial}{\partial x_i} (-(\bar{p} + p')\delta_{ij} + \bar{\tau}_{ij} + \tau'_{ij}) \quad (3.94)$$

After time averaging:

$$\frac{\partial}{\partial t} (\bar{\rho} \bar{u}_j + \overline{\rho' u'_j}) + \frac{\partial}{\partial x_i} (\bar{\rho} \bar{u}_i \bar{u}_j + \bar{u}_j \overline{\rho' u'_i} + \bar{u}_i \overline{\rho' u'_j} + \bar{\rho} \overline{u'_i u'_j} + \overline{\rho' u'_i u'_j}) = \frac{\partial}{\partial x_i} (-\bar{p} \delta_{ij} + \bar{\tau}_{ij}) \quad (3.95)$$

The momentum equation contains many more Reynolds terms, including a term in the time derivative. The compressible RANS equations have a total of 34 Reynolds terms. These equations require 34 closure models, compared to the seven used by incompressible RANS. We will alleviate some of the Reynolds terms by using mass-averaging in the next section. Before moving on, we should examine the compressible RANS equations (Eqs. 3.96 through 3.104), particularly, Eqs. 3.100 and 3.104 show a non-inertial source created by turbulence. The first term of Eq. 104 is identically zero, like Eq. 3.71, but the remaining terms contain $\overline{\rho' u'_{r,i}}$, which will be eliminated by mass-averaging instead of purely time-averaging.

$$\frac{\partial \bar{\mathbf{U}}}{\partial t} + \frac{\partial \bar{\mathbf{F}}_i}{\partial x_{b,i}} = \frac{\partial \bar{\mathbf{F}}_{v,i}}{\partial x_{b,i}} + \bar{\mathbf{S}} \quad (3.96)$$

where

$$\bar{\mathbf{U}} = \left\{ \begin{array}{c} \bar{\rho} \\ \bar{\rho} \bar{u}_{r,j} + \overline{\rho' u'_{r,j}} \\ \bar{\rho} \bar{E}_r + \overline{\rho' e'} + \frac{1}{2} \left(2 \bar{u}_{r,i} \overline{\rho' u'_{r,i}} + \overline{\rho' u'_{r,i} u'_{r,i}} \right) \end{array} \right\} \quad (3.97)$$

$$\bar{\mathbf{F}}_i = \left\{ \begin{array}{c} \bar{\rho} \bar{u}_{r,i} + \overline{\rho' u'_{r,i}} \\ \bar{\rho} \bar{u}_{r,i} \bar{u}_{r,j} + \bar{u}_{r,j} \overline{\rho' u'_{r,i}} + \bar{u}_{r,i} \overline{\rho' u'_{r,j}} - \bar{\rho} \bar{T}_{ij} + \overline{\rho' u'_{r,i} u'_{r,j}} + \bar{\rho} \bar{\delta}_{ij} \\ \bar{u}_{r,i} \bar{\rho} \bar{H}_r + (\bar{H}_r - K) \overline{\rho' u'_{r,i}} + \bar{\rho} \left(\bar{u}'_{r,i} \bar{h}' - \bar{u}_{r,k} \bar{T}_{ik} + \frac{1}{2} \overline{u'_{r,i} u'_{r,k} u'_{r,k}} \right) + \overline{u'_{r,i} \rho' h'} \\ + \bar{u}_{r,i} \left(\overline{\rho' h'} + \bar{u}_{r,k} \overline{\rho' u'_{r,k}} + \frac{1}{2} \overline{\rho' u'_{r,k} u'_{r,k}} \right) + \bar{u}_{r,k} \overline{\rho' u'_{r,i} u'_{r,k}} + \overline{\rho' u'_{r,i} \frac{1}{2} u'_{r,k} u'_{r,k}} \end{array} \right\} \quad (3.98)$$

$$\bar{\mathbf{F}}_{v,i} = \left\{ \begin{array}{c} 0 \\ \bar{\tau}_{ij} \\ \bar{\tau}_{ij} \bar{u}_{r,j} + \overline{\tau'_{ij} u'_{r,j}} - \bar{q}''_i \end{array} \right\} \quad (3.99)$$

$$\bar{\mathbf{S}} = -\bar{\rho} \left\{ \begin{array}{c} 0 \\ \bar{a}_{t,j} + \varepsilon_{jkl} \omega_k \bar{u}_{r,l} \\ \bar{a}_{t,k} \cdot (V_{t,k} + \bar{u}_{r,k}) \end{array} \right\} - \left\{ \begin{array}{c} 0 \\ 2 \varepsilon_{jkl} \omega_k \overline{\rho' u'_{r,l}} \\ \bar{a}_{t,k} \overline{\rho' u'_{r,k}} + \overline{\rho a'_{t,k} \cdot (V_{t,k} + u_{r,k})} \end{array} \right\} \quad (3.100)$$

$$\bar{\rho} \bar{E}_r = \bar{\rho} \bar{e} + \frac{1}{2} \bar{\rho} (\bar{u}_{r,i} \bar{u}_{r,i} - V_{t,i} V_{t,i}) + \bar{\rho} K \quad (3.101)$$

$$\bar{\rho} \bar{H}_r = \bar{\rho} \bar{h} + \frac{1}{2} \bar{\rho} (\bar{u}_{r,i} \bar{u}_{r,i} - V_{t,i} V_{t,i}) + \bar{\rho} K \quad (3.102)$$

$$\bar{a}_{t,i} = a_{o,i} + \varepsilon_{ijk} \omega_j \varepsilon_{klm} \omega_l x_{b,m} + \varepsilon_{ijk} \dot{\omega}_j x_{b,k} + \varepsilon_{ijk} \omega_j \bar{u}_{r,k} \quad (3.103)$$

$$\overline{\rho a'_{t,k} \cdot (V_{t,k} + u_{r,k})} = \varepsilon_{klm} \omega_l \left(\overline{\rho' u'_{r,k} u'_{r,m}} - \bar{\rho} \bar{T}_{km} \right) + \varepsilon_{klm} \omega_l (V_{t,k} + \bar{u}_{r,k}) \overline{\rho' u'_{r,m}} \quad (3.104)$$

3.4.2 Favre-Averaged Navier-Stokes (FANS)

The time-averaged momentum equation can be greatly simplified using Favre-Reynolds-averaging, or mass-averaging. Thermodynamic properties still represented using the Reynolds notation; velocities and energy terms are now represented using Favre notation:

$u = \tilde{u} + u''$, $e = \tilde{e} + e''$, and $h = \tilde{h} + h''$, so that (Wilcox, 2002):

$$\tilde{u}_j = \frac{1}{\bar{\rho}} \frac{1}{T} \int_t^{t+T} \rho u_j dt \quad \tilde{e} = \frac{1}{\bar{\rho}} \frac{1}{T} \int_t^{t+T} \rho e dt \quad \tilde{h} = \frac{1}{\bar{\rho}} \frac{1}{T} \int_t^{t+T} \rho h dt \quad (3.105)$$

Interestingly enough, when the velocity and energy terms are time-averaged **without** mass, the velocity fluctuations do **not** disappear in Favre-averaging:

$$\frac{1}{T} \int_t^{t+T} u_j dt = \tilde{u}_j + \overline{u_j''} \quad \frac{1}{T} \int_t^{t+T} u_j'' dt = \overline{u_j''} \neq 0 \quad (3.106)$$

The Favre-Reynolds-averaged momentum equation is written:

$$\frac{\partial}{\partial t} (\bar{\rho} \tilde{u}_j) + \frac{\partial}{\partial x_i} (\bar{\rho} \tilde{u}_i \tilde{u}_j + \overline{\rho u_i'' u_j''}) = \frac{\partial}{\partial x_i} (-\bar{p} \delta_{ij} + \tilde{\tau}_{ij}) \quad (3.107)$$

which closely resembles the incompressible equation, including its number of terms to close.

If we assume Boussinesq's approximation, the Reynolds stresses can be modeled using the molecular stresses as a pattern. Eddy viscosity is used in place of molecular viscosity:

$$\tilde{\tau}_{ij} = \bar{\mu} \left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} - \frac{2}{3} \frac{\partial \tilde{u}_k}{\partial x_k} \delta_{ij} \right) \quad (3.108)$$

$$-\overline{\rho u_i'' u_j''} = \bar{\rho} \tilde{T}_{ij} \approx \mu_T \left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} - \frac{2}{3} \frac{\partial \tilde{u}_k}{\partial x_k} \delta_{ij} \right) - \frac{2}{3} \bar{\rho} \tilde{K} \delta_{ij} \quad (3.109)$$

The turbulent kinetic energy term is added to the normal stresses so that the trace of the

Reynolds stress tensor is equal to the turbulent kinetic energy²:

$$\bar{\rho} \tilde{T}_{ii} = \mu_T \left(2 \frac{\partial \tilde{u}_i}{\partial x_i} - \left(\frac{2}{3} \frac{\partial \tilde{u}_k}{\partial x_k} \right) (3) \right) - \left(\frac{2}{3} \bar{\rho} \tilde{K} \right) (3) = -2 \bar{\rho} \tilde{K} = \overline{\rho u_i'' u_i''} \quad (3.110)$$

² Again, the three in parentheses (3) arises by taking the trace in three dimensions, since turbulence is truly a three-dimensional process. In other words, in three-dimensions, $\delta_{ii} = 3$.

Notice that Stokes' hypothesis has been specified in the molecular stresses. When the Reynolds stress is patterned after the molecular stress with Stokes' hypothesis, the trace of the first term vanishes, so Stokes' hypothesis is required in Boussinesq's approximation.

Also, notice that the mass-average viscous stress was posed in terms of the mass-averaged mean velocities. If this stress is subtracted from the entire viscous stress, the remaining stress term is calculated:

$$\begin{aligned}\tau_{ij}'' &= \tau_{ij} - \tilde{\tau}_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} + \frac{\lambda}{\mu} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right) - \bar{\mu} \left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} + \frac{\lambda}{\mu} \frac{\partial \tilde{u}_k}{\partial x_k} \delta_{ij} \right) \\ &= \bar{\mu} \left(\frac{\partial u_i''}{\partial x_j} + \frac{\partial u_j''}{\partial x_i} + \frac{\lambda}{\mu} \frac{\partial u_k''}{\partial x_k} \delta_{ij} \right) + \mu' \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} + \frac{\lambda}{\mu} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right)\end{aligned}\quad (3.111)$$

and

$$\frac{1}{T} \int_t^{t+T} \tau_{ij} dt = \tilde{\tau}_{ij} \quad \frac{1}{T} \int_t^{t+T} \tau_{ij} - \tilde{\tau}_{ij} dt = \frac{1}{T} \int_t^{t+T} \tau_{ij}'' dt = 0 \quad (3.112)$$

The momentum equation becomes:

$$\frac{\partial}{\partial t} (\bar{\rho} \tilde{u}_j) + \frac{\partial}{\partial x_i} (\bar{\rho} \tilde{u}_i \tilde{u}_j) = \frac{\partial}{\partial x_i} (-\bar{p} \delta_{ij} + \tilde{\tau}_{ij} + \bar{\rho} \tilde{T}_{ij}) \quad (3.113)$$

The Favre-Reynolds averaged Navier-Stokes equations³ (non-inertial) are written:

$$\frac{\partial \tilde{U}}{\partial t} + \frac{\partial \tilde{F}_i}{\partial x_{b,i}} = \frac{\partial \tilde{F}_{v,i}}{\partial x_{b,i}} + \tilde{S} \quad (3.114)$$

where

³ For comparison, Zhang, et al (1993) presents a very different form of the compressible energy equation where the pressure and viscous stress terms have been broken out using the chain rule. The turbulent energy dissipation rate $\rho \varepsilon$ is included in the energy equation. Many more closure models are required for the expanded form.

$$\tilde{\mathbf{U}} = \begin{Bmatrix} \bar{\rho} \\ \overline{\rho \tilde{u}_{r,j}} \\ \overline{\rho \tilde{E}_r} \end{Bmatrix} \quad \tilde{\mathbf{F}}_i = \begin{Bmatrix} \overline{\rho \tilde{u}_{r,i}} \\ \overline{\rho \tilde{u}_{r,i} \tilde{u}_{r,j}} + \overline{\rho u_{r,i}'' u_{r,j}''} + \bar{p} \delta_{ij} \\ \tilde{u}_{r,i} \overline{\rho \tilde{H}_r} + \bar{u}_{r,j} \overline{\rho u_{r,i}'' u_{r,j}''} + \overline{\rho u_{r,i}'' h''} + \overline{\rho u_{r,i}'' \frac{1}{2} u_{r,j}'' u_{r,j}''} \end{Bmatrix} \quad (3.115)$$

$$\tilde{\mathbf{F}}_{v,i} = \begin{Bmatrix} 0 \\ \tilde{\tau}_{ij} \\ \tilde{\tau}_{ij} \tilde{u}_{r,j} + \overline{\tau_{ij}'' u_{r,j}''} - \tilde{q}_i'' \end{Bmatrix} \quad \tilde{\mathbf{S}} = \begin{Bmatrix} 0 \\ \overline{\rho a_{t,i}} + \varepsilon_{jkl} \omega_k \overline{\rho \tilde{u}_{r,l}} \\ \overline{\rho a_{t,k}} \cdot (V_{t,k} + \tilde{u}_{r,k}) + \overline{\rho a_{t,k} u_{r,k}''} \end{Bmatrix} \quad (3.116)$$

$$\overline{\rho \tilde{u}_i} = B_{ij} (\overline{\rho \tilde{u}_{r,j}} + \overline{\rho V_{t,j}}) \quad (3.117)$$

$$\overline{\rho a_{t,i}} = \bar{\rho} (a_{o,i} + \varepsilon_{ijk} \omega_j \varepsilon_{klm} \omega_l x_{b,m} + \varepsilon_{ijk} \dot{\omega}_j x_{b,k}) + \varepsilon_{ijk} \omega_j \overline{\rho \tilde{u}_{r,k}} \quad (3.118)$$

$$\overline{\rho a_{t,i} u_{r,i}''} = \varepsilon_{ijk} \omega_j \overline{\rho u_{r,i}'' u_{r,k}''} = 0 \quad (3.119)$$

$$\overline{\rho \tilde{E}_r} = \overline{\rho \tilde{e}} + \frac{1}{2} \bar{\rho} (\overline{\tilde{u}_{r,i} \tilde{u}_{r,i}} - V_{t,i} V_{t,i}) + \overline{\rho \tilde{K}} \quad (3.120)$$

$$\overline{\rho \tilde{H}_r} = \overline{\rho \tilde{h}} + \frac{1}{2} \bar{\rho} (\overline{\tilde{u}_{r,i} \tilde{u}_{r,i}} - V_{t,i} V_{t,i}) + \overline{\rho \tilde{K}} = \overline{\rho \tilde{E}_r} + \bar{p} \quad (3.121)$$

The turbulent transport of heat is approximated using Reynolds analogy (Kays, 2005):

$$\tilde{q}_i'' = -\frac{\bar{\mu}}{\text{Pr}} \frac{\partial \tilde{h}}{\partial x_i} \quad \overline{\rho u_{r,i}'' h''} = \tilde{Q}_i \approx -\frac{\mu_T}{\text{Pr}_T} \frac{\partial \tilde{h}}{\partial x_i} \quad (3.122)$$

where Pr_T is the turbulent Prandtl number ($= 0.9$, for air). The molecular diffusion $\overline{\tau_{ij}'' u_j''}$ and transport of turbulent kinetic energy $\overline{\rho u_i'' \frac{1}{2} u_j'' u_j''}$ are approximated using the gradient of K :

$$\overline{\tau_{ij}'' u_j''} - \overline{\rho u_i'' \frac{1}{2} u_j'' u_j''} \approx (\bar{\mu} + \sigma_k \mu_T) \frac{\partial \tilde{K}}{\partial x_i} \quad (3.123)$$

Substituting the Reynolds stresses and other closure terms into Eqs. 3.115 through 3.117, the Favre-average Navier-Stokes equations are written:

$$\frac{\partial \tilde{\mathbf{U}}}{\partial t} + \frac{\partial \tilde{\mathbf{F}}_i^*}{\partial x_{b,i}} = \frac{\partial \tilde{\mathbf{F}}_{v,i}^*}{\partial x_{b,i}} + \tilde{\mathbf{S}} \quad (3.124)$$

where

$$\tilde{\mathbf{F}}_i^* = \begin{Bmatrix} \overline{\rho \tilde{u}_{r,i}} \\ \overline{\rho \tilde{u}_{r,i} \tilde{u}_{r,j}} + \bar{p} \delta_{ij} \\ \tilde{u}_{r,i} \overline{\rho \tilde{H}_r} \end{Bmatrix} \quad \tilde{\mathbf{S}} = -\bar{\rho} \begin{Bmatrix} 0 \\ \tilde{a}_{t,j} + \varepsilon_{jkl} \omega_k \tilde{u}_{r,l} \\ \tilde{a}_{t,k} \cdot (V_{t,k} + \tilde{u}_{r,k}) \end{Bmatrix} \quad (3.125)$$

$$\tilde{\mathbf{F}}_{v,i}^* = \begin{Bmatrix} 0 \\ \tilde{\tau}_{ij} + \bar{\rho} \tilde{\mathbf{T}}_{ij} \\ \tilde{u}_{r,j} (\tilde{\tau}_{ij} + \bar{\rho} \tilde{\mathbf{T}}_{ij}) - \tilde{q}_i'' - \tilde{Q}_i + (\bar{\mu} + \sigma_k \mu_T) \nabla_i \tilde{K} \end{Bmatrix} \quad (3.126)$$

These equations must be supported by a number of thermodynamic equations (Eqs. 3.7, 3.22, 3.23, 3.30, 3.36 through 3.38, and 3.62 through 3.64), which must be written in terms of mass-averaged velocity and energy terms:

$$\overline{\rho \tilde{E}_r} = \bar{\rho} \tilde{E} - \overline{\rho \tilde{u}_i V_{t,i}} \quad (3.127)$$

$$\overline{\rho \tilde{e}} = \bar{\rho} c_v \tilde{T} = \frac{\bar{p}}{\gamma - 1} \quad (3.128)$$

$$\overline{\rho \tilde{h}} = \overline{\rho \tilde{e}} + \bar{p} = \bar{\rho} c_p \tilde{T} = \gamma \overline{\rho \tilde{e}} = \frac{\gamma \bar{p}}{\gamma - 1} \quad (3.129)$$

$$\bar{p} = (\gamma - 1) \left(\overline{\rho \tilde{E}_r} - \frac{1}{2} \overline{\rho (\tilde{u}_{r,i} \tilde{u}_{r,i} - V_{t,i} V_{t,i})} - \overline{\rho \tilde{K}} \right) \quad (3.130)$$

$$\bar{p} = \frac{\gamma - 1}{\gamma} \left(\overline{\rho \tilde{H}_r} - \frac{1}{2} \overline{\rho (\tilde{u}_{r,i} \tilde{u}_{r,i} - V_{t,i} V_{t,i})} - \overline{\rho \tilde{K}} \right) \quad (3.131)$$

$$M^2 = \frac{\overline{\rho u_{r,i} u_{r,i}}}{\rho a^2} = \frac{\overline{\tilde{u}_{r,i} \tilde{u}_{r,i}}}{\tilde{a}^2} \quad \overline{\rho \tilde{a}^2} = \overline{\rho a^2} = \gamma \bar{p} R \tilde{T} = \gamma \bar{p} \quad (3.132)$$

$$\frac{\tilde{T}_t}{\tilde{T}} = \frac{\tilde{H}_r}{\tilde{h}} = \frac{\overline{\rho \tilde{H}_r}}{\overline{\rho \tilde{h}}} = 1 + \frac{\overline{\rho u_{r,i} u_{r,i}} - \bar{\rho} V_{t,i} V_{t,i}}{2 \overline{\rho \tilde{h}}} = 1 + \frac{\gamma - 1}{2} \left(M^2 + M_T^2 - \frac{V_{t,i} V_{t,i}}{\tilde{a}^2} \right) \quad (3.133)$$

where the turbulent Mach number is defined as: $M_T^2 = \overline{\rho u_{r,i}'' u_{r,i}''} / \bar{\rho} \tilde{a}^2 = 2 \bar{\rho} \tilde{K} / \bar{\rho} \tilde{a}^2$.

3.4.2.1 Turbulent Kinetic Energy Equation

Mass-averaged turbulent kinetic energy is defined as:

$$\bar{\rho} \tilde{K} = \frac{1}{2} \overline{\rho u_i'' u_i''} \quad (3.134)$$

The Favre-Reynolds averaged turbulent kinetic energy equation is written (Rubesin, 1990):

$$\frac{D\bar{\rho} \tilde{K}}{Dt} = \bar{\rho} \tilde{T}_{ij} \frac{\partial \tilde{u}_i}{\partial x_j} + \overline{p' \frac{\partial u_i''}{\partial x_i}} - \overline{\tau_{ij} \frac{\partial u_i''}{\partial x_j}} - \overline{u_i'' \frac{\partial \bar{p}}{\partial x_i}} + \frac{\partial}{\partial x_i} \left(\overline{\tau_{ij} u_j''} - \overline{\rho u_i'' \frac{1}{2} u_j'' u_j''} - \overline{u_i'' p'} \right) \quad (3.135)$$

The mass-averaged *turbulent production and dissipation rates* are consolidated:

$$\bar{\rho} \tilde{\Pi} = \bar{\rho} \tilde{T}_{ij} \frac{\partial \tilde{u}_i}{\partial x_j} \quad \bar{\rho} \tilde{\varepsilon} = \overline{\tau_{ij} \frac{\partial u_i''}{\partial x_j}} \quad (3.136)$$

Substituting Eqs. 3.123 and 3.136, the turbulent kinetic energy equation becomes:

$$\frac{D\bar{\rho} \tilde{K}}{Dt} = \bar{\rho} \tilde{\Pi} - \bar{\rho} \tilde{\varepsilon} + \frac{\partial}{\partial x_i} \left((\bar{\mu} + \sigma_k \mu_T) \frac{\partial \tilde{K}}{\partial x_i} - \overline{u_i'' p'} \right) + \overline{p' \frac{\partial u_i''}{\partial x_i}} - \overline{u_i'' \frac{\partial \bar{p}}{\partial x_i}} \quad (3.137)$$

Three terms containing pressure fluctuations p' remain in Eq. 3.137: The pressure diffusion $\overline{u_i'' p'}$, pressure dilatation $\overline{p' \nabla_i u_i''}$, and pressure work $\overline{u_i'' \nabla_i \bar{p}}$ terms. These terms must vanish in the limit of incompressibility, so the pressure terms are often modeled as a function of the freestream Mach number, or the *turbulent Mach number*:

$$M_T^2 = \frac{2\bar{\rho} \tilde{K}}{\bar{\rho} \tilde{a}^2} = \frac{\overline{\rho u_i'' u_i''}}{\bar{\rho} \tilde{a}^2} = \frac{\overline{\rho u_i'' u_i''}}{\bar{\rho} \tilde{u}_i \tilde{u}_i} M^2 \quad (3.138)$$

In general, the three terms are modeled with *ad hoc* relationships that help the overall results match experimental trends, but these relationships are not physics based. The dilatation and *turbulent mass flow* $\overline{u_i''}$ are modeled (Sarkar, 1992; Krishnamurthy, 1997; Wilcox, 2002):

$$\overline{p' \frac{\partial u_i''}{\partial x_i}} \approx 0.15 \bar{\rho} \tilde{\Pi} M_T + 0.2 \bar{\rho} \tilde{\varepsilon} M_T^2 \quad \overline{u_i''} \approx \frac{M_T \tilde{K}}{\bar{\rho} \tilde{\varepsilon}} \tilde{T}_{ij} \frac{\partial \bar{\rho}}{\partial x_j} \quad (3.139)$$

Rubesin (1990) and Krishnamurthy (1997) offer an alternative form of the production term:

$$\overline{p' \frac{\partial u_i''}{\partial x_i}} = \bar{p} \left(\frac{\overline{u_i''} \frac{\partial \bar{\rho}}{\partial x_i}}{\bar{\rho}} - \frac{1}{2} \left(\frac{\partial \beta^2}{\partial t} + \tilde{u}_i \frac{\partial \beta^2}{\partial x_i} \right) \right) \quad \beta^2 = (\gamma - 1)^2 M^4 \frac{2 \bar{\rho} \tilde{K}}{\bar{\rho} \tilde{u}_i \tilde{u}_i} \quad (3.140)$$

Finally, the *pressure diffusion term* $\overline{u_i'' p'}$ was neglected in the RANS equation but cannot rightfully be neglected for compressible flows. Pressure diffusion should become negligible as Mach number approaches zero. Speziale and Sarkar (1991) write the pressure diffusion term in terms of the turbulent heat flux and mass flow:

$$\overline{u_i'' p'} = \bar{p} \left(\frac{1}{\tilde{h}} \frac{\mu_T}{\text{Pr}_T} \frac{\partial \tilde{h}}{\partial x_i} - \overline{u_i''} \right) \quad (3.141)$$

3.4.2.2 Turbulent Dissipation and ω -Equations

The ε - and ω -equations are not developed here using Favre-averaging. The pattern has been developed with the K -equation. Adjustments are made to both equations to obtain empirical trends demonstrated by experimental and DNS data. These models will be presented in the pages that follow.

3.4.3 **Spalart-Allmarus Model (Inertial)**

The Spalart-Allmaras (SA) model is a one-equation model that calculates the transport of eddy viscosity μ_T with a single differential equation (Spalart and Allmaras, 1992; Spalart, 2000). The model was developed at Boeing for standard aerodynamic applications (wings, fuselages, etc.), where the flow is attached to the surface. The model can also be applied to

free shear flows, but free jets should be avoided. This section will cover the original incompressible model and a more modern compressible version used by Deck (2002).

3.4.3.1 Incompressible Spalart-Allmarus

The original SA model uses a single differential equation in conjunction with an algebraic scaling function to calculate the eddy viscosity everywhere within the flow. The differential equation contains advection and diffusion terms that create a semi-realistic model of these advection and diffusion of turbulence. The model also contains production terms that utilize the near-wall region and high strain regions to generate turbulence at appropriate Reynolds numbers. The differential equation calculates a scalar $\hat{\nu}$ similar to (but definitely differing from) eddy viscosity. The scalar $\hat{\nu}$ is then used to calculate the eddy viscosity:

$$\mu_T = \rho \nu_T = f_{\nu l} \rho \hat{\nu} \quad (3.142)$$

The differential equation is assembled through inspection following the patterns of the turbulent kinetic energy equation shown in previous sections. The terms on the left side of Eq. 3.143 simulate the temporal changes to turbulence and its advection on the field. The terms on the right side are the production, destruction, diffusion, and stability:

$$\frac{\partial \hat{\nu}}{\partial t} + \bar{u}_j \frac{\partial \hat{\nu}}{\partial x_j} = c_{b1} \hat{S} \hat{\nu} - c_{w1} f_w \left(\frac{\hat{\nu}}{d} \right)^2 + \frac{\partial}{\partial x_j} \left(\frac{\bar{\nu} + \hat{\nu}}{\sigma} \frac{\partial \hat{\nu}}{\partial x_j} \right) + \frac{c_{b2}}{\sigma} \frac{\partial \hat{\nu}}{\partial x_j} \frac{\partial \hat{\nu}}{\partial x_j} \quad (3.143)$$

The production term is proportional to vorticity through \hat{S} . The destruction is inversely proportional to the distance to the nearest wall. Both the production and destruction of turbulence are scaled by the level of turbulence $\hat{\nu}$. The diffusion term is an analogy to the diffusion of K in Eq. Eq. 3.137, where the SA variable $\hat{\nu}$ has been used in place of the eddy

viscosity. The final term was added by the developers to enhance the stability of the model, but this work has found the term to be unreliable for stabilizing the Galerkin discretization.

Closure. The model has nine closure functions that are ultimately vary with the distance to the nearest wall d , making the model operate very similarly to a mixing length model:

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3} \quad f_{v2} = 1 - \frac{\chi}{1 + f_{v1}\chi} \quad \chi = \frac{\rho\hat{v}}{\mu} \quad (3.144)$$

$$f_w = \hat{g} \left(\frac{1 + c_{w3}^6}{\hat{g}^6 + c_{w3}^6} \right)^{1/6} \quad \hat{g} = r + c_{w2}(r^6 - r) \quad r = \frac{\hat{v}}{\hat{S}\kappa^2 d^2} \quad (3.145)$$

$$\hat{S} = \Omega + f_{v2} \frac{\hat{v}}{\kappa^2 d^2} \quad \Omega = \sqrt{2\Omega_{ij}\Omega_{ij}} \quad \Omega_{ij} = \frac{1}{2} \left(\frac{\partial \bar{u}_i}{\partial x_j} - \frac{\partial \bar{u}_j}{\partial x_i} \right) \quad (3.146)$$

The model also uses eight closure coefficients:

$$c_{b1} = 0.1355 \quad c_{b2} = 0.622 \quad c_{v1} = 7.1 \quad \sigma = \frac{2}{3} \quad (3.147)$$

$$c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1 + c_{b2}}{\sigma} \quad c_{w2} = 0.3 \quad c_{w3} = 2 \quad \kappa = 0.41 \quad (3.148)$$

Trip Transition. Spalart and Allmaras (1992) suggest a transition model created by adding functions to the source terms and a trip transition ΔU :

$$\begin{aligned} \frac{\partial \hat{v}}{\partial t} + \bar{u}_j \frac{\partial \hat{v}}{\partial x_j} = & c_{b1}(1 - f_{t2})\hat{S}\hat{v} - \left(c_{w1}f_w - f_{t2} \frac{c_{b1}}{\kappa^2} \right) \frac{1}{\text{Re}} \left(\frac{\hat{v}}{d} \right)^2 + f_{t1} \text{Re}(\Delta U)^2 \\ & + \frac{\partial}{\partial x_j} \left(\frac{\bar{v} + \hat{v}}{\sigma} \frac{\partial \hat{v}}{\partial x_j} \right) + \frac{c_{b2}}{\sigma} \frac{\partial \hat{v}}{\partial x_j} \frac{\partial \hat{v}}{\partial x_j} \end{aligned} \quad (3.149)$$

where

$$f_{t1} = c_{t1} g_t \exp\left(-c_{t2} \frac{\omega_t^2}{\Delta U^2} (d^2 + g_t^2 d_t^2)\right) \quad \Delta U = |\vec{V} - \vec{V}_t| \quad (3.150)$$

$$f_{t2} = c_{t3} \exp(c_{t4} \chi^2) \quad g_t = \text{MIN}\left(\frac{1}{10}, \frac{\Delta U}{\omega_t \Delta x}\right) \quad (3.151)$$

$$c_{t1} = 1 \quad c_{t2} = 2 \quad c_{t3} = 1.1 \quad c_{t4} = 2 \quad (3.152)$$

which are calculated using the velocity V_t and vorticity ω_t at the trip location and using the distance to the trip location d_t . The specification of the trip location is obscure and difficult for arbitrary three-dimensional flow fields.

Simplifications. The SA model does not include turbulent kinetic energy K , so these terms are neglected in the Reynolds stresses (Eq. 3.72) for simplicity. Molecular diffusion and turbulent transport are generally modeled using the gradient of turbulent kinetic energy (Eq. 3.74), so these terms are also neglected for simplicity in the SA model.

Modifications. Two immediate concerns arise with this model: (1) The model does not predict any decay in eddy viscosity in a uniform field (far from walls). (2) Implementing trip transition on an unstructured domain is difficult. To avoid the first problem, the initial conditions must remain small enough to allow laminar flow at lower Reynolds numbers, but the initial conditions must be sufficiently large to allow the production terms to grow turbulence. The transition model is avoided in this work to alleviate the second problem.

Production in the original SA model was centered around vorticity in near-wall regions. Since its conception, turbulence research has expanded this concept to include the effects of mean strain throughout a flow field. Dacles-Mariani, et. al (1995) and Fluent (2006) suggest modifying the SA production term:

$$\hat{S} = \Omega + C_{prod} MIN(0, S - \Omega) + f_{v2} \frac{\hat{v}}{\kappa^2 d^2} \quad (3.152)$$

$$S = \sqrt{2S_{ij}S_{ij}} \quad S_{ij} = \frac{1}{2} \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) \quad (3.153)$$

and where $C_{prod} = 2$. The original model often over-predicts eddy viscosity because of the exclusive emphasis on rotation. The modification to include both rotation and strain decreases the eddy viscosity to more accurate levels.

Oliver (2008) presents an alternative, which maintains a positive definite \hat{S} . Oliver alters the production term when the \bar{S} becomes negative to maintain positive definite production:

$$\bar{S} = f_{v2} \frac{\hat{v}}{\kappa^2 d^2} \quad \hat{S} = \begin{cases} \Omega + \bar{S} & \text{if } \bar{S} \geq -c_2 \Omega \\ \Omega + \frac{\Omega(c_2^2 \Omega + c_3 \bar{S})}{(c_3 - 2c_2)\Omega - \bar{S}} & \text{if } \bar{S} < -c_2 \Omega \end{cases} \quad (3.154)$$

where $c_2 = 0.7$ and $c_3 = 0.9$. This method was chosen in hopes of increasing convergence.

Vorticity. The Spalart-Allmaras model uses a vorticity term Ω (Eq. 3.146), which represents the magnitude of the vorticity tensor. The vorticity magnitude can be simplified:

$$\Omega^2 = 2\Omega_{ij}\Omega_{ij} = \frac{1}{2} \left(\frac{\partial \tilde{u}_i}{\partial x_j} - \frac{\partial \tilde{u}_j}{\partial x_i} \right) \left(\frac{\partial \tilde{u}_i}{\partial x_j} - \frac{\partial \tilde{u}_j}{\partial x_i} \right) = \frac{\partial \tilde{u}_i}{\partial x_j} \frac{\partial \tilde{u}_i}{\partial x_j} - \frac{\partial \tilde{u}_i}{\partial x_j} \frac{\partial \tilde{u}_j}{\partial x_i} \quad (3.155)$$

In two-dimensions, Ω simplifies the magnitude of Ω_{21} :

$$\Omega^2 = \frac{\partial \tilde{u}}{\partial y} \frac{\partial \tilde{u}}{\partial y} - \frac{\partial \tilde{u}}{\partial y} \frac{\partial \tilde{v}}{\partial x} + \frac{\partial \tilde{v}}{\partial x} \frac{\partial \tilde{v}}{\partial x} - \frac{\partial \tilde{v}}{\partial x} \frac{\partial \tilde{u}}{\partial y} = \left| \frac{\partial \tilde{u}}{\partial y} - \frac{\partial \tilde{v}}{\partial x} \right|^2 = |\Omega_{21}|^2 \quad (3.156)$$

In three-dimensions, Ω simplifies the magnitude of $\vec{\omega}$:

$$\Omega^2 = \left(\frac{\partial \tilde{u}}{\partial y} - \frac{\partial \tilde{v}}{\partial x} \right)^2 + \left(\frac{\partial \tilde{u}}{\partial z} - \frac{\partial \tilde{w}}{\partial x} \right)^2 + \left(\frac{\partial \tilde{v}}{\partial z} - \frac{\partial \tilde{w}}{\partial y} \right)^2 = 4(\Omega_{21}^2 + \Omega_{13}^2 + \Omega_{32}^2) \quad (3.157)$$

3.4.3.2 Compressible Spalart-Allmarus

Deck (2002) suggests a compressible Spalart-Allmaras model as:

$$\frac{\partial \bar{\rho} \hat{v}}{\partial t} + \frac{\partial \bar{\rho} \tilde{u}_j \hat{v}}{\partial x_j} = c_{b1} \hat{S} \bar{\rho} \hat{v} - c_{w1} f_w \bar{\rho} \left(\frac{\hat{v}}{d} \right)^2 + \frac{\partial}{\partial x_j} \left(\frac{\bar{\mu} + \bar{\rho} \hat{v}}{\sigma} \frac{\partial \hat{v}}{\partial x_j} \right) + \frac{c_{b2}}{\sigma} \frac{\partial \bar{\rho} \hat{v}}{\partial x_j} \frac{\partial \hat{v}}{\partial x_j} \quad (3.158)$$

Deck writes the SA equation in a conservative form, adding density to the source and diffusion terms. (Deck suggests a symmetric form of the final diffusion term that requires derivatives of the root of density. This alternative is more computationally intensive and does not increase the accuracy of the solution because the final term is mainly present to increase system stability. Catris (2000) presents another alternative.)

Deck does not utilize the “trip transition” model but suggests seeding the domain with a non-zero initial condition, which is large enough to generate transition (also in Lorin, et al, 2007).

3.4.4 **Menter’s SST Model (Inertial)**

The k - ε and k - ω models are used throughout the literature, industry, and academia. The model uses two differential equations (k and ε , or k and ω) to represent the transport of turbulence through the domain. Both models are useful in a wide variety of flows.

The k - ε model is the most accurate for wall-bounded external flows, where the flow is attached. The k - ε model works well for small and moderate pressure gradients in wall-bound and free-shear flows but performs poorly in strong and any adverse pressure gradients. So

the $k-\varepsilon$ model suffers at internal flows and high curvature. The $k-\varepsilon$ model must often be supplemented with damping functions to encourage convergence near walls.

The $k-\omega$ model is accurate for wall-bounded, internal and external flows. The model can be useful in separated flows, but the magnitude of separation is often over-predicted. The $k-\omega$ model is sensitive to freestream values of ω for transition and shear layer spreading rates.

The $k-\omega$ model does not require damping functions, and the governing equations are less stiff in the near-wall region. $k-\omega$ models often suffer from low dissipation in freestream or low strain regions.

Menter (1992a, 1992b, and 1994) developed a two-equation model that blends a $k-\omega$ model for near-wall accuracy and $k-\varepsilon$ model for freestream independence. Menter converted the $k-\varepsilon$ model into a $k-\omega$ format (Eq. 3.92 and 3.93) and integrated the converted model with a true $k-\omega$ model using a transition function F_l . The $k-\varepsilon$ conversion creates a cross-diffusion term⁴ not seen in Wilcox's model. The Shear-Stress Transport (SST) $k-\omega$ model works well for attached and free shear flows without reference to geometry. Eddy viscosity is calculated:

$$\mu_T = C_\alpha \frac{\rho K}{\omega} \quad (3.159)$$

Transport equations are used in both K and ω :

$$\frac{\partial K}{\partial t} + \bar{u}_i \frac{\partial K}{\partial x_i} = \Pi_k - \varepsilon + \frac{1}{\rho} \frac{\partial}{\partial x_j} \left((\bar{\mu} + \sigma_k \mu_T) \frac{\partial K}{\partial x_j} \right) \quad (3.160)$$

⁴ Kok (1999) also utilizes a cross-diffusion term with different coefficient to eliminate freestream dependence.

$$\frac{\partial \omega}{\partial t} + \bar{u}_i \frac{\partial \omega}{\partial x_i} = \Pi_\omega - \beta \omega^2 + \frac{1}{\rho} \frac{\partial}{\partial x_j} \left((\bar{\mu} + \sigma_\omega \mu_T) \frac{\partial \omega}{\partial x_j} \right) + \frac{C_\omega}{\omega} \frac{\partial K}{\partial x_j} \frac{\partial \omega}{\partial x_j} \quad (3.161)$$

The model has 20 closure functions. Eqs. 3.162 and 3.163 are used to calculate the eddy viscosity. Eqs. 3.164 and 3.165 are terms appearing directly within the K - and ω -equations.

The remaining functions are used to interpolate across the boundary layer.

$$C_\alpha = \text{MIN} \left(1, \frac{a_1 \omega}{F_2 S} \right) \quad S = \sqrt{2 S_{ij} S_{ij}} \quad S_{ij} = \frac{1}{2} \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) \quad (3.162)$$

$$F_2 = \tanh(\Phi_2^2) \quad \Phi_2 = \text{MAX} \left(2 \frac{\sqrt{K}}{\beta_\infty^* \omega d}, \frac{500 \bar{\mu}}{\rho d^2 \omega} \right) \quad (3.163)$$

$$\Pi_k = \text{MIN} \left(T_{ij} \frac{\partial \bar{u}_i}{\partial x_j}, 10 \beta_\infty^* K \omega \right) \quad \Pi_\omega = \alpha \frac{\rho}{\mu_T} \Pi_k \quad (3.164)$$

$$\varepsilon = \beta_\infty^* \omega K \quad C_\omega = 2(1 - F_1) \sigma_{\omega 2} \quad (3.165)$$

$$\sigma_k = \sigma_{k1} F_1 + \sigma_{k2} (1 - F_1) \quad \sigma_\omega = \sigma_{\omega 1} F_1 + \sigma_{\omega 2} (1 - F_1) \quad (3.166)$$

$$\alpha = \alpha_1 F_1 + \alpha_2 (1 - F_1) \quad \beta = \beta_1 F_1 + \beta_2 (1 - F_1) \quad (3.167)$$

$$F_1 = \tanh(\Phi_1^4) \quad \Phi_1 = \text{MIN} \left(\text{MAX} \left(\frac{\sqrt{K}}{\beta_\infty^* \omega d}, \frac{500 \bar{\mu}}{\rho d^2 \omega} \right), \frac{4 \rho K \sigma_{\omega 2}}{D_\omega^+ d^2} \right) \quad (3.168)$$

$$D_\omega^+ = \text{MAX} \left(\frac{2 \rho \sigma_{\omega 2}}{\omega} \frac{\partial K}{\partial x_i} \frac{\partial \omega}{\partial x_i}, 10^{-10} \right) \quad (3.169)$$

The 13 closure coefficients are derived from Wilcox (1992) and Jones and Launder (1972) converted to k - ω form. The interpolated coefficients α , σ_k , σ_ω and β have a subscript of 1 from the k - ω model and subscript of 2 from k - ε . For example, the coefficient $\sigma_{\omega 2}$ appears in the cross-diffusion term (Eq. 3.165) because cross-diffusion is derived from the k - ε model:

$$\alpha_1 = \frac{5}{9} \quad \sigma_{k1} = 0.85 \quad \sigma_{\omega1} = 0.5 \quad \beta_1 = 0.075 \quad (3.170)$$

$$\alpha_2 = 0.44 \quad \sigma_{k2} = 1.0 \quad \sigma_{\omega2} = 0.856 \quad \beta_2 = 0.0828 \quad (3.171)$$

$$a_1 = 0.31 \quad \kappa = 0.41 \quad \beta_{\infty}^* = 0.09 \quad \chi^* = 2.0 \quad M_{T0} = 0.25 \quad (3.172)$$

Menter's model uses Wilcox's k - ω model but adapts the production term Π_{ω} . F_1 is used to switch between the k - ω and k - ε models. ω/K in Wilcox's production term Π_{ω} has been replaced by ρ/μ_T to utilize C_{α} . F_1 switches from k - ω model in the sublayer and log-layer to the k - ε model in the outer layer and external flow. The variable Φ_1 is calculated using three terms: The first term (containing the root of K) models the log-layer; the second term (containing μ) models the sublayer, where $F_1 = 0$; and, the third term safeguards against freestream dependence by setting $F_1 = 1$ in the external flow ($y > \delta$).

3.4.4.1 Compressible SST Model

The k - ω models presented above are converted to compressible formats by adding density to all turbulent kinetic energy K and ω terms (Wilcox, 2002; Menter, 1994). Eddy viscosity is:

$$\mu_T = C_{\alpha} \bar{\rho} \frac{\bar{\rho} \tilde{K}}{\bar{\rho} \tilde{\omega}} \quad (3.173)$$

The transport equations become:

$$\frac{\partial \bar{\rho} \tilde{K}}{\partial t} + \frac{\partial \bar{\rho} \tilde{K} \tilde{u}_i}{\partial x_i} = \bar{\rho} \tilde{\Pi}_k - \bar{\rho} \tilde{\varepsilon} + \frac{\partial}{\partial x_j} \left((\bar{\mu} + \sigma_k \mu_T) \frac{\partial \tilde{K}}{\partial x_j} \right) \quad (3.174)$$

$$\frac{\partial \bar{\rho} \tilde{\omega}}{\partial t} + \frac{\partial \bar{\rho} \tilde{\omega} \tilde{u}_i}{\partial x_i} = \bar{\rho} \tilde{\Pi}_{\omega} - \beta_c \frac{(\bar{\rho} \tilde{\omega})^2}{\bar{\rho}} + \frac{\partial}{\partial x_j} \left((\bar{\mu} + \sigma_{\omega} \mu_T) \frac{\partial \tilde{\omega}}{\partial x_j} \right) + \frac{C_{\omega}}{\bar{\rho} \tilde{\omega}} \frac{\partial \bar{\rho} \tilde{K}}{\partial x_j} \frac{\partial \bar{\rho} \tilde{\omega}}{\partial x_j} \quad (3.175)$$

The closure functions are also modified to include density:

$$C_\alpha = \text{MIN}\left(\alpha, \frac{a_1 \tilde{\omega}}{F_2 \tilde{S}}\right) \quad \tilde{S} = \sqrt{2\tilde{S}_{ij}\tilde{S}_{ij}} \quad \tilde{S}_{ij} = \frac{1}{2}\left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i}\right) \quad (3.176)$$

$$F_2 = \tanh(\Phi_2^2) \quad \Phi_2 = \text{MAX}\left(2\frac{\sqrt{\tilde{K}}}{\beta^* \tilde{\omega} d}, \frac{500\bar{\mu}}{\bar{\rho} d^2 \tilde{\omega}}\right) \quad (3.177)$$

$$\bar{\rho} \tilde{\Pi}_k = \text{MIN}\left(\bar{\rho} \tilde{\Gamma}_{ij} \frac{\partial \tilde{u}_i}{\partial x_j}, 10\beta^* \bar{\rho} \tilde{K} \tilde{\omega}\right) \quad \bar{\rho} \tilde{\Pi}_\omega = \alpha \frac{\bar{\rho}}{\mu_T} \bar{\rho} \tilde{\Pi}_k \quad (3.178)$$

$$\bar{\rho} \tilde{\varepsilon} = \beta^* \bar{\rho} \tilde{\omega} \tilde{K} \quad C_\omega = 2(1 - F_1) \sigma_{\omega 2} \quad (3.179)$$

$$\sigma_k = \sigma_{k1} F_1 + \sigma_{k2} (1 - F_1) \quad \sigma_\omega = \sigma_{\omega 1} F_1 + \sigma_{\omega 2} (1 - F_1) \quad (3.180)$$

$$\alpha = \alpha_1 F_1 + \alpha_2 (1 - F_1) \quad \beta = \beta_1 F_1 + \beta_2 (1 - F_1) \quad (3.181)$$

$$F_1 = \tanh(\Phi_1^4) \quad \Phi_1 = \text{MIN}\left(\text{MAX}\left(\frac{\sqrt{\tilde{K}}}{\beta^* \tilde{\omega} d}, \frac{500\bar{\mu}}{\bar{\rho} d^2 \tilde{\omega}}\right), \frac{4\bar{\rho} \tilde{K} \sigma_{\omega 2}}{D_\omega^+ d^2}\right) \quad (3.182)$$

$$D_\omega^+ = \text{MAX}\left(\frac{2\sigma_{\omega 2}}{\bar{\rho} \tilde{\omega}} \frac{\partial \bar{\rho} \tilde{K}}{\partial x_i} \frac{\partial \bar{\rho} \tilde{\omega}}{\partial x_i}, 10^{-10}\right) \quad (3.183)$$

$$\beta_c = \beta - \chi^* \beta_\infty^* F(M_T) \quad \beta^* = \beta_\infty^* (1 + \chi^* F(M_T)) \quad (3.184)$$

$$M_T^2 = \frac{2\bar{\rho} \tilde{K}}{\bar{\rho} \tilde{\alpha}^2} \quad F(M_T) = \begin{cases} 0 & \text{if } M_T \leq M_{T0} \\ M_T^2 - M_{T0}^2 & \text{if } M_T > M_{T0} \end{cases} \quad (3.185)$$

Adaptations. Spalart and Rumsey (2007) adapted the destruction terms to maintain ambient turbulence levels. The destruction term in the K -equation is shifted to take on a zero value when freestream (ambient) values are present:

$$\bar{\rho} \tilde{\varepsilon} = \beta^* \rho (\omega K - \omega_{amb} K_{amb}) \quad (3.186)$$

The destruction term in the ω -equation is shifted in a similar fashion:

$$\frac{D\bar{\rho}\tilde{\omega}}{Dt} = \bar{\rho}\tilde{\Pi}_\omega - \beta \frac{(\bar{\rho}\tilde{\omega})^2 - (\bar{\rho}\tilde{\omega}_{amb})^2}{\bar{\rho}} + \frac{\partial}{\partial x_j} \left((\bar{\mu} + \sigma_\omega \mu_T) \frac{\partial \tilde{\omega}}{\partial x_j} \right) + \frac{C_\omega}{\bar{\rho}\tilde{\omega}} \frac{\partial \bar{\rho}K}{\partial x_j} \frac{\partial \bar{\rho}\tilde{\omega}}{\partial x_j} \quad (3.187)$$

3.4.5 Non-Inertial RC Correction to Turbulence Models

Experimental results have shown that turbulent production and transport changes in the presence of rotation. Therefore, the turbulence models (and possibly RANS equations) need to be adapted to generate such changes. The Navier-Stokes equations were time- and mass-averaged in the non-inertial frame in previous sections. The momentum, energy, K -, ε -, and ω -equations are expected to contain Reynolds-terms, representing the interaction of turbulence and rotational velocity (or even translation). Eqs. 3.100 and 3.104 illustrate the presence of such terms in the compressible RANS equations, validating the expectation; but, the incompressible RANS (Eqs. 3.67 through 3.71) and compressible FANS (Eqs. 3.116 through 3.119), both of which show the lack of terms after time- or mass-averaging. The latter two equation sets are used in the literature to model rotating fluids, but these equations are a gross over-simplification of the effects of turbulence on the mean flow and production of turbulence in a rotating flow field. Additional terms will need to be added to the turbulence models to supplement the physics and adapt production. The new terms should be Galilean invariant and physical, but effective *ad hoc* methods will also be investigated.

Gorski (1992) uses a channel flow rotated about an off-centered axis to show that non-inertial rotation creates an asymmetric velocity profile in the channel. Without correction to the turbulence model, the CFD solution predicts a symmetric velocity profile. Gorski compares eight methods for correcting the Standard k - ε model to include non-inertial effects. Among those corrections are additional terms in both the k - and ε -equations; adjustments to the

coefficients $C_{\varepsilon 1}$, $C_{\varepsilon 2}$, and C_{μ} ; and alterations to the Reynolds stress tensor to model anisotropy. All of the methods tested by Gorski add some asymmetry to the profile but not enough to match experimental data. Speziale (1989) suggested adding strains, strain rates, and vorticity to the Reynolds stress to simulate the anisotropy created by rotation. Schiestel and Elena (1997) adapted a Reynolds stress model to include the anisotropy created by rotation on turbulence production of individual Reynolds stresses.

Launder and Sharma (1974) have good success with the Standard k - ε model posed in cylindrical form (Eqs. 3.188 and 3.189). Launder and Sharma state: “[The] Extra source terms involving gradients of (V_{θ}/r) appear in the equations for k and ε . Their appearance is due to the conversion of the Cartesian-tensor form of these equations to the present coordinate frame. They are not ad hoc terms.” These terms are not readily expandable to generic Cartesian problems in the frame.

$$\rho \bar{V}_r \frac{\partial K}{\partial r} + \rho \bar{V}_z \frac{\partial K}{\partial z} = \mu_T \left(\left(\frac{\partial \bar{V}_r}{\partial z} \right)^2 + \left(r \frac{\partial (\bar{V}_{\theta}/r)}{\partial z} \right)^2 \right) - \rho \varepsilon + \frac{1}{r} \frac{\partial}{\partial z} \left(r \left(\mu + \frac{\mu_T}{\sigma_K} \right) \frac{\partial K}{\partial z} \right) \quad (3.188)$$

$$\begin{aligned} \rho \bar{V}_r \frac{\partial \varepsilon}{\partial r} + \rho \bar{V}_z \frac{\partial \varepsilon}{\partial z} = C_{\varepsilon 1} \frac{\varepsilon}{K} \mu_T \left(\left(\frac{\partial \bar{V}_r}{\partial z} \right)^2 + \left(r \frac{\partial (\bar{V}_{\theta}/r)}{\partial z} \right)^2 \right) \\ - C_{\varepsilon 2} \frac{\rho \varepsilon^2}{K} + \frac{1}{r} \frac{\partial}{\partial z} \left(r \left(\mu + \frac{\mu_T}{\sigma_{\varepsilon}} \right) \frac{\partial \varepsilon}{\partial z} \right) \end{aligned} \quad (3.189)$$

Rotation of frame and wall curvature affect turbulence production in similar ways, and isotropic eddy viscosity models generally suffer in both areas. Spalart and Shur (1997) combine the effects of rotation and curvature on the SA model into a single correction factor applied to the production of turbulence. Shur, et al. (2000) expands on the original work and plainly presents the final equations. The presence of curvature is measured by the substantial

derivative of the strain tensor (Eq. 3.192). The advection of strain is difficult to calculate on a piece-wise linear domain because the strain is piece-wise constant on a linear domain. The rough estimate can be created using a hybrid element to capture the gradient of the strain tensor. Without the advection of strain, the unsteady term is only active in time-accurate simulations. For simplicity, the entire curvature term is not used in this work (Eq. 3.192). The rotation term, on the other hand, is based in the cross-product of stain and rotation, which is expensive but directly accessible on the piece-wise linear domain.

The RC correction term f_{r1} scales up the SA production term in the presence of rotation:

$$\frac{\partial \bar{\rho} \hat{v}}{\partial t} + \frac{\partial \bar{\rho} \tilde{u}_j \hat{v}}{\partial x_j} = f_{r1} c_{b1} \hat{S} \bar{\rho} \hat{v} - c_{w1} f_w \bar{\rho} \left(\frac{\hat{v}}{d} \right)^2 + \frac{\partial}{\partial x_j} \left(\frac{\bar{\mu} + \bar{\rho} \hat{v}}{\sigma} \frac{\partial \hat{v}}{\partial x_j} \right) + \frac{c_{b2}}{\sigma} \frac{\partial \bar{\rho} \hat{v}}{\partial x_j} \frac{\partial \hat{v}}{\partial x_j} \quad (3.190)$$

where

$$f_{r1} = (1 + c_{r1}) \frac{2r^*}{r^* + 1} (1 - c_{r3} \tan^{-1}(c_{r2} \tilde{r})) - c_{r1} \quad (3.191)$$

$$r^* = \frac{S}{\omega} \quad \tilde{r} = \frac{2\Omega_{ik} S_{jk}}{D^4} \left(\frac{DS_{ij}^0}{Dt} + (\varepsilon_{imn} S_{jn} + \varepsilon_{jmn} S_{in}) \omega_m \right) \quad (3.192)$$

$$\Omega_{ij} = -\Omega_{ji} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i} \right) + \varepsilon_{mji} \omega_m \quad \Omega^2 = 2\Omega_{ij} \Omega_{ij} \quad (3.193)$$

$$S_{ij} = S_{ji} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad S^2 = 2S_{ij} S_{ij} \quad D^4 = \frac{1}{4} (S^2 + \Omega^2)^2 \quad (3.194)$$

Notice that the vorticity tensor Ω_{ij} now includes the rotational rate of the frame. Eq. 192 is also used to adapt the other terms in the SA model in the presence of rotation of the frame (Eqs. 3.154 through 3.157). Shur, et al. (2000) shows good results with the RC correction for a constant rotational rate. Mani, et al. (2004) and Rumsey and Nishino (2011) demonstrated

the improved accuracy of the model for very high curvature and complex flow fields. Eqs. 3.193 and 3.194 show the symmetry of S_{ij} or anti-symmetry of Ω_{ij} . The symmetry is used here to simplify the RC correction equations.

For 2D, the correction term uses Eq. 3.156 to calculate Ω . The remaining terms simplify to:

$$S^2 = 2(S_{xx}^2 + S_{yy}^2) + 4S_{xy}^2 \quad (3.197)$$

$$\tilde{r} = \frac{2\Omega_{21}\omega_z}{D^4} \left((S_{xx} - S_{yy})^2 + 4S_{xy}^2 \right) \quad (3.198)$$

For 3D, the correction term uses Eq. 3.157 to calculate Ω . The remaining terms simplify to:

$$S^2 = 2(S_{xx}^2 + S_{yy}^2 + S_{zz}^2) + 4(S_{xy}^2 + S_{xz}^2 + S_{yz}^2) \quad (3.195)$$

$$\begin{aligned} \frac{D^4 \tilde{r}}{2} = & \left(\begin{aligned} & \Omega_{13} (S_{xy} (2S_{zz} - S_{xx} - S_{yy}) - 3S_{xz} S_{yz}) \\ & + \Omega_{21} (S_{xz} (2S_{xx} - S_{yy} - S_{zz}) - 3S_{xy} S_{yz}) \\ & + \Omega_{32} (4S_{yz}^2 + S_{xy}^2 + S_{xz}^2 + (S_{yy} - S_{zz})^2) \end{aligned} \right) \omega_x \\ & + \left(\begin{aligned} & \Omega_{13} (4S_{xz}^2 + S_{xy}^2 + S_{yz}^2 + (S_{xx} - S_{zz})^2) \\ & + \Omega_{21} (S_{yz} (2S_{xx} - S_{yy} - S_{zz}) - 3S_{xy} S_{xz}) \\ & + \Omega_{32} (S_{xy} (2S_{zz} - S_{xx} - S_{yy}) - 3S_{xz} S_{yz}) \end{aligned} \right) \omega_y \\ & + \left(\begin{aligned} & \Omega_{13} (S_{yz} (2S_{xx} - S_{yy} - S_{zz}) - 3S_{xy} S_{xz}) \\ & + \Omega_{21} (4S_{xy}^2 + S_{xz}^2 + S_{yz}^2 + (S_{xx} - S_{yy})^2) \\ & + \Omega_{32} (S_{xz} (2S_{yy} - S_{xx} - S_{zz}) - 3S_{xy} S_{yz}) \end{aligned} \right) \omega_z \end{aligned} \quad (3.196)$$

Hellsten (1997) developed a simple rotation-curvature correction to the SST model:

$$\frac{\partial \bar{\rho} \tilde{\omega}}{\partial t} + \frac{\partial \bar{\rho} \tilde{\omega} \tilde{u}_i}{\partial x_i} = \bar{\rho} \tilde{\Pi}_\omega - F_4 \beta_c \frac{(\bar{\rho} \tilde{\omega})^2}{\bar{\rho}} + \frac{\partial}{\partial x_j} \left((\bar{\mu} + \sigma_\omega \mu_T) \frac{\partial \tilde{\omega}}{\partial x_j} \right) + \frac{C_\omega}{\bar{\rho} \tilde{\omega}} \frac{\partial \bar{\rho} \tilde{K}}{\partial x_j} \frac{\partial \bar{\rho} \tilde{\omega}}{\partial x_j} \quad (3.199)$$

$$F_4 = \frac{1}{1 + C_{rc} Ri} \quad Ri = \frac{\Omega}{S} \left(\frac{\Omega}{S} - 1 \right) \quad (3.200)$$

The coefficient C_{rc} scales the effects of the Richardson number Ri . The magnitude of this coefficient is debated in the literature and seems to need tuning for the particular problem (Mani, et al., 2004; Rumsey and Nishino, 2011). The need for tuning is undesirable.

Swanson and Rumsey (2009) compared the SA model with the Spalart-Shur correction (SA-RC) with the SST model without any correction. Their results show insensitivity of the SST model to curvature, but the production in the model is unaffected by rotation of frame. Mani, et al. (2004) and Rumsey and Nishino (2011) compared SST using Hellsten's correction (with tuning) with SA-RC and show comparable results with each other and experiment. Smirnov and Menter (2009) tested the Spalart-Shur correction on the production of ω (Eq. 3.201). Smirnov and Menter also limit $f_{r,l}$ to reasonable limits shown in Eq. 3.202:

$$\frac{\partial \bar{\rho} \tilde{\omega}}{\partial t} + \frac{\partial \bar{\rho} \tilde{\omega} \tilde{u}_i}{\partial x_i} = f_{r,l} \bar{\rho} \tilde{\Pi}_\omega - \beta_c \frac{(\bar{\rho} \tilde{\omega})^2}{\bar{\rho}} + \frac{\partial}{\partial x_j} \left((\bar{\mu} + \sigma_\omega \mu_T) \frac{\partial \tilde{\omega}}{\partial x_j} \right) + \frac{C_\omega}{\bar{\rho} \tilde{\omega}} \frac{\partial \bar{\rho} \tilde{K}}{\partial x_j} \frac{\partial \bar{\rho} \tilde{\omega}}{\partial x_j} \quad (3.201)$$

$$0.0 \leq f_{r,l} = (1 + c_{r1}) \frac{2r^*}{r^* + 1} (1 - c_{r3} \tan^{-1}(c_{r2} \tilde{r})) - c_{r1} \leq 1.25 \quad (3.202)$$

Eq. 3.201 was implemented in this research (termed SST-RC). The limitations on $f_{r,l}$ (Eq. 3.202) were used on both SA-RC and SST-RC.

3.5 Non-Dimensional Equations

The previous equations were all dimensional, but the solvers investigated in this work are all dimensionless. This section will remove any dimensionality from the governing and support equations before the equations are discretized using the Galerkin methods. The most important aspect of this process is the introduction of the freestream Reynolds number into the viscous and turbulent components. The Reynolds- and Favre-averaging is denoted by an

over-bar and over-tilda. These marks have been removed for the duration of this document, but the variables remain time- or mass-averaged.

The dimensionality is removed from the Favre-averaged Navier-Stokes equations (Eq. 3.115 and 3.124 – 3.126) by scaling the system of equations by the matrix:

$$[\Delta] = \begin{bmatrix} \frac{L}{\rho_\infty U_\infty} & & \\ & \frac{L}{\rho_\infty U_\infty^2} & \\ & & \frac{L}{\rho_\infty U_\infty^3} \end{bmatrix} \quad (3.203)$$

The system equation is non-dimensionalized:

$$[\Delta] \left(\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x_{b,i}} - \frac{\partial \mathbf{F}_{v,i}}{\partial x_{b,i}} - \mathbf{S} \right) = \frac{\partial \mathbf{U}^*}{\partial t^*} + \frac{\partial \mathbf{F}_i^*}{\partial x_{b,i}^*} - \frac{\partial \mathbf{F}_{v,i}^*}{\partial x_{b,i}^*} - \mathbf{S}^* = 0 \quad (3.204)$$

where

$$t^* = U_\infty t / L \quad x_{b,i}^* = x_{b,i} / L \quad (3.205)$$

The dimensionality is removed from the unknowns vector, flux vectors, and source term:

$$\mathbf{U}^* = \frac{U_\infty}{L} [\Delta] \mathbf{U} = \begin{Bmatrix} \rho / \rho_\infty \\ \rho u_{r,j} / \rho_\infty U_\infty \\ \rho E_r / \rho_\infty U_\infty^2 \end{Bmatrix} = \begin{Bmatrix} \rho^* \\ \rho^* u_{r,j}^* \\ \rho^* E_r^* \end{Bmatrix} \quad (3.206)$$

$$\mathbf{F}_i^* = \frac{1}{L} [\Delta] \mathbf{F}_i = \begin{Bmatrix} \rho u_{r,i} / \rho_\infty U_\infty \\ \rho u_{r,i} u_{r,j} / \rho_\infty U_\infty^2 + (p / \rho_\infty U_\infty^2) \delta_{ij} \\ u_{r,i} \rho H_r / \rho_\infty U_\infty^3 \end{Bmatrix} = \begin{Bmatrix} \rho^* u_{r,i}^* \\ \rho^* u_{r,i}^* u_{r,j}^* + p^* \delta_{ij} \\ u_{r,i}^* \rho^* H_r^* \end{Bmatrix} \quad (3.207)$$

$$\mathbf{F}_{v,i}^* = \frac{1}{L} [\Delta] \mathbf{F}_{v,i} = \left\{ \begin{array}{c} 0 \\ \frac{\tau_{ij}}{\rho_\infty U_\infty^2} + \frac{\rho \mathbf{T}_{ij}}{\rho_\infty U_\infty^2} \\ \frac{u_{r,j}}{U_\infty} \left(\frac{\tau_{ij}}{\rho_\infty U_\infty^2} + \frac{\rho \mathbf{T}_{ij}}{\rho_\infty U_\infty^2} \right) - \frac{q_i'' + Q_i}{\rho_\infty U_\infty^3} + \frac{\mu}{\rho_\infty U_\infty} + \sigma_k \frac{\mu_T}{\mu_\infty} \frac{\partial}{\partial x_{b,i}^*} \left(\frac{K}{U_\infty^2} \right) \end{array} \right\} \quad (3.208)$$

$$\mathbf{F}_{v,i}^* = \left\{ \begin{array}{c} 0 \\ \tau_{ij}^* + \rho \mathbf{T}_{ij}^* \\ u_{r,j}^* (\tau_{ij}^* + \rho \mathbf{T}_{ij}^*) - (q_i'')^* - Q_i^* + \frac{\mu^* + \sigma_k \mu_T^*}{\text{Re}_L} \frac{\partial K^*}{\partial x_{b,i}^*} \end{array} \right\} \quad (3.209)$$

$$\mathbf{S}^* = [\Delta] \mathbf{S} = -\frac{\rho}{\rho_\infty} \left\{ \begin{array}{c} 0 \\ \frac{a_{t,j} L}{U_\infty^2} + \varepsilon_{jkl} \frac{\omega_k L}{U_\infty} \frac{u_{r,l}}{U_\infty} \\ \frac{a_{t,k} L}{U_\infty^2} \cdot \left(\frac{V_{t,k}}{U_\infty} + \frac{u_{r,k}}{U_\infty} \right) \end{array} \right\} = -\rho^* \left\{ \begin{array}{c} 0 \\ a_{t,j}^* + \varepsilon_{jkl} \omega_k^* u_{r,l}^* \\ a_{t,k}^* \cdot (V_{t,k}^* + u_{r,k}^*) \end{array} \right\} \quad (3.210)$$

The properties are non-dimensionalized:

$$\rho^* = \rho / \rho_\infty \quad u_{r,j}^* = u_{r,j} / U_\infty \quad p^* = p / \rho_\infty U_\infty^2 \quad (3.211)$$

$$h^* = h / U_\infty^2 \quad E_r^* = E_r / U_\infty^2 \quad H_r^* = H_r / U_\infty^2 \quad (3.212)$$

The viscous and turbulent terms are non-dimensionalized:

$$\tau_{ij}^* = \tau_{ij} / \rho_\infty U_\infty^2 \quad \rho \mathbf{T}_{ij}^* = \rho \mathbf{T}_{ij} / \rho_\infty U_\infty^2 \quad (3.213)$$

$$(q_i'')^* = q_i'' / \rho_\infty U_\infty^3 \quad Q_i^* = Q_i / \rho_\infty U_\infty^3 \quad K^* = K / U_\infty^2 \quad (3.214)$$

$$\mu^* = \mu / \mu_\infty \quad \lambda^* = \lambda / \mu \quad \mu_T^* = \mu_T / \mu_\infty \quad \text{Re}_L = \rho_\infty U_\infty L / \mu_\infty \quad (3.215)$$

The non-inertial terms are non-dimensionalized:

$$V_{t,j}^* = \frac{V_{t,j}}{U_\infty} \quad a_{t,j}^* = \frac{a_{t,j}L}{U_\infty^2} \quad \omega_j^* = \frac{\omega_j L}{U_\infty} \quad (3.216)$$

Dimensions are removed from the non-inertial mesh coordinate, velocity, and acceleration vectors (Eqs. 3.4, 3.11, 3.117, and 3.118):

$$x_i^* = x_i / L = x_{o,i}^* + B_{ij} x_{b,j}^* \quad (3.217)$$

$$u_i^* = u_i / U_\infty = B_{ij} (u_{r,j}^* + V_{t,j}^*) \quad V_{t,i}^* = V_{o,i}^* + \varepsilon_{ijk} \omega_j^* x_{b,k}^* \quad (3.218)$$

$$a_{t,i}^* = a_{o,i}^* + \varepsilon_{ijk} \omega_j^* \varepsilon_{klm} \omega_l^* x_{b,m}^* + \varepsilon_{ijk} \dot{\omega}_j^* x_{b,k}^* + \varepsilon_{ijk} \omega_j^* u_{r,k}^* \quad (3.219)$$

The viscous and turbulent terms (Eqs. 3.109, 3.111, and 3.122) are non-dimensionalized:

$$\tau_{ij}^* = \frac{\mu_\infty \mu^*}{\rho_\infty U_\infty L} \left(\frac{\partial u_i^*}{\partial x_j^*} + \frac{\partial u_j^*}{\partial x_i^*} + \lambda^* \frac{\partial u_k^*}{\partial x_k^*} \delta_{ij} \right) = \frac{\mu^*}{\text{Re}_L} \left(\frac{\partial u_i^*}{\partial x_j^*} + \frac{\partial u_j^*}{\partial x_i^*} + \lambda^* \frac{\partial u_k^*}{\partial x_k^*} \delta_{ij} \right) \quad (3.220)$$

$$\rho \Gamma_{ij}^* \approx \frac{\mu_T^*}{\text{Re}_L} \left(\frac{\partial u_i^*}{\partial x_j^*} + \frac{\partial u_j^*}{\partial x_i^*} - \frac{2}{3} \frac{\partial u_k^*}{\partial x_k^*} \delta_{ij} \right) - \frac{2}{3} \rho^* K^* \delta_{ij} \quad (3.221)$$

$$(q_i^*)^* = -\frac{\mu_\infty \mu^*}{\rho_\infty U_\infty L \text{Pr}} \frac{\partial (h/U_\infty^2)}{\partial x_i^*} = -\frac{\mu^*}{\text{Re}_L \text{Pr}} \frac{\partial h^*}{\partial x_i^*} \quad Q_i^* \approx -\frac{\mu_T^*}{\text{Re}_L \text{Pr}_T} \frac{\partial h^*}{\partial x_i^*} \quad (3.222)$$

The thermodynamic relationships (Eqs. 3.35, 3.129, and 3.132) are non-dimensionalized:

$$\mu^* = \left(\frac{h^*}{h_\infty^*} \right)^{3/2} \frac{h_\infty^* + \frac{c_p S}{U_\infty^2}}{h^* + \frac{c_p S}{U_\infty^2}} = \left(\frac{h^*}{h_\infty^*} \right)^{3/2} \frac{h_\infty^* + S_{\text{mod}}}{h^* + S_{\text{mod}}} \quad (3.223)$$

$$h^* = \frac{h}{U_\infty^2} = \frac{e}{U_\infty^2} + \frac{p/\rho_\infty U_\infty^2}{\rho/\rho_\infty} = e^* + \frac{p^*}{\rho^*} \quad h^* = \frac{c_p T}{U_\infty^2} = T^* \quad (3.224)$$

$$(a^*)^2 = \frac{a^2}{U_\infty^2} = \frac{\gamma R}{c_p} \frac{c_p T}{U_\infty^2} = (\gamma - 1) T^* = \frac{\gamma p^*}{\rho^*} \quad (3.225)$$

The remaining energy terms are non-dimensionalized:

$$S_{\text{mod}} = c_p S / U_\infty^2 \quad T^* = c_p T / U_\infty^2 = h^* \quad (3.226)$$

$$e^* = e / U_\infty^2 \quad a^* = \frac{a}{U_\infty} \quad (3.227)$$

The remaining dimensions are encompassed in the turbulence models. For the SA model (Eqs. 3.142, 3.144, 3.145, 3.154, 3.190, and 3.193):

$$\mu_T^* = \frac{\mu_T}{\mu_\infty} = f_{v1} \frac{\rho}{\rho_\infty} \frac{\rho_\infty \hat{\nu}}{\mu_\infty} = f_{v1} \rho^* \hat{\nu}^* \quad (3.228)$$

$$\begin{aligned} \frac{\partial \rho^* \hat{\nu}^*}{\partial t^*} + \frac{\partial \rho^* u_j^* \hat{\nu}^*}{\partial x_j^*} &= f_{r1} c_{b1} \hat{S}^* \rho^* \hat{\nu}^* - \frac{c_{w1} f_w \rho^*}{\text{Re}_L} \left(\frac{\hat{\nu}^*}{d^*} \right)^2 \\ &+ \frac{\partial}{\partial x_j^*} \left(\frac{\mu^* + \rho^* \hat{\nu}^*}{\text{Re}_L \sigma} \frac{\partial \hat{\nu}^*}{\partial x_j^*} \right) + \frac{c_{b2}}{\text{Re}_L \sigma} \frac{\partial \rho^* \hat{\nu}^*}{\partial x_j^*} \frac{\partial \hat{\nu}^*}{\partial x_j^*} \end{aligned} \quad (3.229)$$

$$\chi = \frac{\rho^* \hat{\nu}^*}{\mu^*} \quad r = \frac{\hat{\nu}^*}{\hat{S}^* \kappa^2 (d^*)^2 \text{Re}_L} \quad \bar{S}^* = f_{v2} \frac{\hat{\nu}^*}{\kappa^2 (d^*)^2 \text{Re}_L} \quad (3.230)$$

$$\Omega_{ij}^* = \frac{1}{2} \left(\frac{\partial u_i^*}{\partial x_j^*} - \frac{\partial u_j^*}{\partial x_i^*} \right) + \varepsilon_{mji} \omega_m^* \quad (3.231)$$

where

$$\hat{\nu}^* = \frac{\rho_\infty \hat{\nu}}{\mu_\infty} \quad d^* = \frac{d}{L} \quad (3.232)$$

$$\hat{S}^* = \frac{\hat{S}L}{U_\infty} \quad \bar{S}^* = \frac{\bar{S}L}{U_\infty} \quad \Omega_{ij}^* = \frac{\Omega_{ij}L}{U_\infty} \quad (3.233)$$

For the SST model (Eqs. 3.173 through 3.186, and 3.201):

$$\mu_T^* = \frac{\mu_T}{\mu_\infty} = C_\alpha \frac{\rho K / \rho_\infty U_\infty^2}{\omega L / U_\infty} \frac{\rho_\infty U_\infty L}{\mu_\infty} = C_\alpha \frac{\rho^* K^*}{\omega^*} \text{Re}_L \quad (3.234)$$

$$\frac{\partial \rho^* K^*}{\partial t^*} + \frac{\partial \rho^* K^* u_i^*}{\partial x_i^*} = \rho \Pi_k^* - \rho \varepsilon^* + \frac{\partial}{\partial x_j^*} \left(\frac{\mu^* + \sigma_k \mu_T^*}{\text{Re}_L} \frac{\partial K^*}{\partial x_j^*} \right) \quad (3.235)$$

$$\begin{aligned} \frac{D \rho^* \omega^*}{Dt^*} &= f_{r1} \rho \Pi_\omega^* - \beta_c \frac{(\rho^* \omega^*)^2 - (\rho^* \omega_{amb}^*)^2}{\rho^*} \\ &+ \frac{\partial}{\partial x_j^*} \left(\frac{\mu^* + \sigma_\omega \mu_T^*}{\text{Re}_L} \frac{\partial \omega^*}{\partial x_j^*} \right) + \frac{C_\omega}{\rho^* \omega^*} \frac{\partial \rho^* K^*}{\partial x_j^*} \frac{\partial \rho^* \omega^*}{\partial x_j^*} \end{aligned} \quad (3.236)$$

$$C_\alpha = \text{MIN} \left(\alpha, \frac{a_1 \omega^*}{F_2 S^*} \right) \quad S^* = \sqrt{2 S_{ij}^* S_{ij}^*} \quad S_{ij}^* = \frac{1}{2} \left(\frac{\partial u_i^*}{\partial x_j^*} + \frac{\partial u_j^*}{\partial x_i^*} \right) \quad (3.237)$$

$$\Phi_2 = \text{MAX} \left(2 \frac{\sqrt{K^*}}{\beta^* \omega^* d^*}, \frac{500 \mu^*}{\rho^* (d^*)^2 \omega^* \text{Re}_L} \right) \quad (3.238)$$

$$\rho \Pi_k^* = \text{MIN} \left(\rho \Gamma_{ij}^* \frac{\partial u_i^*}{\partial x_j^*}, 10 \beta^* \rho^* K^* \omega^* \right) \quad (3.239)$$

$$\rho \Pi_\omega^* = \alpha \frac{\rho^*}{\mu_T^*} \rho \Pi_k^* \text{Re}_L \quad (3.240)$$

$$\Phi_1 = \text{MIN} \left(\text{MAX} \left(\frac{\sqrt{K^*}}{\beta^* \omega^* d^*}, \frac{500 \mu^*}{\rho^* (d^*)^2 \omega^* \text{Re}_L} \right), \frac{4 \rho^* K^* \sigma_{\omega 2}}{(D_\omega^+)^* (d^*)^2} \right) \quad (3.241)$$

$$(D_\omega^+)^* = \frac{D_\omega^+ L^2}{\rho_\infty U_\infty^2} = \text{MAX} \left(\frac{2 \sigma_{\omega 2}}{\rho^* \omega^*} \frac{\partial \rho^* K^*}{\partial x_i^*} \frac{\partial \rho^* \omega^*}{\partial x_i^*}, 10^{-10} \right) \quad (3.242)$$

$$M_T^2 = \frac{2K^*}{(a^*)^2} \quad (3.243)$$

$$\rho \varepsilon^* = \beta^* \rho^* (\omega^* K^* - \omega_{amb}^* K_{amb}^*) \quad (3.244)$$

where

$$\omega^* = \omega L / U_\infty \quad \rho \Pi_k^* = \rho \Pi_k L / \rho_\infty U_\infty^3 \quad \rho \varepsilon^* = \rho \varepsilon L / U_\infty^3 \quad (3.245)$$

$$\rho \Pi_{\omega}^* = \rho \Pi_{\omega} L^2 / \rho_{\infty} U_{\infty}^2 \qquad (D_{\omega}^+)^* = D_{\omega}^+ L^2 / \rho_{\infty} U_{\infty}^2 \qquad (3.246)$$

$$S^* = S L / U_{\infty} \qquad S_{ij}^* = S_{ij} L / U_{\infty} \qquad (3.247)$$

All of the terms in the RC correction models are already dimensionless ratios, so the above non-dimensionalizations can be applied in the same fashion. The star superscript representing a dimensionless variable will be dropped for the remainder of the document; all equations should be assumed to be dimensionless unless otherwise stated.

3.6 Boundary Conditions

A numerical simulation can only include part of the physical world. Mathematical concepts, such as symmetry, periodicity, and limits, help model the otherwise larger physical world; however, these constructed conditions are often the most difficult to implement. Natural conditions are always created from real, physical situations that exist in the real world, whereas artificial boundaries are a series of simplifications, desirable constraints, or limited views of a larger problem.

Boundary conditions are often grouped into three divisions: Dirichlet (or essential) conditions specify the properties in terms of known quantities. Dirichlet conditions are often the simplest to implement. In fact, developing methods are almost always tested on purely Dirichlet conditions. Neumann (or natural) conditions control the gradients or fluxes along boundaries. Robin (or mixed) conditions describe a boundary gradient in terms of the properties and other known conditions. Robin conditions are the most complex to implement.

Burnett (1987) breaks boundary conditions down into two groups by their definition: Essential and natural conditions. For a boundary value problem of order $2m$, the essential conditions relate properties and derivatives (of order $m - 1$ or less) through an equation.

Natural boundary conditions relative derivatives of order m through $2m - 1$. Fluids, heat conduction, and acoustics are all second order systems (or $m = 1$). Using Burnett's system, CFD boundaries have the possibility of having one essential and one natural boundary condition for each property along a boundary. All of the boundaries can be specified using essential conditions, but not all boundaries can be natural. It is crucial, or essential, that one condition be classified as essential (thus, their name). Without an essential condition, the feasible space is multivalued and likely floats between the possible solutions.

Well-posed boundary conditions are required to properly represent the domain boundary and mesh with the domain equations (and later discretization method) to create a feasible solution field. Ill-posed boundary conditions are the opposite of well-posed conditions and can be divided into under- and over-specified conditions. Under-specified boundaries do not stipulate enough requirements. Numerical solutions will often float along under-specified boundaries. Over-specified boundaries limit too many conditions, which can keep any of the conditions from being properly satisfied or make the implementation unstable.

3.6.1 Far Field

The far field boundary seeks to represent the freestream conditions, or those conditions far from the body of interest, on a finite domain. The far field boundary is simultaneously an inflow and outflow boundary. The far field boundary must imply conditions that are not necessarily seen within the near field and allow the wake and any traveling waves to pass through the boundary without impedance. The impedance of such a boundary (or its ability to reflect perturbations) is often the most complex aspect of the far field boundary.

The far field boundary is the most mathematically complex and diversely approached condition. N'dri (2000), Kallinderis (1994), and Soulaïmani (1994) strictly specify the far field using prescribed or freestream conditions. Gulcat (1997) imposes a freestream at the inflow and a fully developed wake in the outflow. Zienkiewicz (1991) imposes an upstream condition for supersonic flow and suggests using the local characteristics to determine an appropriate mixture of subsonic properties. A less popular approach is to extend the boundary to the true “far field” by using infinite elements. Apart from the approach, the freestream conditions specify the velocity (magnitude and incidence) and thermodynamic properties (density, pressure, and energy). The turbulent properties along the freestream boundary are much more difficult to specify because they represent time-average mathematical concepts. The most realistic turbulent conditions are related to a physical property.

SA Model. The eddy viscosity must be non-zero at all times so that turbulent production and destruction terms are non-zero and so that the Reynolds stress dissipates energy in the energy equation. Eddy viscosity is suggested to start with an initial magnitude much smaller than molecular viscosity (i.e., $\mu_{T,0} < 0.01 \mu \ll \mu$) so that the total effective viscosity is approximately equal to molecular viscosity. In this way, the freestream and laminar regions have very little eddy viscosity, while turbulence is allowed to grow freely according to the model. To produce a fully turbulent solution, the initial condition is much very larger ($\mu_{T,0} = 5 \mu$).

SST Model. The SST model uses turbulent kinetic energy K and its dissipation rate ω to represent the transport of turbulence through the field. Turbulent kinetic energy can be related to the amount of turbulence in the freestream. The turbulent intensity I_{turb} can be constructed from experimental conditions and used to calculate turbulent kinetic energy directly:

$$I = \sqrt{\frac{2}{3} K_\infty} \quad K_\infty = \frac{3}{2} I^2 \quad (3.248)$$

ω represents the rate of dissipation of K . Menter suggests a freestream value limited by:

$$\omega_\infty < \frac{\sqrt{\beta_\infty^*}}{\beta_1} \approx \frac{\sqrt{0.09}}{0.075} = 4 \quad (3.249)$$

or, the freestream value can be back-calculated (Eq. 3.234) from a non-zero eddy viscosity:

$$\omega_\infty = \alpha \frac{\rho_\infty K_\infty}{\mu_{T,\infty}} \text{Re} \quad \text{where} \quad 10^{-5} < \mu_{T,\infty} < 10^{-2} \quad (3.250)$$

3.6.2 Inviscid Wall

The inviscid wall is often misconstrued as a physical condition, but no wall is truly inviscid. Instead, the fluid sticks to the wall forming a boundary layer between the wall and external flow. The inviscid wall is a mathematical construct to represent the “inviscid portion” of the flow away from the boundary layer – a legacy of potential flow solvers. The inviscid wall can be used in an Euler solver to represent solid wall by restricting the flow to be tangent to the wall. The same condition can be used in Navier-Stokes solvers to represent non-critical solid walls: Stings, aft-body fairings, and other surface used to simplify the geometry.

Since the inviscid wall represents a streamline in the flow, no restrictions are made to the heat transfer or turbulence along the inviscid wall. This contrasts the viscous wall.

3.6.3 Viscous Wall

The viscous wall represents a physical wall, to which the fluid sticks. The *no-slip* condition restricts the velocity of the fluid at the wall to be equal to the velocity of the wall (structure):

$$u_i = (u_i)_{wall} \quad (3.251)$$

The viscous wall also restricts the turbulence levels at the wall. The no-slip condition says that the velocity fluctuations in the Reynolds stress tensor $\overline{\rho u_i'' u_j''}$ and turbulent kinetic energy $\overline{\rho u_i'' u_i''}$ must vanish in the limit of the wall. This restriction creates two boundary conditions: The eddy viscosity, and thus Reynolds stresses, goes to zero at the wall; and, the turbulent kinetic energy is zero at the wall. The ω -condition is much more difficult to construct.

Wilcox (2002) provides a mathematical limit for ω near the wall:

$$\lim_{d \rightarrow 0} \omega = \lim_{\substack{K \rightarrow 0 \\ \mu_T \rightarrow 0}} \left(\alpha \frac{\rho K}{\mu_T} \text{Re} \right) \rightarrow \lim_{d \rightarrow 0} \left(\frac{6\mu}{\rho \beta_{o1} d^2} \right) \rightarrow \infty \quad (3.252)$$

An infinite value is impossible to model in a numerical system. Menter (1994) suggests calculating the ω -value at the first node off of the wall ω_{off} and estimating the wall value ω_{wall} as ten times the value off of the wall:

$$\omega_{off} = \frac{6\mu}{\rho \beta_{o1} d^2} \quad \omega_{wall} = 10\omega_{off} \quad (3.253)$$

Menter also suggests placing the first node off the wall within the viscous sublayer: $\Delta y^+ < 3$.

The viscous solid wall is the only boundary condition that represents physical contact between the fluid and solid surface. This contact spawns a series of heat transfer conditions: The *known temperature* condition represents the fluid-solid contact, like the no-slip condition. The alternative condition, *known heat flux*, represents the heat flowing between solid and fluid. A special heat flux condition is used as a mathematical limit, where no heat flux is allowed through the wall – *adiabatic wall*. This condition represents a perfectly

insulated wall. All walls have a finite thermal impedance, so the adiabatic wall is an ideal mathematical construct, often chosen as an alternative to the previous two conditions.

3.6.4 Symmetry Plane

The symmetry plane is a purely mathematical construct used to represent symmetry in the domain. Symmetry in the problem allows the solution space to be split in half, possibly multiple times. The new boundary created by dividing the domain is then modeled by the symmetry plane. The symmetry plane is a stream line so the velocity must be parallel to the symmetry boundary. All other property magnitudes are unrestricted along the symmetry plane, but those properties must reflect across the boundary. The gradients of all properties must be limited by this reflection so that no gradient exists normal to the boundary.

3.6.5 Rocket Exhaust

A rocket engine creates thrust by building a pressure potential that is expanded through a nozzle. The total pressure and temperature generated within the combustion chamber creates this potential. Instead of modeling the combustion within the chamber, this work models the total properties along an *inflow plane*. An air equivalent is used along the inflow plane in place of combustion properties. This simplifies the analysis to avoid tracking the advection of thermodynamic coefficients, like γ , c_p , and c_v . The flow then expands through the nozzle, which is modeled using solid walls. The remaining properties along the inflow plane can be calculated using the thermodynamic relationships presented earlier in this chapter. A third property is needed for the boundary to be well-posed and complete. This third property will be created in the next chapter to work best with the implementation of the condition.

3.6.6 Engine Planes

This work has also created boundary conditions to simulate the flow passing through turbojet engines. A turbojet pulls air in through its inlet, works the flow, adds heat through combustion, and then exhausts the flow through a nozzle. Like the rocket exhaust properties, an air equivalent is used to simplify the engine exhaust. If the engine is operating properly, the engine produces thrust. The engine pulls air into the inlet by decreasing the pressure downstream of the inlet. The engine can reject excess mass flow by increasing the pressure or decrease the mass flow to pull in more mass. These concepts were used to develop an *outflow boundary*, which represents the turbomachinery pulling flow through the inlet.

As the air passes through the turbomachinery, mass, momentum, and energy are added to the flow. Mass is added in the form of fuel flow, which is combusted adding energy; and the turbomachinery works the fluid creating a momentum exchange – *uninstalled thrust*⁵. The net conservation across the turbomachinery was used to link the outflow and inflow planes so that the conservation principles are responsible for the local flow and thrust from the turbojet.

3.6.7 Energy Balance

Care must be used when applying the previous boundary conditions. If the kinetic energy is altered by the boundary condition, the change in energy must be accounted for within the energy balance. If the total energy is changed by the boundary condition, energy has been added or destroyed by the numerical method that seeks to maintain conservation of energy.

⁵ *Uninstalled thrust* is the thrust created by the turbomachinery – compressor, combustor, and turbine. The increased thrust created when the inlet and nozzle are added to the uninstalled thrust of the turbomachinery is termed the *thrust*, or overall thrust.

The imbalance in energy can also create a pressure gradient along or normal to the wall through the ideal gas equation. The flow will be adapted to account for such energy changes.

3.7 Rigid Body Model

The non-inertial formulation has coupled with a rigid body dynamic model capable of modeling all six degrees-of-freedom (Cowan, 2003). O'Neill (2005) adapted the orientation tracking to utilize quaternions to avoid the singular present in Euler angles. This work refines the rigid body model to use the entire inertial matrix read from its file and introduced other research features. These features are outlined here.

3.7.1 Rigid Body Dynamics

The original rigid body system contained mass and inertial matrices with plug-ins for damping and stiffness matrices. The inertia matrix \mathbf{I}_m assumed centerline symmetry ($I_{xy} = I_{yz} = 0$). The previous algorithm also contained a matrix inversion routine that assumed that the inertial matrix was full. This research utilizes the entire inertia matrix and added diagonal stiffness and damping matrices. (The diagonal matrices can be used to simulate simple springs and dampers attached to the body. For instance, a drogue chute can be simulated as a linearized damper.) The inverse of the inertia matrix was applied to all moments induced on the body (e.g., inertial, damping, stiffness, aerodynamic, and external). The inertial forces $\Omega \mathbf{I}_m \vec{\omega}_o$ were filled out to include the full inertia matrix \mathbf{I}_m (Nelson, 1998):

$$\begin{bmatrix} \mathbf{M} & 0 \\ 0 & \mathbf{I}_m \end{bmatrix} \begin{Bmatrix} \dot{\vec{V}}_o \\ \dot{\vec{\omega}} \end{Bmatrix} + \begin{bmatrix} m\Omega + \mathbf{C}_x & 0 \\ 0 & \Omega \mathbf{I}_m + \mathbf{C}_\theta \end{bmatrix} \begin{Bmatrix} \vec{V}_o \\ \vec{\omega} \end{Bmatrix} + \begin{bmatrix} \mathbf{K}_x & 0 \\ 0 & \mathbf{K}_\theta \end{bmatrix} \begin{Bmatrix} \vec{x}_o \\ \vec{\theta} \end{Bmatrix} = \begin{Bmatrix} \vec{F} + m\vec{g} \\ \vec{M} \end{Bmatrix} \quad (3.254)$$

where

$$\mathbf{M} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{bmatrix} \quad \mathbf{C}_x = \begin{bmatrix} C_x & 0 & 0 \\ 0 & C_y & 0 \\ 0 & 0 & C_z \end{bmatrix} \quad \mathbf{K}_x = \begin{bmatrix} K_x & 0 & 0 \\ 0 & K_y & 0 \\ 0 & 0 & K_z \end{bmatrix} \quad (3.255)$$

$$\mathbf{I}_m = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix} \quad \mathbf{C}_\theta = \begin{bmatrix} C_\phi & 0 & 0 \\ 0 & C_\theta & 0 \\ 0 & 0 & C_\psi \end{bmatrix} \quad \mathbf{K}_\theta = \begin{bmatrix} K_\phi & 0 & 0 \\ 0 & K_\theta & 0 \\ 0 & 0 & K_\psi \end{bmatrix} \quad (3.256)$$

$$\mathbf{\Omega} = \vec{\omega} \times \mathbf{I} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \quad (3.257)$$

where m is the mass of the vehicle, I_m is the mass moment of inertia matrix, F is the aerodynamic and external forces, and M is the aerodynamic and external moments calculated about the center of rotation x_o . The systems equations are presented here in the dimensionless form used by Cowan (2003).

Special attention must be taken when applying the translational stiffness matrix. The stiffness matrix has been implemented without the translation \mathbf{B} between the frames. In other words, the spring force is based on the position within the global (inertial) frame and not relative to the body. This assumption is reasonable for a body, which is limited to small rotations, like those seen in a wind tunnel, so that the inertial and body frames are nearly aligned. In general, an aerospace vehicle is free to rotate and translate over great distances and is not constrained by linear stiffness.

The tracking of the center of rotation x_o represents the position of the body in the global (inertial) frame. The velocity has been formulated in the body fixed (non-inertial) frame. The rate of change of tracking position can be expressed in terms of the velocity in the body frame using the translation matrix: $\dot{\vec{x}}_o = \mathbf{B}\vec{V}_o$. If the orientation of the vehicle is tracked

using Euler angles, like in Euler2D and NS2D, the rate of change in orientation can be calculated using the rotational rates: $\dot{\theta} = \omega_z$. The 2D system can be simplified to a 6-degree-of-freedom system:

$$\begin{Bmatrix} \dot{\vec{x}}_o \\ \dot{\vec{V}}_o \\ \dot{\omega}_z \\ \dot{\theta} \end{Bmatrix} = \begin{Bmatrix} 0 \\ \frac{1}{m}\vec{F} + \vec{g} \\ \frac{1}{I_z}M_z \\ 0 \end{Bmatrix} - \begin{bmatrix} 0 & -\mathbf{B} & 0 & 0 \\ \frac{1}{m}\mathbf{K}_x & \frac{1}{m}\mathbf{C}_x & 0 & 0 \\ 0 & 0 & \frac{1}{I_z}\mathbf{C}_\theta & \frac{1}{I_z}\mathbf{K}_\theta \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{Bmatrix} \vec{x}_o \\ \vec{V}_o \\ \omega_z \\ \theta \end{Bmatrix} \quad (3.258)$$

Euler angles have a singularity that can lock the pitch orientation. To avoid this sensitivity, O'Neill (2004) implemented quaternions into Euler3D. The rate of change of quaternions can be expressed: $\dot{\vec{q}} = -\frac{1}{2}\Omega_q\vec{q}$. Quaternions will be discussed in more detail later in this section. The 3D system can be expressed as a 13-degree-of-freedom system:

$$\begin{Bmatrix} \dot{\vec{x}}_o \\ \dot{\vec{V}}_o \\ \dot{\vec{\omega}} \\ \dot{\vec{q}} \end{Bmatrix} = \begin{Bmatrix} 0 \\ \frac{1}{m}\vec{F} + \vec{g} \\ \mathbf{I}_m^{-1}(\vec{M} - \mathbf{K}_\theta\vec{\theta}) \\ 0 \end{Bmatrix} - \begin{bmatrix} 0 & -\mathbf{B} & 0 & 0 \\ \frac{1}{m}\mathbf{K}_x & \Omega + \frac{1}{m}\mathbf{C}_x & 0 & 0 \\ 0 & 0 & \mathbf{I}_m^{-1}(\Omega\mathbf{I}_m + \mathbf{C}_\theta) & 0 \\ 0 & 0 & 0 & \frac{1}{2}\Omega_q \end{bmatrix} \begin{Bmatrix} \vec{x}_o \\ \vec{V}_o \\ \vec{\omega} \\ \vec{q} \end{Bmatrix} \quad (3.259)$$

Gravity is assumed to always occur in the downward direction in the global (inertial) frame. In the non-inertial frame, the gravity vector rotates with the orientation of the body. This relationship between global and body-fixed gravity is expressed in three-dimensions:

$$\begin{Bmatrix} g_x \\ g_y \\ g_z \end{Bmatrix} = \vec{g} = \mathbf{B}^{-1}g\hat{k} = \begin{bmatrix} B_{10} & B_{11} & B_{12} \\ B_{13} & B_{14} & B_{15} \\ B_{16} & B_{17} & B_{18} \end{bmatrix} \begin{Bmatrix} 0 \\ 0 \\ g \end{Bmatrix} = \begin{Bmatrix} B_{12} \\ B_{15} \\ B_{18} \end{Bmatrix} g = \begin{Bmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{Bmatrix} g \quad (3.260)$$

or, in two-dimensions:

$$\begin{Bmatrix} g_x \\ g_y \end{Bmatrix} = \bar{\mathbf{g}} = \mathbf{B}^{-1} \mathbf{g} \hat{\mathbf{j}} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{Bmatrix} 0 \\ g \end{Bmatrix} = \begin{Bmatrix} \sin \theta \\ \cos \theta \end{Bmatrix} g \quad (3.261)$$

The freestream velocity has a similar relationship. In the inertial frame, the freestream velocity is expressed in terms of its magnitude and orientation (α and β). In the non-inertial frame, the freestream is aligned with the global x -direction, which is illustrated in Figure 3.1 for a translating and rotating boundary. The green vector represents the velocity induced by rotation; the blue vector is induced by translation; and, the red velocity is the freestream.

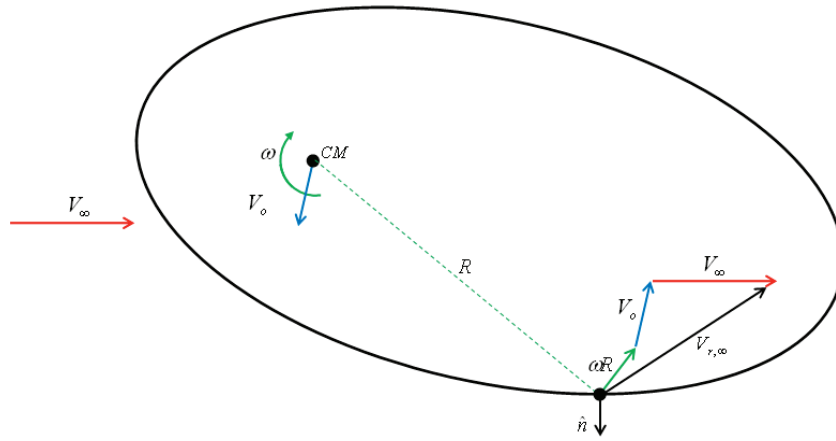


Figure 3.1: Relative Velocity at Boundary Node of a Translating / Spinning Domain.

The orientation of the body is used to rotate the freestream in the non-inertial frame. Using Eq. 3.3, the velocity along the non-inertial freestream boundary is calculated:

$$\begin{Bmatrix} u_{r,\infty} + V_{t,x} \\ v_{r,\infty} + V_{t,y} \\ w_{r,\infty} + V_{t,z} \end{Bmatrix} = \vec{u}_{r,\infty} + \vec{V}_t = \mathbf{B}^{-1} \vec{u}_\infty = \begin{bmatrix} B_{10} & B_{11} & B_{12} \\ B_{13} & B_{14} & B_{15} \\ B_{16} & B_{17} & B_{18} \end{bmatrix} \begin{Bmatrix} V_\infty \\ 0 \\ 0 \end{Bmatrix} = \begin{Bmatrix} B_{10} \\ B_{13} \\ B_{16} \end{Bmatrix} V_\infty \quad (3.262)$$

or, in two-dimensions:

$$\begin{Bmatrix} u_{r,\infty} + V_{t,x} \\ v_{r,\infty} + V_{t,y} \end{Bmatrix} = \vec{u}_{r,\infty} + \vec{V}_t = \mathbf{B}^{-1} \vec{u}_\infty = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{Bmatrix} V_\infty \\ 0 \end{Bmatrix} = \begin{Bmatrix} \cos \theta \\ -\sin \theta \end{Bmatrix} V_\infty \quad (3.263)$$

The relationship can be used to express a velocity along a non-inertial boundary $u_{r,b}$ in terms of its inertial counterpart u_b and the translation and rotation of the body:

$$\vec{u}_{r,b} = \vec{u}_b - \vec{V}_t = \vec{u}_b - \vec{V}_o - \vec{\omega} \times \vec{x}_b \quad (3.264)$$

3.7.2 Quaternions

Some additional attention is necessary to understand quaternions and their relationship to Euler angles. The orientation of the body can be expressed in this work in terms of Euler angles (θ, ϕ, ψ) or quaternions. Quaternions can be calculated from Euler angles:

$$\begin{Bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{Bmatrix} = \begin{Bmatrix} \cos \phi \cos \theta \cos \psi + \sin \phi \sin \theta \sin \psi \\ \sin \phi \cos \theta \cos \psi - \cos \phi \sin \theta \sin \psi \\ \cos \phi \sin \theta \cos \psi + \sin \phi \cos \theta \sin \psi \\ \cos \phi \cos \theta \sin \psi - \sin \phi \sin \theta \cos \psi \end{Bmatrix} \quad (3.265)$$

Euler angles are easier for the human mind to comprehend the orientation of the vehicle. The initial orientation of the body is expressed in terms of quaternions, so Euler3D and NS3D uses Eq. 3.265 to calculate the initial orientation in quaternions. The Euler angles are written to a file after each update. The Euler angles are calculated from the quaternions using:

$$\begin{Bmatrix} \phi \\ \theta \\ \psi \end{Bmatrix} = \begin{Bmatrix} \tan^{-1} \left(\frac{2(q_0 q_1 + q_2 q_3)}{q_0^2 - q_1^2 - q_2^2 + q_3^2} \right) \\ \sin^{-1}(q_{test}) \\ \tan^{-1} \left(\frac{2(q_0 q_3 + q_1 q_2)}{q_0^2 + q_1^2 - q_2^2 - q_3^2} \right) \end{Bmatrix} \quad -1 \leq q_{test} = 2(q_0 q_2 - q_1 q_3) \leq 1 \quad (3.266)$$

The position of the body in the inertial frame is connected to the velocity in the body fixed frame: $\dot{\vec{x}} = \mathbf{B} \vec{V}_o$. O'Neill (2005) writes the transformation matrix \mathbf{B} in terms of quaternions:

$$\mathbf{B} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 - q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \quad (3.267)$$

The quaternions change in time according to the *quaternion differential equation*:

$$\dot{\vec{q}} = \begin{Bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{Bmatrix} = -\frac{1}{2} \begin{bmatrix} 0 & \omega_x & \omega_y & \omega_z \\ -\omega_x & 0 & -\omega_z & \omega_y \\ -\omega_y & \omega_z & 0 & -\omega_x \\ -\omega_z & -\omega_y & \omega_x & 0 \end{bmatrix} \begin{Bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{Bmatrix} = -\frac{1}{2} \mathbf{\Omega}_q \vec{q} \quad (3.268)$$

which is a function of the quaternion position and rotational velocity ω . Finally, it is important to note that quaternions are a unit vector:

$$|\vec{q}|^2 = q_0^2 + q_1^2 + q_2^2 + q_3^2 = (\mathbf{S}_\phi^2 + \mathbf{C}_\phi^2)(\mathbf{S}_\theta^2 + \mathbf{C}_\theta^2)(\mathbf{S}_\psi^2 + \mathbf{C}_\psi^2) = 1 \quad (3.269)$$

3.8 Structural / Controls Model

Many fluids solvers model the deflection of solid walls by moving the boundary to the deformed location and adjusting (or remeshing) the domain to fit the new boundary position. This process is very costly. Small structural and control deflections can be modeled using transpiration (Fisher, 1996; Stephen, 1998; Cowan, 2004), so that the boundary and mesh do not need to be deformed at each step. If the mode shapes of the structure are used to express its motion, the structural dynamics can be modeled as a separate routine within the CFD process, which is much more efficient than progressing a time-accurate FEA model running concurrent with the CFD solver. The modal elastics are sufficient to model the deformations of the structure given enough modes are used to represent the modes that may become active during the simulation.

3.8.1 Transpiration (Inviscid)

Flow along an inviscid wall is restricted to be parallel to the wall (stationary). If the wall is moving, the relative velocity is used to calculate the flow tangency. Under any conditions the flow tangency is dependent upon the local wall normal to restrict the flow. As the structure deforms, the wall normal rotates. Figure 3.2 shows the wall normal before and after rotation. If the wall is not deformed but the normal vector is rotated using transpiration, flow tangency still aligns the flow with the deformed boundary. As long as the boundary does not displace over large distance or through large rotations (limited to 20 degrees or less), the transpired normal and undeformed boundary accurately model the inviscid wall. Cowan (2003) expresses transpiration (flow tangency) in inertial and non-inertial.

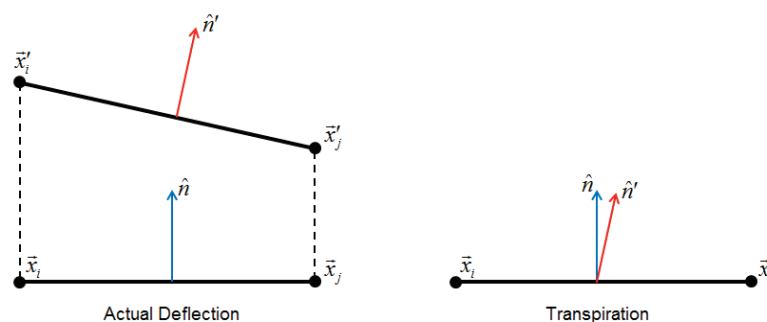


Figure 3.2: Solid Wall Normals with Elastic Motion.

3.8.2 Transpiration (Viscous)

Where the inviscid wall is represented mathematically by flow tangency, the flow senses the viscous wall because the fluid sticks to the wall. The no-slip condition was expressed in Eq. 3.251 in terms of a moving wall, where the velocity of the fluid is equal to the velocity of the wall. This condition has been modeled in NS2D and NS3D and has presented itself as useful

for modeling rotational and lateral velocities (Sukraw, 2008), which do not deform the boundary but rather slide the wall along the boundary surface.

Early in this work, much effort was put into finding a method of “viscous transpiration”, but no valid method has been found. A physical fluid, contrasted with a fluid in a numerical model, senses the presence of a wall when the molecules in the fluid bounce off the wall. If the wall deforms, the location of the wall moves, and the particles bounce off a different boundary to the flow. Mathematically, the deformation is modeled in the no-slip condition, which drives the velocity to zero at the wall. The fluid just off the wall builds a boundary layer that transfers the location of the wall to the external flow.

In a numerical sense, we have a limited number of variables through which to produce transpiration. The velocity is strictly dictated by the no-slip condition, which is an *essential* boundary condition. The temperature (enthalpy or energy) may also be specified in the boundary conditions. The remaining variables are density and pressure. The viscous stresses and heat fluxes could be added to the *unknowns* using a hybrid element approach. The hybrid method was thought to give the solver a means of rotating the stress tensor along the wall to model deformation of the normal, but the viscous stresses proven to be unsuccessful in transmitting wall position to the fluid. Pressure was thought to have the largest effect on the local flow field. The pressure was calculated on an airfoil rotated to 15-degrees incidence. The airfoil was then returned to zero incidence, and the solver was allowed to converge using the wall pressure from the inclined airfoil. The flow solution off of the wall showed minimal effects from pressure. The pressure and viscous stresses should be left as dependent variables because these parameters are used to calculate the forces on the wall.

Inviscid transpiration can be thought of as the deformation of the top of the boundary layer. Concepts were idealized to rotate all of the velocity vectors at the bottom or throughout the boundary layer, but such ideas would only destroy any validity of the near-wall solution.

Viscous walls are more constrained than inviscid walls and therefore lose the ability to apply a viscous transpiration. Inviscid transpiration works because the velocity vector is rotated parallel to the deformed wall. The position of that velocity is much less important than its orientation. In the viscous fluid, the wall velocity is absolutely specified and its position in the flow is the only means that can be used to transmit the position of the wall. From this, elastic deformations can only be incorporated into viscous walls by deforming the wall. This work does not allow for viscous wall deformations and leaves that for future researchers.

3.8.3 Modal Elastics

In modal elastics, the deformation of the solid surface is decomposed into N modes that are represented by mode shapes ϕ_n and linear system dynamics. A linear system of N modes is represented using mass \mathbf{M} , damping \mathbf{C} , and stiffness \mathbf{K} matrices and a generalized forcing vector \mathbf{F} . The generalized forcing vector is comprised of aerodynamic and external forces:

$$[\mathbf{M}]\ddot{\delta} + [\mathbf{C}]\dot{\delta} + [\mathbf{K}]\delta = \mathbf{F} \quad (3.270)$$

The systems equation is presented here in the dimensionless form used by Cowan (2003).

The solid walls are deformed according to the generalized displacements δ_n and mode shapes ϕ_n . If the i^{th} node on the wall is \bar{x}_i , the deformation and velocity of that node are calculated:

$$\bar{x}'_i = \bar{x}_i + \sum_N \delta_n \phi_n(\bar{x}_i) \quad \bar{V}_{b,i} = \sum_N \dot{\delta}_n \phi_n(\bar{x}_i) \quad (3.271)$$

CHAPTER IV

DEVELOPMENT

The previous chapter discussed the equations and models that are explored in this work. This chapter will discretize those equations using two versions of Galerkin's method. The four in-house OSU codes are discretized using a piece-wise linear Bubnov-Galerkin method, while NASA-CFDsol is discretized using a first-order Taylor-Galerkin method. The in-house codes use Gauss's theorem to create boundary integrals, where boundary conditions used to adapt the boundary fluxes. CFDsol applies the chain rule instead of Gauss's theorem on the inviscid fluxes so the boundary conditions are all explicitly applied between updates of the governing equations. The viscous terms in CFDsol are discretized using Gauss's theorem and its boundary integrals. The two types of solvers are developed independently to highlight their differences. The source terms (non-inertial, quasi-combustion, and turbulence) are developed in the same manner for all five solvers, and these terms are developed after the other aspects of the solvers. The chapter closes with the development of forces and moments through the integration of momentum and stability enhancement through local time stepping.

4.1 In-House Codes

Two in-house Euler codes were developed by Cowan (2003). These codes were developed using a piece-wise linear Bubnov-Galerkin method. Euler2D is a two-dimensional Euler code which was used as an efficient basis for developing understanding before expanding the method to Euler3D, the three-dimensional Euler code. Moffitt (2004) added viscous terms to Euler2D, creating the first implementation of NS2D. These methods, along with alterations for efficiency and accuracy, have been used to expand Euler2D/3D into NS2D/3D. Propulsion models were developed in all four codes, starting in the Euler codes and later expanding to the viscous codes.

4.1.1 Shape Function

A piece-wise linear property distribution is created using linear shape functions as a basis. The distribution is created by the linear combination of the shape functions and the properties at the nodes. The shape functions are defined to be equal to unity at one node and zero at all other nodes of the element. A one-dimensional linear element is illustrated in Figure 4.1. The shape functions are defined in terms of the local coordinates: $\Phi_e = \{\xi_1 \xi_2\}$. ξ_1 is the local coordinate that starts opposite of node 1 (value of 0) and extends across the element to node 1 (value of 1). ξ_2 is the other local coordinate, which starts opposite of node 2 with a value of 0 and takes on a value of 1 at node 2. The direction of the local coordinates may seem backwards, but this convention will make more sense in two-dimensions. The local coordinates are related in that their sum is always unity: $\xi_1 + \xi_2 = 1$. The second coordinate can now be written in terms of the first: $\xi_2 = 1 - \xi_1$, which allows the element to be defined in

terms of one degree-of-freedom ξ_1 . The properties are linearly distributed using the shape functions: $p = \Phi_e p_e = p_1 \xi_1 + p_2 \xi_2$. This pressure distribution is illustrated in Figure 4.1.

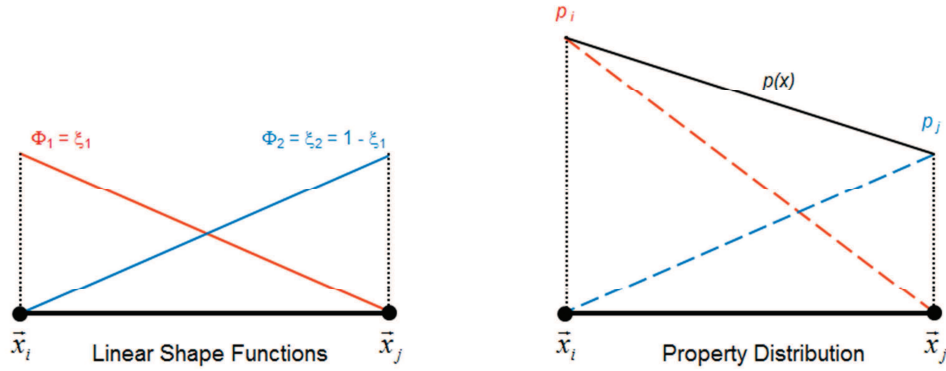


Figure 4.1: Linear Shape Function and Property Distribution.

A two-dimensional linear element is a triangular element defined by its three vertices. The shape function is defined using the local coordinates: $\Phi_e = \{\xi_1 \xi_2 \xi_3\}$. Each coordinate originates from the center of the opposite edge ($= 0$) and extends back to the node ($= 1$). The local coordinates and shape functions are illustrated in Figure 4.2. The local coordinates again sum to unity: $\xi_1 + \xi_2 + \xi_3 = 1$, so the third coordinate can be written in terms of the other two: $\xi_3 = 1 - \xi_1 - \xi_2$.

A linear three-dimensional element is a tetrahedral and can be defined by analogy to the previous elements. The shape functions are equal to the local coordinates: $\Phi_e = \{\xi_1 \xi_2 \xi_3 \xi_4\}$. The local coordinates originate from the center of the opposing face ($= 0$) and extending back to the node ($= 1$). The coordinates are related through their unit sum: $\xi_1 + \xi_2 + \xi_3 + \xi_4 = 1$. The fourth coordinate can be written in terms of the other three: $\xi_4 = 1 - \xi_1 - \xi_2 - \xi_3$.

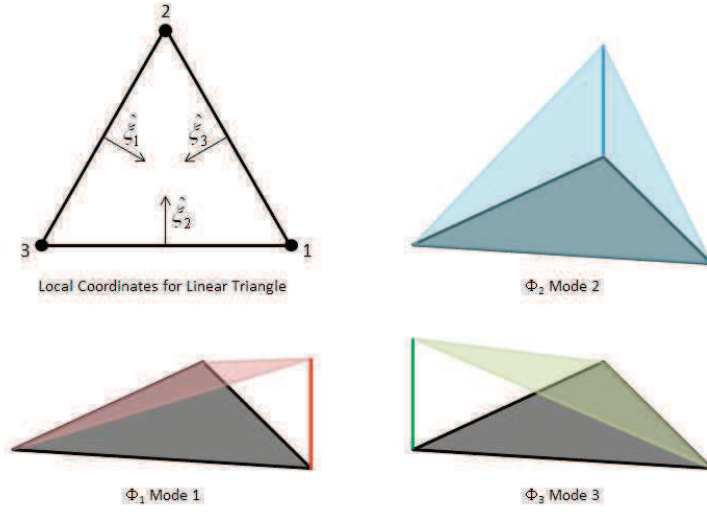


Figure 4.2: Local Coordinates and Shape Functions for Linear Triangular Element.

4.1.2 Element Jacobian

The global position within the element can be written using the shape functions as a basis.

For the linear one-dimensional element, the global position is written:

$$x = x_1\xi_1 + x_2\xi_2 = x_1(1 - \xi_2) + x_2\xi_2 = x_1 + J_e\xi_2 \quad (4.1)$$

where \mathbf{J}_e is the Jacobian. The Jacobian is equal to the length of the element: $J_e = l_e = x_2 - x_1$.

A similar relationship can be written for the global and local coordinates on the two- and three-dimensional elements:

$$\vec{x} = \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} x_3 \\ y_3 \end{Bmatrix} + \begin{bmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{bmatrix} \begin{Bmatrix} \xi_1 \\ \xi_2 \end{Bmatrix} = \vec{x}_3 + \mathbf{J}_e^T \vec{\xi} \quad (4.2)$$

$$\vec{x} = \begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{Bmatrix} x_4 \\ y_4 \\ z_4 \end{Bmatrix} + \begin{bmatrix} x_1 - x_4 & x_2 - x_4 & x_3 - x_4 \\ y_1 - y_4 & y_2 - y_4 & y_3 - y_4 \\ z_1 - z_4 & z_2 - z_4 & z_3 - z_4 \end{bmatrix} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix} = \vec{x}_4 + \mathbf{J}_e^T \vec{\xi} \quad (4.3)$$

The determinants of the Jacobians are related to the area and volume of their elements,

respectively: $|\mathbf{J}_e|_{tri} = 2A_e$ and $|\mathbf{J}_e|_{tet} = 6V_e$. Generically, the coordinates are related:

$$\vec{x} = \vec{x}_{d+1} + \mathbf{J}_e^T \vec{\xi} \quad (4.4)$$

where d is the number of dimensions in the element. The Jacobian can be used to convert integrals on the element from global to local coordinates:

$$\int_{l_e} f(\vec{x}) d\Omega = \int_0^1 f(\xi_1) |\mathbf{J}_e| d\xi_1 = l_e \int_0^1 f(\xi_1) d\xi_1 \quad (4.5)$$

$$\int_{A_e} f(\vec{x}) d\Omega = \int_0^1 \int_0^{(1-\xi_2)} f(\vec{\xi}) |\mathbf{J}_e| d\xi_1 d\xi_2 = 2A_e \int_0^1 \int_0^{(1-\xi_2)} f(\vec{\xi}) d\xi_1 d\xi_2 \quad (4.6)$$

$$\int_{V_e} f(\vec{x}) d\Omega = \int_0^1 \int_0^{(1-\xi_3)} \int_0^{(1-\xi_2-\xi_3)} f(\vec{\xi}) |\mathbf{J}_e| d\xi_1 d\xi_2 d\xi_3 = 6V_e \int_0^1 \int_0^{(1-\xi_3)} \int_0^{(1-\xi_2-\xi_3)} f(\vec{\xi}) d\xi_1 d\xi_2 d\xi_3 \quad (4.7)$$

Eqs. 4.5 – 4.7 will be used to convert element integrals into local coordinates, where the integrals can be simplified for each element before assembly into a global system of equations. Eq. 4.8 is created as a form of shorthand for the previous three relationships:

$$\int_{\Omega_e} f(\vec{x}) d\Omega = |\mathbf{J}_e| \int_{0 \leq \xi \leq 1} f(\vec{\xi}) d\vec{\xi} = d! \Omega_e \int_{0 \leq \xi \leq 1} f(\vec{\xi}) d\vec{\xi} \quad (4.8)$$

where Ω_e is equal to the length, area, and volume of the element, respectively. The Jacobian can be written generically: $|\mathbf{J}_e| = d! \Omega_e$.

4.1.3 Element Gradients

The Jacobian matrix \mathbf{J}_e can be used to define the property gradient. The properties are distributed using the shape functions Φ_e and properties at the nodes (here, pressure) p_e . The properties at the nodes are a function of solution progress, whether time accurate or steady-state; while the shape functions are a spatial distribution that does not change in time. The

spatial gradient of the properties can be calculated using the shape functions and the chain rule:

$$\frac{\partial p}{\partial \bar{x}} = \frac{\partial \bar{\xi}}{\partial \bar{x}} \frac{\partial p}{\partial \bar{\xi}} = \frac{\partial \bar{\xi}}{\partial \bar{x}} \frac{\partial \Phi_e}{\partial \bar{\xi}} p_e \quad (4.9)$$

From Eq. 4.4, the derivative between coordinates $\partial \bar{\xi} / \partial \bar{x}$ is related using the Jacobian matrix:

$$\frac{\partial}{\partial \bar{\xi}} = \frac{\partial \bar{x}}{\partial \bar{\xi}} \frac{\partial}{\partial \bar{x}} = \mathbf{J}_e \frac{\partial}{\partial \bar{x}} \quad \frac{\partial}{\partial \bar{x}} = \frac{\partial \bar{\xi}}{\partial \bar{x}} \frac{\partial}{\partial \bar{\xi}} = \mathbf{J}_e^{-1} \frac{\partial}{\partial \bar{\xi}} = \frac{1}{|\mathbf{J}_e|} \mathbf{A}_e \frac{\partial}{\partial \bar{\xi}} \quad (4.10)$$

The local gradient of the shape function is unity for the first d coordinates and -1 for the last:

$$\frac{\partial \Phi_{e,j}}{\partial \xi_i} = \frac{\partial \xi_j}{\partial \xi_i} = \delta_{ij} \quad \frac{\partial \Phi_{e,d+1}}{\partial \xi_i} = \frac{\partial}{\partial \xi_i} \left(1 - \sum_{j=1, \dots, d} \xi_j \right) = -1 \quad (4.11)$$

$$\frac{\partial \Phi_e}{\partial \bar{\xi}} = [\mathbf{I} \quad \{-1\}] \quad (4.12)$$

The global-gradient becomes:

$$\frac{\partial p}{\partial \bar{x}} = \frac{\partial \bar{\xi}}{\partial \bar{x}} \frac{\partial \Phi_e}{\partial \bar{\xi}} p_e = \mathbf{J}_e^{-1} [\mathbf{I} \quad \{-1\}] p_e = \frac{1}{|\mathbf{J}_e|} [\mathbf{A}_e \quad A_{e,d+1}] p_e \quad (4.13)$$

where \mathbf{I} is the identity matrix ($d \times d$) and $\{1\}$ is a column vector of ones (d -terms). The first d columns are \mathbf{A}_e , and the last column is the negative sum of the other d columns:

$$A_{i(d+1)} = -\sum_{j=1}^d A_{ij} \quad (4.14)$$

For the one-dimensional element, the gradient is calculated:

$$\frac{\partial p}{\partial x} = \frac{1}{|\mathbf{J}_e|} \mathbf{A}_e p_e = \frac{1}{l_e} \{-1 \quad 1\} \begin{Bmatrix} p_1 \\ p_2 \end{Bmatrix} \quad (4.15)$$

For the two-dimensional element, the gradients become:

$$\frac{\partial p}{\partial \bar{x}} = \frac{1}{|\mathbf{J}_e|} \mathbf{A}_e p_e = \frac{1}{2A_e} \begin{bmatrix} y_{23} & -y_{13} & y_{12} \\ -x_{23} & x_{13} & -x_{12} \end{bmatrix} \begin{Bmatrix} p_1 \\ p_2 \\ p_3 \end{Bmatrix} \quad (4.16)$$

where $x_{ij} = x_i - x_j$. For the three-dimensional element, the inverse Jacobian takes on the form:

$$|\mathbf{J}_e| \mathbf{J}_e^{-1} = \begin{bmatrix} y_{24}z_{34} - y_{34}z_{24} & y_{34}z_{14} - y_{14}z_{34} & y_{14}z_{24} - y_{24}z_{14} \\ z_{24}x_{34} - z_{34}x_{24} & z_{34}x_{14} - z_{14}x_{34} & z_{14}x_{24} - z_{24}x_{14} \\ x_{24}y_{34} - x_{34}y_{24} & x_{34}y_{14} - x_{14}y_{34} & x_{14}y_{24} - x_{24}y_{14} \end{bmatrix} \quad (4.17)$$

Eq. 4.17 represents the first three columns of the 3x4 gradient matrix. The fourth column is calculate using Eq. 4.14.

4.1.4 Consistent and Lumped Mass Matrices

A term called the *consistent mass matrix* will arise several times during the development of Galerkin's method. The elemental consistent mass matrix $\mathbf{M}_{c,e}$ is calculated:

$$\mathbf{M}_{c,e} = \int_{\Omega_e} \Phi_e^T \Phi_e d\Omega = |\mathbf{J}_e| \int_{0 \leq \bar{\xi} \leq 1} \Phi_e^T \Phi_e d\bar{\xi} = \frac{|\mathbf{J}_e|}{(d+2)!} (\mathbf{I} + [1]) = \frac{d! \Omega_e}{(d+2)!} (\mathbf{I} + [1]) \quad (4.18)$$

where The consistent mass matrix for the one-, two-, and three-dimensional elements are:

$$\mathbf{M}_{c,e} = \int_{l_e} \Phi_e^T \Phi_e d\Omega = \frac{l_e}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \quad (4.19)$$

$$\mathbf{M}_{c,e} = \int_{A_e} \Phi_e^T \Phi_e d\Omega = \frac{A_e}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \quad (4.20)$$

$$\mathbf{M}_{c,e} = \int_{V_e} \Phi_e^T \Phi_e d\Omega = \frac{V_e}{20} \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix} \quad (4.21)$$

For simplicity, the terms of the consistent mass matrix can be lumped together onto the diagonal. This form of the mass matrix is called the *lumped mass matrix*:

$$\mathbf{M}_{L,e} = \frac{d!\Omega_e}{(d+2)!}(d+2)\mathbf{I} = \frac{\Omega_e}{d+1}\mathbf{I} \quad (4.22)$$

$$\mathbf{M}_{L,e} = \frac{l_e}{2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (4.23)$$

$$\mathbf{M}_{L,e} = \frac{A_e}{3} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.24)$$

$$\mathbf{M}_{L,e} = \frac{V_e}{4} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.25)$$

4.1.5 Gauss Quadrature

Quadrature is used to convert integrals over regular regions into a linear combination of values taken evenly from that region. Gauss quadrature is written (Li, 2006; Jinyun, 1984):

$$\int_{0 \leq \xi \leq 1} f(\xi) d\xi = \sum_{N_p} \frac{w_j}{d!} f(\xi_j) \quad (4.26)$$

where w_k is the weight applied to the k^{th} point in the quadrature, and ξ_k is the local coordinates of the k^{th} point. If m^{th} order quadrature is used to integrate an n^{th} order integrand, the quadrature result is exactly equal to the analytical integral, if and only if $m \geq n$. In other words, if the integrand f is a third order polynomial, then quadrature of third order or higher will result in an exact solution. First and second order quadrature are less expensive and can be used to approximate the integral of the third order polynomial. Table 4.1 shows the

weights and locations for the lowest three orders of Gauss quadrature on triangle elements.

Table 4.2 shows the first three orders for tetrahedral elements.

Table 4.1: Gauss Quadrature – Triangle Elements.

Order	# of Pts	Weights	Local Coordinates		
m	---	w_j	$\Phi_{e,j}$		
1	1	1	1/3	1/3	1/3
2	3	1/3	1/2	1/2	0
		1/3	1/2	0	1/2
		1/3	0	1/2	1/2
3	4	-27/48	1/3	1/3	1/3
		25/48	0.6	0.2	0.2
		25/48	0.2	0.6	0.2
		25/48	0.2	0.2	0.6

Table 4.2: Gauss Quadrature – Tetrahedra Elements.

Order	# of Pts	Weights	Local Coordinates			
m	---	w_j	$\Phi_{e,j}$			
1	1	1	0.25	0.25	0.25	0.25
2	4	0.25	0.58541020	0.13819660	0.13819660	0.13819660
		0.25	0.13819660	0.58541020	0.13819660	0.13819660
		0.25	0.13819660	0.13819660	0.58541020	0.13819660
		0.25	0.13819660	0.13819660	0.13819660	0.58541020
3	5	-4/5	0.25	0.25	0.25	0.25
		9/20	0.5	1/6	1/6	1/6
		9/20	1/6	0.5	1/6	1/6
		9/20	1/6	1/6	0.5	1/6
		9/20	1/6	1/6	1/6	0.5

Cowan (2003) implemented first and second order Gauss quadrature to integrate the flux terms. In Euler2D, first and second order quadrature requires one and three points, respectively. In Euler3D, one and four points are required for the same integral orders. Cowan's analysis showed that higher order quadrature did not improve the accuracy and created longer run times.

Given a piece-wise linear distribution of properties, the inviscid flux is at worst third order on each element. Early in this work, third order integration was implemented in Euler2D in two ways: Gauss quadrature and analytical integration. (The analytical integrals are demonstrated in Appendix B.) These routines were tested by Brown (2009). Brown's results showed that quadrature order increased run time and that analytical integration was faster than third order quadrature. Brown's tests were not definitive because artificial dissipation dominated the error convergence rather than quadrature order. Brown's study should be repeated after implemented an artificial dissipation model with less overall impact.

If viscosity and thermal conductivity are assumed to be piece-wise constant on the domain, the viscous momentum fluxes are piece-wise constant and the viscous energy flux is piece-wise linear because of velocity. Pair with Cowan and Brown's results, first order quadrature was selected exclusively in the development of NS2D and NS3D. For first order quadrature, the location is defined at the center of the element ($\xi_1 = 1/d+1$) with a weight of unity ($w_1 = 1$). In other words, the integral is approximated using the average properties on that element. Two- and three-dimensional integrals over triangle and tetrahedra are simplified with one quadrature point:

$$\int_0^1 \int_0^{(1-\xi_2)} f(\xi_1, \xi_2) d\xi_1 d\xi_2 = \frac{1}{2!} f(1/3, 1/3) = \frac{1}{2} \bar{f}_e \quad (4.27)$$

$$\int_0^1 \int_0^{(1-\xi_3)} \int_0^{(1-\xi_2-\xi_3)} f(\xi_1, \xi_2, \xi_3) d\xi_1 d\xi_2 d\xi_3 = \frac{1}{3!} f(1/4, 1/4, 1/4) = \frac{1}{6} \bar{f}_e \quad (4.28)$$

4.1.6 Galerkin Formulation

The governing equations for the time-averaged Navier-Stokes equations (Eqs. 3.204 thru 3.210), SA model (Eq. 3.229), and SST model (Eq. 3.235 and 3.236) are combined together

into a single system of equations. The turbulence model is then selected by using its particular governing equation(s) and ignoring those from the other model. A source term \mathbf{S}_C is also included in the governing equations for later inclusion of the quasi-combustion model. The combined system is represented in Eqs. 4.29 thru 4.32:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x_{b,i}} = \frac{\partial \mathbf{F}_{v,i}}{\partial x_{b,i}} + \mathbf{S}_{NI} + \mathbf{S}_C + \mathbf{S}_T \quad (4.29)$$

$$\mathbf{U} = \begin{Bmatrix} \rho \\ \rho u_{r,j} \\ \rho E_r \\ \rho \hat{v} \\ \rho K \\ \rho \omega \end{Bmatrix} \quad \mathbf{F}_i = \begin{Bmatrix} \rho u_{r,i} \\ \rho u_{r,i} u_{r,j} + p \delta_{ij} \\ u_{r,i} \rho H_r \\ u_{r,i} \rho \hat{v} \\ u_{r,i} \rho K \\ u_{r,i} \rho \omega \end{Bmatrix} \quad \mathbf{S}_{NI} = -\rho \begin{Bmatrix} 0 \\ a_{t,j} + \varepsilon_{jkl} \omega_k u_{r,l} \\ a_{t,k} \cdot (V_{t,k} + u_{r,k}) \\ 0 \\ 0 \\ 0 \end{Bmatrix} \quad (4.30)$$

$$\mathbf{F}_{v,i} = \begin{Bmatrix} 0 \\ \tau_{ij} + \rho \Gamma_{ij} \\ u_{r,j} (\tau_{ij} + \rho \Gamma_{ij}) - q_i'' - Q_i + \frac{1}{\text{Re}} (\mu + \sigma_k \mu_T) \nabla_i K \\ \frac{1}{\sigma \text{Re}} (\mu + \rho \hat{v}) \nabla_i \hat{v} \\ \frac{1}{\text{Re}} (\mu + \sigma_k \mu_T) \nabla_i K \\ \frac{1}{\text{Re}} (\mu + \sigma_\omega \mu_T) \nabla_i \omega \end{Bmatrix} \quad (4.31)$$

$$\mathbf{S}_T = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ f_{r1} c_{b1} \hat{S} \rho \hat{v} - \frac{1}{\text{Re}} c_{w1} f_w \rho \left(\frac{\hat{v}}{d}\right)^2 + \frac{1}{\sigma \text{Re}} \nabla_i \rho \hat{v} \nabla_i \hat{v} \\ \rho \Pi_k - \rho \varepsilon \\ f_{r1} \rho \Pi_\omega - \beta_c \rho (\omega^2 - \omega_{amb}^2) + \frac{C_\omega}{\rho \omega} \nabla_i \rho K \nabla_i \rho \omega \end{Bmatrix} \quad (4.32)$$

Eq. 4.29 is integrated used the Galerkin finite element method:

$$\int_{\Omega} \Phi^T \left(\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial}{\partial x_{b,i}} (\mathbf{F}_i - \mathbf{F}_{v,i}) - \mathbf{S}_{NI} - \mathbf{S}_C - \mathbf{S}_T \right) d\Omega = 0 \quad (4.33)$$

The viscous flux terms (momentum) are piece-wise constant on the domain. Their gradient is identically zero. This eliminates the possibility of representing the Navier-Stokes equations in this form. If the Gauss's theorem is applied to the inviscid and viscous flux terms, then the gradient is shifted from the fluxes to the shape function:

$$\int_{\Omega} \Phi^T \left(\frac{\partial \mathbf{U}}{\partial t} - \mathbf{S}_{NI} - \mathbf{S}_C - \mathbf{S}_T \right) d\Omega - \int_{\Omega} \frac{\partial \Phi^T}{\partial x_{b,i}} (\mathbf{F}_i - \mathbf{F}_{v,i}) d\Omega + \int_{\Gamma} \Phi^T (\mathbf{F}_i - \mathbf{F}_{v,i}) \cdot \hat{n}_i d\Gamma = 0 \quad (4.34)$$

Gauss's theorem also creates integrals over the boundary Γ , which can be used to imply the boundary conditions. The properties are distributed in a piece-wise linear manner across the domain using the shape function Φ_e , simplifying the domain and boundary into a series of domain and boundary element integrals:

$$\begin{aligned} \sum_e \int_{\Omega_e} \Phi_e^T \left(\frac{\partial \mathbf{U}}{\partial t} - \mathbf{S}_{NI} - \mathbf{S}_C - \mathbf{S}_T \right) d\Omega - \sum_e \int_{\Omega_e} \frac{\partial \Phi_e^T}{\partial x_{b,i}} (\mathbf{F}_i - \mathbf{F}_{v,i}) d\Omega \\ + \sum_{be} \int_{\Gamma_{be}} \Phi_{be}^T (\mathbf{F}_i - \mathbf{F}_{v,i}) \cdot \hat{n}_i d\Gamma = 0 \end{aligned} \quad (4.35)$$

where Γ_{be} is the edge length l_{be} in two-dimensions and face area A_{be} in three-dimensions.

The equations can be used to construct a residual vector:

$$\begin{aligned} \mathbf{R}(\mathbf{U}_{n+1}, \mathbf{U}_n, \mathbf{U}_{n-1}) = \sum_e \Delta t_n \int_{\Omega_e} \Phi_e^T \left(\frac{\partial \mathbf{U}}{\partial t} - \mathbf{S}_{NI} - \mathbf{S}_C - \mathbf{S}_T \right) d\Omega \\ - \sum_e \Delta t_n \int_{\Omega_e} \frac{\partial \Phi_e^T}{\partial x_{b,i}} (\mathbf{F}_i - \mathbf{F}_{v,i}) d\Omega + \Delta t_n \sum_{be} \int_{\Gamma_{be}} \Phi_{be}^T (\mathbf{F}_i - \mathbf{F}_{v,i}) \cdot \hat{n}_i d\Gamma \end{aligned} \quad (4.36)$$

4.1.7 Gauss's Theorem

Gauss's theorem was used to help smooth a piece-wise constant pressure gradient. The resulting distribution was susceptible to numerical errors and degraded near the boundaries.

Both issues brought into question the validity of Gauss's theorem on discontinuous functions, like the piece-wise constant viscous flux terms:

$$\int_{\Omega} \Phi_e^T \frac{\partial \mathbf{F}_{v,i}}{\partial x_i} d\Omega = \int_{\Gamma} \Phi_e^T (\mathbf{F}_{v,i} \cdot \hat{n}_i) d\Gamma - \int_{\Omega} \frac{\partial \Phi_e^T}{\partial x_i} \mathbf{F}_{v,i} d\Omega \quad (4.37)$$

Gauss's theorem requires that the flux $\mathbf{F}_{v,i}$ be continuous on the domain Ω and its boundary Γ . The viscous flux is constant on each element and discontinuous between elements, violating the mathematical requirements of Gauss's theorem. Instead of applying Gauss's theorem to Eq. 4.33 and then discretize the result into element and boundary element integrals, Eq. 4.33 was first discretized into elements that are discontinuous in viscous fluxes $\mathbf{F}_{v,i}$ and then Gauss's theorem was applied to the integral on each element:

$$\int_{\Omega} \Phi_e^T \frac{\partial \mathbf{F}_{v,i}}{\partial x_i} d\Omega = \sum_e \int_{\Omega_e} \Phi_e^T \frac{\partial \mathbf{F}_{v,i}}{\partial x_i} d\Omega = \sum_f \int_{\Gamma_f} \Phi_e^T (\mathbf{F}_{v,i} \cdot \hat{n}) d\Gamma - \sum_e \int_{\Omega_e} \frac{\partial \Phi_e^T}{\partial x_i} \mathbf{F}_{v,i} d\Omega \quad (4.38)$$

Gauss's theorem is applied element-wise, which satisfies the requirement that the flux $\mathbf{F}_{v,i}$ be continuous on the domain (element) Ω_e and its boundary (or face) Γ_f . The difference in the two methods requires the integrals over the internal interfaces (interfaces between elements) to be included in the summation. If the derivatives were to match, which they seldom do, then the interface integrals for two adjacent elements cancel each other. The interface integrals should correct any error from applying Gauss's theorem to a discontinuous domain. The new "Gauss friendly" scheme was found to be more accurate over a finite band of element sizes. For elements outside of this band, the original scheme was much more stable and more accurate. Concerns with Gauss's theorem creating the instabilities in NS3D and CFDsol were disregarded. The problem is purely a numerical error propagation issue. The numerical errors themselves need to be eliminated. Viscous local time stepping is developed

at the end of the chapter to not only stabilize the local flow but minimize the propagation (feedback) of numerical errors.

4.1.8 Unsteady Term

Cowan (2003) developed the unsteady terms by linearly distributing the unknowns vector \mathbf{U} on the element. The integration simplifies to the consistent mass matrix $\mathbf{M}_{c,e}$:

$$\int_{\Omega_e} \Phi_e^T \frac{\partial \mathbf{U}}{\partial t} d\Omega = \int_{\Omega_e} \Phi_e^T \Phi_e d\Omega \frac{\partial \mathbf{U}_e}{\partial t} = \mathbf{M}_{c,e} \frac{\partial \mathbf{U}_e}{\partial t} \quad (4.39)$$

The two- and three-dimensional mass matrices are given in Eqs. 4.20 and 4.21, respectively. For steady solutions Cowan neglects the unsteady term. For unsteady solutions, Cowan models the temporal derivative using first and second order finite difference approximations, respectively:

$$\frac{\partial \mathbf{U}_e}{\partial t} = \frac{\mathbf{U}_{e,n+1} - \mathbf{U}_{e,n}}{\Delta t} \quad (4.40)$$

$$\frac{\partial \mathbf{U}_e}{\partial t} = \frac{1.5\mathbf{U}_{e,n+1} - 2\mathbf{U}_{e,n} + \mathbf{U}_{e,n-1}}{\Delta t} \quad (4.41)$$

4.1.9 Element Fluxes Terms

The inviscid and viscous fluxes in the domain element integrals in Eq. 4.36 are integrated here using first order Gauss quadrature (one point):

$$\int_{\Omega_e} \frac{\partial \Phi_e^T}{\partial x_{b,i}} (\mathbf{F}_i - \mathbf{F}_{v,i}) d\Omega = \int_{0 \leq \bar{\xi} \leq 1} \frac{\partial \Phi_e^T}{\partial x_{b,i}} (\mathbf{F}_i - \mathbf{F}_{v,i}) |\mathbf{J}_e| d\bar{\xi} = \frac{1}{d!} A_i^T (\bar{\mathbf{F}}_i - \bar{\mathbf{F}}_{v,i}) \quad (4.42)$$

where A_i is the i^{th} row of the shape function gradient (Eqs. 4.13 and 4.14). The two-dimensional flux integrals are represented using the properties on the element:

$$\int_{\Omega_e} \frac{\partial \Phi_e^T}{\partial x_{b,i}} (\mathbf{F}_i - \mathbf{F}_{v,i}) d\Omega = \frac{1}{2} \begin{Bmatrix} A_{i1} \\ A_{i2} \\ A_{i3} \end{Bmatrix} (\bar{\mathbf{F}}_i - \bar{\mathbf{F}}_{v,i}) \quad (4.43)$$

The three-dimensional flux integrals are similarly:

$$\int_{\Omega_e} \frac{\partial \Phi_e^T}{\partial x_{b,i}} (\mathbf{F}_i - \mathbf{F}_{v,i}) d\Omega = \frac{1}{6} \begin{Bmatrix} A_{i1} \\ A_{i2} \\ A_{i3} \\ A_{i4} \end{Bmatrix} (\bar{\mathbf{F}}_i - \bar{\mathbf{F}}_{v,i}) \quad (4.44)$$

The inviscid and viscous fluxes are evaluated with Eqs. 4.30 and 4.31 using the properties at the center of each element. The properties at the center of the element are equal to the average of the properties at the vertices.

4.1.10 Boundary Flux Terms

The inviscid and viscous boundary integrals are simplified using different approaches.

Cowan (2003) linearly distributes the inviscid fluxes across the boundary element to increase stability. The linearization puts two shape functions inside the integral, which simplifies to the consistent mass matrix (Eqs. 4.19 and 4.20).

$$\int_{\Gamma_{be}} \Phi_{be}^T (\mathbf{F}_i \cdot \hat{\mathbf{n}}_i) d\Gamma = |\mathbf{J}_{be}| \int_{0 \leq \xi \leq 1} \Phi_{be}^T \Phi_{be} d\Gamma (\mathbf{F}_{i,be} \cdot \hat{\mathbf{n}}_i) = [\mathbf{M}_{c,be}] (\mathbf{F}_{i,be} \cdot \hat{\mathbf{n}}_i) \quad (4.45)$$

The two-dimensional inviscid boundary integral is evaluated:

$$\int_{\Gamma_{be}} \Phi_{be}^T (\mathbf{F}_i \cdot \hat{\mathbf{n}}_i) d\Gamma = \frac{l_{be}}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{Bmatrix} \mathbf{F}_{i,1} \cdot \hat{\mathbf{n}}_i \\ \mathbf{F}_{i,2} \cdot \hat{\mathbf{n}}_i \end{Bmatrix} \quad (4.46)$$

The three-dimensional inviscid boundary integral is modeled:

$$\int_{\Gamma_{be}} \Phi_{be}^T(\mathbf{F}_i \cdot \hat{\mathbf{n}}_i) d\Gamma = \frac{A_{be}}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \begin{Bmatrix} \mathbf{F}_{i,1} \cdot \hat{\mathbf{n}}_i \\ \mathbf{F}_{i,2} \cdot \hat{\mathbf{n}}_i \\ \mathbf{F}_{i,3} \cdot \hat{\mathbf{n}}_i \end{Bmatrix} \quad (4.47)$$

Moffitt (2004) uses Gauss quadrature on the boundary integral to match the adjacent domain integral. The stresses and heat fluxes in the viscous flux (Eq. 4.31) are evaluated using velocity and enthalpy gradients (Eqs. 3.220 and 3.222). The boundary element does not contain enough information to calculate the gradient, so the gradients are taken from the adjacent element.

$$\int_{\Gamma_{be}} \Phi_{be}^T(\mathbf{F}_{v,i} \cdot \hat{\mathbf{n}}_i) d\Gamma = |\mathbf{J}_{be}| \int_{0 \leq \xi \leq 1} \Phi_{be}^T(\mathbf{F}_{v,i} \cdot \hat{\mathbf{n}}_i) d\Gamma = \frac{\Gamma_{be}}{d} \{1\} (\bar{\mathbf{F}}_{v,i} \cdot \hat{\mathbf{n}}_i) \quad (4.48)$$

The two-dimensional viscous boundary integral is modeled:

$$\int_{\Gamma_{be}} \Phi_{be}^T(\mathbf{F}_i \cdot \hat{\mathbf{n}}_i) d\Gamma = \frac{l_{be}}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix} (\bar{\mathbf{F}}_{v,i} \cdot \hat{\mathbf{n}}_i) \quad (4.49)$$

The three-dimensional viscous boundary integral is represented:

$$\int_{\Gamma_{be}} \Phi_{be}^T(\mathbf{F}_i \cdot \hat{\mathbf{n}}_i) d\Gamma = \frac{A_{be}}{3} \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix} (\bar{\mathbf{F}}_{v,i} \cdot \hat{\mathbf{n}}_i) \quad (4.50)$$

4.1.10.1 Riemann Invariants

Many of the boundary conditions applied in the in-house codes imply properties and fluxes on the boundary that do not necessarily match the properties or fluxes on the element just inside of the domain. This discontinuity is demonstrated in Figure 4.3. Riemann invariants are used to calculate an appropriate inviscid normal flux to evaluate the boundary integrals.

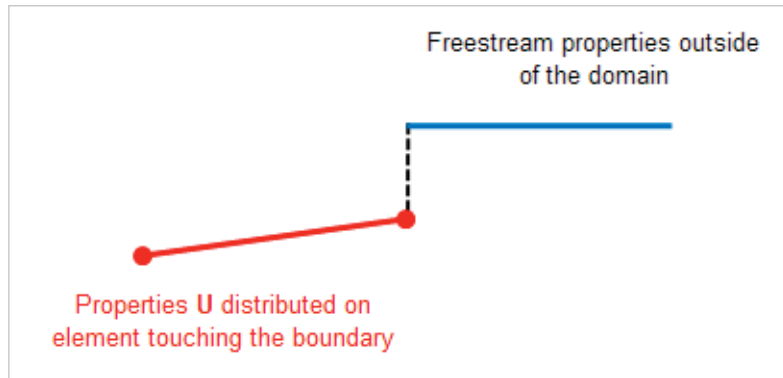


Figure 4.3: Discontinuity at the Boundary Element.

In seeking a stable and accurate solution along a discontinuous boundary, Givoli's (1991) criteria for a far field boundary can be applied to any discontinuous boundary:

- Approximate the desired properties outside of the domain on a finite space,
- Be compatible with the current numerical scheme,
- Not create spurious reflections on the boundary, and
- Reach a steady-state or quasi-steady-state rapidly for use in time-accurate solutions.

Cowan (2003) applied the Riemann invariant to the far field boundaries in Euler2D and Euler3D. Riemann invariants are correct the normal flux in the boundary integrals, and the invariant allows the boundary to flex with the internal properties while guiding the residual at the boundary toward the desired properties in an error minimization type approach.

1D Riemann Problem. The one-dimensional Euler equations are written in conservative form:

$$\frac{\partial}{\partial t} \begin{Bmatrix} \rho \\ \rho u \\ \rho e \end{Bmatrix} + \frac{\partial}{\partial x} \begin{Bmatrix} \rho u \\ \rho u^2 + p \\ (\rho e + p)u \end{Bmatrix} = 0 \quad (4.51)$$

The governing equations can be written in terms of the primitive variables, ρ , u , and p , using the ideal gas equation (Ivings, 1998):

$$\frac{\partial}{\partial t} \begin{Bmatrix} \rho \\ u \\ p \end{Bmatrix} + \begin{bmatrix} u & \rho & 0 \\ 0 & u & 1/\rho \\ 0 & \rho a^2 & u \end{bmatrix} \frac{\partial}{\partial x} \begin{Bmatrix} \rho \\ u \\ p \end{Bmatrix} = 0 \quad (4.52)$$

or, in short hand notation:

$$\frac{\partial \mathbf{U}}{\partial t} + [\mathbf{A}(\mathbf{U})] \frac{\partial \mathbf{U}}{\partial x} = 0 \quad (4.53)$$

The eigenvalues of $[\mathbf{A}(\mathbf{U})]$ represent the wave speed of information traveling in the x -direction:

$$\begin{vmatrix} u - \lambda & \rho & 0 \\ 0 & u - \lambda & 1/\rho \\ 0 & \rho a^2 & u - \lambda \end{vmatrix} = (u - \lambda)((u - \lambda)^2 - a^2) = 0 \quad (4.54)$$

The eigenvalues are $u - a$, u , and $u + a$, which are all real values. The corresponding eigenvectors are:

$$\begin{Bmatrix} \rho \\ -a \\ \rho a^2 \end{Bmatrix}, \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix}, \begin{Bmatrix} \rho \\ a \\ \rho a^2 \end{Bmatrix} \quad (4.55)$$

The eigenvalues and eigenvectors tell us about how information travels through the one-dimensional Euler system:

- The first wave travels downstream at a velocity $u - a$ and contains changes to density, velocity, and pressure (energy).

- The second wave travels downstream at a velocity u and contains only information about the density changes in the flow.
- The third wave travels downstream at a velocity $u + a$ and contains information from all of the flow properties, like the first wave.

The three characteristics transfer the information at different speeds and directions depending upon the speed of the flow relative to the acoustic speed. The behavior of the characteristics can be broken down by flow regime:

- *Acoustic* ($u = 0$): The first and third characteristics travel left and right (respectively) at the acoustic speed, and the other characteristic does not move.
- *Subsonic* ($u < a$): The first characteristic travels upstream (left), while the other two travel downstream (right) at two different speeds.
- *Sonic* ($u = a$): The first characteristic does not travel at all, while the other two travel downstream at two different speeds.
- *Supersonic* ($u > a$): All three characteristics travel downstream at different speeds.

The behavior of the three characteristics in the four regimes is illustrated in Figure 4.4. The differences in behavior explain why an entire subsonic flow field is affected by the presence of a solid body, whereas a supersonic flow field is only affected downstream of the solid body. Figure 4.5 shows the characteristics in 2D flow field. A point sound source is traveling at a velocity V . Acoustic waves emanate in all directions from the source at a velocity a . For $V < a$ (subsonic), the wave travels in all direction. When $V = a$ (sonic), the acoustic waves culminate into a normal shock. For $V > a$ (supersonic), the acoustic waves create two oblique shocks.

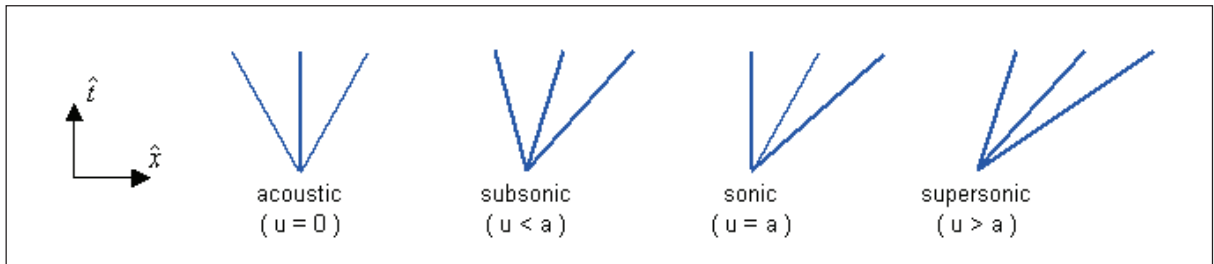


Figure 4.4: Behavior of Characteristic Waves in Different Flow Regimes.

The inviscid governing equations can be written generically using the Riemann invariant matrix \mathbf{A} as (Toro, 1997):

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} = \frac{\partial \mathbf{U}}{\partial t} + [\mathbf{A}] \frac{\partial \mathbf{U}}{\partial x} = 0 \quad [\mathbf{A}] = \frac{\partial \mathbf{F}}{\partial \mathbf{U}} \quad (4.56)$$

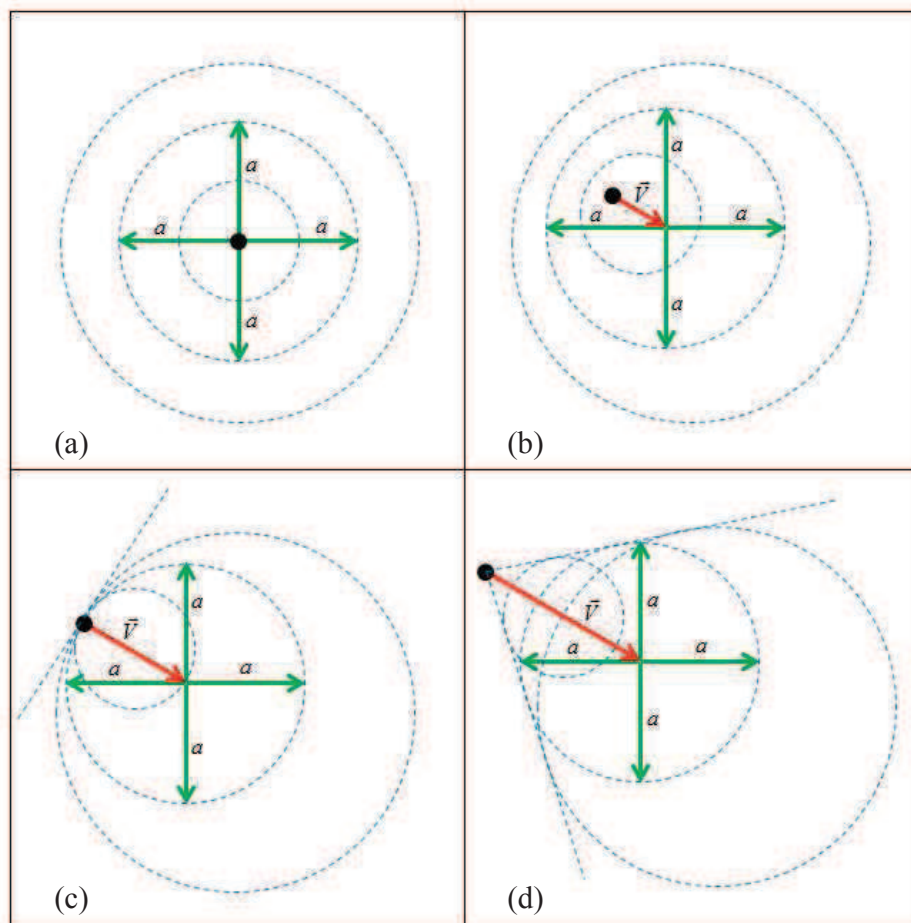


Figure 4.5: 2D Characteristics: (a) Acoustic, (b) Subsonic, (c) Sonic, & (d) Supersonic.

The invariant matrix \mathbf{A} is used to calculate the flux using a first-order Taylor series expansion:

$$\mathbf{F}_b = \mathbf{F} + \frac{\partial \mathbf{F}}{\partial \mathbf{U}} \Delta \mathbf{U} = \mathbf{F} + [\mathbf{A}] \Delta \mathbf{U} \quad (4.57)$$

The flux is discontinuous across the boundary, as shown in Figure 4.3. The flux on the element side of the boundary (left in Figure 4.3) is calculated from the distribution of properties on the element and corrected with the characteristics flowing from the domain to the boundary:

$$\mathbf{F}_{b,e} = \mathbf{F}_e - [\mathbf{A}_+](\mathbf{U}_e - \mathbf{U}_{BC}) \quad (4.58)$$

The flux on the other side of the boundary (right in Figure 4.3) is calculated using the boundary conditions and corrected with the characteristics flowing from outside to the boundary:

$$\mathbf{F}_{b,BC} = \mathbf{F}_{BC} + [\mathbf{A}_-](\mathbf{U}_e - \mathbf{U}_{BC}) \quad (4.59)$$

$[\mathbf{A}_+]$ and $[\mathbf{A}_-]$ are generated by using only the positive and negative characteristics, respectively, to reconstruct the \mathbf{A} -matrix from its eigenvalues and eigenvectors (Toro, 1997):

$$[\mathbf{A}_+] = [\Phi_A][\Lambda_A^+][\Phi_A]^{-1} \quad \text{and} \quad [\mathbf{A}_-] = [\Phi_A][\Lambda_A^-][\Phi_A]^{-1} \quad (4.60)$$

$$(\Lambda_A^+)_{ii} = \text{Max}(\lambda_i, 0) \quad \text{and} \quad (\Lambda_A^-)_{ii} = \text{Min}(\lambda_i, 0) \quad (4.61)$$

The fluxes can be averaged to approximate the flux along the boundary (LeVeque, 1992):

$$\mathbf{F}_c = \frac{\mathbf{F}_{b,e} + \mathbf{F}_{b,BC}}{2} = \frac{\mathbf{F}_e + \mathbf{F}_{BC}}{2} - \frac{1}{2}([\mathbf{A}_+] - [\mathbf{A}_-])(\mathbf{U}_e - \mathbf{U}_{BC}) \quad (4.62)$$

$$\mathbf{F}_c = \frac{\mathbf{F}_e + \mathbf{F}_{BC}}{2} - \frac{1}{2}[\mathbf{A}^*](\mathbf{U}_e - \mathbf{U}_{BC}) \quad (4.63)$$

$$[\mathbf{A}^*] = [\Phi_A][\Lambda_A^*][\Phi_A]^{-1} \quad \text{and} \quad (\Lambda_A^*)_{ii} = |\lambda_i| \quad (4.64)$$

2D and 3D Riemann Problems. Riemann invariants are applied along the boundaries through the boundary integrals. These integrals require the normal flux along the boundary. The invariant matrix will be used as a first-order expansion of the normal flux, converting the problem from a two- or three-dimensional problem to a one-dimensional problem. Riemann invariants are most, if not only, appropriate for one-dimensional cases (Godon, 1993). The \mathbf{A} -matrix is defined by taking the derivative of the normal flux vector \mathbf{F}_n with respect to the unknowns vector \mathbf{U} :

$$\mathbf{F}_n = \mathbf{F}_i \cdot \hat{n}_i = \begin{Bmatrix} \rho V_n \\ \rho u_{r,j} V_n + p \hat{n}_j \\ V_n (\rho E_r + p) \end{Bmatrix} = \mathbf{U} V_n + p \begin{Bmatrix} 0 \\ \hat{n} \\ V_n \end{Bmatrix} \quad \mathbf{U} = \begin{Bmatrix} \rho \\ \rho u_{r,j} \\ \rho E_r \end{Bmatrix} \quad (4.65)$$

The \mathbf{A} -matrix is calculated:

$$[\mathbf{A}] = \frac{\partial \mathbf{F}_n}{\partial \mathbf{U}} = [\mathbf{I}] V_n + \mathbf{U} \frac{\partial V_n}{\partial \mathbf{U}} + \frac{\partial p}{\partial \mathbf{U}} \begin{Bmatrix} 0 \\ \hat{n} \\ V_n \end{Bmatrix} + p \begin{Bmatrix} 0 \\ 0 \\ \partial V_n / \partial \mathbf{U} \end{Bmatrix} \quad (4.66)$$

From Eq. 3.130, the pressure derivative is evaluated:

$$\frac{\partial p}{\partial \mathbf{U}} = \left\{ \frac{\gamma-1}{2} (u_{r,i} u_{r,i} + V_{t,i} V_{t,i}) \quad (1-\gamma) u_{r,i} \quad \gamma-1 \right\} \quad (4.67)$$

The normal velocity can be written in terms of the unknowns:

$$V_n = \frac{\rho u_{r,i} \cdot \hat{n}_i}{\rho} \quad \frac{\partial V_n}{\partial \mathbf{U}} = \frac{1}{\rho} \{-V_n \quad \hat{n}_j \quad 0\} \quad (4.68)$$

The \mathbf{A} -matrix is written:

$$[\mathbf{A}] = \begin{bmatrix} 0 & \hat{n}_j & 0 \\ \frac{\gamma-1}{2}(u_{r,k}u_{r,k} + V_{t,k}V_{t,k})\hat{n}_i - u_{r,i}V_n & V_n\delta_{ij} + u_{r,i}\hat{n}_j + (1-\gamma)u_{r,j}\hat{n}_i & (\gamma-1)\hat{n}_i \\ \frac{\gamma-1}{2}(u_{r,k}u_{r,k} + V_{t,k}V_{t,k})V_n - H_rV_n & H_r\hat{n}_j + (1-\gamma)u_{r,j}V_n & \gamma V_n \end{bmatrix} \quad (4.69)$$

where i indicates the i^{th} momentum flux (rows) and j indicates the j^{th} velocity component (columns). The \mathbf{A} -matrix is calculated using the Roe-average properties (Toro, 1997):

$$u_{r,i}^{(R)} = \frac{u_{r,i,bc}\sqrt{\rho_{bc}} + u_{r,i,e}\sqrt{\rho_e}}{\sqrt{\rho_{bc}} + \sqrt{\rho_e}} \quad H_r^{(R)} = \frac{H_{r,bc}\sqrt{\rho_{bc}} + H_{r,e}\sqrt{\rho_e}}{\sqrt{\rho_{bc}} + \sqrt{\rho_e}} \quad (4.70)$$

$$K^{(R)} = \frac{K_{bc}\sqrt{\rho_{bc}} + K_e\sqrt{\rho_e}}{\sqrt{\rho_{bc}} + \sqrt{\rho_e}} \quad (4.71)$$

The speed of sound is calculated from the Roe-averaged properties:

$$(a^{(R)})^2 = (\gamma-1) \left(H_r^{(R)} - \frac{1}{2}u_{r,i}^{(R)}u_{r,i}^{(R)} + \frac{1}{2}V_{t,i}V_{t,i} - K^{(R)} \right) \quad (4.72)$$

Cowan (2003) uses the following algorithm that follows to reduce the computational time on Riemann invariants. This work adapts Cowan's algorithm by adding turbulent kinetic energy K to the total energy. The eigenvalues and unknowns step are calculated (Dubois, 1993):

$$\lambda_1 = |u_{r,i}^{(R)} \cdot \hat{n}_i + a^{(R)}| \quad \lambda_2 = |u_{r,i}^{(R)} \cdot \hat{n}_i - a^{(R)}| \quad \lambda_3 = |u_{r,i}^{(R)} \cdot \hat{n}_i| \quad (4.73)$$

$$\Delta\mathbf{U} = \begin{Bmatrix} \Delta\mathbf{U}_1 \\ \Delta\mathbf{U}_{j+1} \\ \Delta\mathbf{U}_E \end{Bmatrix} = \begin{Bmatrix} \rho_{bc} - \rho_e \\ \rho u_{r,j,bc} - \rho u_{r,j,e} \\ \rho E_{r,bc} - \rho E_{r,e} \end{Bmatrix} \quad (4.74)$$

$$\Delta\mathbf{U}_K = \rho K_{bc} - \rho K_e \quad (4.75)$$

The matrix assembly handled quickly by calculating a_1 , a_2 , c_1 , and c_2 . a_1 represents the pressure jump Δp , and a_2 represents the normal velocity jump $\Delta\rho V_n$:

$$a_1 = (\gamma - 1) \left(\frac{1}{2} (u_{r,i}^{(R)} u_{r,i}^{(R)} + V_{t,i} V_{t,i}) \Delta U_1 - u_{r,j}^{(R)} \Delta U_{j+1} + \Delta U_E - \Delta U_K \right) \quad (4.76)$$

$$a_2 = \hat{n}_j \Delta U_{j+1} - (u_{r,i}^{(R)} \cdot \hat{n}_i) \Delta U_1 \quad (4.77)$$

c_1 and c_2 represent the combination of the eigenvalues in a common and repeating form:

$$c_1 = \frac{1}{2} \frac{(\lambda_1 + \lambda_2 - 2\lambda_3)a_1}{(a^{(R)})^2} + \frac{1}{2} \frac{(\lambda_1 - \lambda_2)a_2}{a^{(R)}} \quad (4.78)$$

$$c_2 = \frac{1}{2} \frac{(\lambda_1 - \lambda_2)a_1}{a^{(R)}} + \frac{1}{2} (\lambda_1 + \lambda_2 - 2\lambda_3)a_2 \quad (4.79)$$

These values are used to calculate the corrected normal flux on the boundary:

$$\mathbf{F}_{inv,n,c} = \frac{1}{2} \left(\mathbf{F}_{inv,n,e} + \mathbf{F}_{inv,n,bc} - \lambda_3 \Delta \mathbf{U} - c_1 \begin{Bmatrix} 1 \\ u_{r,j}^{(R)} \\ H_r^{(R)} \end{Bmatrix} - c_2 \begin{Bmatrix} 0 \\ \hat{n}_j \\ u_{r,i}^{(R)} \cdot \hat{n}_i \end{Bmatrix} \right) \quad (4.80)$$

4.1.10.2 Viscous Boundary Fluxes

Riemann invariants are appropriate for inviscid fluxes, but viscous fluxes must be evaluated on the discontinuous boundary using conceptual understanding of the viscous boundary. Moffitt (2004) assumed that the viscous fluxes through the far field boundary were zero, which is the desired quantity in the far field. Moffitt found that when viscous fluxes (vortices and gradients) attempt to flow through the far field boundary, the equations become imbalanced and reflections are imparted into the domain. Neglecting the viscous fluxes, where the boundary condition suggests no viscous flux, creates a boundary with excessive impedance. To allow the viscous fluxes to pass through the boundary (lower impedance), the viscous fluxes are taken from the adjacent element within the domain. Using this method, the viscous integrals (domain and boundary) in the Galerkin equations are complete and

balanced by the current distribution of viscous fluxes. (The symmetry plane violates this rule as defined in the next chapter.)

4.1.10.3 Turbulent Fluxes

The turbulent equations have advection and diffusion terms, which are similar to the inviscid and viscous flux terms discussed above. The diffusion terms are modeled using the gradients from the adjacent element, like the viscous terms. The advection terms can be modeled by expanding the Riemann problem. The SA and k- ω equations are added to Eq. 4.52:

$$\frac{\partial}{\partial t} \begin{Bmatrix} \rho \\ u \\ p \\ \hat{v} \\ K \\ \omega \end{Bmatrix} + \begin{bmatrix} u & \rho & 0 & 0 & 0 & 0 \\ 0 & u & \rho^{-1} & 0 & 0 & 0 \\ 0 & \rho a^2 & u & 0 & 0 & 0 \\ 0 & 0 & 0 & u & 0 & 0 \\ 0 & 0 & 0 & 0 & u & 0 \\ 0 & 0 & 0 & 0 & 0 & u \end{bmatrix} \frac{\partial}{\partial x} \begin{Bmatrix} \rho \\ u \\ p \\ \hat{v} \\ K \\ \omega \end{Bmatrix} = 0 \quad (4.81)$$

The eigenvalues of \mathbf{A} represent the wave speed of the information traveling in the x -directions:

$$\begin{vmatrix} u - \lambda & \rho & 0 & 0 & 0 & 0 \\ 0 & u - \lambda & \rho^{-1} & 0 & 0 & 0 \\ 0 & \rho a^2 & u - \lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & u - \lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & u - \lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & u - \lambda \end{vmatrix} = 0 \quad (4.82)$$

The eigenvalues are $u - a$, u , u , u , u , and $u + a$, which adds three new real characteristics to the eigenvalues. The corresponding eigenvectors are:

$$\begin{pmatrix} \rho \\ -a \\ \rho a^2 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} \rho \\ a \\ \rho a^2 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad (4.83)$$

which shows that the original eigenvalues and eigenvectors of the invariant matrix are unchanged by the addition of the turbulent equations. The turbulent properties advect downstream at the local flow velocity u . Using Eq. 4.80 as a pattern, three new correction equations are created to adapt the boundary normal flux of turbulent quantities across a discontinuous boundary:

$$\begin{pmatrix} F_{\hat{v},n,c} \\ F_{K,n,c} \\ F_{\omega,n,c} \end{pmatrix} = \frac{1}{2} \left(\begin{pmatrix} \rho \hat{v}_e V_{n,e} \\ \rho K_e V_{n,e} \\ \rho \omega_e V_{n,e} \end{pmatrix} + \begin{pmatrix} \rho \hat{v}_{bc} V_{n,bc} \\ \rho K_{bc} V_{n,bc} \\ \rho \omega_{bc} V_{n,bc} \end{pmatrix} - |V_n| \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{pmatrix} \rho \hat{v}_e \\ \rho K_e \\ \rho \omega_e \end{pmatrix} - \begin{pmatrix} \rho \hat{v}_{bc} \\ \rho K_{bc} \\ \rho \omega_{bc} \end{pmatrix} \right) \right) \quad (4.84)$$

The K -correction equation can be rearranged:

$$F_{K,n,c} = \frac{1}{2} (\rho K_e (V_{n,e} - |V_n|) + \rho K_{bc} (V_{n,bc} + |V_n|)) \quad (4.85)$$

showing that two conditions exist: (1) When the fluid is flowing into the domain, the turbulent kinetic energy K along the boundary is taken from the boundary condition. (2) When the fluid is flowing out of the domain, the local element distribution is used to define K . The flow direction can be determined by the element side of the discontinuity to avoid using Roe-averaging:

$$F_{K,n,c} = \begin{cases} \rho K_{bc} V_{n,bc} & \text{if } V_{n,e} \geq 0 \\ \rho K_e V_{n,e} & \text{if } V_{n,e} < 0 \end{cases} \quad (4.86)$$

The other two turbulent variables have similar transport:

$$F_{\hat{v},n,c} = \begin{cases} \rho \hat{v}_{bc} V_{n,bc} & \text{if } V_{n,e} \geq 0 \\ \rho \hat{v}_e V_{n,e} & \text{if } V_{n,e} < 0 \end{cases} \quad F_{\omega,n,c} = \begin{cases} \rho \omega_{bc} V_{n,bc} & \text{if } V_{n,e} \geq 0 \\ \rho \omega_e V_{n,e} & \text{if } V_{n,e} < 0 \end{cases} \quad (4.87)$$

4.1.11 Boundary Conditions

Burnett (1987) classifies boundary conditions by their implementation: Unconstrained boundary conditions correspond to boundary terms that appear in the finite element system, usually introduced because of the use of integration by parts. The condition is substituted into the governing equations in place of domain variables. Unconstrained conditions are satisfied approximately at best. Constrained boundary conditions have no boundary terms but are applied directly using a constraint equation. Constraint equations are satisfied exactly.

The use of Gauss's theorem in the finite element method changes the application of boundary conditions. If integration by parts is not used in a system of order $2m$, all of the boundaries will require constrained conditions. If integration by parts is introduced m times into the same system, the essential conditions must be constrained while the natural conditions can be unconstrained. If the system is integrated again m times by parts, then all of the boundary conditions can be specified using unconstrained methods. Of course, constrained methods can always be used in place of unconstrained, and constraint equations can be reposed in an implicit form and integrated into the governing equations. By the mid-1980s, the majority of finite element CFD solvers were using integration by parts once in their development. This trend has carried on into the present, where natural conditions are regularly applied through boundary integrals, and essential conditions are applied explicitly to the properties.

Cowan (2003) applied three conditions through the flux in the boundary integral: Inviscid wall, symmetry plane, and far field. This work adds four new conditions: Viscous walls, rocket exhaust, turbojet inflow, and turbojet exhaust planes. The normal flux is calculated:

$$(\mathbf{F}_{i,be} - \mathbf{F}_{V,i,be}) \cdot \hat{n}_i = \begin{pmatrix} \rho V_n \\ \rho V_n u_{r,j} + p \hat{n}_j \\ V_n \rho H_r \\ V_n \rho \hat{v} \\ V_n \rho K \\ V_n \rho \omega \end{pmatrix} \begin{pmatrix} 0 \\ (\tau_{ij} + \rho \Gamma_{ij}) \cdot \hat{n}_i \\ u_{r,j} (\tau_{ij} + \rho \Gamma_{ij}) \cdot \hat{n}_i - q_n'' - Q_n + \frac{1}{\text{Re}} (\mu + \sigma_k \mu_T) \nabla_n K \\ \frac{1}{\sigma \text{Re}} (\mu + \rho \hat{v}) \nabla_n \hat{v} \\ \frac{1}{\text{Re}} (\mu + \sigma_k \mu_T) \nabla_n K \\ \frac{1}{\text{Re}} (\mu + \sigma_\omega \mu_T) \nabla_n \omega \end{pmatrix} \quad (4.88)$$

where V_n is the normal velocity ($V_n = u_{r,i} \cdot \hat{n}_i$), ∇_n is the normal gradient ($\nabla_n = \nabla_i \cdot \hat{n}_i$), and q_n'' and Q_n are the normal heat fluxes ($q_n'' = q_i'' \cdot \hat{n}_i$ and $Q_n = Q_i \cdot \hat{n}_i$).

4.1.11.1 Far Field

The far field boundary applies the freestream conditions to the flow, while maintaining an appropriate level of fidelity with the local flow. The local and freestream properties are combined using Riemann invariants, which introduce the correct amount of upstream and downstream influence to calculate the inviscid normal flux. The Riemann corrected normal flux is applied directly in the boundary integrals. The viscous heat fluxes are taken from the adjacent element. The normal flux is calculated using Eqs. 4.63, 4.80, 4.86, and 4.87:

$$(\mathbf{F}_{i,be} - \mathbf{F}_{V,i,be}) \cdot \hat{n}_{i,be} = \frac{\mathbf{F}_{inv,n,e} + \mathbf{F}_{inv,n,\infty}}{2} - \frac{1}{2} [\mathbf{A}^*] (\mathbf{U}_e - \mathbf{U}_\infty) - \bar{\mathbf{F}}_{V,i,e} \cdot \hat{n}_{i,be} \quad (4.89)$$

4.1.11.2 Inviscid Wall

The inviscid wall boundary condition restricts the flow at each node to be tangent to the wall. Unlike the symmetry plane, the solid wall cannot be implied through the boundary integral.

The integral is not strong enough to force the flow to become tangent to the boundary.

Instead, flow tangency is used to constrain the normal velocity along the solid wall:

$$u_{r,i} = u'_{r,i} - (u'_{r,i} \cdot \hat{n}_i - V_{b,i} \cdot \hat{n}_i) \hat{n}_i \quad (4.90)$$

where $u'_{r,i}$ is the velocity along the wall before correction and $V_{b,i}$ is the velocity of a moving wall. Flow tangency is not applied on an element-by-element basis. If the condition was applied by elements, the velocity vector would only be tangent to the latest element applied to that node. In fact, a curved boundary is represented by many flat boundary faces so that the velocity would be removed along the entire wall. The normal component on one face would be removed from that element's nodes and then its neighbors would remove the remaining velocity as velocity normal to their faces. Instead, the flow tangency condition is applied at the nodes using an area-weighted normal vector, shown for two- and three- dimensions on left and right, respectively:

$$\hat{n}_{iwl} = \frac{\sum_{be} l_{be} \hat{n}_{be}}{\sum_{be} l_{be}} \quad \hat{n}_{iwl} = \frac{\sum_{be} A_{be} \hat{n}_{be}}{\sum_{be} A_{be}} \quad (4.91)$$

The boundary integrals along the solid wall are evaluated as the flux (properties) stand, to maintain conservation of mass, momentum, and energy. No assumptions are made to the viscous stresses or heat flux along the inviscid wall. This process allows some flow to leak through the wall at the center of the elements due to the linear Interpolation. The flow is forced to match the boundary curvature at the nodes, where the accuracy is most desirable.

The normals along the trailing edge of an airfoil are shown in Figure 4.6. The boundary element normals are shown in blue, and the normals at the nodes are shown in green. The normal at the trailing edge is averaged from the adjacent elements, which do not necessarily

have the same area. The TE normal in Figure 4.6 is inclined slightly due to this imbalance in areas. According to the Kutta condition (Anderson, 2001), the flow should leave smoothly at the trailing edge. Flow tangency trims off all of the velocity, creating a stagnated region in the inviscid solution. Cowan allows for such points through *singular points*, where the flow tangency condition is not applied. Singular nodes are also applied in the viscous codes. The number of viscous singular nodes is tracked to help parse the data. The no-slip condition is unaffected by the accuracy of the wall normal.

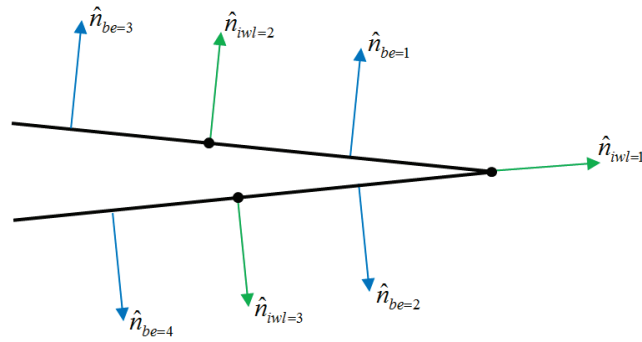


Figure 4.6: Normals along Trailing Edge of an Airfoil.

The kinetic energy is reduced by applying Eq. 4.90, and the loss in energy is reflected in the ideal gas equation (Eq. 3.131). If the other properties are not adapted to this loss in kinetic energy, then the pressure will increase along the wall. We desire to maintain the original density, pressure, and temperature along the wall while applying Eq. 4.90. The internal energy e is proportional to the temperature T (Eq. 3.23). The new kinetic energy (Eq. 4.92) must be reflected in the total energy (Eq. 4.93). The total energy is the sum of energies (Eq. 3.120), so the total energy and enthalpy are adapted by the change created by Eq. 4.94.

$$u_{r,i}u_{r,i} = u'_{r,i}u'_{r,i} - (u'_{r,i}n_i)^2 \quad (4.92)$$

$$c_v T = e = E_r - \frac{1}{2}(u_{r,i}u_{r,i} - V_{t,i}V_{t,i}) - K = E'_r - \frac{1}{2}(u'_{r,i}u'_{r,i} - V_{t,i}V_{t,i}) - K \quad (4.93)$$

$$\rho E_r = \rho E'_r - \frac{1}{2}\rho(u'_{r,i}n_i)^2 \quad \rho H_r = \rho H'_r - \frac{1}{2}\rho(u'_{r,i}n_i)^2 \quad (4.94)$$

4.1.11.3 Viscous Wall

The viscous wall creates a solid boundary in the flow. Like the inviscid wall, the fluid cannot pass through the wall, and the fluid sticks to the wall. The no-slip condition (Eq. 4.95) is used to constrain the velocity to this essential condition.

$$u_{r,i} = V_{b,i} \quad (4.95)$$

The total energy and enthalpy must again be adapted to maintain the thermodynamic properties. Eqs. 3.120 and 3.121 are adapted to the new kinetic energy created by the no-slip condition:

$$\rho E = \rho E'_r - \frac{1}{2}\rho u'_{r,i}u'_{r,i} \quad \rho H = \rho H'_r - \frac{1}{2}\rho u'_{r,i}u'_{r,i} \quad (4.96)$$

Heat transfer conditions along the viscous solid wall are specified by one of three conditions: (1) Known temperature, (2) known heat flux, and (3) adiabatic wall. The adiabatic wall is a special case of the second condition, where the specified heat flux is zero. The temperature is not stored in the solution; instead total enthalpy is tracked at the domain nodes. The static enthalpy is specified along the wall (Eq. 4.97) and used to calculate total enthalpy.

$$h_w = c_p T_w \quad \rho H = \rho(h_w + \frac{1}{2}V_{b,i}V_{b,i}) \quad (4.97)$$

The heat flux is either specified by the user (types 2 or 3) or calculated on the adjacent element (type 1) by combining Eqs. 3.222, 4.16, and 4.17. The heat flux is implied in boundary integrals:

$$q'' = q_i'' \cdot n_i = -\frac{\mu}{\text{Pr Re}} \frac{\partial h}{\partial x_j} \cdot n_i \quad (4.98)$$

The no-slip condition also drives the turbulent oscillations to zero at the wall. From this, the Reynolds stresses ρT_{ij} and turbulent kinetic energy ρK are mathematically zero at the no-slip wall. The Reynolds stresses are calculated using Bousinesq's approximation (Eq. 3.221). The eddy viscosity is driven to zero at the wall to make the Reynolds stresses, turbulent transport of heat, and other Reynolds terms identically zero at the wall. If the normal velocity is driven to zero (no-slip) and the turbulent conditions are applied, the boundary normal flux is calculated:

$$(\mathbf{F}_{i,be} - \mathbf{F}_{V,i,be}) \cdot \hat{n}_i = \begin{pmatrix} 0 \\ p \delta_{ij} \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ \tau_{ij} \cdot \hat{n}_i \\ u_{r,j} \tau_{ij} \cdot \hat{n}_i - q_n'' + \frac{1}{\text{Re}} \mu \nabla_n K \\ \frac{1}{\sigma \text{Re}} \mu \nabla_n \hat{v} \\ \frac{1}{\text{Re}} \mu \nabla_n K \\ \frac{1}{\text{Re}} \mu \nabla_n \omega \end{pmatrix} \quad (4.99)$$

Boundary condition has already been discussed for w (Eq. 3.253). The SA variable must be driven to zero at the wall because \hat{v} is proportional to eddy viscosity (Eq. 3.228).

Constant Density Corner. Problems occur if a single element spans a no-slip corner, such that all but one of its sides create the no-slip boundaries. The density at the corner node becomes constant. The continuity residual is discretized according to Eq. 4.36:

$$\mathbf{R}_\rho = \Delta t_n \mathbf{M}_c \frac{\partial \rho_e}{\partial t} - \Delta t_n \sum_e \int_{A_e} \frac{\partial \Phi_e^T}{\partial x_i} \rho u_i d\Omega + \Delta t_n \sum_{be} \int_{l_{be}} \Phi_e^T \rho u_i \cdot n_i d\Gamma \quad (4.100)$$

Contributions to the corner node come from the corner nodes of this element alone. All of the nodes exist on the no-slip wall, where the velocities are identically zero, so that the

velocity on the entire element is zero. From this, the domain and boundary integrals are zero for the corner element. The unsteady term is neglected in steady solutions so the density residual for the corner node is always zero. For unsteady solutions, the unsteady term weakly connects nodes through the mass matrix, and the corner node density is no longer time-accurate.

This issue is not commonly seen in the literature, yet many triangular and tetrahedral meshing packages have the option to split corner elements. The corner element can be split into multiple elements so that each has only one no-slip edge. This method requires an extra processing step between mesh generation and CFD solution.

4.1.11.4 Symmetry Plane

The velocity normal to a symmetry element is assumed to be zero in order to satisfy the symmetry condition. In fact, any fluxes or derivatives normal to the symmetry element are assumed to be zero to satisfy the symmetry condition. The symmetry boundary condition is applied implicitly through the boundary integral. If the normal velocity and normal gradients are removed from the flux, the normal flux is calculated using the pressure and stresses:

$$(\mathbf{F}_{i,be} - \mathbf{F}_{V,i,be}) \cdot \hat{n}_i = \begin{Bmatrix} 0 \\ p\hat{n}_j \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix} - \begin{Bmatrix} 0 \\ (\tau_{ij} + \rho\mathbf{T}_{ij}) \cdot \hat{n}_i \\ u_{r,j} (\tau_{ij} + \rho\mathbf{T}_{ij}) \cdot \hat{n}_i \\ 0 \\ 0 \\ 0 \end{Bmatrix} \quad (4.101)$$

The viscous and Reynolds stress must be calculated in the presence of the zero normal velocity and zero normal gradient assumptions. To accomplish this, the velocities and gradients are written in the normal-tangential coordinate system:

$$\vec{V}_{nt} = \begin{Bmatrix} V_n \\ V_t \\ V_b \end{Bmatrix} = \begin{Bmatrix} un_x + vn_y + wn_z \\ ut_x + vt_y + wt_z \\ ub_x + vb_y + wb_z \end{Bmatrix} = \begin{bmatrix} n_x & n_y & n_z \\ t_x & t_y & t_z \\ b_x & b_y & b_z \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = [\theta]^T \vec{V} \quad (4.102)$$

$$\nabla_{nt} = \begin{Bmatrix} \frac{\partial}{\partial \hat{n}} \\ \frac{\partial}{\partial \hat{t}} \\ \frac{\partial}{\partial \hat{b}} \end{Bmatrix} = \begin{Bmatrix} \hat{n} \cdot \frac{\partial}{\partial \vec{x}} \\ \hat{t} \cdot \frac{\partial}{\partial \vec{x}} \\ \hat{b} \cdot \frac{\partial}{\partial \vec{x}} \end{Bmatrix} = \begin{Bmatrix} \hat{n} \\ \hat{t} \\ \hat{b} \end{Bmatrix} \cdot \frac{\partial}{\partial \vec{x}} = [\theta]^T \nabla \quad (4.103)$$

The inverse relationships can also be formulated:

$$\vec{V} = [\theta]^T \vec{V}_{nt} = [\theta] \vec{V}_{nt} \quad \nabla = [\theta]^{-T} \nabla_{nt} = [\theta] \nabla_{nt} \quad (4.104)$$

The normal velocity and gradient can be removed from the velocity gradient in the normal-tangential coordinate system and then rotated the tensor back into the global frame using:

$$\nabla \vec{V}^T = [\theta] \nabla_n ([\theta] \vec{V}_n)^T = [\theta] \nabla_n \vec{V}_n^T [\theta]^T \quad (4.105)$$

The divergence of velocity is independent of the reference:

$$\nabla^T \vec{V} = ([\theta] \nabla_n)^T [\theta] \vec{V}_n = \nabla_n^T [\theta]^T [\theta] \vec{V}_n = \nabla_n^T \vec{V}_n \quad (4.106)$$

Combining the previous transformations, the stress in the global frame is calculated the velocities and gradients in the normal-tangential frame:

$$[\vec{\tau}] = \frac{\mu}{\text{Re}} \left(\nabla \vec{V}^T + (\nabla \vec{V}^T)^T + \lambda \nabla^T \vec{V} [\mathbf{I}] \right) = \frac{\mu}{\text{Re}} \left([\theta] \left(\nabla_n \vec{V}_n^T + (\nabla_n \vec{V}_n^T)^T \right) [\theta]^T + \lambda \nabla_n^T \vec{V}_n [\mathbf{I}] \right) \quad (4.107)$$

The exchange of momentum through the boundary element is calculated by dotting the stress tensor with the normal vector. Using Eq. 4.102 for the definition of $[\theta]^T$, the dot product is:

$$[\vec{\tau}] \hat{n} = \frac{\mu}{\text{Re}} \left([\theta] \left(\nabla_n \vec{V}_n^T + (\nabla_n \vec{V}_n^T)^T \right) \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} + \lambda \nabla_n^T \vec{V}_n \hat{n} \right) \quad (4.108)$$

2D Symmetry Element. In two-dimensions, the viscous stress simplifies:

$$\nabla_n \vec{V}_n^T + (\nabla_n \vec{V}_n^T)^T = \begin{bmatrix} 2 \frac{\partial V_n}{\partial \hat{n}} & \frac{\partial V_n}{\partial \hat{t}} + \frac{\partial V_t}{\partial \hat{n}} \\ \frac{\partial V_n}{\partial \hat{t}} + \frac{\partial V_t}{\partial \hat{n}} & 2 \frac{\partial V_t}{\partial \hat{t}} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 2 \frac{\partial V_t}{\partial \hat{t}} \end{bmatrix} \quad (4.109)$$

$$\left(\nabla_n \vec{V}_n^T + (\nabla_n \vec{V}_n^T)^T \right) \begin{Bmatrix} 1 \\ 0 \end{Bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 2 \frac{\partial V_t}{\partial \hat{t}} \end{bmatrix} \begin{Bmatrix} 1 \\ 0 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \quad (4.110)$$

$$[\vec{\tau}] \hat{n} = \frac{\mu}{\text{Re}} \lambda \frac{\partial V_t}{\partial \hat{t}} \begin{Bmatrix} n_x \\ n_y \end{Bmatrix} \quad \tau_{ij} \hat{n}_i = \frac{\mu}{\text{Re}} \lambda \frac{\partial V_t}{\partial \hat{t}} \hat{n}_j \quad (4.111)$$

The tangential derivative can be approximated along the boundary edge:

$$\frac{\partial V_t}{\partial \hat{t}} = \frac{V_{t2} - V_{t1}}{\ell_{21}} \quad (4.112)$$

and the tangential velocity can be calculated:

$$V_t = ut_x + vt_y = un_y - vn_x \quad (4.113)$$

3D Symmetry Element. In three-dimensions, the viscous stress simplifies:

$$\nabla_n \vec{V}_n^T + (\nabla_n \vec{V}_n^T)^T = \begin{bmatrix} 2 \frac{\partial V_n}{\partial \hat{n}} & \frac{\partial V_n}{\partial \hat{t}} + \frac{\partial V_t}{\partial \hat{n}} & \frac{\partial V_n}{\partial \hat{b}} + \frac{\partial V_b}{\partial \hat{n}} \\ \frac{\partial V_n}{\partial \hat{t}} + \frac{\partial V_t}{\partial \hat{n}} & 2 \frac{\partial V_t}{\partial \hat{t}} & \frac{\partial V_t}{\partial \hat{b}} + \frac{\partial V_b}{\partial \hat{t}} \\ \frac{\partial V_n}{\partial \hat{b}} + \frac{\partial V_b}{\partial \hat{n}} & \frac{\partial V_t}{\partial \hat{b}} + \frac{\partial V_b}{\partial \hat{t}} & 2 \frac{\partial V_b}{\partial \hat{b}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 \frac{\partial V_t}{\partial \hat{t}} & \frac{\partial V_t}{\partial \hat{b}} + \frac{\partial V_b}{\partial \hat{t}} \\ 0 & \frac{\partial V_t}{\partial \hat{b}} + \frac{\partial V_b}{\partial \hat{t}} & 2 \frac{\partial V_b}{\partial \hat{b}} \end{bmatrix} \quad (4.114)$$

$$\left(\nabla_n \vec{V}_n^T + (\nabla_n \vec{V}_n^T)^T \right) \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 \frac{\partial V_t}{\partial \hat{t}} & \frac{\partial V_t}{\partial \hat{b}} + \frac{\partial V_b}{\partial \hat{t}} \\ 0 & \frac{\partial V_t}{\partial \hat{b}} + \frac{\partial V_b}{\partial \hat{t}} & 2 \frac{\partial V_b}{\partial \hat{b}} \end{bmatrix} \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix} \quad (4.115)$$

$$[\vec{\tau}] \hat{n} = \frac{\mu}{\text{Re}} \lambda \left(\frac{\partial V_t}{\partial \hat{t}} + \frac{\partial V_b}{\partial \hat{b}} \right) \begin{Bmatrix} n_x \\ n_y \\ n_z \end{Bmatrix} \quad \tau_{ij} \hat{n}_i = \frac{\mu}{\text{Re}} \lambda \left(\frac{\partial V_t}{\partial \hat{t}} + \frac{\partial V_b}{\partial \hat{b}} \right) \hat{n}_j \quad (4.116)$$

The tangent vector can be calculated using two of the nodes on the boundary element:

$$\hat{t} = \frac{\vec{t}}{|\vec{t}|} = \frac{\vec{x}_{13}}{\ell_{13}} \quad \ell_{13} = \sqrt{\vec{x}_{13} \cdot \vec{x}_{13}} = \sqrt{\ell_{13x}^2 + \ell_{13y}^2 + \ell_{13z}^2} \quad (4.117)$$

The binormal vector is calculated from the cross product of normal and tangent vectors:

$$\hat{b} = \hat{n} \times \hat{t} = \hat{n} \times \frac{\vec{x}_{13}}{\ell_{13}} = \frac{1}{\ell_{13}} \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ n_x & n_y & n_z \\ \ell_{13x} & \ell_{13y} & \ell_{13z} \end{vmatrix} = \frac{1}{\ell_{13}} \begin{Bmatrix} \ell_{13z}n_y - \ell_{13y}n_z \\ \ell_{13x}n_z - \ell_{13z}n_x \\ \ell_{13y}n_x - \ell_{13x}n_y \end{Bmatrix} \quad (4.118)$$

The velocity and position vectors can be written in terms of the tangent-binormal directions, where the problem can be simplified to a two-dimensional problem:

$$\begin{Bmatrix} V_{t,i} \\ V_{b,i} \end{Bmatrix} = \begin{Bmatrix} u_i t_x + v_i t_y + w_i t_z \\ u_i b_x + v_i b_y + w_i b_z \end{Bmatrix} = \begin{bmatrix} t_x & t_y & t_z \\ b_x & b_y & b_z \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ w_i \end{Bmatrix} \quad \begin{Bmatrix} x_{t,i} \\ x_{b,i} \end{Bmatrix} = \begin{Bmatrix} \hat{t} \cdot \vec{x}_{i3} \\ \hat{b} \cdot \vec{x}_{i3} \end{Bmatrix} \quad (4.119)$$

$$\begin{Bmatrix} x_{t,1} \\ x_{b,1} \end{Bmatrix} = \begin{Bmatrix} \frac{\vec{x}_{13}}{\ell_{13}} \cdot \vec{x}_{13} \\ \hat{b} \cdot \vec{x}_{13} \end{Bmatrix} = \begin{Bmatrix} \ell_{13} \\ 0 \end{Bmatrix} \quad \begin{Bmatrix} x_{t,2} \\ x_{b,2} \end{Bmatrix} = \begin{Bmatrix} \hat{t} \cdot \vec{x}_{23} \\ \hat{b} \cdot \vec{x}_{23} \end{Bmatrix} \quad \begin{Bmatrix} x_{t,3} \\ x_{b,3} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \quad (4.120)$$

The local gradients are calculated by treating the boundary element (in tangent-binormal coordinates) like a two-dimensional element:

$$\begin{Bmatrix} x_t \\ x_b \end{Bmatrix} = \begin{bmatrix} x_{t,1} - x_{t,3} & x_{t,2} - x_{t,3} \\ x_{b,1} - x_{b,3} & x_{b,2} - x_{b,3} \end{bmatrix} \begin{Bmatrix} \xi_1 \\ \xi_2 \end{Bmatrix} + \begin{Bmatrix} x_{t,3} \\ x_{b,3} \end{Bmatrix} = \begin{bmatrix} x_{t,1} & x_{t,2} \\ x_{b,1} & x_{b,2} \end{bmatrix} \begin{Bmatrix} \xi_1 \\ \xi_2 \end{Bmatrix} + \begin{Bmatrix} x_{t,3} \\ x_{b,3} \end{Bmatrix} \quad (4.121)$$

$$\begin{Bmatrix} \frac{\partial}{\partial \xi_1} \\ \frac{\partial}{\partial \xi_2} \end{Bmatrix} = \begin{bmatrix} \frac{\partial x_t}{\partial \xi_1} & \frac{\partial x_b}{\partial \xi_1} \\ \frac{\partial x_t}{\partial \xi_2} & \frac{\partial x_b}{\partial \xi_2} \end{bmatrix} \begin{Bmatrix} \frac{\partial}{\partial x_t} \\ \frac{\partial}{\partial x_b} \end{Bmatrix} = \begin{bmatrix} x_{t,1} & x_{b,1} \\ x_{t,2} & x_{b,2} \end{bmatrix} \begin{Bmatrix} \frac{\partial}{\partial x_t} \\ \frac{\partial}{\partial x_b} \end{Bmatrix} \quad (4.122)$$

$$\begin{Bmatrix} \frac{\partial}{\partial x_t} \\ \frac{\partial}{\partial x_b} \end{Bmatrix} = \frac{1}{|J_{tb}|} \begin{bmatrix} x_{b,2} & -x_{b,1} \\ -x_{t,2} & x_{t,1} \end{bmatrix} \begin{Bmatrix} \frac{\partial}{\partial \xi_1} \\ \frac{\partial}{\partial \xi_2} \end{Bmatrix} \quad (4.123)$$

$$|J_{tb}| = x_{t,1}x_{b,2} - x_{b,1}x_{t,2} = \ell_{13}x_{b,2} - (0)x_{t,2} = \ell_{13}x_{b,2} \quad (4.124)$$

$$\begin{Bmatrix} \frac{\partial}{\partial x_t} \\ \frac{\partial}{\partial x_b} \end{Bmatrix} = \frac{1}{\ell_{13} x_{b,2}} \begin{bmatrix} x_{b,2} & 0 \\ -x_{t,2} & \ell_{13} \end{bmatrix} \begin{Bmatrix} \frac{\partial}{\partial \xi_1} \\ \frac{\partial}{\partial \xi_2} \end{Bmatrix} \quad (4.125)$$

$$\frac{\partial V_t}{\partial \hat{t}} = \frac{\partial V_t}{\partial x_t} = \frac{1}{\ell_{13} x_{b,2}} x_{b,2} \frac{\partial V_t}{\partial \xi_1} = \frac{V_{t,1} - V_{t,3}}{\ell_{13}} \quad (4.126)$$

$$\frac{\partial V_b}{\partial \hat{b}} = \frac{\partial V_b}{\partial x_b} = \frac{1}{\ell_{13} x_{b,2}} \left(\ell_{13} \frac{\partial V_b}{\partial \xi_2} - x_{t,2} \frac{\partial V_b}{\partial \xi_1} \right) = \frac{1}{x_{b,2}} \left(V_{b,2} - V_{b,3} - x_{t,2} \frac{V_{b,1} - V_{b,3}}{\ell_{13}} \right) \quad (4.127)$$

The divergence of velocity is calculated:

$$\frac{\partial V_t}{\partial \hat{t}} + \frac{\partial V_b}{\partial \hat{b}} = \frac{V_{t,1} - V_{t,3}}{\ell_{13}} + \frac{1}{x_{b,2}} \left(V_{b,2} - V_{b,3} - x_{t,2} \frac{V_{b,1} - V_{b,3}}{\ell_{13}} \right) \quad (4.128)$$

4.1.11.5 Rocket Exhaust

The rocket boundary condition defines a total pressure p_t and total enthalpy H at the outflow plane. A third piece of information – static pressure p – is necessary to determine all of the properties at the outflow plane. The static pressure is taken from the adjacent element and not the boundary condition, which gives the boundary condition adaptability. As the static pressure increases, the velocity decreases to maintain the total pressure. Isentropic relationships are used to obtain the chamber Mach number and static temperature (internal enthalpy):

$$\frac{T_t}{T} = \left(\frac{p_t}{p} \right)^{\frac{\gamma-1}{\gamma}} \quad h = \frac{H}{T_t/T} \quad (4.129)$$

$$M = \sqrt{\frac{2}{\gamma-1} \left(\frac{T_t}{T} - 1 \right) - M_T^2 + \frac{V_{t,i} V_{t,i}}{a^2}} \quad M_T^2 = \frac{2K}{a^2} \quad (4.130)$$

The remaining properties are calculated using thermodynamic relationships:

$$e = \frac{h}{\gamma} \quad \rho = \frac{p}{(\gamma-1)e} \quad a = \sqrt{\gamma \frac{p}{\rho}} \quad (4.131)$$

The local velocity is calculated element by element using the normal and mesh velocities:

$$V_n = M a \quad u_{r,i} = V_{t,i} - V_n n_i \quad (4.132)$$

The inviscid boundary integrals are assembled using Riemann invariants to ensure that the appropriate characteristics are present in the flow along the boundary. The viscous fluxes are calculated using the stress tensor, heat flux, and gradients from the adjacent element:

$$(\mathbf{F}_{i,be} - \mathbf{F}_{V,i,be}) \cdot \hat{n}_{i,be} = \frac{\mathbf{F}_{inv,n,e} + \mathbf{F}_{inv,n,rckt}}{2} - \frac{1}{2} [\mathbf{A}^*] (\mathbf{U}_e - \mathbf{U}_{rckt}) - \bar{\mathbf{F}}_{V,i,e} \cdot \hat{n}_{i,be} \quad (4.133)$$

Initial testing showed that the growth of total properties at the rocket boundary needed to be limited for solution stability. A large jump in properties near the wall (from total properties to initial conditions) created a traveling wave (shown in the figure below). The wave travels downstream striking the nozzle. The nozzle returns part of the wave back upstream, and the solution diverges within the iteration that the wave strikes the rocket boundary.

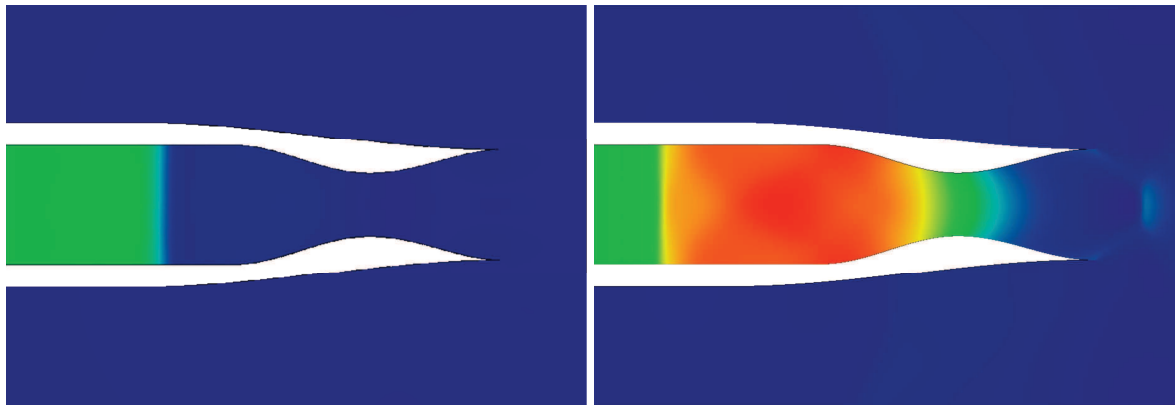


Figure 4.7: Traveling Waves within Rocket Combustion Chamber.

The method was adapted by increasing the total properties from the initial conditions over a series of iterations (Eq. 4.134). The ramp-up keeps the properties from coalescing into a strong traveling wave. Instead, small traveling waves, seen as chamber noise, bounce around

inside of the combustion chamber and are dissipated by the artificial viscosity. The new method showed that the total properties could be varied over 3000 iterations to arrive at a stable solution or over 10,000 iterations to minimize the acoustic responses in the chamber. The later is on the order of the time to converge the entire solution, and the solution converges progressively as the engine “starts up”. Real rockets have a finite time over which the pressure and temperature in the combustion chamber increases to its running properties.

$$p_t = p_{t,IC} + (p_{t,spec} - p_{t,IC}) \frac{istp}{N_{rstp}} \quad H = H_{IC} + (H_{spec} - H_{IC}) \frac{istp}{N_{rstp}} \quad (4.134)$$

Figure 4.8 shows the progressive growth of pressure in the combustion chamber. Figure 4.9 shows the distribution of pressure and Mach number within and downstream from the nozzle.

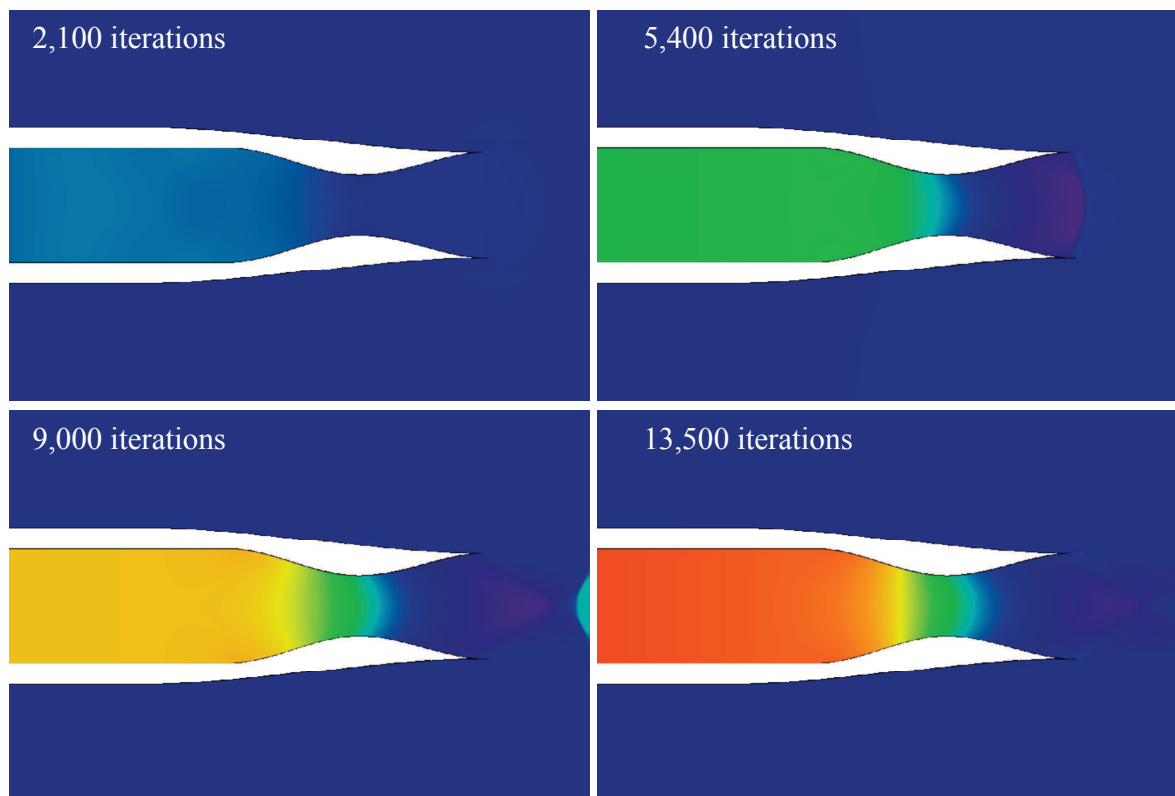


Figure 4.8: Progressive Growth of Pressure within Rocket Combustion Chamber.

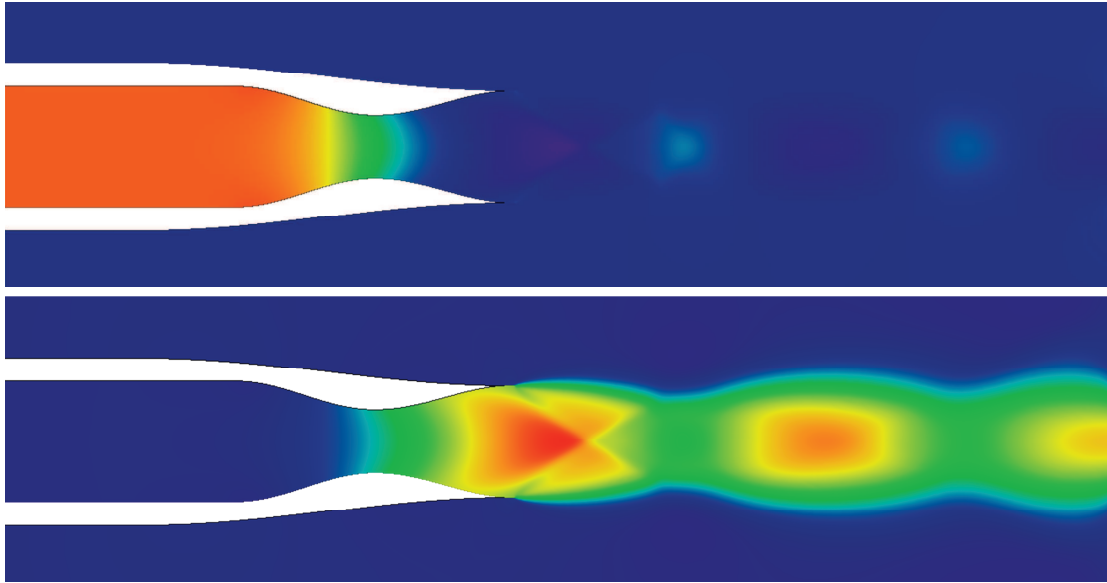


Figure 4.9: Rocket Engine – Pressure (Top) and Mach Number (Bottom) Distributions.

4.1.11.6 Turbojet Engine Planes

Engine boundary conditions simulate the plane upstream of the compressor (or fan) and the plane downstream of the turbine (or fan). The two planes remove any rotating mechanisms from the aerodynamics model while still adding the essential mass, momentum, and enthalpy. The engine is defined by the fuel flow, uninstalled thrust (without inlet or nozzle), and enthalpy production.

Inflow Plane. The turbojets are spun up using starter motors, but once running the turbine powers the compressor and fan to pull in a given mass flow rate. This mass flow rate varies depending on the altitude and load on the engine, but the mass flow rate does not vary much about on-design conditions (Mattingly, 1996) (e.g., perturbation in altitude and speed in normal flight). This mass flow rate is pulled through the engine by the pressure gradient in the inlet.

At the design speed (set by the throttle), the engine pulls a stream of air through the engine equivalent in diameter to the engine inlet. This air path is called the *stream tube*. For speeds lower than the design speed, the stream tube broadens to pull in more mass. When the speed increases beyond the design speed, the mass flow rate exceeds the capacity of the engine and the pressure increases at the inlet. Some of the flow is forced to divert around the edges of the inlet, called *spillover*. This decreases the size of the stream tube. At supersonic speeds the pressure is transferred upstream through the boundary layer so that spillover still occurs.

The mass flow is thus controlled by the pressure. The turbojet inflow boundary controls the mass flow rate through the engine using the pressure, or more accurately the momentum flux: $\rho V_n^2 + p$. The energy flux through the inlet plane will reflect the natural flow. The initial pressure at the inflow plane is calculated by averaging the pressure over the plane from the initial conditions or previous solution. The mass flow is controlled by changing the static inflow pressure to adapt the mass flow, as necessary to match the design condition.

The first pressure controller was constructed using an IF statement on the mass flow rate. The rate of change in pressure was constant and *a priori*, and the stability of the inflow plane was governed by the magnitude of the rate. Figure 4.10 shows the controller phase diagram.

Figure 4.10 shows that a linear scheme can be used to control the mass flow rate:

$$p^{(i+1)} = p^{(i)} + k \left(\frac{\dot{m}^{(i)}}{\dot{m}_D} - 1 \right) \quad (4.135)$$

Subsonic inlets react quite well to the linear controller and inflow boundary condition.

Supersonic inlets require a normal shock to slow the flow down to subsonic speeds before entering the compressor (and fan) section. The normal shock does not change mass flow

rate, so two solutions provide the same mass flow rate through the engine: One case corresponding to a subsonic compression and the other a supersonic expansion. Subsonic flow into the turbomachinery is desired in all cases. Subsonic speeds can be achieved through higher boundary pressures, ensuring the normal shock. To maintain higher pressures, the initial control pressure $p^{(0)}$ is set equal to the freestream total pressure, which will create a zero mass flow rate. A gain of 0.0004 provides an appropriate level of damping in the system to create a stable solution. Tests were run in an attempt to find an optimal gain k , but unique situations were found that require higher or lower gains for a stable solution. The gain has been left for the user to adjust to the situation.

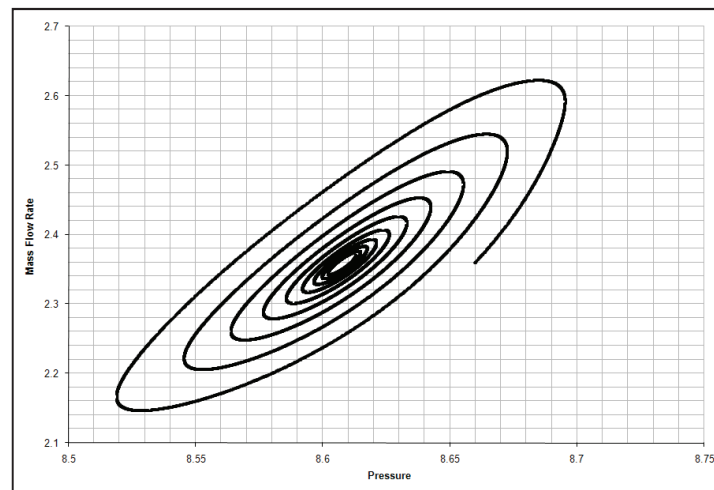


Figure 4.10: Phase Plane for Inlet Mass Flow Rate and Static Pressure.

Figure 4.11 shows the oscillations in mass flow rate and boundary pressure for a subsonic inlet with a gain of 0.0004. At subsonic speeds, the inlet condition creates many high frequency oscillations in mass flow rate, attributed to the acoustics of the inlet geometry.

Figure 4.12 shows similar oscillations for a supersonic inlet. If the solution starts with supersonic flow within the inlet, the gain must be scheduled to allow a normal shock to form

and advance to the inlet lip. The static pressure at the boundary plane is not affected at changing the mass flow rate while the normal shock exists within the inlet. The shock advances upstream to the inlet lip where mass flow can be relieved by spilling over the lip. The gain was scheduled in three steps, where the desired mass flow rate and gain are changed at each step: The first step forms a normal shock in the inlet ($k = 0.0004$). The second step ($k = 0.004$) lowers the mass flow rate (0.7%), pushing the shock out of the engine. The final step ($k = 0.004$) raises the mass flow rate back to design, pulling the shock back onto the inlet lip. The gain during the first step was kept small to create small changes in the pressure while the mass flow rate is relatively constant. The gain was increased for the latter two steps to speed up convergence while the shock advances upstream and then pulls back onto the lip. Information travels at a finite rate between the control boundary and normal shock. The time lag created by the exchange of information must be taken into account when selecting appropriate gain values.

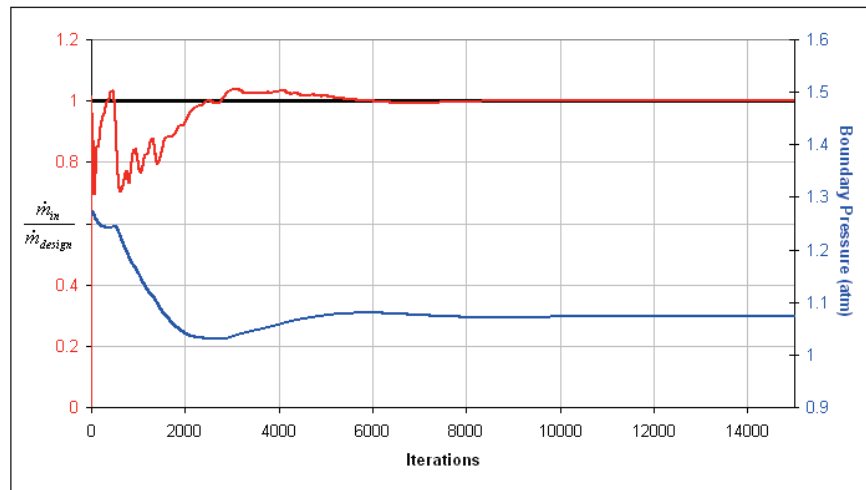


Figure 4.11: Oscillating Mass Flow Rate and Boundary Pressure (Subsonic).

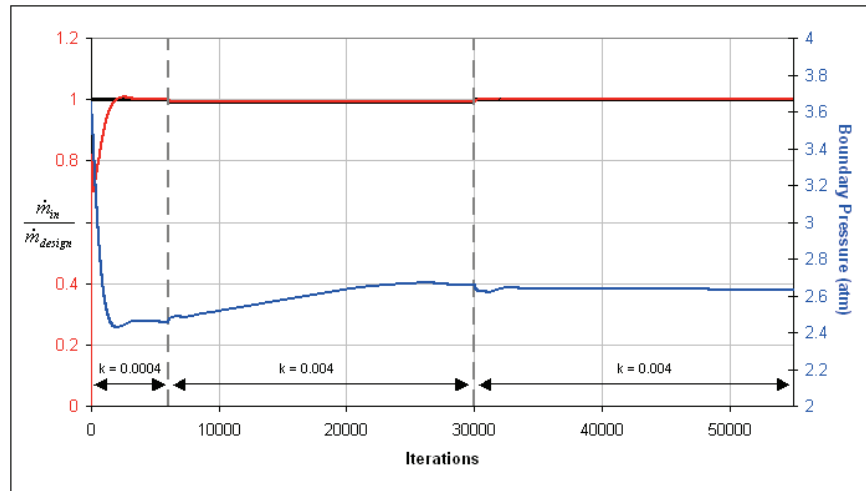


Figure 4.12: Oscillating Mass Flow Rate and Boundary Pressure (Supersonic).

The inflow boundary has been tested under various conditions to verify that the appropriate flow physics are present in the solution. The results from these tests are illustrated in Figure 4.13. The inlets were designed and compared with the principles outlined by Farokhi (2009):

- Case A shows a sharp-edged supersonic inlet designed to have a normal shock at the inlet at Mach 1.5. The solution in Figure 4.13 was created using the process described for Figure 4.12, which is necessary to push the normal shock out of the inlet.
- Case B is a subsonic inlet with round leading edges flying at its design speed – Mach 0.6. The solution shows that the flow goes transonic at the narrowest point in the inlet. Stagnation points can be found on the leading edges of the inlet lips and the spinner.
- Case C is the same inlet tested in Case B held at an angle of attack. The asymmetric flow field causes the lower lip to see even higher transonic speeds while the upper lip sees lower speeds. The stagnation points on the inlet lips both rotate down to account for the angle of attack. The asymmetry in the flow field exists until very near the inflow plane, where the inflow boundary corrects the flow to nearly parallel. The spinner stagnation shows how the flow has turned.

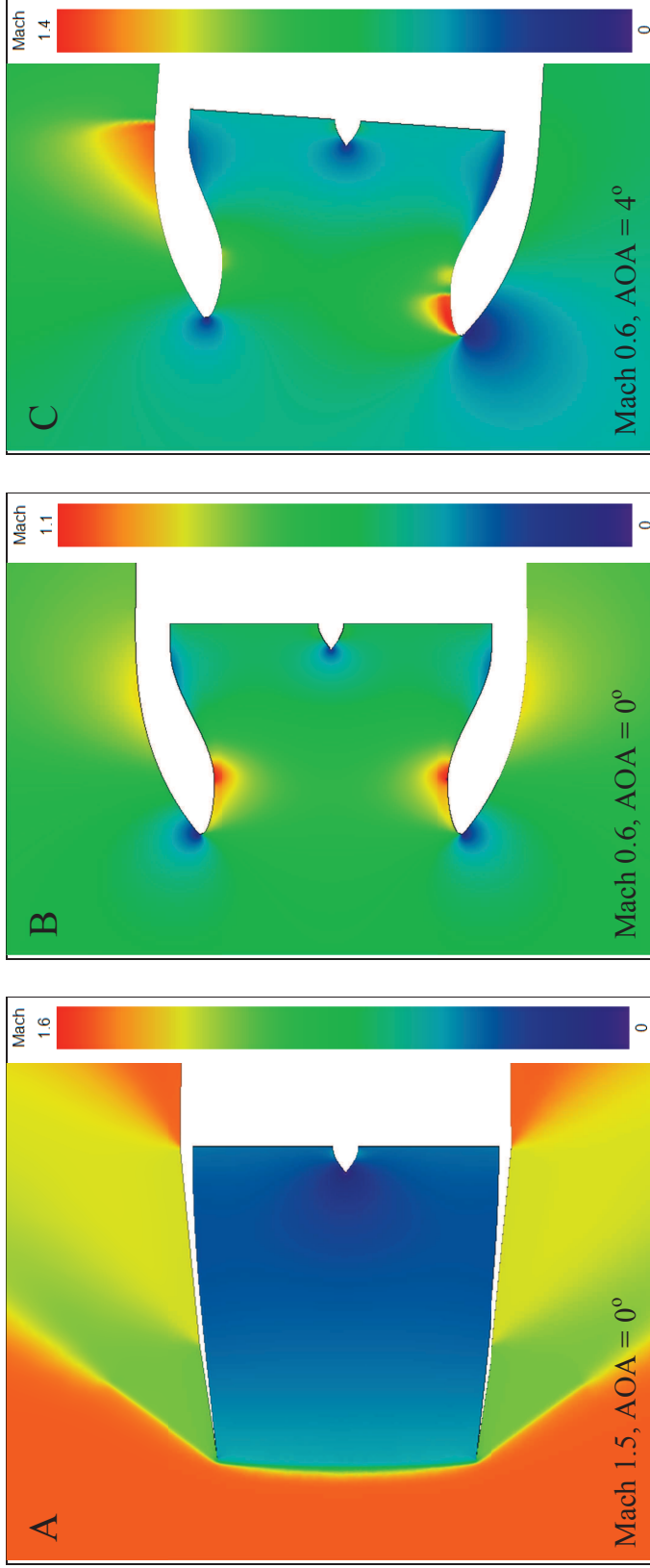


Figure 4.13: Engine Inlets under Various Conditions.

Outlet Boundary Conditions. The engine boundary condition defines the additional mass, momentum, and energy added to the flow by the engine, assuming that the engine is adiabatic. The boundary planes are assumed to be parallel, on average. (Each boundary is assumed to be a revolution about the shaft of the engine. The average of the boundary normal vectors for each plane will then align with the engine shaft. The additional momentum is thus aligned with the engine shaft and the normal vectors of the boundary planes.) The influx of mass, momentum, and energy are integrated over the inflow plane. The fuel flow \dot{m}_f , uninstalled thrust F , and enthalpy

production $\Delta\dot{m}H$ are added to these values to get the total out flux. The out flux is evenly distributed over the plane and applied to each element.

The mass, momentum, and energy flowing into the engine is integrated at the inflow plane. The additional flow rates are added to the inflow rate to obtain the total mass, momentum, and energy outflow rates. The outflow rates are divided by the total outflow area to calculate the boundary fluxes at the outflow plane and applied using Riemann invariants:

$$\dot{m}_{in} = \int_{A_{in}} \rho(u_i \cdot \hat{n}_i) dA \quad \dot{m}_{out} = \dot{m}_{in} + \dot{m}_f = \overline{\rho_{out} V_{n,out}} A_{out} \quad (4.136)$$

$$P_{j,in} = \int_{A_{in}} (\rho u_j (u_i \cdot \hat{n}_i) + p \hat{n}_j) dA \quad P_{j,out} = \left| \dot{m} \bar{V}_{in} \right| + F = \overline{\rho_{out} V_{n,out}^2} + p_{out} A_{out} \quad (4.137)$$

$$\dot{m}H_{in} = \int_{A_{in}} \rho H (u_i \cdot \hat{n}_i) dA \quad \dot{m}H_{out} = \dot{m}H_{in} + \Delta\dot{m}H = \overline{\rho H_{out} V_{n,out}} A_{out} \quad (4.138)$$

The Riemann invariant matrix \mathbf{A} (Eq. 4.69) is constructed using properties and not fluxes, so the properties must be reconstructed from the average fluxes along the boundary. Given the fluxes and the ideal gas equation (Eq. 3.131) and normal velocity (Eq. 4.144):

$$f_1 = \rho V_n \quad f_{j+1} = \rho u_{r,j} V_n + p \hat{n}_j \quad f_5 = \rho H_r V_n \quad (4.139)$$

$$p = \frac{\gamma-1}{\gamma} \left(\rho H_r - \frac{1}{2} \rho (u_{r,i} u_{r,i} - V_{t,i} V_{t,i}) + \rho K \right) \quad (4.140)$$

These equations can be combined to solve for static pressure p :

$$A p^2 + B p + C = 0 \quad (4.141)$$

$$A = \frac{\gamma+1}{2} \quad B = -f_j \hat{n}_j \quad C = (\gamma-1) \left(f_1 f_5 - \frac{1}{2} f_{j+1} f_{j+1} + f_1^2 \left(\frac{1}{2} V_{t,i} V_{t,i} + K \right) \right) \quad (4.142)$$

The solution to the quadratic equation above comes in the form of two pressures: One corresponding to a subsonic property set, and the other a supersonic property set. The properties exiting the outlet plane are almost always subsonic. The subsonic pressure is:

$$p = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \quad (4.143)$$

The velocity vector can be found using the momentum flux vector f_{j+1} :

$$u_j = \frac{f_{j+1} - p\hat{n}_j}{f_1} \quad V_n = u_i \cdot \hat{n}_i \quad (4.144)$$

The density and total enthalpy are calculated using the mass and enthalpy fluxes f_1 and f_5 :

$$\rho = \frac{f_1}{V_n} \quad \rho H = \frac{f_5}{V_n} \quad \rho E = \rho H - p \quad (4.145)$$

Example: Consider a flow with the properties: $\rho = 0.8$, $u = 1.25$, $v = 0$, $w = 0$, $p = 2$, $\rho E = 5.625$, and $\rho H = 7.625$. For a unit normal $(1,0,0)$, the fluxes are: $f_1 = 1$, $f_2 = 3.25$, $f_3 = f_4 = 0$, and $f_5 = 9.53125$. The properties are reconstructed (Eqs. 4.140 through 4.144):

$$A = \frac{1.4+1}{2} = 1.2 \quad B = -(3.25)(1) - (0)(0) - (0)(0) = -3.25$$

$$C = (1.4 - 1) \left((1)(9.53125) - \frac{1}{2} \left((3.25)^2 + (0)^2 + (0)^2 \right) \right) = 1.7$$

$$p = \frac{-(-3.25) + \sqrt{(-3.25)^2 - 4(1.2)(1.7)}}{2(1.2)} = 2$$

$$u = \frac{3.25 - (2)(1)}{1} = 1.25 \quad v = \frac{0 - (2)(0)}{1} = 0 \quad w = \frac{0 - (2)(0)}{1} = 0$$

$$V_n = (1.25)(1) + (0)(0) + (0)(0) = 1.25$$

$$\rho = \frac{1}{1.25} = 0.8 \quad \rho H = \frac{9.53125}{1.25} = 7.625 \quad \rho E = 7.625 - 2 = 5.625$$

Improved Performance. During the startup process, the fluxes and properties through at the engine inflow plane contain many perturbations. These disturbances may arise from the propagating “acoustics” due to the sudden startup or from the iteration of the pressure controller with the inlet flow field. These disturbances are transferred to the outflow plane through the conservation relationships (Eqs. 4.136 through 4.138). To avoid stability problems caused by these disturbances, the static properties at the outflow plane are ramped up from their initial values. Eq. 4.146 shows the ramp-up of static pressure, for example:

$$p = p_{IC} + (p_{spec} - p_{IC}) \frac{istp}{N_{rstp}} \quad (4.146)$$

Figure 4.14 shows the relationship between the outflow properties estimated by the engine conservation (dashed lines) and ramped properties (solid lines). The ramped properties are much smoother and present a more stable outflow boundary. These properties are then used to calculate the fluxes through the outflow planes (Eq. 4.88). The viscous flux is taken directly from the adjacent element. Figure 4.15 shows the fluxes calculated from the estimated (dashed) and ramped (solid) properties. The disturbances are minimized by interpolating between the estimated properties and initial conditions.

4.1.12 Artificial Dissipation

Cowan added two artificial dissipation models to the residual (Eq. 4. 36) for stability:

$$\mathbf{R}(\mathbf{U}_{n+1}, \mathbf{U}_n, \mathbf{U}_{n-1}) = [\mathbf{M}_c] \frac{\partial \mathbf{U}}{\partial t} - \mathbf{A}(\mathbf{U}_{n+1}) + \mathbf{B}(\mathbf{U}_{n+1}) - \mathbf{S}(\mathbf{U}_{n+1}) - \mathbf{D}(\mathbf{U}_{n+1}) \quad (4.147)$$

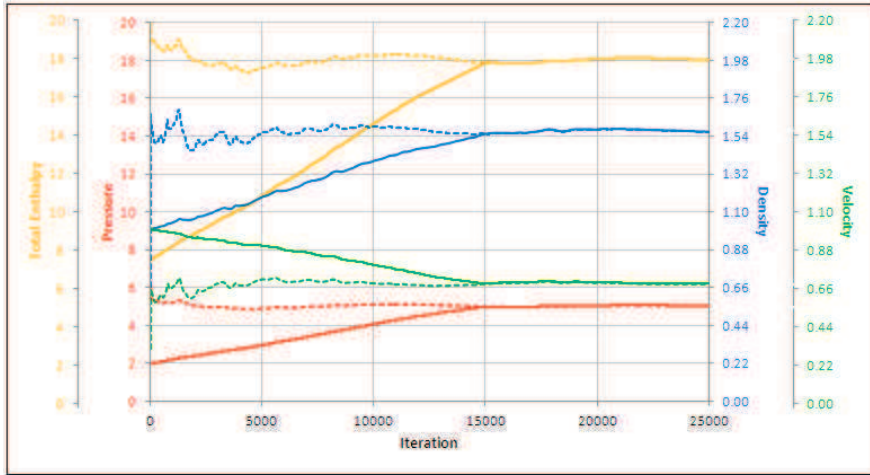


Figure 4.14: Properties along Turbojet Inflow Boundary over 15k Ramp-up Steps.

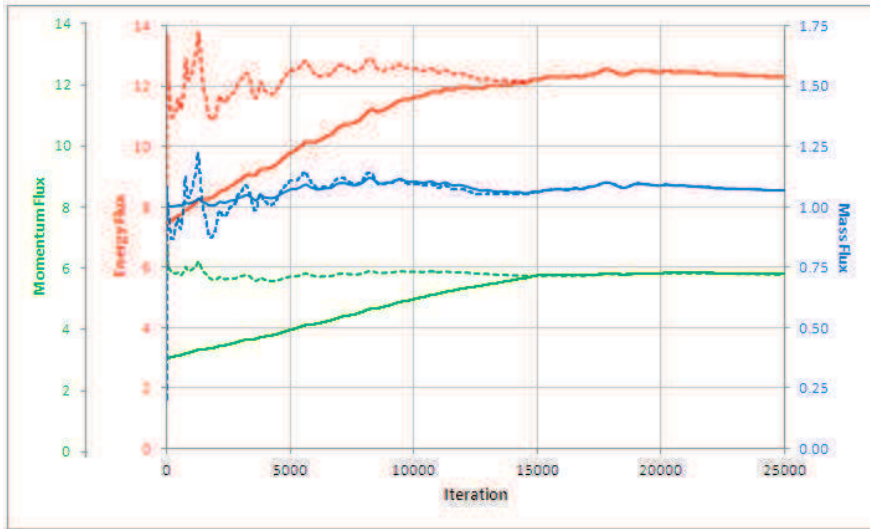


Figure 4.15: Fluxes along Turbojet Inflow Boundary over 15k Ramp-up Steps.

where \mathbf{A} is shorthand for the element flux vectors, \mathbf{B} is shorthand for the boundary flux vectors, and \mathbf{S} represents all of the source terms. For the lower order model, the artificial dissipation is summed over all segments in the domain (Cowan, 2003; Peiro, 1993):

$$\mathbf{D}(U_{n+1}) = \sum_{nsg} \frac{1}{d+1} \text{diss} \left(f_1 \left| \vec{\nabla}_t \right| a^* \Delta U + f_2 \vec{\nabla}_t \Delta p \right) \quad (4.148)$$

where $diss$ is the dissipation scaling factor, ∇_t is the gradient along the segment (constructed using the inverse element Jacobians and boundary area-normal vector), a^* is the speed of sound, $\Delta\mathbf{U}$ is the difference in properties along the segment, and Δp is the difference in pressure along the segment. The functions f_1 and f_2 are calculated based on local Mach:

$$M_1 = \frac{\bar{\nabla}_t \cdot \bar{\mathbf{u}}_{r,1}}{|\bar{\nabla}_t|} \sqrt{\frac{\rho_1}{\gamma p_1}} \quad M_2 = \frac{\bar{\nabla}_t \cdot \bar{\mathbf{u}}_{r,2}}{|\bar{\nabla}_t|} \sqrt{\frac{\rho_2}{\gamma p_2}} \quad (4.149)$$

$$M^+ = \begin{cases} \frac{1}{4}(1 + M_2)^2 & \text{if } |M_2| < 1 \\ \frac{1}{2}(M_2 + |M_2|) & \text{if } |M_2| \geq 1 \end{cases} \quad M^- = \begin{cases} \frac{1}{4}(1 + M_1)^2 & \text{if } |M_1| < 1 \\ \frac{1}{2}(M_1 + |M_1|) & \text{if } |M_1| \geq 1 \end{cases} \quad (4.150)$$

$$M^* = M^+ + M^- \quad (4.151)$$

$$f_1 = \begin{cases} \frac{1}{2}(1 + (M^*)^2) & \text{if } |M^*| < 1 \\ |M^*| & \text{if } |M^*| \geq 1 \end{cases} \quad f_2 = \begin{cases} \frac{1}{2}M^*(3 - M^*) & \text{if } |M^*| < 1 \\ M^*/|M^*| & \text{if } |M^*| \geq 1 \end{cases} \quad (4.152)$$

$$a^* = \frac{1}{2}(a_1(1 - f_2) + a_2(1 + f_2)) \quad (4.153)$$

The lower order model is suggested for supersonic solutions. The higher order model is calculated using the same equations, where the small differences $\Delta\mathbf{U}$ and Δp are neglected. The differences are removed so that the model can be used in rotating and subsonic domains.

Brown (2009) found that artificial dissipation is required to maintain positive entropy growth. The stagnation point on a circular cylinder begins to drift for dissipation constants lower than 0.5, so that a solution with 0.1 dissipation constant resembles a spinning cylinder, where the forward and aft stagnation points approach each other on one side of the cylinder. The governing equations enforce the conservation of mass, momentum, and energy. The second law is not enforced by a governing equation but rather by artificial dissipation, which skews the solution to produce a positive entropy growth.

4.1.13 Predictor-Corrector and Temporal Discretization

Cowan (2003) does not solve the governing equations directly, but uses a predictor-corrector scheme to minimize the error on the governing equations while progressing the solution with inherent stability. The predictor-corrector algorithm advances the solution using a Taylor series expansion of the residual (Eq. 4.147). If the solution residual is driven to zero at the next step then the expansion becomes:

$$\mathbf{R}(\mathbf{U}_{n+1}^{i+1}, \mathbf{U}_n, \mathbf{U}_{n-1}) = \mathbf{R}(\mathbf{U}_{n+1}^i, \mathbf{U}_n, \mathbf{U}_{n-1}) + \frac{\partial \mathbf{R}}{\partial \mathbf{U}} (\mathbf{U}_{n+1}^{i+1} - \mathbf{U}_{n+1}^i) = 0 \quad (4.154)$$

where the superscript indicates the iterations to convergence. The residual derivative can be approximated using the consistent mass matrix, and the consistent mass matrix can be converted to a lumped mass matrix to simplify inversion:

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}} \approx [\mathbf{M}_C] \approx [\mathbf{M}_L] \quad (4.155)$$

$$\mathbf{U}_{n+1}^{i+1} = \mathbf{U}_{n+1}^i - [\mathbf{M}_L]^{-1} \mathbf{R}(\mathbf{U}_{n+1}^i, \mathbf{U}_n, \mathbf{U}_{n-1}) \quad (4.156)$$

The steady update is further modified so that the unknowns are only updated from the previous iteration:

$$\mathbf{U}_{n+1}^{i+1} = \mathbf{U}_n - [\mathbf{M}_L]^{-1} \mathbf{R}(\mathbf{U}_{n+1}^i) \quad (4.157)$$

In both cases, the residual goes to zero as the governing equations become satisfied, and the predictor-corrector does not change the unknowns for a zero residual.

Meaningful Solutions. The properties can be written in their most primitive states as: ρ , u , v , w , p , and T . The components of velocity can take on any negative or positive values, even zero.

The thermodynamic properties must be positive definite: $\rho > 0$, $p > 0$, $T > 0$. These three

quantities are interrelated through the equation of state (Eq. 3.26). The internal energy, enthalpy, and acoustic speed are also calculated from the thermodynamic quantities (Eq. 3.5, 3.23, 3.25, and 3.36), and these properties must be positive definite: $e > 0$, $h > 0$, $E > 0$, $H > 0$, and $a > 0$.

Only two of these quantities need to be checked for validity (i.e., positive definite values) and the thermodynamic relationships will maintain the others. Density is an obvious choice since density is updated by the continuity equation and stored in the unknowns vector. Total energy is updated through the energy equation, and total enthalpy is tracked in \mathbf{U} . Neither of these properties should be checked because positive-definite total properties do not ensure positive-definite static properties; although the opposite is true. Cowan checks pressure over internal energy.

Changes in density are calculated directly from the continuity residual. To ensure that the density is positive definite, excessively negative changes in density are scaled down:

$$\rho_2 = \rho_1 + \frac{\Delta\rho}{1 - 4(0.1 + \Delta\rho/\rho)} \quad \text{if} \quad \frac{\Delta\rho}{\rho} < -0.1 \quad (4.158)$$

Figure 4.16 shows the effective density change with and without the limitation. With the limitation, the density change can only approach -0.25 in the limit as the actual change goes to negative infinity. In other words, this function limits the size of the effective density drop in one iteration to be less than one quarter of the density. If the density change were held constant, the solver could approach a very small density (< 0.01) in only a few iterations (see Figure 4.16).

With the limiting function, the solver would need at least 17 iterations to approach a very small density while the governing equations are predicting a very large negative density change. This function keeps large negative perturbations from destroying a reasonable solution. The function does *not* hold back unstable solutions, because an unstable solution would persist beyond the 17 iterations that the function provides protection.

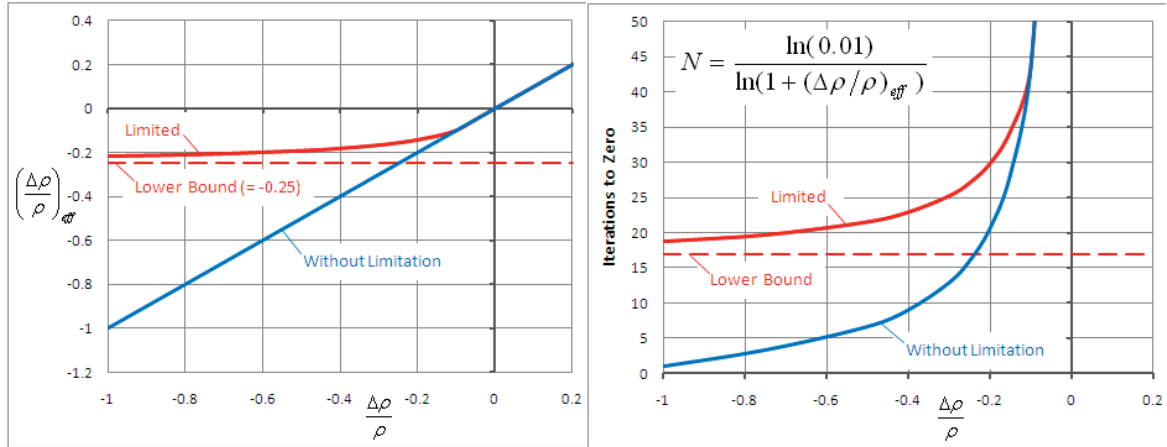


Figure 4.16: Effective Density Change (Left) and Number of Iterations to Zero (Right).

Using this method, the density can only approach a zero value, but never reach zero, ensuring a positive definite value. Changes in pressure are slightly more complicated. The changes in total and turbulent kinetic energy are calculated using their residuals and predictor corrector methods. The change in pressure is calculated from the change in total energy and kinetic energy in the current and previous iterations. The pressure is corrected using the same method as density. The new pressure is used to calculate the new total enthalpy to be stored.

$$\Delta p = p_2 - p_1 = (\gamma - 1) \left(\Delta \rho E - \frac{1}{2} \rho_2 |\vec{V}_2|^2 + \frac{1}{2} \rho_1 |\vec{V}_1|^2 - \Delta \rho K \right) \quad (4.159)$$

$$p_2 = p_1 + \frac{\Delta p}{1 - 4(0.1 + \Delta p/p)} \quad \text{if} \quad \frac{\Delta p}{p} < -0.1 \quad (4.160)$$

The turbulent properties are limited so that the turbulent intensity and eddy viscosity are never less than their freestream values. Because this limit occurs much more frequently than Eqs. 4.158 and 4.165, the residual is corrected before being summed into the RMS tracking calculations.

The correction is necessary for the residuals to show convergence properly.

4.1.14 Residual and Boundary Conditions

The flow tangency condition was described previously in its application to the inviscid solid wall boundary condition. The flow tangency equation is actually applied in three different places: Initial conditions, velocity vector at each iteration, and the residual vector at each iteration. The first two applications are made directly to the velocity vector using Eq. 4.91.

The application to the residual is less obvious. The flow tangency condition is written in terms of the unknowns vector as:

$$\mathbf{U}_{n+1}^{i+1} = \left(\mathbf{U}_{n+1}^{i+1}\right)^* - \left(\left(\mathbf{U}_{n+1}^{i+1}\right)^* \cdot \begin{Bmatrix} \mathbf{0} \\ \hat{\mathbf{n}} \\ \mathbf{0} \end{Bmatrix} \right) \begin{Bmatrix} \mathbf{0} \\ \hat{\mathbf{n}} \\ \mathbf{0} \end{Bmatrix} \quad (4.161)$$

where $\left(\mathbf{U}_{n+1}^{i+1}\right)^*$ is the unknowns calculated using the predictor-corrector, and \mathbf{U}_{n+1}^{i+1} is the unknowns after one complete iteration. During a single iteration, the previous unknowns are used to calculate the residual vector and update to new unknowns. The flow tangency condition is applied to the unknowns vector to arrive at the solution for this iteration. These two equations can be combined into one step:

$$\mathbf{U}_{n+1}^{i+1} = \mathbf{U}_{n+1}^i - [\mathbf{M}_L]^{-1} \mathbf{R}(\mathbf{U}_{n+1}^i, \mathbf{U}_n) - \left(\left(\mathbf{U}_{n+1}^i - [\mathbf{M}_L]^{-1} \mathbf{R}(\mathbf{U}_{n+1}^i, \mathbf{U}_n)\right) \cdot \begin{Bmatrix} \mathbf{0} \\ \hat{\mathbf{n}} \\ \mathbf{0} \end{Bmatrix} \right) \begin{Bmatrix} \mathbf{0} \\ \hat{\mathbf{n}} \\ \mathbf{0} \end{Bmatrix} \quad (4.162)$$

Because the initial conditions and each iteration are corrected for flow tangency, the previous vector is always tangent to the wall surface. If the residual vector is also tangent to the wall:

$$\mathbf{R}(\mathbf{U}_{n+1}^i, \mathbf{U}_n) \cdot \begin{Bmatrix} 0 \\ \hat{n} \\ 0 \end{Bmatrix} = 0 \quad (4.163)$$

The predictor-corrector and tangency iteration resolves to:

$$\mathbf{U}_{n+1}^{i+1} = \mathbf{U}_{n+1}^i - [\mathbf{M}_L]^{-1} \mathbf{R}(\mathbf{U}_{n+1}^i, \mathbf{U}_n) \quad (4.164)$$

which is the same as the original predictor corrector equation. Therefore, the flow tangency condition is applied to the residual before entering the predictor-corrector algorithm to maintain flow tangency at that step, and at the end of each iteration to ensure flow tangency between iterations:

$$\mathbf{R}(\mathbf{U}_{n+1}^i, \mathbf{U}_n) = \mathbf{R}'(\mathbf{U}_{n+1}^i, \mathbf{U}_n) \cdot \left(\begin{Bmatrix} 0 \\ \hat{n} \\ 0 \end{Bmatrix} \right) \begin{Bmatrix} 0 \\ \hat{n} \\ 0 \end{Bmatrix} \quad (4.165)$$

When the velocity and enthalpy are constrained along the no-slip wall, the residual is not used to calculate these properties. The corresponding contributions are removed before the RMS summation so that convergence tracking is true to the solution.

4.2 CFDsol

CFDsol was developed by Dr. Gupta (2000) at NASA. CFDsol was originally written as a supersonic, inviscid FEM code with vision to expand the solver to viscous and aeroelastic cases. Several groups have worked on CFDsol under the guidance of Dr. Gupta. One group has added viscous terms and a Spalart-Allmaras turbulence model to the solver, although the SA model does not operate accurately. This research alleviated several problems with time accuracy, subsonic boundaries, and turbulence model. Routines were added to give the CFDsol the ability to model aeroelastics, rigid body dynamics, and propulsion cases.

4.2.1 Taylor Formulation

CFDsol was originally developed with the inertial Navier-Stokes equations (Gupta, 2000). This work extended that derivation to include all of the terms in Eqs. 4.29 through 4.32, excluding the SST model. These equations are discretized here using the Taylor-Galerkin method. A first-order Taylor series expansion can be written on the unknown vector \mathbf{U} :

$$\mathbf{U}_{n+1} = \mathbf{U}_n + \frac{\Delta t}{1!} \frac{\partial \mathbf{U}}{\partial t} + \frac{\Delta t^2}{2!} \frac{\partial^2 \mathbf{U}}{\partial t^2} + \frac{\Delta t^3}{3!} \frac{\partial^3 \mathbf{U}}{\partial t^3} + \dots \quad (4.166)$$

where the subscripts n and $n+1$ represent the values at the previous and current (to be calculated) time steps. The expansion can be truncated after the first derivative, making the approximation first order accurate in time. Eq. 4.29 is substituted back into this equation:

$$\mathbf{U}_{n+1} = \mathbf{U}_n + \Delta t \frac{\partial \mathbf{U}}{\partial t} \quad (4.167)$$

$$\mathbf{U}_{n+1} = \mathbf{U}_n + \Delta t \left(-\frac{\partial \mathbf{F}_i}{\partial x_{b,i}} + \frac{\partial \mathbf{F}_{v,i}}{\partial x_{b,i}} + \mathbf{S}_{NI} + \mathbf{S}_C + \mathbf{S}_T \right) \quad (4.168)$$

Gupta expands the inviscid term using the chain rule so that Gauss's theorem is no longer needed for the inviscid flux. (The SST model has been dropped in the rest of the derivation.)

$$\frac{\partial \mathbf{F}_i}{\partial x_{b,i}} = \frac{\partial}{\partial x_{b,i}} \begin{pmatrix} \rho u_{r,i} \\ \rho u_{r,i} u_{r,j} + p \delta_{ij} \\ u_{r,i} (\rho E_r + p) \\ u_{r,i} \rho \hat{v} \end{pmatrix} = \frac{\partial}{\partial x_{b,i}} \left(u_{r,i} \mathbf{U} + p \begin{pmatrix} 0 \\ \delta_{ij} \\ u_{r,i} \\ 0 \end{pmatrix} \right) \quad (4.169)$$

$$\frac{\partial \mathbf{F}_i}{\partial x_{b,i}} = \frac{\partial u_{r,i}}{\partial x_{b,i}} \mathbf{U} + u_{r,i} \frac{\partial \mathbf{U}}{\partial x_{b,i}} + \frac{\partial p}{\partial x_{b,i}} \begin{pmatrix} 0 \\ \delta_{ij} \\ u_{r,i} \\ 0 \end{pmatrix} + \frac{\partial u_{r,i}}{\partial x_{b,i}} \begin{pmatrix} 0 \\ 0 \\ p \\ 0 \end{pmatrix} \quad (4.170)$$

4.2.2 Galerkin Formulation

Eqs. 4.170 is substituted into Eq. 4.168 and integrated using Galerkin's method:

$$\int_{\forall} \Phi_e^T (\mathbf{U}_{n+1} - \mathbf{U}_n) d\Omega = - \int_{\forall} \Phi_e^T \Delta t \left(\frac{\partial u_{r,i}}{\partial x_{b,i}} \mathbf{U} + u_{r,i} \frac{\partial \mathbf{U}}{\partial x_{b,i}} + \frac{\partial p}{\partial x_{b,i}} \begin{pmatrix} 0 \\ \delta_{ij} \\ u_{r,i} \\ 0 \end{pmatrix} + \frac{\partial u_{r,i}}{\partial x_{b,i}} \begin{pmatrix} 0 \\ 0 \\ p \\ 0 \end{pmatrix} \right) d\Omega \quad (4.171)$$

$$+ \int_{\forall} \Phi_e^T \Delta t \left(\frac{\partial \mathbf{F}_{v,i}}{\partial x_{b,i}} + \mathbf{S}_{NI} + \mathbf{S}_C + \mathbf{S}_T \right) d\Omega$$

The viscous term requires Gauss's theorem to reduce the order of derivatives:

$$\int_{\forall} \Phi_e^T (\Delta \mathbf{U}) d\Omega = - \int_{\forall} \Phi_e^T \Delta t \left(\frac{\partial u_{r,i}}{\partial x_{b,i}} \mathbf{U} + u_{r,i} \frac{\partial \mathbf{U}}{\partial x_{b,i}} + \frac{\partial p}{\partial x_{b,i}} \begin{pmatrix} 0 \\ \delta_{ij} \\ u_{r,i} \\ 0 \end{pmatrix} + \frac{\partial u_{r,i}}{\partial x_{b,i}} \begin{pmatrix} 0 \\ 0 \\ p \\ 0 \end{pmatrix} \right) d\Omega \quad (4.172)$$

$$+ \int_{\forall} \Phi_e^T \Delta t (\mathbf{S}_{NI} + \mathbf{S}_C + \mathbf{S}_T) d\Omega - \int_{\forall} \frac{\partial \Phi_e^T}{\partial x_{b,i}} \Delta t \mathbf{F}_{v,i} d\Omega + \int_S \Phi_e^T \Delta t (\mathbf{F}_{v,i} \cdot \hat{n}_i) d\Gamma$$

Gupta uses the shape functions Φ_e to create as many mass matrices as possible. The remaining properties are averaged on each element:

$$\int_{\forall} \Phi_e^T (\Phi_e \Delta \mathbf{U}_e) d\Omega = - \int_{\forall} \Phi_e^T \Delta t \left(\frac{\partial u_{r,i}}{\partial x_{b,i}} \Phi_e \mathbf{U}_e + \bar{u}_{r,i} \frac{\partial \mathbf{U}}{\partial x_{b,i}} + \frac{\partial p}{\partial x_{b,i}} \begin{pmatrix} 0 \\ \delta_{ij} \\ \bar{u}_{r,i} \\ 0 \end{pmatrix} + \frac{\partial u_{r,i}}{\partial x_{b,i}} \begin{pmatrix} 0 \\ 0 \\ p \\ 0 \end{pmatrix} \right) d\Omega \quad (4.173)$$

$$+ \int_{\forall} \Phi_e^T \Delta t (\mathbf{S}_{NI} + \mathbf{S}_C + \mathbf{S}_T) d\Omega - \int_{\forall} \frac{\partial \Phi_e^T}{\partial x_{b,i}} \Delta t \bar{\mathbf{F}}_{v,i} d\Omega + \int_S \Phi_e^T \Delta t (\bar{\mathbf{F}}_{v,i} \cdot \hat{n}_i) d\Gamma$$

Gupta collects common terms into matrices and vectors:

$$\sum_e \mathbf{M}_{c,e} \Delta \mathbf{U}_e = - \sum_e \Delta t \left(\frac{\partial u_{r,i}}{\partial x_{b,i}} \mathbf{M}_{c,e} + \mathbf{K}_e \right) \mathbf{U}_e - \sum_e \Delta t (\mathbf{f}_{1,e} + \mathbf{f}_{2,e}) + \mathbf{S} + \mathbf{K}_\sigma + \mathbf{f}_\sigma \quad (4.174)$$

$$\mathbf{M}_{c,e} = \int_{V_e} \Phi_e^T \Phi_e d\Omega \quad \mathbf{K}_e = \int_{V_e} \Phi_e^T \bar{u}_{r,i} \frac{\partial \Phi_e}{\partial x_{b,i}} d\Omega \quad (4.175)$$

$$\mathbf{f}_{1,e} = \int_{V_e} \Phi_e^T \frac{\partial p}{\partial x_{b,i}} \begin{Bmatrix} 0 \\ \delta_{ij} \\ \bar{u}_{r,i} \\ 0 \end{Bmatrix} d\Omega \quad \mathbf{f}_{2,e} = \int_{V_e} \Phi_e^T \frac{\partial u_{r,i}}{\partial x_{b,i}} \begin{Bmatrix} 0 \\ 0 \\ p \\ 0 \end{Bmatrix} d\Omega \quad (4.176)$$

$$\mathbf{S} = \int_{\forall} \Phi_e^T \Delta t (\mathbf{S}_{MI} + \mathbf{S}_C + \mathbf{S}_T) d\Omega \quad (4.177)$$

$$\mathbf{K}_\sigma = - \int_{\forall} \frac{\partial \Phi_e^T}{\partial x_{b,i}} \Delta t \bar{\mathbf{F}}_{v,i} d\Omega \quad \mathbf{f}_\sigma = \int_S \Phi_e^T \Delta t (\bar{\mathbf{F}}_{v,i} \cdot \hat{n}_i) d\Gamma \quad (4.178)$$

The mass matrix $\mathbf{M}_{c,e}$ has already been developed in Eq. 4.21. The viscous terms have been derived using the same method applied to the OSU in-house codes (Eqs. 4.36 and 4.55). The source terms are the same for all five codes and will be developed near the end of the chapter. The stiffness matrix \mathbf{K}_e and pressure vectors $\mathbf{f}_{1,e}$ and $\mathbf{f}_{2,e}$ need to be develop here.

4.2.2.1 Stiffness Matrix

The stiffness matrix \mathbf{K}_e is integrated, using the gradient of the basis function. The last term of the integrand in Eq. 4.175 is defined using the inverse Jacobian matrix \mathbf{A}_e for the element (Eqs. 4.13 and 4.17). Dotting \mathbf{A}_e with the velocity yields a constant vector \mathbf{T}_e :

$$\mathbf{T}_e = \bar{u}_{r,i} \frac{\partial \Phi_e}{\partial x_{b,i}} = \{ \bar{u}_r \quad \bar{v}_r \quad \bar{w}_r \} \frac{1}{|\mathbf{J}_e|} \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \end{bmatrix} \quad (4.179)$$

$$\mathbf{T}_e^T = \frac{1}{|\mathbf{J}_e|} \begin{Bmatrix} A_{11}\bar{u}_n + A_{21}\bar{v}_n + A_{31}\bar{w}_n \\ A_{12}\bar{u}_n + A_{22}\bar{v}_n + A_{32}\bar{w}_n \\ A_{13}\bar{u}_n + A_{23}\bar{v}_n + A_{33}\bar{w}_n \\ A_{14}\bar{u}_n + A_{24}\bar{v}_n + A_{34}\bar{w}_n \end{Bmatrix} = \begin{Bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{Bmatrix} \quad (4.180)$$

Eq. 4.180 is substituted into Eq. 4.175. \mathbf{T}_e is pulled out of the integral because it is constant:

$$\mathbf{K}_e = \int_{0 \leq \bar{\xi} \leq 1} \Phi_e^T |J_e| d\bar{\xi} \mathbf{T}_e = \frac{V_e}{4} \begin{Bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{Bmatrix} \mathbf{T}_e = \frac{V_e}{4} \begin{bmatrix} T_1 & T_2 & T_3 & T_4 \\ T_1 & T_2 & T_3 & T_4 \\ T_1 & T_2 & T_3 & T_4 \\ T_1 & T_2 & T_3 & T_4 \end{bmatrix} \quad (4.181)$$

Gupta takes advantage of the similarity in the $\mathbf{M}_{c,e}$ and \mathbf{K}_e matrices. $\mathbf{M}_{c,e}$ only contains two terms, one for the diagonal and one for the off-diagonal; and the columns of \mathbf{K}_e are constant. Gupta stores two values for each column, one for the diagonal and one for the off-diagonal.

4.2.2.2 Pressure Vector

Starting with Eq. 4.176 and using the stiffness matrix as a guide, $\mathbf{f}_{1,e}$ is shown in Eq. 4.182.

Gupta distributes the pressure in Eq. 4.176 using the shape function so that $\mathbf{f}_{2,e}$ becomes Eq. 4.183. The momentum terms of Eq. 4.182 can be dotted with the velocity vector to calculate contributions to the energy equation. Eq. 4.183 only contributes to the energy equation.

$$\mathbf{f}_{1,e} = \int_{V_e} \Phi_e^T d\Omega \begin{Bmatrix} 0 \\ \delta_{ij} \\ \bar{u}_{r,i} \\ 0 \end{Bmatrix} \frac{\partial p}{\partial x_{b,i}} = \frac{V_e}{4} \begin{Bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{Bmatrix} \begin{Bmatrix} \mathbf{0} \\ \mathbf{A}_e \\ \mathbf{T}_e \\ \mathbf{0} \end{Bmatrix} p_e \quad (4.182)$$

$$\mathbf{f}_{2,e} = \frac{\partial u_{r,i}}{\partial x_{b,i}} \int_{V_e} \Phi_e^T \Phi_e d\Omega p_e \begin{Bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{Bmatrix} = \frac{\partial u_{r,i}}{\partial x_{b,i}} \mathbf{M}_{c,e} p_e \begin{Bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{Bmatrix} \quad (4.183)$$

4.2.3 Boundary Conditions

Without boundary integrals, the boundary conditions in CFDsol must be applied by constraining the properties along the boundaries. This method sounds simple, but this section will demonstrate how the rigid nature of a constrained boundary condition is often unforgiving, unstable, and inaccurate. The essential conditions are the exception.

4.2.3.1 Far Field

The far field boundary is broken down into inflow and outflow regions. The two are distinguished from each other by the sign of the velocity normal. If the normal velocity is greater than zero, the boundary is considered to be outflow. Originally, the properties are not changed on the outflow boundary. The properties on the rest of the far field boundary (normal velocity less than or equal to zero) are set to the freestream properties.

Floating Outflow. The original implementation of CFDsol exhibited several boundary instabilities. The boundary conditions are constrained at the nodes after updating the governing equations. Strong boundary conditions constrain the inflow properties to match the freestream properties exactly. At subsonic speeds, the outflow properties are unrestricted and “float” to lower pressures without limit (Figure 4.17). The outflow pressure decreases with each iteration of the solution. The other properties follow pressure.

The velocity distribution (Figure 4.18) shows that the flow accelerates along the length of the domain and exits at an oblique angle with respect to the freestream. The outflow boundary allows the flow to rotate without restraint. The rotated flow converts some of the outflow to inflow. The reduced area of the outflow accelerates the flow like a nozzle.

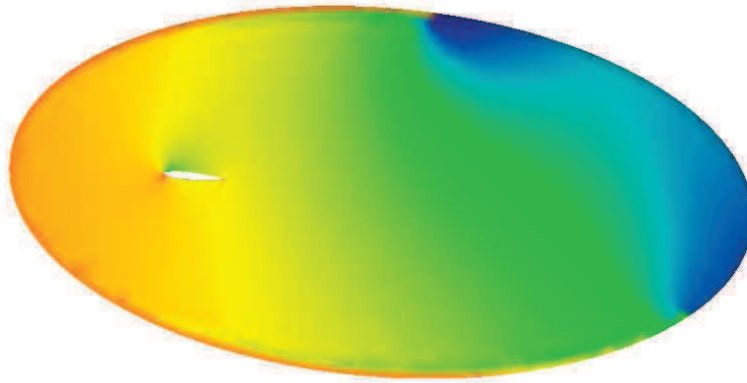


Figure 4.17: Pressure Distribution around NACA 0012 Airfoil (Mach 0.5, 5°, CFDsol).

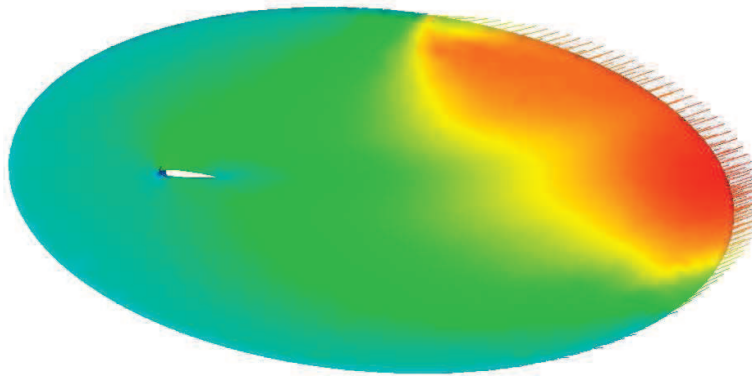


Figure 4.18: Velocity Distribution around NACA 0012 Airfoil (Mach 0.5, 5°, CFDsol).

Figure 4.19 shows the CFDsol solution compared to a potential flow solution. The CFDsol solution has the appropriate shape, but the entire pressure distribution is shifted by approximately $-1 C_p$ (shown upward because the vertical axis has been flipped). The pressure distribution near the airfoil is lower than normal because of a pressure gradient across the domain.

The pressure gradient in the CFDsol solution can be seen in Figure 4.20. The CFDsol solution sustains the freestream pressure ($C_p = 0$) over what normal velocity defines as the inflow boundary, approx. 80% of the boundary. The outflow boundary, on the other hand, has a much lower pressure, which creates the pressure gradient shown by the red line. The pressure gradient is so strong and influential that it creates a jump in pressure from the inflow boundary to the neigh-

boring nodes. The airfoil disturbance in pressure can be seen in Figure 4.20 as a perturbation to the pressure gradient. The shift ($\Delta C_p \approx -1$) in Figure 4.19 can be clearly seen in the pressure gradient in Figure 4.20. Similar gradients are found in the density and velocity (Figure 4.21).

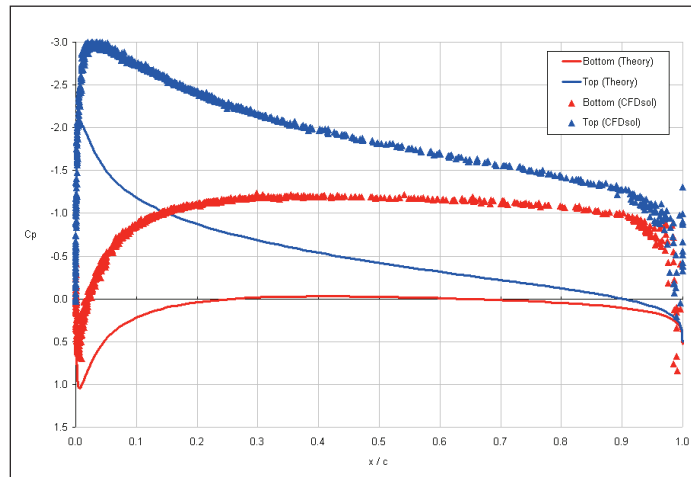


Figure 4.19: Coefficient of Pressure from CFDsol and Potential Flow Solution.

To isolate the problem on the boundary, several other factors were tested: The dissipation scalar *factor* was varied from 0.025 to 0.5, which showed no change in the outflow boundary.

Additional dissipation smoothed out the pressure gradient, which realigned the flow; but, the misalignment had already occurred, reducing the area of the outflow boundary and in turn accelerating the flow. Dissipation could not alleviate the boundary problem. The relaxation factor *tau* was varied from 0.1 to 2.0 and did not help to control the solution. Changing *tau* only affected the progress of the solution, not its stability. The freestream Mach number was found to be the most influential. Supersonic solutions had a stable outflow boundary because the boundary models the appropriate characteristics.

The shape of the far field boundary was also tested. Figure 4.22 shows an airfoil rotated at 5 degrees angle of attack with respect to a rectangular domain. The rectangular domain was

modeled as a far field boundary. The rotation of the flow due to the presence of the airfoil creates an asymmetric pressure load on the outflow boundary. The solution exhibits the same accelerated boundary phenomenon as the elliptical boundary. The velocity vectors are angled toward the corner.

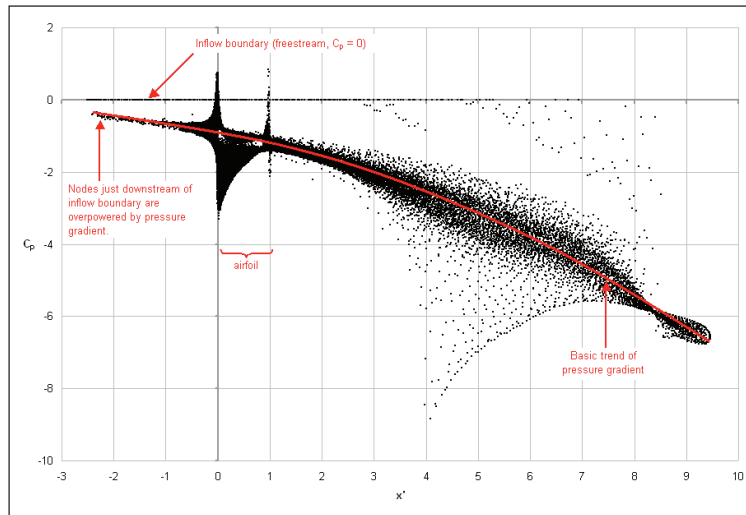


Figure 4.20: Domain Pressure through NACA 0012 Solution (Mach 0.5, 5°, CFDsol).

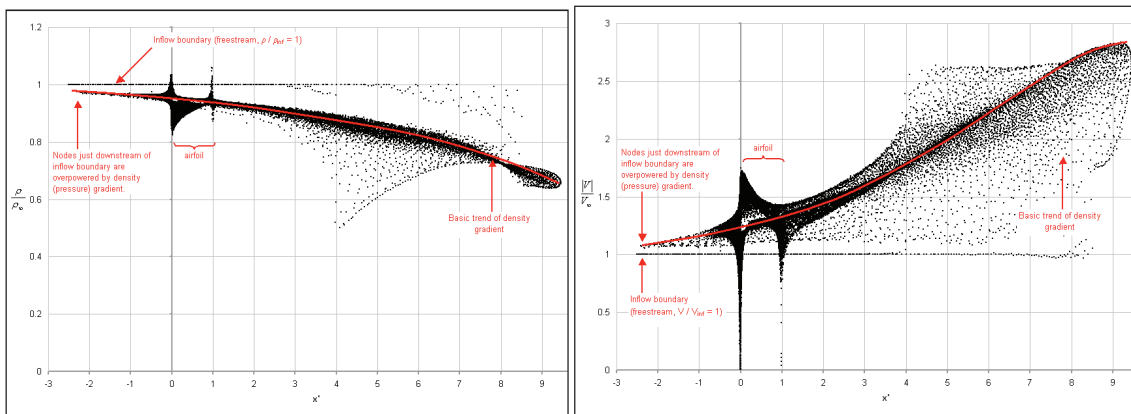


Figure 4.21: Density (Left) and Velocity (Right) from NACA 0012 (Mach 0.5, CFDsol).

Intermediate Solution. The two-part far field boundary was replaced by a “reflective” boundary condition, which applied the freestream conditions along the entire far field boundary. The condition was much more stable than the composite boundary, but changes in properties could

not propagate through the boundary. The vortices and acoustic waves generated by the structure were reflected from both the inflow and outflow boundaries. The energy change (positive or negative) could not leave the domain because of the impedance of the far field boundary. This boundary is not appropriate for elastic or rigid body motion.

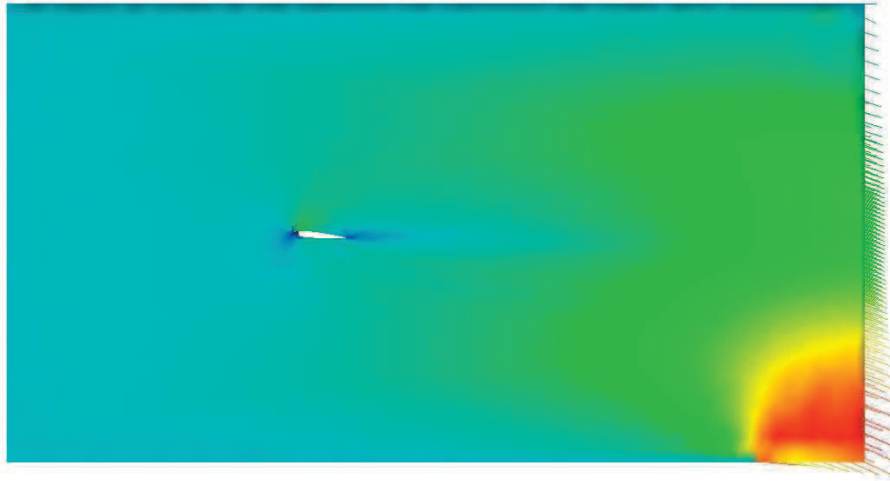


Figure 4.22: Velocity Distribution for NACA 0012 in Rectangular Domain (Mach 0.5).

Improved Far Field. The far field boundary must consider the characteristics of the flow because the governing equations couple all of the properties together in a very specific manner. The characteristics were demonstrated when developing the 1D Riemann problem (between Eqs. 4.55 and 4.56). The far field boundary was adjusted to take these characteristics into account and utilize a mixture of freestream and domain properties.

According to the Riemann invariants of the flow, three characteristics transfer information throughout the field. At subsonic speeds, at least one characteristic transfers information in each direction. At supersonic speeds, information is only transferred downstream and laterally in the flow. The difference in behavior explains why the suggested original boundary worked well at supersonic speeds where all three characteristics point downstream and that the boundary is

missing the upstream component at subsonic speeds. The loss in characteristic keeps pressure information from moving into the domain, allowing the pressure to float.

In other words, at supersonic speeds the far field boundary should only consider the upstream properties. Ideally, a subsonic far field would consider a mixture of both the upstream and downstream. The alternative is to choose the freestream properties for subsonic far fields, neglecting the characteristic traveling from the domain because its properties should be close to the freestream properties. This approach is summed up:

$$\mathbf{U}_{FF} = \begin{cases} \mathbf{U}_e & \text{for } M < 0 \text{ and } u_{r,i} \cdot \hat{n}_i \\ \mathbf{U}_\infty & \text{otherwise} \end{cases} \quad (4.184)$$

When these criteria are applied to the far field boundary, the subsonic far field maintains its pressure for the airfoil case. Compare Figure 4.23 (after correction) with Figure 4.19 and Figure 4.20 (before). The CFDsol solution now matches the potential flow solution.

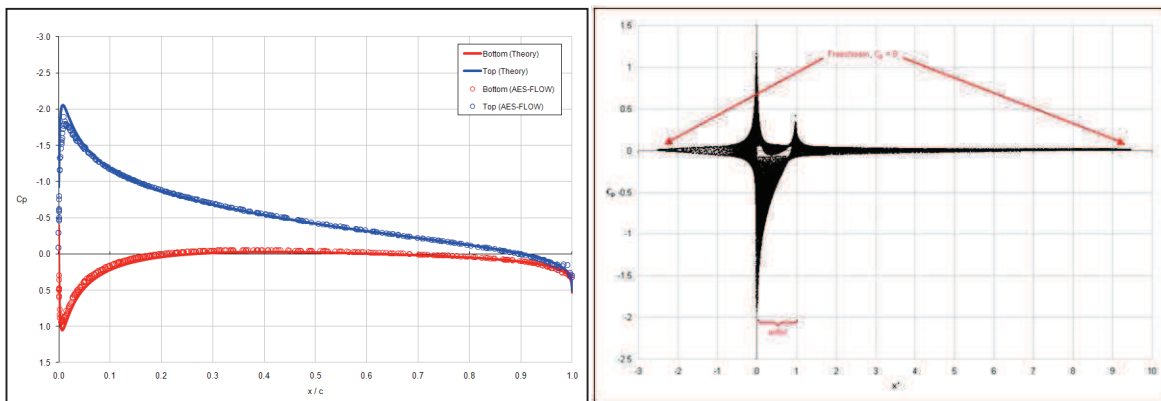


Figure 4.23: Surface (Left) and Domain Pressure (Right) for NACA 0012 (Improved).

4.2.3.2 Inviscid Wall

The inviscid wall boundary is maintained using flow tangency at the wall nodes. Eqs. 4.90 and 4.94 are applied to each node along the inviscid wall, using the area-weighted wall normals (Eq.

4.91). Flow tangency is lifted at singular nodes, which were added during this work by AES engineers. The singular nodes were modeled after the OSU in-house codes and alleviated problems along start leading and trailing edges.

4.2.3.3 Viscous Wall

Heat transfer is not included in CFDsol. The no-slip condition is the only boundary condition applied along the viscous solid wall. The no-slip condition (Eq. 4.95) is applied explicitly along with the correction to total energy (Eq. 4.96).

4.2.3.4 Symmetry Plane

The symmetry plane has to be applied explicitly in CFDsol. The presence of the symmetry plane implies a zero normal velocity and a zero normal derivative. The velocity can be constrained to have no normal velocity using flow tangency (Eq. 4.90). Like the solid walls, the total energy must be corrected using Eq. 4.94 to avoid inducing a pressure gradient normal to the boundary. The gradient condition is unconstrained; flow tangency is assumed to be sufficient to create a symmetry field.

CFDsol assumes that the symmetry plane is *planar* so that flow tangency is applied to each boundary element individually. Velocity is removed from all three nodes using the element's normal vector. If the symmetry plane is faceted (not planar), two neighboring elements will remove all of the velocity normal to their surface. This process may take several iterations to occur to take full effect, but eventually the faceted boundary will remove all of the velocity.

The explicit symmetry boundary also has advantages. If inviscid walls are used to model two surfaces that come together at a 90° angle, the normal within the junction are averaged between

the two surfaces. (The normal is area-weighted using Eq. 4.91.) The symmetry plane can be used to create a planar wall, like a wind tunnel wall or flat ground, that applies flow tangency. The nodes in the corner feel flow tangency from the symmetry plane (elemental) and wall (nodal) so that the flow is properly constrained. (In the OSU in-house codes, the symmetry plane implies no normal velocity and should not be used in place of a wall.)

4.2.3.5 Rocket Exhaust

The rocket boundary condition was designed and implemented in Euler2D and expanded to Euler3D. The rocket boundary is created by calculating the boundary normal flux using the rocket properties. The routines were then transferred to CFDsol, where the rocket properties are used to constrain the rocket boundary between iterations. A third property is needed to close the isentropic and thermodynamic equations (Eqs. 4.129 through 4.132). The OSU codes use the static pressure from the element side of the discontinuity. Static pressure was not an appropriate choice for CFDsol. If the static pressure is taken from the element and then used to constrain the pressure on that element, then the pressure has not really been constraint. Like the badly constrained far field, the pressure along the rocket boundary floats.

Instead the exhaust Mach number is used to close the isentropic relationships. CFDsol requires the user to specify the exhaust Mach number for the duration of the run. The isentropic relationships are used to calculate the static properties from their totals:

$$\frac{T_t}{T} = 1 + \frac{\gamma - 1}{2} M^2 \qquad \frac{p_t}{p} = \left(\frac{T_t}{T} \right)^{\frac{\gamma}{\gamma - 1}} \qquad (4.185)$$

$$h = \frac{H}{T_t/T} \qquad p = \frac{P_t}{p_t/p} \qquad (4.186)$$

The other properties are calculated using Eq. 4.131 and the following:

$$V_n = M a \qquad u_{r,i} = -V_n n_i \qquad \rho E = \rho H - p \qquad (4.187)$$

To maintain solution stability, the total properties along the inflow plane are increased linearly over N_{rstp} iterations using Eq. 4.134. This work found that 1000 to 5000 iterations were found to be sufficient to step up the total properties, depending on the pressure rise and element size.

Smaller N_{rstp} can be used with care. Figure 4.24 shows the static properties at the rocket boundary as the total properties are increased over 2000 iterations. The plot on the left shows the properties calculated by CFDsol. Since all three values are specified by the user, the properties are known *a priori*, even during the ramp-up period. The right plot illustrates the pressure feedback in Euler3D through the boundary properties. The constraint boundary specifies the properties precisely at the boundary, whereas the implied boundary has lower impedance at the cost of the domain affecting the boundary properties indirectly. Surprisingly, the implied boundary is more stable and less susceptible to perturbations traveling back through the boundary.

Figure 4.25 shows the static pressure throughout the chamber and nozzle as the total properties are ramped up from freestream to off-design conditions. The consecutive slices show the pressure at the initial conditions, choke condition, and as the normal shock forms in the diverging section of the nozzle.

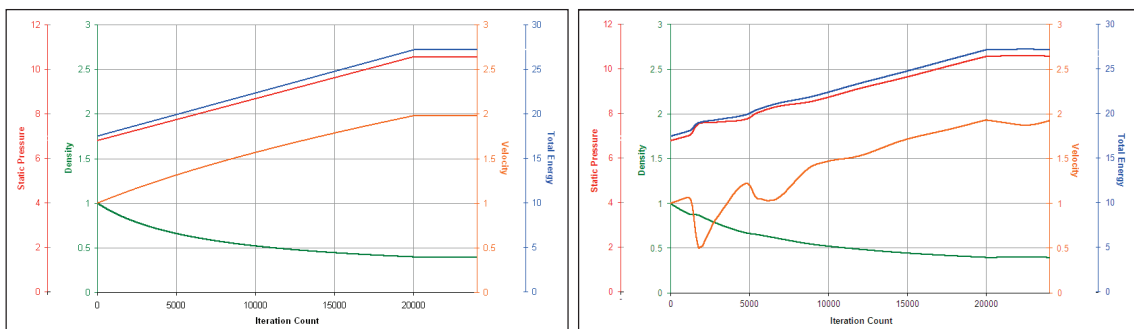


Figure 4.24: Static Properties at Rocket Boundary (left – CFDsol, right – Euler3D).

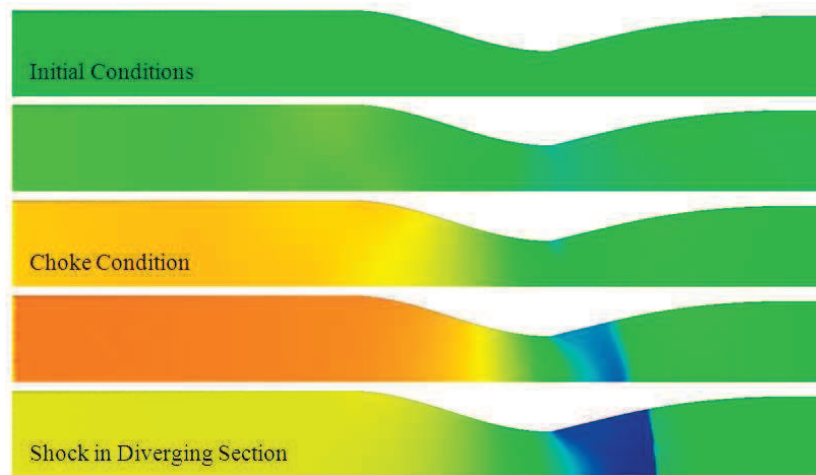


Figure 4.25: Development of Static Pressure Downstream of Rocket Inflow Plane.

4.2.4 Stability

Several instabilities arose in CFDsol during the course of this work. The outflow boundary was already demonstrated with the boundary conditions. Coupling between the explicit solid wall, symmetry planes, and other boundaries can deplete the velocity at intersections. Unique situations can arise where the domain solution is locally or absolutely unstable.

Solid Wall and Far Field. The rectangular domain in Figure 4.22 was modified to have solid walls on the top and bottom to simulate a wind tunnel. The velocity along the walls was then constrained to be parallel with freestream, leaving only one direction in which the outflow boundary can travel. The results from the wind tunnel simulation are shown in Figure 4.26. The iteration numbers are shown in the corner in white. The velocity at the limits of the outflow boundary perturbs slightly, and the solid walls eliminate all normal velocity. The perturbations continue until all of the velocity in the corners is reduced to zero. The conservation of momentum forms a “boundary layer” upstream of the outflow corners, even though the flow is inviscid. The slowed flow creates a nozzle around the airfoil. After 3000 iterations, the slowed

flow extends to the inflow boundary, which reacts with a checkerboard of properties. The checkerboard slowly propagates downstream, destroying the rest of the field.

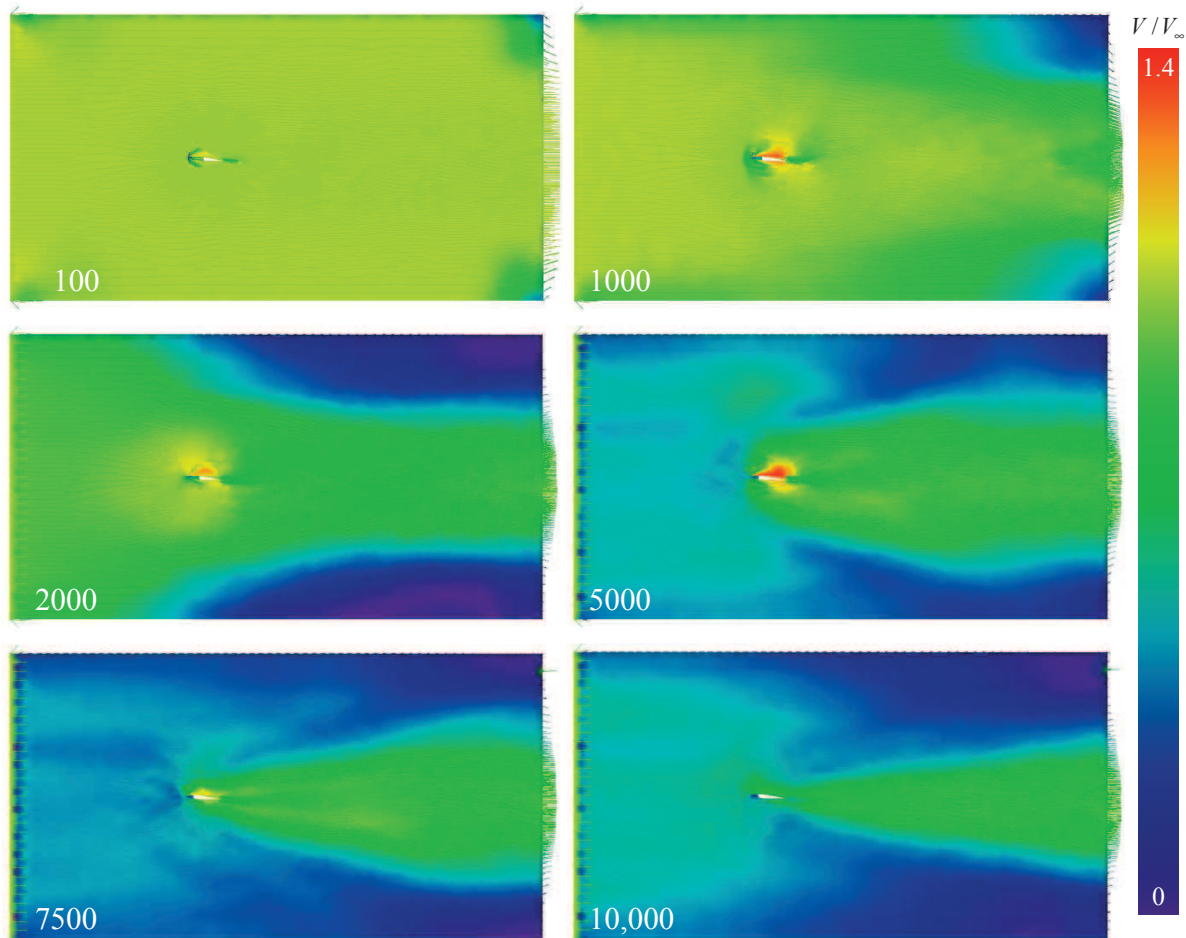


Figure 4.26: Velocity around NACA 0012 in “Wind Tunnel” (CFDsol).

Instabilities. Local instabilities were also found within the domain. These instabilities appear as small flutters in properties that appear and disappear like “dust floating in the air shown by sunlight through a window”. An example of these instabilities is shown in Figure 4.27. The relaxation factor τ can be used to increase the local stability. An “appropriately small τ ” is difficult to determine for all geometries. For most meshes, a τ of 0.5 is appropriate. Some

meshes have an undesirable coupling between flow characteristics and mesh spacing, which requires values of 0.1 to 0.01. This instability slows progress to a crawl.

The global time step $dtfix$ can be used to relax unsteady solutions. Several cases were generated by scaling down $dtfix$ until a stable solution occurred. Three meshes on different geometries were found that could not be stabilized. The domain had to be remeshed in a very different manner to achieve stability. For one two-dimensional case, the domain was one element wide, and the solution was only contained by the explicit symmetry planes.

4.2.5 Boundary Integrals

The majority of fluids research in finite element utilizes Gauss's theorem to shift the derivative off the flux term and create boundary integrals (Baker, 1983; Brenner, 2002; Thomasset, 1981; Thomee, 2006; Zienkiewicz, 2000). This method is utilized in the OSU codes, which shows enhanced stability over CFDsol. The differences between the two methods can be seen in Eqs. 4.188 and 4.189, which bring together the developments above:

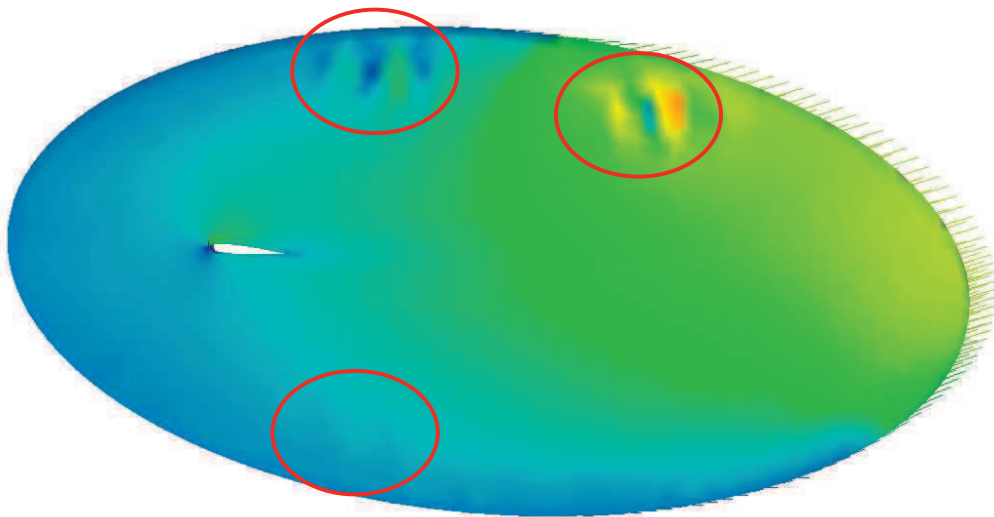


Figure 4.27: Oscillations in Pressure Distribution from CFDsol.

$$\mathbf{M}_c \Delta \mathbf{U}_e - \sum_e \frac{V_e \Delta t}{|J_e|} A_i^T \bar{\mathbf{F}}_i + \sum_{be} \Delta t [\mathbf{M}_{c,be}] (\mathbf{F}_{i,be} \cdot \hat{\mathbf{n}}_i) = \mathbf{K}_\sigma + \mathbf{f}_\sigma + \mathbf{S} \quad (4.188)$$

$$\mathbf{M}_c \Delta \mathbf{U}_e + \mathbf{K} \mathbf{U}_e + \sum_e \Delta t (\mathbf{f}_{1,e} + \mathbf{f}_{2,e}) = \mathbf{K}_\sigma + \mathbf{f}_\sigma + \mathbf{S} \quad (4.189)$$

Gupta's derivation (2007) produces a matrix-vector formulation that can be inverted using sparse storage and inversion techniques. Gauss's theorem can be applied to the governing equations (Eq. 4.188) and still arrive at an invertible matrix-vector formulation. The unsteady, viscous flux, and source terms are identical in CFDsol and the first-order temporal NS3D. The difference is in the inviscid flux terms. Gupta's derivation used the chain rule to split the inviscid flux into velocity, unknowns, and pressure terms. The inviscid flux can be similarly decomposed using Eq. 4.169. The average and normal fluxes are represented:

$$\bar{\mathbf{F}}_i = \bar{u}_{r,i} \bar{\mathbf{U}} + \bar{p} \begin{Bmatrix} 0 \\ \delta_{ij} \\ \bar{u}_{r,i} \\ 0 \end{Bmatrix} \quad \mathbf{F}_{i,be} \cdot \hat{\mathbf{n}}_i = (u_{r,i,be} \cdot \hat{\mathbf{n}}_i) \mathbf{U}_{be} + p_{be} \begin{Bmatrix} 0 \\ \hat{\mathbf{n}}_j \\ u_{r,i,be} \cdot \hat{\mathbf{n}}_i \\ 0 \end{Bmatrix} \quad (4.190)$$

If a Riemann invariant is used to bridge a discontinuous boundary, the inviscid flux needs to be written in terms of the invariant matrix \mathbf{A} , using Eq. 4.89 as an example:

$$\mathbf{F}_{n,c} = \frac{1}{2} \left((u_{r,i,be} \cdot \hat{\mathbf{n}}_i) \mathbf{U}_{be} + p_{be} \begin{Bmatrix} 0 \\ \hat{\mathbf{n}}_j \\ u_{r,i,be} \cdot \hat{\mathbf{n}}_i \\ 0 \end{Bmatrix} + \mathbf{F}_{inv,n,\infty} - [\mathbf{A}^*] (\mathbf{U}_{be} - \mathbf{U}_\infty) \right) \quad (4.191)$$

The inviscid terms in Eqs. 4.188 and 4.189 are equated, where \mathbf{K}^* and \mathbf{f}^* are created using Gauss's theorem instead of the derivative chain rule. Substituting Eqs. 4.190 and 4.191 and separating the unknowns \mathbf{U} and pressure terms, the inviscid flux terms are written:

$$\mathbf{M}_c \Delta \mathbf{U}_e + \Delta t \mathbf{K}^* \mathbf{U}_e + \Delta t \mathbf{f}^* = \mathbf{K}_\sigma + \mathbf{f}_\sigma + \mathbf{S} \quad (4.192)$$

$$\mathbf{K}^* \mathbf{U}_e + \mathbf{f}^* = - \sum_e \frac{V_e}{|J_e|} A_i^T \bar{\mathbf{F}}_i + \sum_{\substack{be \\ no RI}} [\mathbf{M}_{c,be}] (\mathbf{F}_{i,be} \cdot \hat{\mathbf{n}}_i) + \sum_{\substack{be \\ RI}} [\mathbf{M}_{c,be}] \mathbf{F}_{n,c} \quad (4.193)$$

$$\begin{aligned} \mathbf{K}^* \mathbf{U}_e = & - \sum_e \frac{V_e}{|J_e|} A_i^T \bar{\mathbf{u}}_{r,i} \frac{1}{4} \{1\}^T \mathbf{U}_e + \sum_{\substack{be \\ no RI}} [\mathbf{M}_{c,be}] (\bar{\mathbf{u}}_{r,i,be} \cdot \hat{\mathbf{n}}_i) \mathbf{U}_{be} \\ & + \sum_{\substack{be \\ RI}} \frac{1}{2} [\mathbf{M}_{c,be}] ((u_{r,i,be} \cdot \hat{\mathbf{n}}_i) [\mathbf{I}] - [\mathbf{A}^*]) \mathbf{U}_{be} \end{aligned} \quad (4.194)$$

$$\begin{aligned} \mathbf{f}^* = & - \sum_e \frac{V_e}{|J_e|} A_i^T \frac{1}{4} \{1\}^T p_e \begin{Bmatrix} 0 \\ \delta_{ij} \\ \bar{\mathbf{u}}_{r,i} \\ 0 \end{Bmatrix} + \sum_{\substack{be \\ no RI}} [\mathbf{M}_{c,be}] p_e \begin{Bmatrix} 0 \\ \hat{\mathbf{n}}_j \\ u_{r,i} \cdot \hat{\mathbf{n}}_i \\ 0 \end{Bmatrix} \\ & + \sum_{\substack{be \\ RI}} \frac{1}{2} [\mathbf{M}_{c,be}] \begin{Bmatrix} p_{be} \\ \begin{Bmatrix} 0 \\ \hat{\mathbf{n}}_j \\ u_{r,i,be} \cdot \hat{\mathbf{n}}_i \\ 0 \end{Bmatrix} \end{Bmatrix} + \mathbf{F}_{inv,n,BC} + [\mathbf{A}^*] \mathbf{U}_{BC} \end{aligned} \quad (4.195)$$

Using as much of the original derivation as possible, the new stiffness matrix and pressure vector can be written in terms of the inverse Jacobian A and its dot product with velocity \mathbf{T}_e :

$$\begin{aligned} \mathbf{K}^* = & - \sum_e \mathbf{K}_e^T + \sum_{\substack{be \\ no RI}} [\mathbf{M}_{c,be}] (\bar{\mathbf{u}}_{r,i,be} \cdot \hat{\mathbf{n}}_i) \mathbf{U}_{be} \\ & + \sum_{\substack{be \\ RI}} \frac{1}{2} [\mathbf{M}_{c,be}] ((u_{r,i,be} \cdot \hat{\mathbf{n}}_i) [\mathbf{I}] - [\mathbf{A}^*]) \mathbf{U}_{be} \end{aligned} \quad (4.196)$$

Eq. 4.196 contains the negative transpose of \mathbf{K}_e matrix (Eq. 4.181). The term constructed from the divergence of velocity and mass matrix (Eq. 4.174) has been replaced by boundary integrals terms. To incorporate smoothly into Gupta's original purpose, the boundary integrals have been developed as matrices, both with and without Riemann invariants.

$$\begin{aligned}
\mathbf{f}^* = & -\sum_e \frac{V_e}{|J_e|} \begin{Bmatrix} 0 \\ A_j^T \\ \mathbf{T}_e^T \\ 0 \end{Bmatrix} \bar{p} + \sum_{\substack{be \\ no\ RI}} [\mathbf{M}_{c,be}] p_e \begin{Bmatrix} 0 \\ \hat{n}_j \\ \mathbf{u}_{r,i} \cdot \hat{n}_i \\ 0 \end{Bmatrix} \\
& + \sum_{\substack{be \\ RI}} \frac{1}{2} [\mathbf{M}_{c,be}] \left(p_{be} \begin{Bmatrix} 0 \\ \hat{n}_j \\ \mathbf{u}_{r,i,be} \cdot \hat{n}_i \\ 0 \end{Bmatrix} + \mathbf{F}_{inv,n,BC} + [\mathbf{A}^*] \mathbf{U}_{BC} \right)
\end{aligned} \tag{4.197}$$

The first term in Eq. 4.197 resembles $\mathbf{f}_{1,e}$ (Eq. 4.182). $\mathbf{f}_{2,e}$ has been replaced by the boundary integral terms. The pressure terms in Eq. 4.197 can be implemented in a similar manner to $\mathbf{f}_{1,e}$. The energy contributions can be assembled as the dot product of the momentum terms and the velocity vector. And for Riemann boundaries, the boundary condition flux and properties are constant (after the rocket is ramped up).

Eqs. 4.196 and 4.197 have kept the matrix-vector form of the original CFDsol formulation while shifting the derivative from the properties to the shape function. The result is a constant gradient on the shape function, which has an increased stability over the property derivatives in the original derivation. Property derivatives suffer from numerical errors, as discussed in Section 4.5.3. Eq. 4.35 was implemented in CFDsol during testing, and the adapted implementation showed enhanced stability. (Eq. 4.35 was used to derive Eq. 4.196 and 4.197.) Eqs. 4.196 and 4.197 could not be fully implemented in CFDsol because of the limitations of the NASA contract. This section has been left as a record for anyone revisiting the numerical stability of CFDsol. For now, artificial dissipation is used to control any local instability. This artificial dissipation model is overly diffuse in order to control the perturbations and provide traditional algorithm stability.

The boundary integrals also incorporate the boundary conditions implicitly into the governing equations so that both the governing equations and boundary conditions are balanced in the same

system. The current implementation applies the boundary conditions between iterations of the governing equations to constrain the properties along the boundaries. The resulting residuals do not converge to machine precision. Experiments have shown that CFDsol only converges to 10^{-6} while the boundary integral implementation in Euler3D produces residuals on the order of 10^{-15} . (Both CFDsol and Euler3D are double precision codes.) Boundary integrals, although implied, produce a better convergence on the final solution, and Riemann invariants can be implemented to incorporate the natural characteristics into the boundary discontinuity, instead of simplified characteristic assumptions.

4.2.6 Artificial Dissipation

CFDsol uses a simple algebraic artificial dissipation model (Nithiarasu, 1998) to increase algorithm stability:

$$\mathbf{M}_c \Delta \mathbf{U}_e + \mathbf{K} \mathbf{U}_e + \sum_e \Delta t (\mathbf{f}_{1,e} + \mathbf{f}_{2,e}) = \mathbf{K}_\sigma + \mathbf{f}_\sigma + \mathbf{S} + \Delta t \hat{\mathbf{R}} \quad (4.198)$$

$$\hat{\mathbf{R}} = - \sum_e \frac{C_s S_e \tau}{\Delta t_e} \mathbf{M}_L^{-1} (\mathbf{M}_c - \mathbf{M}_L) \mathbf{U}_n \quad \Delta t_e = \frac{l_{min}}{V_{max} + a_{max}} \quad (4.199)$$

where C_s is the dissipation scalar, τ is the relaxation factor, and Δt is the local time step. l_{min} is the minimum characteristic length of the element calculate using the ratio of element volume V_e to face area A_f . When Δt is replaced by element l_{min} and speed, then the artificial dissipation resembles the spatial-flux terms (i.e. $V_i \partial \mathbf{U} / \partial x_i$) more closely than diffusion. S_e is the average pressure switch on the element, where the pressure switch S_i at the nodes is:

$$S_i = \frac{\left| \sum_e (p_i - p_k) \right|}{\sum_e |p_i - p_k|} \quad (4.200)$$

p_i is the pressure at the node of interest, and p_k is the pressure at nodes attached by segments.

4.2.6.1 Time Accuracy

The artificial dissipation model was derived around a local time step ($1/\Delta t_e$). The model was implemented correctly for steady solutions, but time accurate solutions use a global time step. If the global time step is used in Eqs. 4.198 and 4.199, then they cancel each other and the contributions are independent of time step size, which brackets performance of the solver. The solver is limited by numerical stability at large time steps and the artificial dissipation model dominates the solver at small time steps. Figure 4.28 shows the lift history for the same geometry with different time steps Δt and dissipation scalars C_s . The dissipation scalar C_s has the same effect on the solution as the inverse time step $1/\Delta t$.

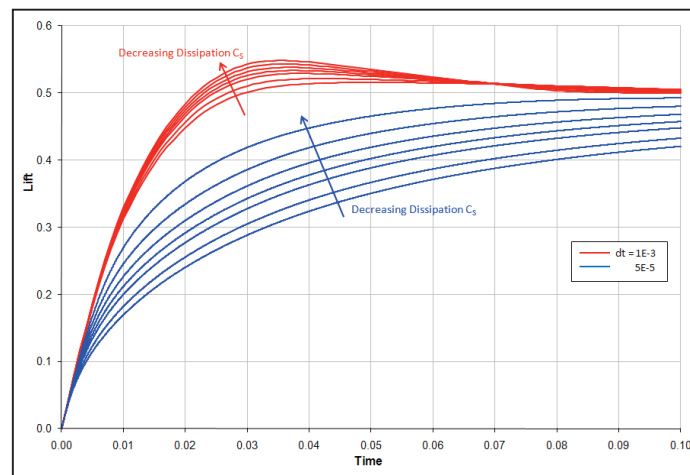


Figure 4.28: Unsteady Solution Accuracy vs. Time Step Δt and Dissipation C_s .

The implementation was adapted so that the time step in Eq. 4.199 is the elemental time step, and the global time step is used for Eq. 4.198. The proper applied of time steps allows the artificial dissipation $\hat{\mathbf{R}}$ to form as posed and then be scaled by the global time step. This was accomplished by retasking an unused array *DELTE* to store the element time steps. The nodal time step

Δt_n (local time step for steady and global time step for unsteady) is stored in *DELTP*. *CFDsol* averages the time step at the nodes to calculate the elemental time step Δt_e . The local time step (average over the element) is now stored in *DELTE* for both steady and unsteady schemes. The artificial dissipation routine uses *DELTE* for its time step. For smaller global time steps, the contributions from artificial dissipation to the residual are scaled down, just like the inviscid, viscous, and source terms. The steady solution is unchanged, and the unsteady solution can now be time accurate with artificial dissipation.

4.2.7 Solution Updates

Gupta's (2007) original formulation was intended to be solved through sparse matrix storage and inversion techniques. For the time being, the system is iterated. This section will discuss the time accuracy of the original implementation and the method for adapting the algorithm to Cowan's (2003) predictor-corrector to simulate time accurate solutions.

4.2.7.1 Consolidated Form

After simplifying all of the terms, Gupta (2007) collects similar terms and applies a Crank-Nicolson approach to the flux terms. Starting with Eq. 4.174, the unsteady matrices assemble into the global mass matrix \mathbf{M}_c , the inviscid flux matrices combine into a global stiffness matrix \mathbf{K} , and the remaining terms are lumped together into a single vector \mathbf{R} :

$$\mathbf{M}_c \Delta \mathbf{U} = -\mathbf{K} \mathbf{U}_e + \Delta t (\mathbf{R} + \hat{\mathbf{R}}) \quad (4.201)$$

$$\mathbf{M}_c = \sum_e \mathbf{M}_{c,e} \quad \mathbf{K} = \sum_e \Delta t \left(\frac{\partial u_{r,i}}{\partial x_{b,i}} \mathbf{M}_{c,e} + \mathbf{K}_e \right) \quad (4.202)$$

$$\Delta t \mathbf{R} = \mathbf{S} + \mathbf{K}_\sigma + \mathbf{f}_\sigma - \sum_e \Delta t (\mathbf{f}_{1,e} + \mathbf{f}_{2,e}) \quad (4.203)$$

The jump condition ΔU_e is replaced with $U_{n+1} - U_n$. The unknowns vector is averaged using the Crank-Nicolson method: $U_e = (U_{n+1} + U_n) / 2$. The system simplifies:

$$\mathbf{M}_c (\mathbf{U}_{n+1} - \mathbf{U}_n) = -\frac{1}{2} \mathbf{K} (\mathbf{U}_{n+1} + \mathbf{U}_n) + \Delta t (\mathbf{R} + \hat{\mathbf{R}}) \quad (4.204)$$

Eq. 4.204 is rearranged so that the previous unknowns U_n appear on the right side of the equation and the most current unknowns U_{n+1} are on the left side:

$$(\mathbf{M}_c + \frac{1}{2} \mathbf{K}) \mathbf{U}_{n+1} = (\mathbf{M}_c - \frac{1}{2} \mathbf{K}) \mathbf{U}_n + \Delta t (\mathbf{R} + \hat{\mathbf{R}}) \quad (4.205)$$

Gupta then renames the grouped matrices:

$$\mathbf{M}_+ \mathbf{U}_{n+1} = \mathbf{M}_- \mathbf{U}_n + \Delta t (\mathbf{R} + \hat{\mathbf{R}}) \quad (4.206)$$

$$\mathbf{M}_+ = \mathbf{M}_c + \frac{1}{2} \mathbf{K} \quad \mathbf{M}_- = \mathbf{M}_c - \frac{1}{2} \mathbf{K} \quad (4.207)$$

Gupta desired to invert the \mathbf{M}_+ matrix in the future using efficient methods for storing and inverting very sparse matrices. For the time being, the system is iterated until convergence. The \mathbf{M}_+ matrix is split into its diagonal \mathbf{D}_+ and off-diagonal \mathbf{M}'_+ :

$$(\mathbf{D}_+ + \mathbf{M}'_+) \mathbf{U}_{n+1} = \mathbf{M}_- \mathbf{U}_n + \Delta t (\mathbf{R} + \hat{\mathbf{R}}) \quad (4.208)$$

$$\mathbf{D}_+ \mathbf{U}_{n+1} = \mathbf{M}_- \mathbf{U}_n - \mathbf{M}'_+ \mathbf{U}_{n+1} + \Delta t (\mathbf{R} + \hat{\mathbf{R}}) \quad (4.209)$$

$$\mathbf{U}_{n+1}^{i+1} = \mathbf{D}_+^{-1} (\mathbf{M}_- \mathbf{U}_n - \mathbf{M}'_+ \mathbf{U}_{n+1}^i + \Delta t (\mathbf{R} + \hat{\mathbf{R}})) \quad \mathbf{U}_{n+1}^0 = \mathbf{U}_n \quad (4.210)$$

The equations are then iterated using Gauss iteration. In practice, Eq. 4.210 is unstable for most solutions. Gupta has used the solver to explicitly step to steady solutions.

4.2.7.2 Addition of Predictor-Corrector

Explicitly stepping steady solutions with Eq. 4.210 is accurate and stable for most meshes. For unsteady solutions, the explicit stepping of Eq. 4.210 is stable but not time accurate. Figure 4.29

shows the startup vortex traveling downstream from a suddenly accelerated airfoil. Figure 4.29 shows three time slices calculated by explicitly stepping Eq. 4.210. The vortex travels downstream at 2.55 times the freestream velocity. The increased speed of the vortex is due to a loss in effective “mass” in the governing equations, which allows the same “forces” to over accelerate that “mass”. Here, “force” refers to the advection and pressure terms, while “mass” refers to the volume weighted nature of the Galerkin equation.

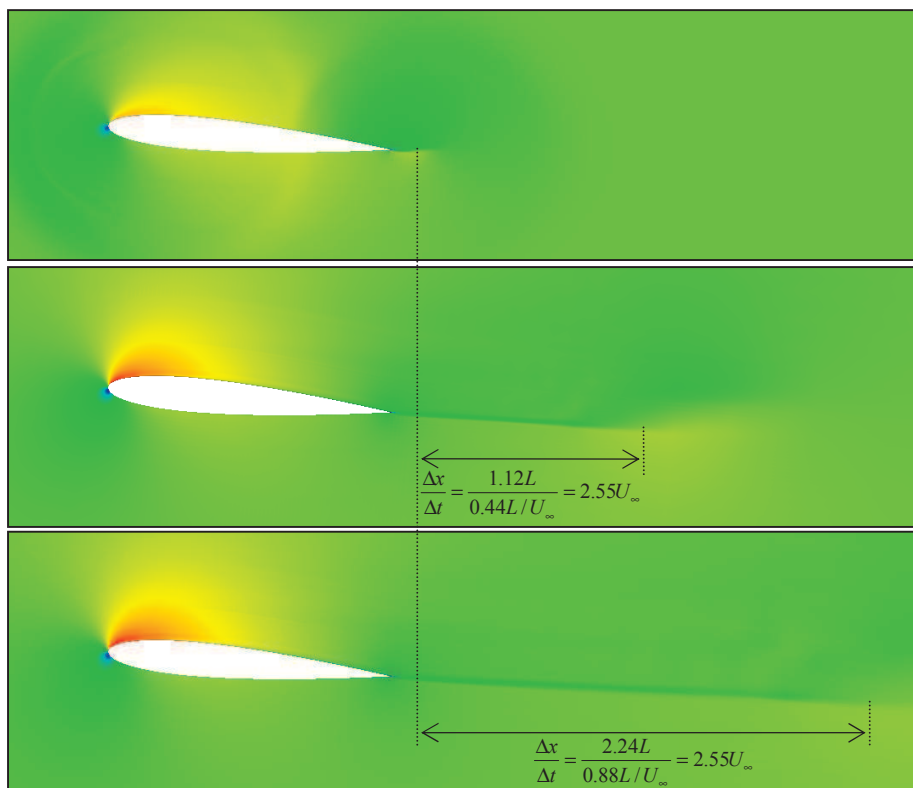


Figure 4.29: Inaccurate Motion of Wagner Vortex.

The over-acceleration can be illustrated using the mathematics of the governing FEM equations. To simplify the \mathbf{M}_+ and \mathbf{M}_- equations, we neglect advection so that only the unsteady and pressure terms are present in Eq. 4.210. \mathbf{M}_+ and \mathbf{M}_- are both simplified to \mathbf{M}_C . Further, we assume explicit iteration used so that $\mathbf{U}_{n+1} = \mathbf{U}_n$ on the right side of Eq. 4.210:

$$\mathbf{U}_{n+1} \approx \mathbf{D}_C^{-1}(\mathbf{M}_C \mathbf{U}_n - \mathbf{M}'_C \mathbf{U}_n + \Delta t(\mathbf{R} + \hat{\mathbf{R}})) = \mathbf{D}_C^{-1}(\mathbf{D}_C \mathbf{U}_n + \Delta t(\mathbf{R} + \hat{\mathbf{R}})) \quad (4.211)$$

$$\mathbf{U}_{n+1} \approx \mathbf{U}_n + \Delta t \mathbf{D}_C^{-1}(\mathbf{R} + \hat{\mathbf{R}}) \quad (4.212)$$

where \mathbf{D}_c and \mathbf{M}'_c are used to represent the diagonal and off-diagonal terms of the consistent mass matrix \mathbf{M}_c . The consistent mass matrix is shown in Eq. 4.21. The diagonal terms \mathbf{D}_c can be written in terms of the lumped mass matrix \mathbf{M}_L (Eq. 4.25):

$$\mathbf{D}_c = \frac{\rho_e}{20} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} = \frac{2\rho_e}{5 \cdot 4} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \frac{2}{5} \mathbf{M}_L \quad (4.213)$$

The diagonal terms \mathbf{D}_c carry 2/5 of the mass of the equivalent lumped mass matrix, which would give the flow roughly 40% of its effective mass. The changes created by pressure and dissipation would then create 2.5 times the acceleration on a given mass of air. This analysis only works well far from the airfoil, but the reduced mass exists closer to the airfoil. The diagonal terms of \mathbf{M}_+ contain \mathbf{D}_c and advection terms, which would change this estimate according to the mesh and advection properties.

Cowan (2003) uses a predictor-corrector algorithm to maintain the correct mass and apply the governing equations in implicit format. The FEM equations in CFDsol are reformulated:

$$\mathbf{R}_s(\mathbf{U}_{n+1}, \mathbf{U}_n) = \mathbf{M}_+ \mathbf{U}_{n+1} - \mathbf{M}_- \mathbf{U}_n - \Delta t(\mathbf{R} + \hat{\mathbf{R}}) \quad (4.214)$$

The methodology outlined in Section 4.1.13. The residual is minimized using Eqs. 4.156:

$$\mathbf{U}_{n+1}^{i+1} = \mathbf{U}_{n+1}^i - \mathbf{M}_L^{-1} \mathbf{R}_s(\mathbf{U}_{n+1}^i, \mathbf{U}_n) \quad (4.215)$$

The predictor-corrector method can be used in explicit or iterated for implicit form. Both forms utilize the complete system mass so that forcing is time-accurate. Explicit iteration is demon-

strated in Figure 4.30 for the Wagner problem, where the trailing vortex advects downstream at the freestream velocity.

4.3 Source Terms

Three source terms have been lumped together into \mathbf{S} for all five solvers. These source terms are addressed here for both CFDsol and the in-house codes at the same time. The non-inertial source term \mathbf{S}_{NI} is used to model the acceleration due to rotation and translation. The quasi-combustion source terms \mathbf{S}_C are used to model mass and heat generation due to combustion without modeling the chemistry or species relations. Finally, the turbulence models have been included in the previous equations, but only the advection and diffusion terms have been simplified. The remaining turbulent source terms \mathbf{S}_T are simplified here.

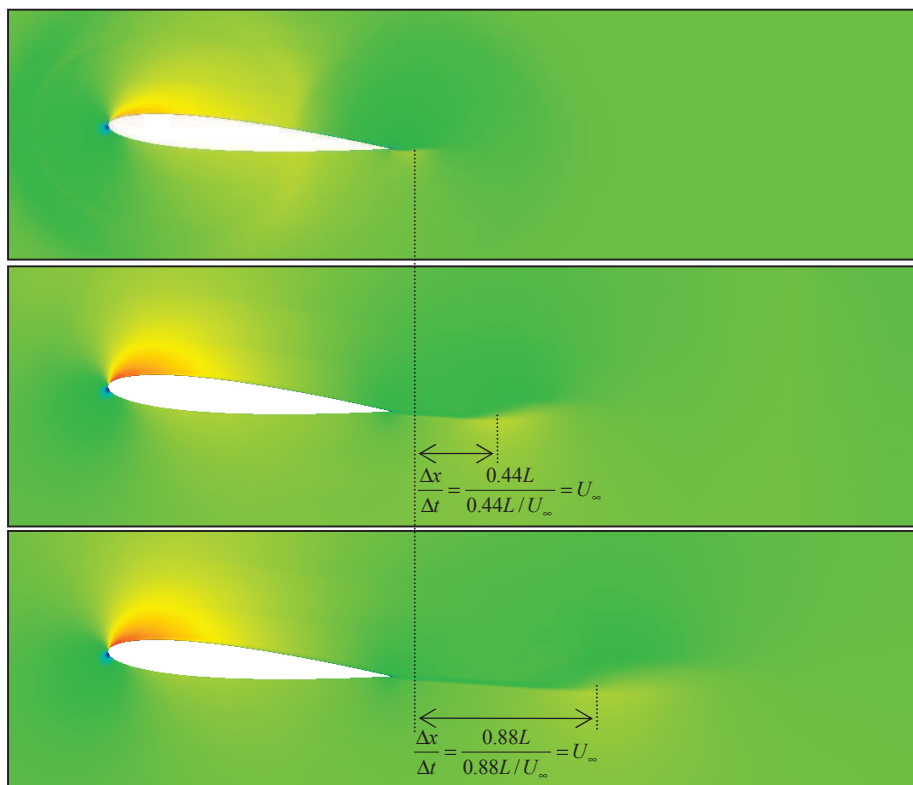


Figure 4.30: Corrected Motion of Wagner Vortex.

4.3.1 Non-Inertial Terms

The non-inertial source term Eqs. 4.30 and 4.36 is integrated using the techniques applied to the flux terms. Multiple Gauss points can be used in the quadrature using Eq. 4.26. If one Gauss point is used to integrate the source term, the integral simplifies:

$$\sum_e \Delta t_n \int_{\Omega_e} \Phi_e^T \mathbf{S}_{NI} d\Omega = \sum_e \Delta t_n \int_{0 \leq \xi \leq 1} \Phi_e^T \mathbf{S}_{NI} |\mathbf{J}_e| d\xi = - \sum_e \frac{\Delta t_n \Omega_e}{d+1} \{1\} \bar{\mathbf{S}}_{NI} \quad (4.216)$$

where $\bar{\mathbf{S}}_{NI}$ is the source term evaluated using average properties, accelerations, and velocities on the element.

Moffitt (2004) implemented these equations in NS2D. Sukraw (2008) experimented with translating and spinning cylinders using the non-inertial frame. Figure 4.31 shows a cylinder translating at a 45° angle with respect to the horizontal. Figure 4.31 was created using the non-inertial frame, moving at equal velocities in the x - and y -directions. The Reynolds number of the flow was changed by varying the velocity of the cylinder or the velocity of the reference frame. Figure 4.31 shows the drag coefficient calculated using the non-inertial frame in NS2D compared to empirical trends and experimental data. The drag coefficient matches very well for all Reynolds numbers shown.

Sukraw had less success with spinning cylinders in the non-inertial frame. The solutions had spurious oscillations in velocity (see Figure 4.32). Sukraw experimented with the type of artificial dissipation, $diss$, CFL , and Reynolds number Re all changed the oscillations, but the magnitude and frequencies did not scale with any of these factors. Sukraw proposed investigating the governing equations for development or implementation problems.

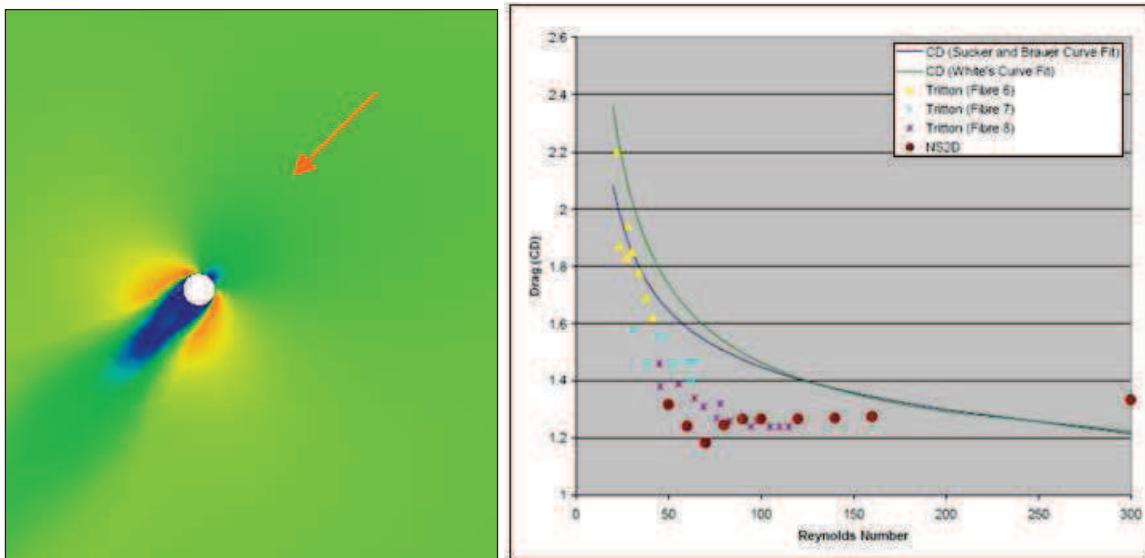


Figure 4.31: Cylinder Translating at 45° (left) and C_D vs. Re (right) (Sukraw, 2008).

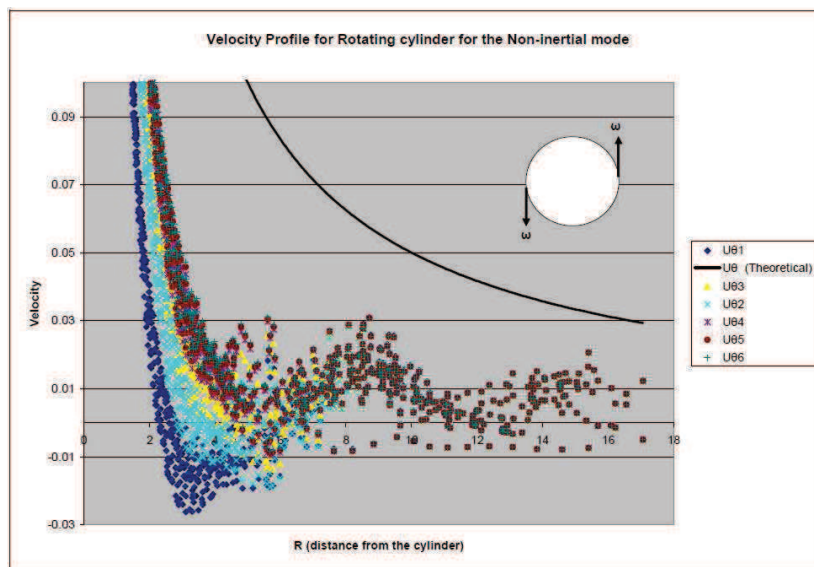


Figure 4.32: Oscillations in Velocity Profile around Spinning Cylinder (Sukraw, 2008).

The spinning cylinder was repeated in this research. Sukraw's mesh utilized a fine boundary layer mesh near the surface of the cylinder – a remnant of the mesh used on the translating cylinder. The domain was remeshed with a more coarse spacing. The mesh size was $0.04D$ normal to the cylinder surface and increased linearly with the radius, maintaining a constant

(inviscid) local time step. The global time step was selected to create viscous stability at the cylinder surface: $\Delta t = 2.2 \times 10^{-3}$ for $V_{wall} = 1$. The time step was scaled by 0.1 for the higher velocity ($V_{wall} = 10$).

Figure 4.33 shows the velocity profiles created at the two different rotational speeds. At the slower speed, the velocity profiles converge to the theoretical solution in 3 million iterations. At the faster speed, the velocity profile progresses more slowly and diverges after 290 thousand iterations. Just before the solution diverges, the profile begins oscillating like Sukraw's solution. The high speed case and Sukraw's test were run using local time steps greater than the viscous local time stepping limit.

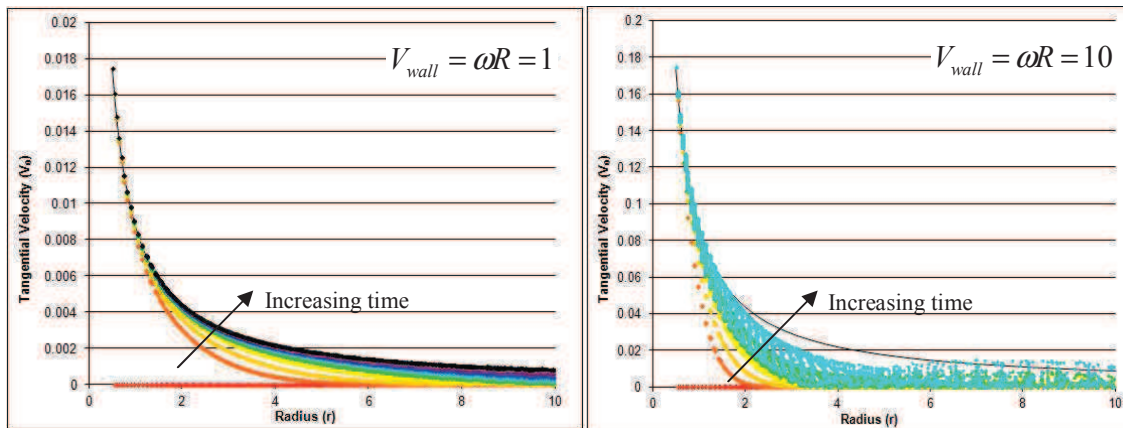


Figure 4.33: Circumferential Velocity Profile around Spinning Cylinder.

4.3.2 Turbulent Source Terms

The turbulent source term Eqs. 4.32 and 4.36 is integrated using quadrature with one Gauss point. is used to integrate the source term, the integral simplifies:

$$\sum_e \Delta t_n \int_{\Omega_e} \Phi_e^T \mathbf{S}_T d\Omega = \sum_e \Delta t_n \int_{0 \leq \xi \leq 1} \Phi_e^T \mathbf{S}_T |\mathbf{J}_e| d\xi = - \sum_e \frac{\Delta t_n \Omega_e}{d+1} \{1\} \bar{\mathbf{S}}_T \quad (4.217)$$

where $\bar{\mathbf{S}}_T$ is the source term evaluated using average properties on the element. The turbulent destruction terms in the SA and ω -equations have been rearranged so that conservative unknowns can be average where possible (Eq. 4.29). The primitive unknowns \hat{v} and ω are calculated from the conservative unknowns using the average density. The second term in Eq. 4.218 contains the stability and cross-diffusion terms, which are calculated on each element using derivatives from Eq. 4.13:

$$\bar{\mathbf{S}}_T = \left\{ \begin{array}{c} 0 \\ 0 \\ 0 \\ \left(\overline{f_{r1} c_{b1} \hat{S}} - \frac{c_{w1} \overline{f_v \hat{v}}}{d^2 \text{Re}} \right) \overline{\rho \hat{v}} \\ \overline{\rho \Pi_k} - \overline{\rho \mathcal{E}} \\ f_{r1} \overline{\rho \Pi_\omega} - \beta_c \left(\overline{\omega \rho \omega} - \overline{\rho} \omega_{amb}^2 \right) \end{array} \right\} + \left\{ \begin{array}{c} 0 \\ 0 \\ 0 \\ \frac{1}{\sigma \text{Re}} \nabla_i \rho \hat{v} \nabla_i \hat{v} \\ 0 \\ \frac{C_\omega}{\rho \omega} \nabla_i \rho K \nabla_i \rho \omega \end{array} \right\} \quad (4.218)$$

4.3.3 Quasi-Combustion Terms

In general, combustion is modeled by tracking the transfer of constituents from one species to another, calculating the energy produced in the process, and changing the properties to reflect the new constituents. The net effects of combustion increase the temperature and pressure. These effects are accomplished by adding mass, in the form of fuel, and enthalpy to the flow. Often momentum is transferred to the flow when fuel is added, but this momentum is small compared to the momentum change due to pressure and temperature effects. The model used here, termed “quasi-combustion”, only models the net effects of combustion on propulsion through the transfer of mass (density generation) and energy (total enthalpy generation) to the flow.

A scramjet engine cannot be designed using heat generation alone because the heat generation must be known *a priori* (Curran, 1996). But the heat generation is not known until the com-

bustion, which depends on the constituents in the flow field, is known. The “quasi-combustion” terms can be used to initially size the net energy exchange across the combustor and select a reasonable scramjet for testing. The scramjet would presumably be designed separate from the aircraft and then installed later in the design. Off-design characteristics (i.e., perturbed flight, acceleration, gust response, etc.) are tested with the entire vehicle.

A source term is added to the Euler equation to track the density and enthalpy generation in the combustor (Eq. 4.36). The source term takes on the form shown in Eq. 4.219:

$$\mathbf{S}_c = \begin{Bmatrix} \dot{\rho}_c \\ 0 \\ \dot{q}_c''' \\ 0 \\ 0 \\ 0 \end{Bmatrix} \quad \sum_e \Delta t_n \int_{\Omega_e} \Phi_e^T \mathbf{S}_c d\Omega = \sum_e \Delta t_n \int_{0 \leq \xi \leq 1} \Phi_e^T \mathbf{S}_c |\mathbf{J}_e| d\xi \quad (4.219)$$

The combustion source term is simplified in two ways, to accommodate different combustion models. The first method assumes that the source term is known at the nodes in the domain.

The source term is piecewise linearly distributed on the domain using Φ_e :

$$\sum_e \Delta t_n \int_{\Omega_e} \Phi_e^T \mathbf{S}_c d\Omega = \sum_e \Delta t_n \int_{\Omega_e} \Phi_e^T \Phi_e d\Omega \mathbf{S}_{c,e} = \sum_e \Delta t_n \mathbf{M}_{c,e} \mathbf{S}_{c,e} \quad (4.220)$$

For two- and three-dimensions, Eq. 4.220 becomes:

$$\sum_e \Delta t_n \int_{A_e} \Phi_e^T \Phi_e \mathbf{S}_{c,e} d\Omega = \sum_e \frac{\Delta t_n A_e}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \mathbf{S}_{c,e} \quad (4.221)$$

$$\sum_e \Delta t_n \int_{V_e} \Phi_e^T \Phi_e \mathbf{S}_{c,e} d\Omega = \sum_e \frac{\Delta t_n V_e}{20} \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix} \mathbf{S}_{c,e} \quad (4.222)$$

The second method assumes that the source term is known at the center of each element in the domain. The source term is considered piecewise constant on the domain:

$$\sum_e \Delta t_n \int_{\Omega_e} \Phi_e^T \mathbf{S}_C d\Omega = \sum_e \Delta t_n \int_{\Omega_e} \Phi_e^T \bar{\mathbf{S}}_C d\Omega = \sum_e \frac{\Delta t_n \Omega_e}{d+1} \{1\} \bar{\mathbf{S}}_C \quad (4.223)$$

For two- and three-dimensions, Eq. 4.223 becomes:

$$\sum_e \Delta t_n \int_{A_e} \Phi_e^T \bar{\mathbf{S}}_C d\Omega = \sum_e \frac{\Delta t_n A_e}{3} \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix} \bar{\mathbf{S}}_C \quad (4.224)$$

$$\sum_e \Delta t_n \int_{V_e} \Phi_e^T \bar{\mathbf{S}}_C d\Omega = \sum_e \frac{\Delta t_n V_e}{4} \begin{Bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{Bmatrix} \bar{\mathbf{S}}_C \quad (4.225)$$

This model has been tested in a quasi-one-dimensional afterburner simulation. The pressure, density, and Mach number distribution through the afterburner domain can be seen in the figures below. The first case, shown in Figure 4.34, represents a step distribution in both combustion values. The second case, shown in Figure 4.35, generates mass in the same step-wise manner, but rounds the sharp transition using cosine distributions. Both distributions calculate the correct outflow properties. In other words, the distribution does not affect the overall conservation, but sharp transitions cause the solver to apply artificial dissipation to smooth out the solution. The cosine transitions help reduce the amount of dissipation used by all of the solvers in this work.

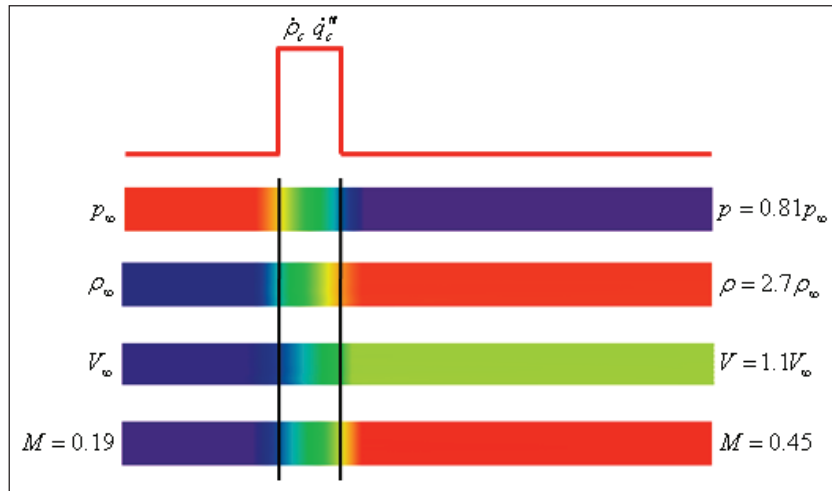


Figure 4.34: Afterburner Properties Simulated with a “Step” Generation.

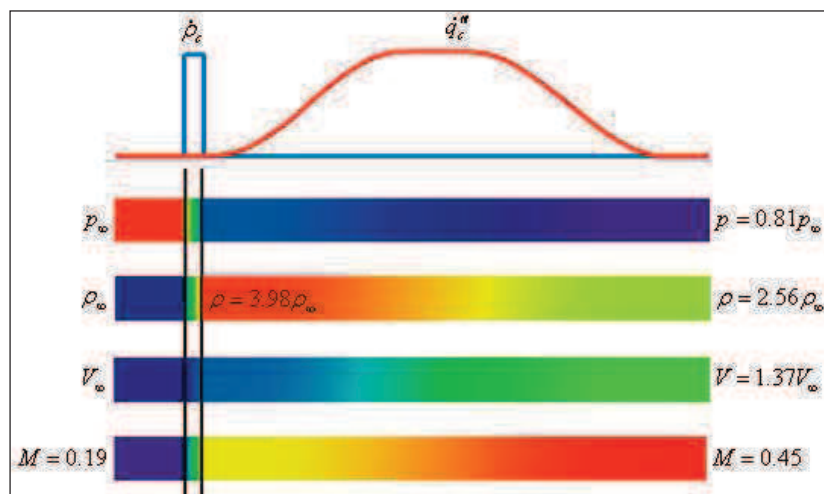


Figure 4.35: Afterburner Properties Simulated with a “Cosine” Generation.

The method uses the “air standard” for flow through and downstream of the combustion. The “air standard” says that the flow can be modeled using the properties of air, ignoring the changes in properties due to combustion. For example, the ratio of specific heats for air ($\gamma = 1.4$) is used to calculate the properties upstream of the combustor. If the “air standard” is maintained downstream of the combustor, the pressure on the aft section of a generic scramjet vehicle is

10.04 p_∞ . If the properties change during combustion ($\gamma = 1.33$), then the pressure on the aft section of the same vehicle is 10.81 p_∞ . The pressure has increased 7.6%. The “air standard” predicts a lower thrust on the vehicle, so the “air standard” can be used to make conservative design estimates, if used early in the design process.

4.4 Integration of Momentum

The aerodynamics and propulsive forces on an arbitrary body are formulated using the momentum exchange on a control volume. Figure 4.36 shows an engine, representing an arbitrary body with surface traction and momentum exchange through inlet and outlet. The aerodynamic and propulsive forces are summed together in the body force F_{body} . The engine is held in place by a support strut and creates a reaction force F_{react} at its attachment to the floor. The summation of aerodynamic and propulsive forces on the body F_{body} is equal and opposite to the reaction force F_{react} at the base of the support strut:

$$\vec{F}_{body} + \vec{F}_{react} = 0 \qquad \vec{F}_{body} = -\vec{F}_{react} \qquad (4.226)$$

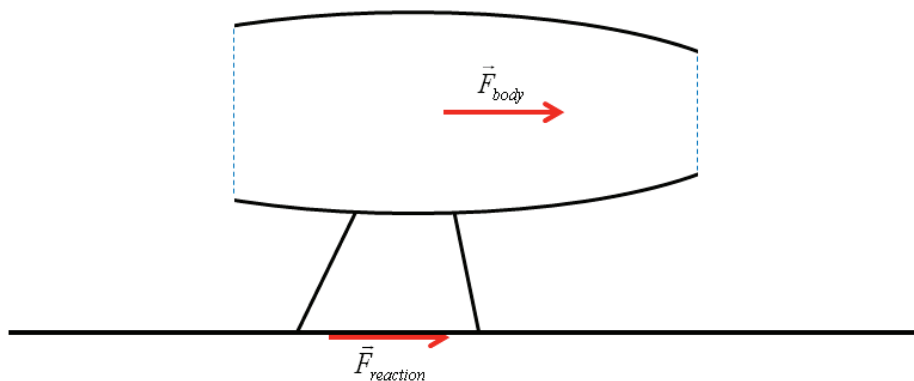


Figure 4.36: Body and Reaction Forces.

Figure 4.37 shows the engine and strut, surrounded by a control volume. The pressures and velocities are defined around the boundary of the control volume, and the reaction force F_{react} is shown at the base of the support strut. The velocity along the surface is assumed to be tangent to the surface in inviscid flow and zero at the bottom of a viscous boundary layer.

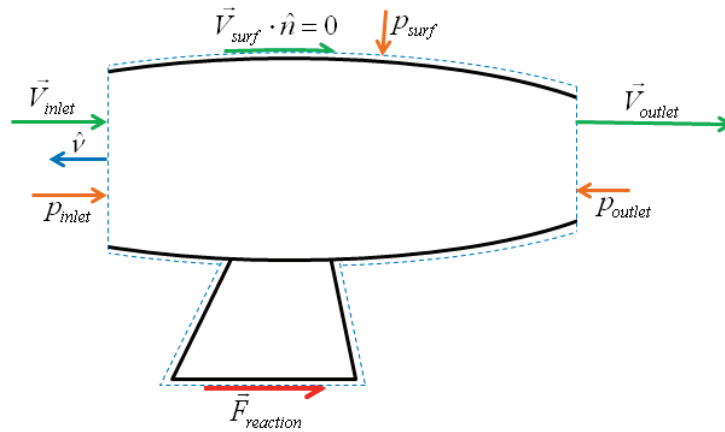


Figure 4.37: Control Volume around Body.

The momentum exchange on the control volume is calculated using Newton's second law (Munson, 1998):

$$\left(\frac{D(m\vec{V})}{Dt} \right)_{sys} = \sum \vec{F} \quad (4.227)$$

Applying Reynolds transport theorem and assuming quasi-steady flow:

$$\frac{\partial}{\partial t} \int_V \rho \vec{V} d\Omega + \int_S \rho \vec{V} (\vec{V} \cdot \hat{v}) d\Gamma = \vec{F}_{react} + \int_S \vec{T} d\Gamma \quad (4.228)$$

$$\vec{F}_{react} = \int_S \rho \vec{V} (\vec{V} \cdot \hat{v}) d\Gamma - \int_S \vec{T} d\Gamma \quad (4.229)$$

The traction is calculated from the surface stresses using Cauchy's equations (Allen, 1985):

$$\vec{T} = \begin{Bmatrix} T_x \\ T_y \\ T_z \end{Bmatrix} = \begin{Bmatrix} \sigma_{xx}\hat{v}_x + \sigma_{yx}\hat{v}_y + \sigma_{zx}\hat{v}_z \\ \sigma_{xy}\hat{v}_x + \sigma_{yy}\hat{v}_y + \sigma_{zy}\hat{v}_z \\ \sigma_{xz}\hat{v}_x + \sigma_{yz}\hat{v}_y + \sigma_{zz}\hat{v}_z \end{Bmatrix} = [\sigma]\hat{v} \quad (4.230)$$

where \hat{v} is the unit normal pointing out of the solid surface. The stress tensor for a fluid is composed of the pressure and viscous stresses:

$$[\sigma] = [\tau] - p[\mathbf{I}] = \begin{bmatrix} \tau_{xx} - p & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \tau_{yy} - p & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \tau_{zz} - p \end{bmatrix} \quad (4.231)$$

Notice that the viscous stress tensor is symmetric, so the overall stress tensor is also symmetric.

The traction created by a viscous fluid on a solid surface is calculated:

$$\vec{T} = ([\tau] - p[\mathbf{I}])\hat{v} = \begin{Bmatrix} (\tau_{xx} - p)\hat{v}_x + \tau_{xy}\hat{v}_y + \tau_{xz}\hat{v}_z \\ \tau_{xy}\hat{v}_x + (\tau_{yy} - p)\hat{v}_y + \tau_{yz}\hat{v}_z \\ \tau_{xz}\hat{v}_x + \tau_{yz}\hat{v}_y + (\tau_{zz} - p)\hat{v}_z \end{Bmatrix} \quad (4.232)$$

These variables are substituted into Eq. 4.229:

$$\vec{F}_{react} = \int_S \rho \vec{V} (\vec{V} \cdot \hat{v}) d\Gamma - \int_S ([\tau] - p[\mathbf{I}])\hat{v} d\Gamma \quad (4.233)$$

The force on the body is equal and opposite the reaction at the base. Taking into account the normals on the CFD domain are opposite the normals on the surface:

$$\vec{F}_{body} = \int_S \rho \vec{V} (\vec{V} \cdot \hat{n}) d\Gamma - \int_S ([\tau] - p[\mathbf{I}])\hat{n} d\Gamma \quad (4.234)$$

The moment on the body is created by integrating the force crossed with the distance to the center of mass:

$$M_{body} = \int_S \vec{r} \times \rho \vec{V} (\vec{V} \cdot \hat{n}) d\Gamma - \int_S \vec{r} \times ([\tau] - p[\mathbf{I}])\hat{n} d\Gamma \quad (4.235)$$

The generalized forces used in the aeroelastic calculations are calculated by integrating the force times the generalized displacement:

$$F_{A,i} = \int_S \vec{\Phi}_i \cdot \rho \vec{V} (\vec{V} \cdot \hat{n}) d\Gamma - \int_S \vec{\Phi}_i \cdot ([\tau] - p[\mathbf{I}]) \hat{n} d\Gamma \quad (4.236)$$

Dimensionless Form. All unknown properties are specified in non-dimensional form, so the force calculations should also be dimensionless. For the three-dimensional solvers, Eqs. 4.234 through 4.241 become:

$$\vec{F}_{body}^* = \frac{\vec{F}_{body}}{\frac{1}{2} \rho_\infty U_\infty^2 L_{ref}^2} = 2 \int_{S^*} \rho^* \vec{V}^* (\vec{V}^* \cdot \hat{n}) d\Gamma - 2 \int_{S^*} ([\tau^*] - p^* [\mathbf{I}]) \hat{n} d\Gamma \quad (4.237)$$

$$\vec{M}_{body}^* = \frac{\vec{M}_{body}}{\frac{1}{2} \rho_\infty U_\infty^2 L_{ref}^3} = 2 \int_{S^*} \vec{r}^* \times \rho^* \vec{V}^* (\vec{V}^* \cdot \hat{n}) d\Gamma - 2 \int_{S^*} \vec{r}^* \times ([\tau^*] - p^* [\mathbf{I}]) \hat{n} d\Gamma \quad (4.238)$$

$$F_{A,i}^* = \frac{F_{A,i}}{\frac{1}{2} \rho_\infty U_\infty^2 L_{ref}^3} = 2 \int_{S^*} \vec{\Phi}_i^* \cdot \rho^* \vec{V}^* (\vec{V}^* \cdot \hat{n}) d\Gamma - 2 \int_{S^*} \vec{\Phi}_i^* \cdot ([\tau^*] - p^* [\mathbf{I}]) \hat{n} d\Gamma \quad (4.239)$$

The domain volume of the two-dimensional solvers is strictly equal the area of integration times a unit thickness: $V = (A)(1)$. The boundary area is similarly the total length of boundary times a unit thickness: $S = (l) (1)$. The quantity S is replaced by l in Eqs. 4.237 through 4.239. One of the length dimensions L_{ref} is likewise removed from the denominator:

$$\vec{F}_{body}^* = \frac{\vec{F}_{body}}{\frac{1}{2} \rho_\infty U_\infty^2 L_{ref}} = 2 \int_{l^*} \rho^* \vec{V}^* (\vec{V}^* \cdot \hat{n}) d\Gamma - 2 \int_{l^*} ([\tau^*] - p^* [\mathbf{I}]) \hat{n} d\Gamma \quad (4.240)$$

$$\vec{M}_{body}^* = \frac{\vec{M}_{body}}{\frac{1}{2} \rho_\infty U_\infty^2 L_{ref}^2} = 2 \int_{l^*} \vec{r}^* \times \rho^* \vec{V}^* (\vec{V}^* \cdot \hat{n}) d\Gamma - 2 \int_{l^*} \vec{r}^* \times ([\tau^*] - p^* [\mathbf{I}]) \hat{n} d\Gamma \quad (4.241)$$

$$F_{A,i}^* = \frac{F_{A,i}}{\frac{1}{2} \rho_\infty U_\infty^2 L_{ref}^2} = 2 \int_{l^*} \vec{\Phi}_i^* \cdot \rho^* \vec{V}^* (\vec{V}^* \cdot \hat{n}) d\Gamma - 2 \int_{l^*} \vec{\Phi}_i^* \cdot ([\tau^*] - p^* [\mathbf{I}]) \hat{n} d\Gamma \quad (4.242)$$

Three-Dimensional FEM Form. The integrals above are reposed as a piece-wise element-by-element integral over the boundaries. (The star notation is dropped here, but all terms are presented in dimensionless form.) Eqs. 4.237 and 4.238 are discretized into the summation of forces and moments on each boundary element, using the average properties on each:

$$\begin{Bmatrix} F_{body,x} \\ F_{body,y} \\ F_{body,z} \end{Bmatrix} = \sum_{be} \begin{Bmatrix} \overline{F_x} \\ \overline{F_y} \\ \overline{F_z} \end{Bmatrix} \quad \begin{Bmatrix} M_{body,x} \\ M_{body,y} \\ M_{body,z} \end{Bmatrix} = \sum_{be} \begin{Bmatrix} \overline{r_y F_z - r_z F_y} \\ \overline{r_z F_x - r_x F_z} \\ \overline{r_x F_y - r_y F_x} \end{Bmatrix} \quad (4.243)$$

$$\overline{F_x} = 2A_{be}(\overline{\rho u} \overline{V_n} - \overline{T_x}) \quad \overline{T_x} = (\tau_{xx} - \overline{p})\hat{n}_x + \tau_{xy}\hat{n}_y + \tau_{xz}\hat{n}_z \quad (4.244)$$

$$\overline{F_y} = 2A_{be}(\overline{\rho v} \overline{V_n} - \overline{T_y}) \quad \overline{T_y} = \tau_{xy}\hat{n}_x + (\tau_{yy} - \overline{p})\hat{n}_y + \tau_{yz}\hat{n}_z \quad (4.245)$$

$$\overline{F_w} = 2A_{be}(\overline{\rho w} \overline{V_n} - \overline{T_z}) \quad \overline{T_z} = \tau_{xz}\hat{n}_x + \tau_{yz}\hat{n}_y + (\tau_{zz} - \overline{p})\hat{n}_z \quad (4.246)$$

Similarly, the generalized force (Eq. 4.239) for elastic deflections is calculated:

$$F_{A,i} = \sum_{be} (\overline{\Phi_{x,i} F_x} + \overline{\Phi_{y,i} F_y} + \overline{\Phi_{z,i} F_z}) \quad (4.247)$$

Two-Dimensional FEM Form. Eqs. 4.240 and 4.241 are similarly discretized on the boundary edges and rewritten in terms of average properties on each boundary edge:

$$\begin{Bmatrix} F_{body,x} \\ F_{body,y} \end{Bmatrix} = \sum_{be} \begin{Bmatrix} \overline{F_x} \\ \overline{F_y} \end{Bmatrix} \quad M_{body} = \sum_{be} (\overline{r_x F_y} - \overline{r_y F_x}) \quad (4.248)$$

$$\overline{F_x} = 2l_{be}(\overline{\rho u} \overline{V_n} - \overline{T_x}) \quad \overline{T_x} = (\tau_{xx} - \overline{p})\hat{n}_x + \tau_{xy}\hat{n}_y \quad (4.249)$$

$$\overline{F_y} = 2l_{be}(\overline{\rho v} \overline{V_n} - \overline{T_y}) \quad \overline{T_y} = \tau_{xy}\hat{n}_x + (\tau_{yy} - \overline{p})\hat{n}_y \quad (4.250)$$

$$F_{A,i} = \sum_{be} (\overline{\Phi_{x,i} F_x} + \overline{\Phi_{y,i} F_y}) \quad (4.251)$$

Flat Plate Example: The first example is a flat plate with chord 2, span 1, and negligible thickness. The pressure on the top and bottom surfaces of the plate are 1 and 2, respectively. The velocity along the plate is parallel to the plate. The surface normals are represented:

$$\hat{n}_{top}^T = \{ 0 \quad -1 \quad 0 \} \quad \hat{n}_{bot}^T = \{ 0 \quad 1 \quad 0 \}$$

With a Reynolds number 10, the flow has a viscosity of 0.1. The velocity gradients and shear stress at the wall are calculated:

$$\frac{\partial u^*}{\partial y^*} = \begin{cases} 1 & \text{for } y^* > 0 \\ -1 & \text{for } y^* < 0 \end{cases} \quad \tau_{xy}^* = \frac{1}{\text{Re}_L} \frac{\partial u^*}{\partial y^*} = \begin{cases} 0.1 & \text{for } y^* > 0 \\ -0.1 & \text{for } y^* < 0 \end{cases}$$

The pressure and shear stress creates the load (Eq. 4.237):

$$\vec{F}_{body}^* = 2 \int_{S^*} \rho^* \vec{V}^* (\vec{V}^* \cdot \hat{n}) d\Gamma - 2 \int_{S^*} ([\tau^*] - p^* [\mathbf{I}]) \hat{n} d\Gamma$$

$$\vec{F}_{body}^* = -2 \int_{S^*} \begin{bmatrix} -p^* & \tau_{xy}^* & 0 \\ \tau_{xy}^* & -p^* & 0 \\ 0 & 0 & -p^* \end{bmatrix} \hat{n} d\Gamma$$

$$\vec{F}_{body}^* = -2 \begin{bmatrix} -p_{top}^* & \tau_{xy,top}^* & 0 \\ \tau_{xy,top}^* & -p_{top}^* & 0 \\ 0 & 0 & -p_{top}^* \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} cb - 2 \begin{bmatrix} -p_{bot}^* & \tau_{xy,bot}^* & 0 \\ \tau_{xy,bot}^* & -p_{bot}^* & 0 \\ 0 & 0 & -p_{bot}^* \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} cb$$

$$\vec{F}_{body}^* = 2 \begin{bmatrix} \tau_{xy,top}^* - \tau_{xy,bot}^* \\ p_{bot}^* - p_{top}^* \\ 0 \end{bmatrix} cb = 2 \begin{bmatrix} 0.1 - (-0.1) \\ 2 - 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 4 \\ 0 \end{bmatrix}$$

The viscous terms create a non-zero drag while the difference in pressure creates lift.

If the center of rotation is placed at the port side of the leading edge of the plate, the moment is calculated (Eq. 4.238):

$$\vec{M}_{body}^* = 2 \int_{S^*} \vec{r}^* \times \rho^* \vec{V}^* (\vec{V}^* \cdot \hat{n}) d\Gamma - 2 \int_{S^*} \vec{r}^* \times ([\tau^*] - p^* [\mathbf{I}]) \hat{n} d\Gamma = \int_{S^*} \vec{r}^* d\Gamma \times \frac{\vec{F}_{body}^*}{2x1}$$

$$\vec{M}_{body}^* = \int_{-b0}^0 \int_0^c \begin{Bmatrix} x^* \\ 0 \\ z^* \end{Bmatrix} dx^* dz^* \times \frac{\vec{F}_{body}^*}{2x1} = \begin{Bmatrix} 2 \\ 0 \\ -1 \end{Bmatrix} \times \frac{1}{2} \begin{Bmatrix} 0.8 \\ 4 \\ 0 \end{Bmatrix} = \begin{Bmatrix} 2 \\ -0.4 \\ 4 \end{Bmatrix}$$

The plate generates a rolling, pitching, and yawing moment about its port corner. The yaw moment is created by the viscous shear stress while the pitch and roll moments come from the difference in pressure across the plate (inviscid).

This case was tested in Euler3D using a plate of negligible thickness (0.01), chord 2, and unit span. The center of rotation (origin) was specified at the centerline, port side of the leading edge. All surfaces of the plate were specified as inviscid walls. The pressure domain was specified as initial conditions: The pressure above and below the centerline of the domain (plate) were specified as 1 and 2, respectively. All other properties were specified using freestream conditions. Euler3D predicted a lift and pitching moment of 4, roll moment of 2, and zero yawing moment, matching the pressure portion of the results above. (The sides of the plate create errors in force on the order of 10^{-3} and in the moment on the order of 10^{-2} .)

This case was also tested in NS3D using the same geometry and center of rotation (origin). The top and bottom surfaces of the plate were specified as viscous walls; the other surfaces were specified as inviscid walls to minimize numerical errors. The pressure and velocity domains were specified as initial conditions: The pressure above and below the centerline of the domain

(plate) were specified as 1 and 2, respectively. The velocity above and below the centerline were specified as $u = 1 + y$ and $u = 1 - y$, respectively. All other properties were specified using freestream conditions. NS3D predicted a lift and pitching moment of 4, drag due to viscous shear of 0.8, roll moment of 2, and yaw moment of -0.4, matching the results above. (The sides of the plate create errors in force on the order of 10^{-3} and in the moment on the order of 10^{-2} . Making the inviscid walls into viscous walls skews all of the results.)

Engine Example: The second example is a rectangular engine with 1x1 cross-section and length 2. The pressure on the inlet, outlet, and solid surfaces of the engine are freestream pressure so that the pressure on either side of the engine cancels out the effects from the opposing side. The inflow velocity is unity, and the outflow velocity is 4. The inlet and outlet normals are represented:

$$\hat{n}_{in}^T = \{ 1 \quad 0 \quad 0 \} \qquad \hat{n}_{out}^T = \{ -1 \quad 0 \quad 0 \}$$

The force on the engine is calculated (Eq. 4.237):

$$\vec{F}_{body}^* = 2 \int_{S^*} \rho^* \vec{V}^* (\vec{V}^* \cdot \hat{n}) d\Gamma + 2 \int_{S^*} p^* \hat{n} d\Gamma$$

$$\vec{F}_{body}^* = 2\rho_{in}^* \vec{V}_{in}^* (\vec{V}_{in}^* \cdot \hat{n}_{in}) A_{in}^* + 2\rho_{out}^* \vec{V}_{out}^* (\vec{V}_{out}^* \cdot \hat{n}_{out}) A_{out}^*$$

$$\vec{F}_{body}^* = 2(1) \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} \left(\begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} \cdot \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} \right) (1x1) + 2(1) \begin{Bmatrix} 4 \\ 0 \\ 0 \end{Bmatrix} \left(\begin{Bmatrix} 4 \\ 0 \\ 0 \end{Bmatrix} \cdot \begin{Bmatrix} -1 \\ 0 \\ 0 \end{Bmatrix} \right) (1x1) = \begin{Bmatrix} -30 \\ 0 \\ 0 \end{Bmatrix}$$

The engine generates a negative drag (or thrust). If the center of rotation is placed at the lower-port corner of the leading edge of the engine, the moment is calculated (Eq. 4.238):

$$\vec{M}_{body}^* = 2 \int_{S^*} \vec{r}^* \times \rho^* \vec{V}^* (\vec{V}^* \cdot \hat{n}) d\Gamma + 2 \int_{S^*} \vec{r}^* \times p^* \hat{n} d\Gamma = - \int_{S^*} \vec{r}^* d\Gamma \times \frac{\vec{F}_{body}^*}{1 \times 1}$$

$$\vec{M}_{body}^* = \int_{-1}^0 \int_0^1 \begin{Bmatrix} 0 \\ y^* \\ z^* \end{Bmatrix} dy^* dz^* \times \frac{\vec{F}_{body}^*}{1 \times 1} = \begin{Bmatrix} 0 \\ \frac{1}{2} \\ -\frac{1}{2} \end{Bmatrix} \times \begin{Bmatrix} -30 \\ 0 \\ 0 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 15 \\ 15 \end{Bmatrix}$$

The engine generates a pitching and yawing moment about its lower-port corner.

This case was tested in Euler3D using an engine with 1x1 inlet/outlet and length 2. The center of rotation (origin) was specified at the lower, port, leading corner. The inlet and outlet were specified using engine BCs, and the nacelle skins were specified as inviscid walls. The velocity domain was specified as initial conditions: The velocity forward and aft of the mid-length were specified as $u = 1$ and $u = 4$, respectively. All other properties were specified using freestream conditions. Euler3D predicted a drag of -30 (or thrust of 30), pitching and yawing moments of 15, matching the results above.

4.5 Local Time Stepping

The residual in Eq. 4.36 has been scaled by the time step. If an appropriate local time step is chosen in place of the global time step, the preconditioned residual can be used to accelerate the convergence of steady solutions – called *local time stepping*. Cowan (2003) uses the local time step to speed up convergence where possible and relax convergence where needed. The unsteady term in Eq. 4.36 includes the ratio of local time step over global time step: $\Delta t_n / \Delta t$. The ratio acts like a relaxation factor. In fact, Cowan restricts the local time step to be less than or equal the global time step so that the ratio always acts like a relaxation factor. This idea of local time stepping will be extended here to viscous flows to enhance stability.

4.5.1 Inviscid Local Time Step

The inviscid local time step is selected to be the time required for the local flow to cross a fluid element. The question often arises as to which portion of the element the fluid is crossing.

Many locations can be chosen. If the fluid velocity vector crosses along the edge of a side, then the distance across the element is the length of that edge. If the fluid crosses at the opposing vertex with the same vector, then the distance is mathematically zero.

Cowan implemented a local time stepping routine that calculates the local velocity in the direction of each segment and then calculates the time for the local flow to cross that segment. The maximum velocity in the local flow is calculated using the fastest characteristic. From the Riemann problem (Eq. 4.54), the fastest characteristic is $V+a$. Cowan's routine is based in the weighted segments used in the artificial dissipation model (Eq. 4.148). The algorithm is presented in its conceptual form here:

$$\Delta t_i = MIN \left(\frac{CFL |\bar{x}_i - \bar{x}_j|}{(\bar{x}_i - \bar{x}_j) \cdot \bar{V}_{r,j} + a_j} \right) \quad (4.252)$$

where the subscript i denotes the node of interest and the subscript j denotes the nodes attached to node i through segments. The velocity and acoustic speed are taken from the attached node to calculate the time taken to transmit information from the attached node to the node of interest.

The use of weighted segments allows Cowan to construct the local time step in a much more continuous manner than the minimum function in Eq. 4.252. (The weighted segments have been avoided here because they obscure the concept.) Cowan scales the local time step by CFL (or the Courant-Fredrick-Lewis, originators of the concept of local time stepping) number CFL to relax the local time step. For inviscid solutions, CFL number less than 0.7 are suggested,

although some cases have proven to be stable up to a CFL of 0.8. A nominal value of 0.5 is generally used as a safety cushion.

4.5.2 Inviscid vs. Viscous Stability

Test data from NS2D and CFDsol have shown that there are strict inviscid and viscous stability limits. The inviscid stability limit is more stringent for element Reynolds numbers greater than half: $Re_{\Delta x} = \rho_{\infty} u_{\infty} \Delta x / \mu_{\infty} = Re_L \Delta x / L < 0.5$. The viscous stability is more stringent for element Reynolds numbers less than half. These limits are illustrated in Figure 4.38 for both NS2D and CFDsol. *CFL* in Figure 4.38 represents the ratio of maximum stable time step to local inviscid time step: $CFL = \Delta t_{max} / \Delta t_{inv}$. NS2D was tested on flat plate, cylinder, and other simple geometries. Figure 4.38 shows that NS2D is limited by inviscid stability for $Re_{\Delta x} > 0.5$ and viscous stability below, and the viscous stability decreases proportional to the element Reynolds number. CFDsol was tested on a isotropic mesh on a rectangular domain. The same trends are seen in Figure 4.38 for CFDsol where the viscous stability limit occurs at $Re_{\Delta x} = 200$.

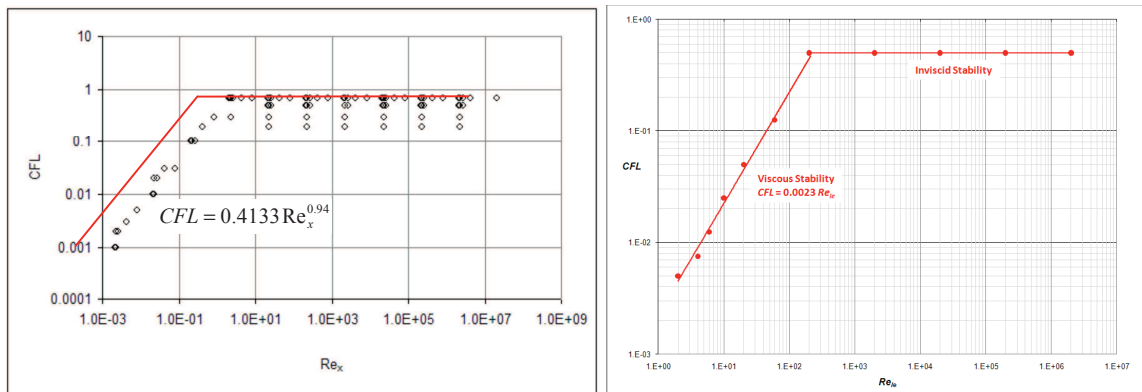


Figure 4.38: Stable CFL Values vs. Element Reynolds Number Re_x (left – Moffitt, 2004; right – CFDsol).

A rectangular isotropic domain was used to qualitatively explore the stability of NS2D, NS3D, and CFDsol under various conditions. The 3D solvers were found to be less stable than

NS2D, likely caused by the added degree-of-freedom. The SA model was found to be more stable than the RANS equations, where the velocity distribution diverged before the turbulence model. Conduction was tested on the same mesh with zero velocity between viscous walls. The heat transfer boundary conditions were shown to have little effect on stability, but the magnitude of Prandtl number Pr was shown to directly affect the stability of the conduction problem. Using air properties, the conduction problem was less stable than RANS or SA with a freestream.

4.5.3 Numerical Error in Derivatives

Similar instabilities were found in both NS3D and CFDsol when modeling a laminar boundary layer along the plate. The boundary layer was modeled in the xy -plane, and the instability was tracked to the viscous stresses τ_{yz} and τ_{zz} . (The solver becomes stable when these derivatives are neglected in NS3D.) These stresses were generated by numerical errors in the cross flow velocity w . The three z -stresses are shown below (assuming $\nabla \cdot \vec{V} = 0$):

$$\tau_{xz} = \frac{\mu}{\text{Re}} \left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \right) \quad \tau_{yz} = \frac{\mu}{\text{Re}} \left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \right) \quad \tau_{zz} = \frac{\mu}{\text{Re}} \left(2 \frac{\partial w}{\partial z} \right) \quad (4.253)$$

If the flow is truly two-dimensional, then the z -derivatives should vanish, leaving the derivatives containing w . If w is modeled as a zero mean with random numerical perturbations on the order of machine precision ($w \sim \delta_w$), then the z -stresses can be modeled:

$$\tau_{xz} \approx \frac{\mu}{\text{Re}} \left(\frac{\partial \delta_w}{\partial x} \right) \quad \tau_{yz} \approx \frac{\mu}{\text{Re}} \left(\frac{\partial \delta_w}{\partial y} \right) \quad \tau_{zz} \approx \frac{\mu}{\text{Re}} \left(2 \frac{\partial \delta_w}{\partial z} \right) \quad (4.254)$$

Under the best of conditions, random perturbations would cancel each other so that all three stresses become zero. In the worst case scenario in this example, two perturbations have equal

magnitude δ_w with opposite sign, and these two perturbations are used to evaluate the derivative and therefore the flux. Considering the worst case as the upper limit, Eq. 4.254 becomes:

$$\tau_{xz} \leq \frac{\mu}{\text{Re}} \left(\frac{2\delta_w}{\ell_x} \right) \quad \tau_{yz} \leq \frac{\mu}{\text{Re}} \left(\frac{2\delta_w}{\ell_y} \right) \quad \tau_{zz} \leq \frac{\mu}{\text{Re}} \left(\frac{4\delta_w}{\ell_z} \right) \quad (4.255)$$

where ℓ_x , ℓ_y , and ℓ_z are the characteristic lengths of the element used to evaluate their respective derivatives. Eq. 4.255 shows that problems with numerical errors are greatest on the smallest elements in the domain, which is supported by the observations in NS3D and CFDsol. The large elements far from the body experience the least problems from numerical errors in derivatives.

Returning to the two-dimensional boundary layer case that began this discussion, Eq. 4.255 shows that fluctuations in δ_w feedback into the three momentum equations and energy equation. The effects on the x - and y -momentum are minimized because the streamwise contributions are much larger than the terms seen in Eq. 4.255. The z -momentum is altogether susceptible to the numerical error in the z -derivatives, and the w -velocity is adapted to balance the z -momentum equation. The w -velocity resulting from this balance is not promised to be smooth because the z -derivatives are not smooth. Instead, the feedback of w -velocity into the stresses in Eq. 4.253 often aggravates the problem. Locking the w -velocity to a zero value eliminates the problem in the two-dimensional case (and not in general) and allows a similar error to be seen in the energy equations. The viscous dissipation created by the z -stresses is estimated in Eq. 4.256 with fluctuations in the w -velocity δ_w . The terms in Eq. 4.256 keep the energy residual from converging to a desirable level. The energy residual is used to measure convergence on a time step and overall.

$$\Phi_z = u_r \tau_{xz} + v_r \tau_{yz} + w_r \tau_{zz} \leq \frac{2\mu}{\text{Re}} \left(\frac{u_r \delta_w}{\ell_x} + \frac{v_r \delta_w}{\ell_y} + \frac{2\delta_w^2}{\ell_z} \right) \quad (4.256)$$

All of these problems can be eliminated if derivatives can be grouped into realistic estimates and numerical errors. A gradient filter has been created in `grad_filter.f` but not tested as an appropriate means of limiting gradients in NS3D. The gradient limiter uses Eq. 4.257 to create a lower limit for the gradient of a property p on an element with characteristic length ℓ_i :

$$\frac{\partial p}{\partial x_i} \geq \frac{2\delta_p}{\ell_i} \quad (4.257)$$

The method has not been tested in NS3D because appropriate values for ℓ_i and δ_p have not yet been determined. A characteristic length *could* be the minimum distance across an element, and δ_p *might* be estimated using 10^{-15} times the freestream property. For example, an enthalpy derivative on an element with minimum size 10^{-5} would be limited by $2 \times 10^{-10} h_\infty$. Derivatives below this limit *would* be eliminated. These ideas need testing before full implementation.

The application of the Gauss's theorem was thought to create or magnify the sensitivity of the algorithm to numerical derivatives. This idea is explored in the next section. The problem is thought to be contained at present through the use of viscous local time stepping, which is developed in the final sections of this chapter as a means of increasing the local stability.

4.5.4 Viscous Local Time Step

The derivation of local viscous time stepping is much more complex. The residual (Eq. 4.36) is rewritten in a stability format, where all of the unknowns have been pulled out of matrices:

$$\mathbf{R}(\mathbf{U}_{n+1}, \mathbf{U}_n) = [\mathbf{M}_C] \Delta \mathbf{U} + \Delta t [\mathbf{K}_{inv}] \mathbf{U}_{n+1} + \Delta t [\mathbf{K}_{vis}] \mathbf{U}_{n+1} - \Delta t \mathbf{S} - \Delta t \mathbf{D} \quad (4.258)$$

$$[\mathbf{K}_{inv}] \mathbf{U}_e^{i+1,k} = \sum_{be} \int_{\Gamma_{be}} \Phi_{be}^T (\mathbf{F}_i \cdot \hat{\mathbf{n}}_i) d\Gamma - \sum_e \int_{\Omega_e} \nabla_i \Phi_e^T \mathbf{F}_i d\Omega \quad (4.259)$$

$$[\mathbf{K}_{vis}] \mathbf{U}_e^{i+1,k} = \sum_e \int_{\Omega_e} \nabla_i \Phi_e^T \mathbf{F}_{v,i} d\Omega - \sum_{be} \int_{\Gamma_{be}} \Phi_{be}^T (\mathbf{F}_{v,i} \cdot \hat{\mathbf{n}}_i) d\Gamma - \sum_e \int_{\Omega_e} \Phi_e^T \mathbf{S}_v d\Omega \quad (4.260)$$

Notice that the source terms have been split into two parts: The source terms that contain derivatives and act like diffusion terms \mathbf{S}_v , and all other source terms \mathbf{S} . The diffusion-like source terms \mathbf{S}_v are included in the viscous stiffness matrix \mathbf{K}_{vis} .

$$\mathbf{S} = \begin{Bmatrix} \dot{\rho}_c \\ 0 \\ \dot{q}_c''' \\ 0 \\ 0 \\ 0 \end{Bmatrix} - \rho \begin{Bmatrix} 0 \\ \mathbf{a}_t + \vec{\omega} \times \vec{V} \\ \mathbf{a}_t \cdot (\vec{V} + \vec{V}_t) \\ 0 \\ 0 \\ 0 \end{Bmatrix} + \begin{Bmatrix} 0 \\ 0 \\ 0 \\ (c_{b1} \hat{S} - \frac{c_{w1} f_w \hat{v}}{d^2 \text{Re}}) \rho \hat{v} \\ \rho \tilde{\Pi}_k - \rho \tilde{\mathcal{E}} \\ \rho \tilde{\Pi}_\omega - \beta \rho \omega^2 \end{Bmatrix} \quad \mathbf{S}_v = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ \frac{c_{b2}}{\sigma \text{Re}} \nabla_i \hat{v} \nabla_i \rho \hat{v} \\ 0 \\ \frac{C_\omega}{\rho \omega} \nabla_i \rho K \nabla_i \rho \omega \end{Bmatrix} \quad (4.261)$$

The inviscid local time step can be thought of as the largest time step for that node according to the characteristics from \mathbf{K}_{inv} . The viscous local time step could be calculated in a similar manner from the eigenvalues of \mathbf{K}_{vis} , but this process would be very expensive. Alternative methods are available for diagonal-dominant matrices solved through iteration.

The viscous matrix \mathbf{K}_{vis} can be used to calculate local time steps appropriate for viscous stability. Neglecting the inviscid terms \mathbf{K}_{inv} , “non-viscous” portion of the source term \mathbf{S} , and artificial dissipation \mathbf{D} , the remaining system represents the finite element equations when the element Reynolds number is very low.

$$[\mathbf{M}_C] (\mathbf{U}_{n+1}^{i+1} - \mathbf{U}_n) + \Delta t [\mathbf{K}_{vis}] \mathbf{U}_{n+1}^i = 0 \quad (4.262)$$

$$\mathbf{U}_{n+1}^{i+1} \approx \mathbf{U}_n - \Delta t [\mathbf{M}_L]^{-1} [\mathbf{K}_{vis}] \mathbf{U}_{n+1}^i \quad \mathbf{U}_{n+1}^0 = \mathbf{U}_n \quad (4.263)$$

$$\mathbf{U}_{n+1}^{i+1} = \mathbf{U}_n + [\mathbf{A}]\mathbf{U}_{n+1}^i = \left([\mathbf{I}] + [\mathbf{A}] + \dots + [\mathbf{A}]^i\right)\mathbf{U}_n + [\mathbf{A}]^{i+1}\mathbf{U}_{n+1}^0 = \sum_{j=0}^{i+1} [\mathbf{A}]^j \mathbf{U}_n \quad (4.264)$$

The system converges when $[\mathbf{A}]^k \ll [\mathbf{I}]$, or the $k+1^{\text{th}}$ term can be neglected. The quickest convergence occurs for systems where the smallest value of k can be used to approximate the final value. Convergence can be assured for a diagonally dominant system if the system matrix is less than unity: $A_{ii} < 1$.

The viscous stiffness matrix \mathbf{K}_{vis} is diagonal dominant. Scaling by the inverse lumped mass matrix, which is primary diagonal, does not change the diagonal dominance. A different Δt can be applied to each row of the matrix so that the diagonal is less than unity: $A_{ii} = \Delta t_i K_{ii} / M_{L,ii} < 1$ or $\Delta t_i < M_{L,ii} / K_{ii}$.

The previous definition of the stiffness matrix can be simplified:

$$[\mathbf{K}_{vis}] \mathbf{U}_{n+1}^i = \sum_e \frac{1}{d!} \begin{Bmatrix} A_{i1} \\ \vdots \\ A_{i(d+1)} \end{Bmatrix} A_{im}^T \bar{\mathbf{F}}_{v,i} - \sum_{be} \frac{\Gamma_{be}}{d} \begin{Bmatrix} 1 \\ \vdots \\ 1 \end{Bmatrix}_d (\bar{\mathbf{F}}_{v,i} \cdot \hat{n}_i) - \sum_e \frac{\Omega_e}{d+1} \begin{Bmatrix} 1 \\ \vdots \\ 1 \end{Bmatrix}_{d+1} \bar{\mathbf{S}}_v \quad (4.264)$$

where the subscript on the vector of 1's designates the size of the vector. The domain integrals contributed to $d+1$ unknowns equations while the boundary integrals contributed to d unknowns equations.

Stability of Momentum Equations. The momentum equations are differential equations of a vector space, whereas the other differential equations represent the transport of scalar properties.

The momentum equations (designated by j) is discussed first:

$$\mathbf{K}^{(j\alpha)}(\rho u_\alpha)_e = \sum_e \frac{1}{d!} \begin{Bmatrix} A_{i1} \\ \vdots \\ A_{i(d+1)} \end{Bmatrix} (\tau_{ij} + \rho \tilde{\Gamma}_{ij}) - \sum_{be} \frac{\Gamma_{be}}{d} \begin{Bmatrix} 1 \\ \vdots \\ 1 \end{Bmatrix}_d (\tau_{ij} + \rho \tilde{\Gamma}_{ij}) \cdot \hat{n}_i \quad (4.265)$$

$$\mathbf{K}^{(j\alpha)} = \sum_e \frac{1}{d!} \begin{Bmatrix} A_{i1} \\ \vdots \\ A_{i(d+1)} \end{Bmatrix} \frac{\partial}{\partial(\rho u_\alpha)_e} (\tau_{ij} + \rho \tilde{\Gamma}_{ij}) - \sum_{be} \frac{\Gamma_{be}}{d} \begin{Bmatrix} 1 \\ \vdots \\ 1 \end{Bmatrix}_d \frac{\partial}{\partial(\rho u_\alpha)_e} (\tau_{ij} + \rho \tilde{\Gamma}_{ij}) \cdot \hat{n}_i \quad (4.266)$$

The velocity gradients can be rewritten in terms of the conservative properties:

$$\nabla_i u_j = \nabla_i \frac{\rho u_j}{\rho} = \frac{1}{|J_e|} A_{in} \left(\frac{\rho u_j}{\rho} \right)_n = \frac{1}{|J_e|} \left(\frac{A_j}{\rho} \right)_n (\rho u_j)_n \quad (4.267)$$

The velocity gradients are used to calculate the stress terms:

$$\tau_{ij} + \rho \tilde{\Gamma}_{ij} = \frac{\mu + \mu_T}{\text{Re}_L |J_e|} \left(\left(\frac{A_i}{\rho} \right)_n (\rho u_j)_n + \left(\frac{A_j}{\rho} \right)_n (\rho u_i)_n \right) + \frac{\lambda \mu - \frac{2}{3} \mu_T}{\text{Re}_L |J_e|} \left(\frac{A_k}{\rho} \right)_n (\rho u_k)_n \delta_{ij} \quad (4.268)$$

$$\frac{\partial}{\partial(\rho u_\alpha)_e} (\tau_{ij} + \rho \tilde{\Gamma}_{ij}) = \frac{\mu + \mu_T}{\text{Re}_L |J_e|} \left(\left(\frac{A_i}{\rho} \right)_n \delta_{j\alpha} + \left(\frac{A_j}{\rho} \right)_n \delta_{i\alpha} \right) + \frac{\lambda \mu - \frac{2}{3} \mu_T}{\text{Re}_L |J_e|} \left(\frac{A_k}{\rho} \right)_n \delta_{k\alpha} \delta_{ij} \quad (4.269)$$

which, in turn, are used to calculate the viscous fluxes. The stiffness matrix $\mathbf{K}^{(j\alpha)}$ represents the relationship between the j^{th} momentum equation and α^{th} velocity:

$$\mathbf{K}^{(j\alpha)} = \sum_e K_{mn,e}^{(j\alpha)} - \sum_{be} K_{mn,be}^{(j\alpha)} \quad (4.270)$$

$$K_{mn,e}^{(j\alpha)} = \frac{1}{d!} \begin{Bmatrix} A_{i1} \\ \vdots \\ A_{i(d+1)} \end{Bmatrix} \left(\frac{\mu + \mu_T}{\text{Re}_L |J_e|} \left(\left(\frac{A_i}{\rho} \right)_n \delta_{j\alpha} + \left(\frac{A_j}{\rho} \right)_n \delta_{i\alpha} \right) + \frac{\lambda \mu - \frac{2}{3} \mu_T}{\text{Re}_L |J_e|} \left(\frac{A_\alpha}{\rho} \right)_n \delta_{ij} \right) \quad (4.271)$$

$$K_{mn,be}^{(j\alpha)} = \frac{\Gamma_{be}}{d} \begin{Bmatrix} 1 \\ \vdots \\ 1 \end{Bmatrix}_d \left(\frac{\mu + \mu_T}{\text{Re}_L |J_e|} \left(\left(\frac{A_i}{\rho} \right)_n \delta_{j\alpha} + \left(\frac{A_j}{\rho} \right)_n \delta_{i\alpha} \right) + \frac{\lambda \mu - \frac{2}{3} \mu_T}{\text{Re}_L |J_e|} \left(\frac{A_\alpha}{\rho} \right)_n \delta_{ij} \right) \cdot \hat{n}_i \quad (4.272)$$

Contributions from the element matrices ($K_{mn,e}$) and boundary elements ($K_{mn,be}$) are summed for all terms on the diagonal K_{ii} . These terms are then used to calculate the local Δt necessary to

stabilize the u -, v -, and w -equations independently. The minimum of these time steps is the local stability of the momentum equations. The other equations have similar stiffness matrices and thus similar stability.

Stability of the Energy Equation. The energy equation represents the transport of total energy ρE and contains diffusion in the form of conduction. Conduction is calculated using Newton's law of cooling, which has been formulated with enthalpy h . The enthalpy gradient is reformulated here in terms of the total energy:

$$\frac{\partial h}{\partial x_i} = \gamma \frac{\partial e}{\partial x_i} = \gamma \frac{\partial}{\partial x_i} \left(\frac{\rho E}{\rho} - \frac{u_j u_j}{2} - K \right) = \gamma \frac{A_{in}}{|J_e|} \left(\frac{\rho E}{\rho} - \frac{u_j u_j}{2} - K \right)_n \quad (4.273)$$

$$q_i'' + \tilde{Q}_i'' = -\frac{1}{\text{Re}_L} \left(\frac{\mu}{\text{Pr}} + \frac{\mu_T}{\text{Pr}_T} \right) \frac{\partial h}{\partial x_i} = -\frac{\gamma}{\text{Re}_L} \left(\frac{\mu}{\text{Pr}} + \frac{\mu_T}{\text{Pr}_T} \right) \frac{1}{|J_e|} \left(\frac{A_i}{\rho} \right)_n \rho E_n + \dots \quad (4.274)$$

The stiffness equation associated with the energy equation and total energy diffusion is:

$$\mathbf{K}^{(ht)} \rho E_e = \sum_e \frac{1}{d!} \begin{Bmatrix} A_{i1} \\ \vdots \\ A_{i(d+1)} \end{Bmatrix} \left(\dots - q_i'' - \tilde{Q}_i'' + \dots \right) - \sum_{be} \frac{\Gamma_{be}}{d} \begin{Bmatrix} 1 \\ \vdots \\ 1 \end{Bmatrix}_d \left(\dots - q_i'' - \tilde{Q}_i'' + \dots \right) \cdot \hat{n}_i \quad (4.275)$$

$$\mathbf{K}^{(ht)} = -\sum_e \frac{1}{d!} \begin{Bmatrix} A_{i1} \\ \vdots \\ A_{i(d+1)} \end{Bmatrix} \left\{ \frac{\partial}{\partial \rho E_e} (q_i'' + \tilde{Q}_i'') + \sum_{be} \frac{\Gamma_{be}}{d} \begin{Bmatrix} 1 \\ \vdots \\ 1 \end{Bmatrix}_d \frac{\partial}{\partial \rho E_e} (q_i'' + \tilde{Q}_i'') \cdot \hat{n}_i \right\} \quad (4.276)$$

$$\frac{\partial}{\partial \rho E_e} (q_i'' + \tilde{Q}_i'') = -\frac{\gamma}{\text{Re}_L} \left(\frac{\mu}{\text{Pr}} + \frac{\mu_T}{\text{Pr}_T} \right) \frac{1}{|J_e|} \left(\frac{A_i}{\rho} \right)_n \quad (4.277)$$

The stiffness matrix $\mathbf{K}^{(ht)}$ represents the relationship between the energy equation and nodal total energy:

$$\mathbf{K}^{(ht)} = \sum_e K_{mn,e}^{(ht)} - \sum_{be} K_{mn,be}^{(ht)} \quad (4.278)$$

$$\mathbf{K}_{mn,e}^{(ht)} = \frac{1}{d!} \frac{\gamma}{\text{Re}_L} \left(\frac{\mu}{\text{Pr}} + \frac{\mu_T}{\text{Pr}_T} \right) \frac{1}{|J_e|} \begin{Bmatrix} A_{i1} \\ \vdots \\ A_{i(d+1)} \end{Bmatrix} \begin{Bmatrix} A_{i1} & \dots & A_{i(d+1)} \\ \rho_1 & & \rho_{d+1} \end{Bmatrix} \quad (4.279)$$

$$\mathbf{K}_{mn,be}^{(ht)} = \frac{\Gamma_{be}}{d} \frac{\gamma}{\text{Re}_L} \left(\frac{\mu}{\text{Pr}} + \frac{\mu_T}{\text{Pr}_T} \right) \frac{1}{|J_e|} \begin{Bmatrix} 1 \\ \vdots \\ 1 \end{Bmatrix}_d \begin{Bmatrix} A_{i1} & \dots & A_{i(d+1)} \\ \rho_1 & & \rho_{d+1} \end{Bmatrix} \cdot \hat{n}_i \quad (4.280)$$

where $K_{mn,e}$ and $K_{mn,be}$ represent the Laplacian ∇^2 and gradient normal to the boundary $\nabla \cdot \hat{n}$.

Stability of Turbulent Equations. The turbulent transport equations contain diffusion and turbulent source terms (SA stability and k - ω cross-diffusion), which are calculated using gradients of the turbulent properties:

$$\nabla_i \hat{v} = \nabla_i \frac{\rho \hat{v}}{\rho} = \frac{1}{|J_e|} A_{in} \left(\frac{\rho \hat{v}}{\rho} \right)_n = \frac{1}{|J_e|} \left(\frac{A_i}{\rho} \right)_n \rho \hat{v}_n \quad (4.281)$$

$$\nabla_i K = \nabla_i \frac{\rho K}{\rho} = \frac{1}{|J_e|} A_{in} \left(\frac{\rho K}{\rho} \right)_n = \frac{1}{|J_e|} \left(\frac{A_i}{\rho} \right)_n \rho K_n \quad (4.282)$$

$$\nabla_i \omega = \nabla_i \frac{\rho \omega}{\rho} = \frac{1}{|J_e|} A_{in} \left(\frac{\rho \omega}{\rho} \right)_n = \frac{1}{|J_e|} \left(\frac{A_i}{\rho} \right)_n \rho \omega_n \quad (4.283)$$

The turbulent diffusion and source terms can be written generically:

$$\mathbf{F}_{v,i} = C_F \frac{\partial}{\partial x_i} \left(\frac{\mathbf{U}}{\rho} \right) = \frac{C_F}{|J_e|} \left(\frac{A_i}{\rho} \right)_n \mathbf{U}_n \quad \mathbf{S}_v = C_{S,i} \frac{\partial \mathbf{U}}{\partial x_i} = \frac{C_{S,i}}{|J_e|} A_{in} \mathbf{U}_n \quad (4.284)$$

$$C_F = \begin{cases} \frac{1}{\sigma \text{Re}} (\mu + \rho \hat{v}) & \text{for } \mathbf{U} = \rho \hat{v} \\ \frac{1}{\text{Re}} (\mu + \sigma_k \mu_T) & \text{for } \mathbf{U} = \rho K \\ \frac{1}{\text{Re}} (\mu + \sigma_\omega \mu_T) & \text{for } \mathbf{U} = \rho \omega \end{cases} \quad C_{S,i} = \begin{cases} \frac{c_{b2}}{\sigma \text{Re}} \nabla_i \hat{v} & \text{for } \mathbf{U} = \rho \hat{v} \\ 0 & \text{for } \mathbf{U} = \rho K \\ \frac{C_\omega}{\rho \omega} \nabla_i \rho K & \text{for } \mathbf{U} = \rho \omega \end{cases} \quad (4.285)$$

From this, the stiffness matrices are written generically:

$$\mathbf{K}^{(turb)} = \sum_e K_{mn,e}^{(turb)} - \sum_{be} K_{mn,be}^{(turb)} \quad (4.286)$$

$$K_{mn,e}^{(turb)} = \frac{1}{d!} \frac{C_{F1}}{|J_e|} \begin{Bmatrix} A_{i1} \\ \vdots \\ A_{i(d+1)} \end{Bmatrix} \begin{Bmatrix} A_{i1} & \dots & A_{i(d+1)} \\ \rho_1 & \dots & \rho_{d+1} \end{Bmatrix} - \frac{\Omega_e}{d+1} \frac{C_{S,i}}{|J_e|} \begin{Bmatrix} 1 \\ \vdots \\ 1 \end{Bmatrix}_{d+1} \{A_{i1} \dots A_{i(d+1)}\} \quad (4.287)$$

$$K_{mn,be}^{(turb)} = \frac{\Gamma_{be}}{d} \frac{C_{F1}}{|J_e|} \begin{Bmatrix} 1 \\ \vdots \\ 1 \end{Bmatrix}_d \begin{Bmatrix} A_{i1} & \dots & A_{i(d+1)} \\ \rho_1 & \dots & \rho_{d+1} \end{Bmatrix} \cdot \hat{n}_i \quad (4.288)$$

The turbulent stiffness matrices resemble the heat transfer and momentum matrices, in that they have element and boundary element matrices that represent Laplacian and normal-gradients.

The turbulent stiffness also contains a source term matrix with considerations from an alternative gradient in $C_{S,i}$.

Generic Viscous Stability. All of the finite element equations are scaled by the inverse lumped mass matrix \mathbf{M}_L^{-1} in the predictor-corrector:

$$[\mathbf{A}] = [\mathbf{M}_L]^{-1} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{K}^{(uu)} & \mathbf{K}^{(uv)} & \mathbf{K}^{(uw)} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{K}^{(vu)} & \mathbf{K}^{(vv)} & \mathbf{K}^{(vw)} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{K}^{(wu)} & \mathbf{K}^{(wv)} & \mathbf{K}^{(ww)} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{K}^{(ht)} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{K}^{(sa)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{K}^{(k)} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{K}^{(\omega)} \end{bmatrix} \quad (4.289)$$

The system matrix \mathbf{A} is diagonal dominant under normal operating conditions. The diagonal can now be used to determine the stability of the system. The viscous local time step is calculated by inverting the terms on the diagonal:

$$\Delta t_{vis}^{(k)} = \frac{CFL}{2\mathbf{A}_{ii}^{(k)}} \quad (4.290)$$

where $k = 1$ is density, $k = 2, 3,$ and 4 are momentum; and etc. The 2 in the denominator was selected through experimentation. The minimum of the global, local inviscid, and local viscous time steps is chosen as the most stable local time step:

$$\Delta t_{loc} = MIN\left(dt, \Delta t_{inv}, \Delta t_{vis}^{(1)}, \Delta t_{vis}^{(2)}, \dots, \Delta t_{vis}^{(8)}\right) \quad (4.291)$$

With reasonable numbers for the viscosity ratio, eddy viscosity, Prandtl number, and other model properties, the heat transfer stability is generally the most sensitive, followed by the momentum equations and then the turbulence models. For this reason, the viscous stability has been broken apart into four options:

- $itime = -1$, uses the one-dimension simplification demonstrated below
- $itime = 0$, only assesses the stability of the energy equation (heat transfer only)
- $itime = 1$, assesses the momentum and heat transfer
- $itime = 2$, assesses the momentum, heat transfer, and turbulence

The method is more assured of stability and convergence for higher values of $itime$ because more equations are checked. Remember, the method developed here decoupled by the viscous equations by only considering the diagonal matrices, so the method is not unconditionally stable. The viscous stability is now controllable using CFL , like the inviscid stability, where CFL can be lowered to relax the solution. So that the user can see how the stability is being affected by the equations, a ratio is written out for each of four successive comparisons:

- $htime$, ratio of inviscid to heat transfer time;
- $vtime$, ratio of momentum time to previous;
- $stime$, ratio of turbulent (ω or ν) time to previous; and,
- $utime$, ratio of local to global time steps.

Each value represents the minimum ratio at any node in the domain. $utime$ is always checked, so it is written out in the first column. If $utime = 1$, then the global time step is smaller than any of

the local time steps on the domain, and the global time step dt can be increased for efficiency.

The remaining three values are tested with increasing levels of $itime$ and are added in successive columns. If any of the later columns are consistently listing unity, then the value of $itime$ can be lowered to exclude those equation from the tests.

A simple one-dimensional analogy was discussed for $itime = -1$. This analogy will first be developed by considering a simple one-dimensional test case with equal sized elements. Then the one-dimensional case will be expanded to multiple dimensions, using the minimum distance across the element. This method is popular in the literature, but this research found that it's application is limited.

One-Dimensional Stability. Consider a one-dimensional domain with two equal size elements of length l_e far from any boundary. The density and other properties are constant on the domain.

Ignoring the C_s -term, the stability matrix is 3x3 and derived:

$$\mathbf{K}^{(uu)} = \sum_e \frac{(2 + \lambda)\mu + \frac{4}{3}\mu_T}{\text{Re}_L l_e} \begin{Bmatrix} A_{11} \\ A_{12} \end{Bmatrix} \begin{Bmatrix} A_{11} & A_{12} \\ \rho_1 & \rho_2 \end{Bmatrix} = \frac{C_F}{l_e \rho} \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \quad (4.292)$$

$$[\mathbf{M}_L] = \sum_e \frac{l_e}{2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \frac{l_e}{2} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.293)$$

$$[\mathbf{A}] = [\mathbf{M}_L]^{-1} [\mathbf{K}^{(uu)}] = \frac{2}{l_e} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \frac{C_F}{l_e \rho} \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} = \frac{2C_F}{l_e^2 \rho} \begin{bmatrix} 1 & -1 & 0 \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ 0 & -1 & 1 \end{bmatrix} \quad (4.294)$$

Similar equations can be developed for the other equations so that an appropriate local time step for the center node would be:

$$\begin{aligned}
\Delta t_{vis}^{(\rho u)} &= \frac{1}{2} \left(\frac{2C_{F1}^{(\rho u)}}{l_e^2 \rho} \right)^{-1} = \frac{l_e^2 \rho}{4} \left(2 \frac{\mu + \mu_T}{\text{Re}} + \frac{\lambda \mu - \frac{2}{3} \mu_T}{\text{Re}} \right)^{-1} \leq 0.375 \frac{l_e^2 \rho \text{Re}}{\mu + \mu_T} \\
0.125 \frac{l_e^2 \rho \text{Re}}{\mu + \mu_T} &\leq \Delta t_{vis}^{(\rho E)} = \frac{1}{2} \left(\frac{2C_{F1}^{(\rho E)}}{l_e^2 \rho} \right)^{-1} = \frac{l_e^2 \rho}{4} \left(\frac{\gamma}{\text{Re}} \left(\frac{\mu}{Pr} + \frac{\mu_T}{Pr_T} \right) \right)^{-1} \leq 0.161 \frac{l_e^2 \rho \text{Re}}{\mu + \mu_T} \\
0.166 \frac{l_e^2 \rho \text{Re}}{\mu + \mu_T} &\leq \Delta t_{vis}^{(\rho \hat{v})} = \frac{1}{2} \left(\frac{2C_{F1}^{(\rho \hat{v})}}{l_e^2 \rho} \right)^{-1} = \frac{l_e^2 \rho}{4} \left(\frac{1}{\sigma \text{Re}} (\mu + \rho \hat{v}) \right)^{-1} \\
0.212 \frac{l_e^2 \rho \text{Re}}{\mu + \mu_T} &\leq \Delta t_{vis}^{(\rho K)} = \frac{1}{2} \left(\frac{2C_{F1}^{(\rho K)}}{l_e^2 \rho} \right)^{-1} = \frac{l_e^2 \rho}{4} \left(\frac{1}{\text{Re}} (\mu + \sigma_K \mu_T) \right)^{-1} \leq 0.250 \frac{l_e^2 \rho \text{Re}}{\mu + \mu_T} \\
0.250 \frac{l_e^2 \rho \text{Re}}{\mu + \mu_T} &\leq \Delta t_{vis}^{(\rho \omega)} = \frac{1}{2} \left(\frac{2C_{F1}^{(\rho \omega)}}{l_e^2 \rho} \right)^{-1} = \frac{l_e^2 \rho}{4} \left(\frac{1}{\text{Re}} (\mu + \sigma_\omega \mu_T) \right)^{-1} \leq 0.500 \frac{l_e^2 \rho \text{Re}}{\mu + \mu_T}
\end{aligned}$$

Notice that for air ($\lambda^* = -2/3$; $\gamma = 1.4$, $Pr = 0.7$, $Pr_T = 0.9$) and given turbulence models ($\rho v \leq \mu_T$, $\sigma = 2/3$; $\sigma_\omega = 2$), the heat transfer is always smallest:

$$0.125 \frac{l_e^2 \rho \text{Re}}{\mu + \mu_T} \leq \Delta t_{vis}^{(\rho E)} \leq \Delta t_{vis}^{(\rho \hat{v})} \leq \Delta t_{vis}^{(\rho K)} \leq \Delta t_{vis}^{(\rho u)} \leq \Delta t_{vis}^{(\rho \omega)} \quad (4.295)$$

The heat transfer equation can be used to assemble a quick stability check on orthogonal grids. For two- and three-dimensional domains, the minimum distance across an element is used in place of the element length. The minimum distance across an element is found using the “volume” of the element and maximum “area” of one of its faces:

$$h_{\min, 2D} = \frac{2A_e}{\text{MAX}(l_{f,12}, l_{f,13}, l_{f,23})} \quad (4.296)$$

$$h_{\min, 3D} = \frac{3V_e}{\text{MAX}(A_{f,123}, A_{f,124}, A_{f,134}, A_{f,234})} \quad (4.297)$$

The one-dimensional analogy has been adequate for orthogonal grids, whether mathematically generated on the entire domain or just in the near-wall region. If the viscous gradients are strong outside of the orthogonal regions of the mesh, a more robust ($itime \geq 0$) method should be used.

Generic Form. A generic form was implemented in NS2D and NS3D.

$$\mathbf{K}^{(a)} = \sum_e K_{mn,e}^{(a)} - \sum_{be} K_{mn,be}^{(a)} \quad (4.298)$$

$$K_{mn,e}^{(*)} = \psi_e \left(A_{im} \frac{A_{in}}{\rho_n} \delta_{j\alpha} + \varepsilon A_{cm} \frac{A_{jn}}{\rho_n} \right) + \phi_e A_{jm} \frac{A_{cn}}{\rho_n} - \{1\}_{d+1} \sigma_{e,i} A_{in} \quad (4.299)$$

$$K_{mn,be}^{(*)} = \{1\}_d \left(\theta_{be} \left(\frac{A_{in}}{\rho_n} \hat{n}_i \delta_{j\alpha} + \varepsilon \frac{A_{jn}}{\rho_n} \hat{n}_\alpha \right) + \zeta_{be} \frac{A_{cn}}{\rho_n} \hat{n}_j \right) \quad (4.300)$$

$$\psi_e = \frac{C_{F1}}{d! |J_e|} \quad \phi_e = \frac{C_{F2}}{d! |J_e|} \quad \sigma_{e,i} = \frac{\Omega_e}{d+1} \frac{C_{S,i}}{|J_e|} \quad (4.301)$$

$$\theta_{be} = \frac{\Gamma_{be}}{d} \frac{C_{F1}}{|J_e|} \quad \zeta_{be} = \frac{\Gamma_{be}}{d} \frac{C_{F2}}{|J_e|} \quad (4.302)$$

$$C_{F1} = \begin{cases} \frac{1}{\text{Re}} (\mu + \mu_T) & \text{for } \mathbf{U} = \rho \mathbf{u}_j \\ \frac{\gamma}{\text{Re}} \left(\frac{\mu}{\text{Pr}} + \frac{\mu_T}{\text{Pr}_T} \right) & \text{for } \mathbf{U} = \rho E \\ \frac{1}{\sigma \text{Re}} (\mu + \rho \hat{v}) & \text{for } \mathbf{U} = \rho \hat{v} \\ \frac{1}{\text{Re}} (\mu + \sigma_k \mu_T) & \text{for } \mathbf{U} = \rho K \\ \frac{1}{\text{Re}} (\mu + \sigma_\omega \mu_T) & \text{for } \mathbf{U} = \rho \omega \end{cases} \quad C_{S,i} = \begin{cases} 0 & \text{for } \mathbf{U} = \rho \mathbf{u}_j \\ 0 & \text{for } \mathbf{U} = \rho E \\ \frac{c_{b2}}{\sigma \text{Re}} \nabla_i \hat{v} & \text{for } \mathbf{U} = \rho \hat{v} \\ 0 & \text{for } \mathbf{U} = \rho K \\ \frac{C_\omega}{\rho \omega} \nabla_i \rho K & \text{for } \mathbf{U} = \rho \omega \end{cases} \quad (4.303)$$

$$C_{F2} = \begin{cases} \frac{1}{\text{Re}} (\lambda \mu - \frac{2}{3} \mu_T) & \text{for } \mathbf{U} = \rho \mathbf{u}_j \\ 0 & \text{otherwise} \end{cases} \quad \varepsilon = \begin{cases} 1 & \text{for } \mathbf{U} = \rho \mathbf{u}_j \\ 0 & \text{otherwise} \end{cases} \quad (4.304)$$

The subscripts j and α are equal and correspond to the component of momentum (u, v, w). For heat transfer or turbulence stability, j and α have no special meaning, but j equals α creates the Jacobian and normal gradient that represents the diffusion terms.

Two-Dimensional Stability. Now consider a two-dimensional domain with an arbitrary geometry (element distribution) and a progressing solution. The viscous stability matrix is assessed:

$$\begin{aligned}
\mathbf{K}^{(*)} = & \sum_e \frac{C_{F1} \delta_{j\alpha}}{4A_e} \begin{bmatrix} A_{11} \frac{A_{11}}{\rho_1} & A_{11} \frac{A_{12}}{\rho_2} & A_{11} \frac{A_{13}}{\rho_3} \\ A_{12} \frac{A_{11}}{\rho_1} & A_{12} \frac{A_{12}}{\rho_2} & A_{12} \frac{A_{13}}{\rho_3} \\ A_{13} \frac{A_{11}}{\rho_1} & A_{13} \frac{A_{12}}{\rho_2} & A_{13} \frac{A_{13}}{\rho_3} \end{bmatrix}_e + \sum_e \frac{C_{F1} \delta_{j\alpha}}{4A_e} \begin{bmatrix} A_{21} \frac{A_{21}}{\rho_1} & A_{21} \frac{A_{22}}{\rho_2} & A_{21} \frac{A_{23}}{\rho_3} \\ A_{22} \frac{A_{21}}{\rho_1} & A_{22} \frac{A_{22}}{\rho_2} & A_{22} \frac{A_{23}}{\rho_3} \\ A_{23} \frac{A_{21}}{\rho_1} & A_{23} \frac{A_{22}}{\rho_2} & A_{23} \frac{A_{23}}{\rho_3} \end{bmatrix}_e \\
& + \sum_e \frac{C_{F1} \mathcal{E}}{4A_e} \begin{bmatrix} A_{\alpha 1} \frac{A_{j1}}{\rho_1} & A_{\alpha 1} \frac{A_{j2}}{\rho_2} & A_{\alpha 1} \frac{A_{j3}}{\rho_3} \\ A_{\alpha 2} \frac{A_{j1}}{\rho_1} & A_{\alpha 2} \frac{A_{j2}}{\rho_2} & A_{\alpha 2} \frac{A_{j3}}{\rho_3} \\ A_{\alpha 3} \frac{A_{j1}}{\rho_1} & A_{\alpha 3} \frac{A_{j2}}{\rho_2} & A_{\alpha 3} \frac{A_{j3}}{\rho_3} \end{bmatrix}_e + \sum_e \frac{C_{F2}}{4A_e} \begin{bmatrix} A_{j1} \frac{A_{\alpha 1}}{\rho_1} & A_{j1} \frac{A_{\alpha 2}}{\rho_2} & A_{j1} \frac{A_{\alpha 3}}{\rho_3} \\ A_{j2} \frac{A_{\alpha 1}}{\rho_1} & A_{j2} \frac{A_{\alpha 2}}{\rho_2} & A_{j2} \frac{A_{\alpha 3}}{\rho_3} \\ A_{j3} \frac{A_{\alpha 1}}{\rho_1} & A_{j3} \frac{A_{\alpha 2}}{\rho_2} & A_{j3} \frac{A_{\alpha 3}}{\rho_3} \end{bmatrix}_e \\
& - \sum_e \frac{C_{S,1}}{6} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{11} & A_{12} & A_{13} \\ A_{11} & A_{12} & A_{13} \end{bmatrix}_e - \sum_e \frac{C_{S,2}}{6} \begin{bmatrix} A_{21} & A_{22} & A_{23} \\ A_{21} & A_{22} & A_{23} \\ A_{21} & A_{22} & A_{23} \end{bmatrix}_e \\
& - \sum_{be} \frac{l_{be} C_{F1} \hat{n}_x \delta_{j\alpha}}{4A_e} \begin{bmatrix} \frac{A_{11}}{\rho_1} & \frac{A_{12}}{\rho_2} & \frac{A_{13}}{\rho_3} \\ \frac{A_{11}}{\rho_1} & \frac{A_{12}}{\rho_2} & \frac{A_{13}}{\rho_3} \\ \frac{A_{11}}{\rho_1} & \frac{A_{12}}{\rho_2} & \frac{A_{13}}{\rho_3} \end{bmatrix}_{be,e} - \sum_{be} \frac{l_{be} C_{F1} \hat{n}_y \delta_{j\alpha}}{4A_e} \begin{bmatrix} \frac{A_{21}}{\rho_1} & \frac{A_{22}}{\rho_2} & \frac{A_{23}}{\rho_3} \\ \frac{A_{21}}{\rho_1} & \frac{A_{22}}{\rho_2} & \frac{A_{23}}{\rho_3} \\ \frac{A_{21}}{\rho_1} & \frac{A_{22}}{\rho_2} & \frac{A_{23}}{\rho_3} \end{bmatrix}_{be,e} \\
& - \sum_{be} \frac{l_{be} C_{F1} \mathcal{E} \hat{n}_\alpha}{4A_e} \begin{bmatrix} \frac{A_{j1}}{\rho_1} & \frac{A_{j2}}{\rho_2} & \frac{A_{j3}}{\rho_3} \\ \frac{A_{j1}}{\rho_1} & \frac{A_{j2}}{\rho_2} & \frac{A_{j3}}{\rho_3} \\ \frac{A_{j1}}{\rho_1} & \frac{A_{j2}}{\rho_2} & \frac{A_{j3}}{\rho_3} \end{bmatrix}_{be,e} - \sum_{be} \frac{l_{be} C_{F2} \hat{n}_j}{4A_e} \begin{bmatrix} \frac{A_{\alpha 1}}{\rho_1} & \frac{A_{\alpha 2}}{\rho_2} & \frac{A_{\alpha 3}}{\rho_3} \\ \frac{A_{\alpha 1}}{\rho_1} & \frac{A_{\alpha 2}}{\rho_2} & \frac{A_{\alpha 3}}{\rho_3} \\ \frac{A_{\alpha 1}}{\rho_1} & \frac{A_{\alpha 2}}{\rho_2} & \frac{A_{\alpha 3}}{\rho_3} \end{bmatrix}_{be,e}
\end{aligned} \tag{4.305}$$

$$[\mathbf{M}_L] = \sum_e \frac{A_e}{3} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}_e \quad \Delta t_{vis}^{(k)} = \frac{1}{2\mathbf{A}_{vis,ii}^{(k)}} = \frac{\mathbf{M}_{L,ii}}{2\mathbf{K}_{vis,ii}^{(k)}} \tag{4.306}$$

For efficiency, only the diagonal terms ($i = j$) are added to the global matrix. Stability is tested on the heat transfer ($k = 4$), momentum ($k = 2, 3; j = \alpha = 1, 2$), and turbulence model ($k = 5$ or 7). The values of C_{F1} , C_{F2} , and $C_{S,i}$ change with the test. The four tests have been listed in order of likelihood of instability and the order of implementation in *itime*.

Three-Dimensional Stability. Now consider a three-dimensional domain with an arbitrary geometry (element distribution) and a progressing solution. The viscous stability matrix is assessed:

$$\mathbf{K}_e^{(*)} = \mathbf{K}_e^{(*)} - \mathbf{K}_{be}^{(*)} \quad (4.307)$$

$$\begin{aligned} \mathbf{K}_e^{(*)} = & \sum_e \frac{C_{F1} \delta_{j\alpha}}{36V_e} \begin{bmatrix} A_{11}^{A_1} \rho_1 & A_{12}^{A_1} \rho_1 & A_{13}^{A_1} \rho_1 & A_{14}^{A_1} \rho_1 \\ A_{11}^{A_2} \rho_2 & A_{12}^{A_2} \rho_2 & A_{13}^{A_2} \rho_2 & A_{14}^{A_2} \rho_2 \\ A_{11}^{A_3} \rho_3 & A_{12}^{A_3} \rho_3 & A_{13}^{A_3} \rho_3 & A_{14}^{A_3} \rho_3 \\ A_{11}^{A_4} \rho_4 & A_{12}^{A_4} \rho_4 & A_{13}^{A_4} \rho_4 & A_{14}^{A_4} \rho_4 \end{bmatrix}_e \\ & + \sum_e \frac{C_{F1} \delta_{j\alpha}}{36V_e} \begin{bmatrix} A_{21}^{A_1} \rho_1 & A_{22}^{A_1} \rho_1 & A_{23}^{A_1} \rho_1 & A_{24}^{A_1} \rho_1 \\ A_{21}^{A_2} \rho_2 & A_{22}^{A_2} \rho_2 & A_{23}^{A_2} \rho_2 & A_{24}^{A_2} \rho_2 \\ A_{21}^{A_3} \rho_3 & A_{22}^{A_3} \rho_3 & A_{23}^{A_3} \rho_3 & A_{24}^{A_3} \rho_3 \\ A_{21}^{A_4} \rho_4 & A_{22}^{A_4} \rho_4 & A_{23}^{A_4} \rho_4 & A_{24}^{A_4} \rho_4 \end{bmatrix}_e \\ & + \sum_e \frac{C_{F1} \delta_{j\alpha}}{36V_e} \begin{bmatrix} A_{31}^{A_1} \rho_1 & A_{32}^{A_1} \rho_1 & A_{33}^{A_1} \rho_1 & A_{34}^{A_1} \rho_1 \\ A_{31}^{A_2} \rho_2 & A_{32}^{A_2} \rho_2 & A_{33}^{A_2} \rho_2 & A_{34}^{A_2} \rho_2 \\ A_{31}^{A_3} \rho_3 & A_{32}^{A_3} \rho_3 & A_{33}^{A_3} \rho_3 & A_{34}^{A_3} \rho_3 \\ A_{31}^{A_4} \rho_4 & A_{32}^{A_4} \rho_4 & A_{33}^{A_4} \rho_4 & A_{34}^{A_4} \rho_4 \end{bmatrix}_e \\ & + \sum_e \frac{C_{F1} \delta_{j\alpha}}{36V_e} \begin{bmatrix} A_{\alpha 1}^{A_1} \rho_1 & A_{\alpha 2}^{A_1} \rho_1 & A_{\alpha 3}^{A_1} \rho_1 & A_{\alpha 4}^{A_1} \rho_1 \\ A_{\alpha 1}^{A_2} \rho_2 & A_{\alpha 2}^{A_2} \rho_2 & A_{\alpha 3}^{A_2} \rho_2 & A_{\alpha 4}^{A_2} \rho_2 \\ A_{\alpha 1}^{A_3} \rho_3 & A_{\alpha 2}^{A_3} \rho_3 & A_{\alpha 3}^{A_3} \rho_3 & A_{\alpha 4}^{A_3} \rho_3 \\ A_{\alpha 1}^{A_4} \rho_4 & A_{\alpha 2}^{A_4} \rho_4 & A_{\alpha 3}^{A_4} \rho_4 & A_{\alpha 4}^{A_4} \rho_4 \end{bmatrix}_e \\ & + \sum_e \frac{C_{F2}}{36V_e} \begin{bmatrix} A_{j1}^{A_{e1}} \rho_1 & A_{j2}^{A_{e1}} \rho_1 & A_{j3}^{A_{e1}} \rho_1 & A_{j4}^{A_{e1}} \rho_1 \\ A_{j1}^{A_{e2}} \rho_2 & A_{j2}^{A_{e2}} \rho_2 & A_{j3}^{A_{e2}} \rho_2 & A_{j4}^{A_{e2}} \rho_2 \\ A_{j1}^{A_{e3}} \rho_3 & A_{j2}^{A_{e3}} \rho_3 & A_{j3}^{A_{e3}} \rho_3 & A_{j4}^{A_{e3}} \rho_3 \\ A_{j1}^{A_{e4}} \rho_4 & A_{j2}^{A_{e4}} \rho_4 & A_{j3}^{A_{e4}} \rho_4 & A_{j4}^{A_{e4}} \rho_4 \end{bmatrix}_e \\ & - \sum_e \frac{C_{S,2}}{24} \begin{bmatrix} A_{21} & A_{22} & A_{23} & A_{24} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{21} & A_{22} & A_{23} & A_{24} \end{bmatrix}_e \\ & - \sum_e \frac{C_{S,3}}{24} \begin{bmatrix} A_{31} & A_{32} & A_{33} & A_{34} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{31} & A_{32} & A_{33} & A_{34} \end{bmatrix}_e \\ & - \sum_e \frac{C_{S,1}}{24} \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{11} & A_{12} & A_{13} & A_{14} \\ A_{11} & A_{12} & A_{13} & A_{14} \\ A_{11} & A_{12} & A_{13} & A_{14} \end{bmatrix}_e \\ & - \sum_e \frac{C_{\alpha 4}}{36V_e} \begin{bmatrix} A_{\alpha 1}^{A_4} \rho_4 & A_{\alpha 2}^{A_4} \rho_4 & A_{\alpha 3}^{A_4} \rho_4 & A_{\alpha 4}^{A_4} \rho_4 \\ A_{\alpha 1}^{A_3} \rho_3 & A_{\alpha 2}^{A_3} \rho_3 & A_{\alpha 3}^{A_3} \rho_3 & A_{\alpha 4}^{A_3} \rho_3 \\ A_{\alpha 1}^{A_2} \rho_2 & A_{\alpha 2}^{A_2} \rho_2 & A_{\alpha 3}^{A_2} \rho_2 & A_{\alpha 4}^{A_2} \rho_2 \\ A_{\alpha 1}^{A_1} \rho_1 & A_{\alpha 2}^{A_1} \rho_1 & A_{\alpha 3}^{A_1} \rho_1 & A_{\alpha 4}^{A_1} \rho_1 \end{bmatrix}_e \end{aligned} \quad (4.308)$$

$$\begin{aligned}
\mathbf{K}_{be}^{(*)} = & \sum_{be} \frac{A_{be} C_{F1} \hat{n}_x \delta_{j\alpha}}{18V_e} \begin{bmatrix} \frac{A_{11}}{\rho_1} & \frac{A_{12}}{\rho_2} & \frac{A_{13}}{\rho_3} & \frac{A_{14}}{\rho_4} \\ \frac{A_{21}}{\rho_1} & \frac{A_{22}}{\rho_2} & \frac{A_{23}}{\rho_3} & \frac{A_{24}}{\rho_4} \\ \frac{A_{31}}{\rho_1} & \frac{A_{32}}{\rho_2} & \frac{A_{33}}{\rho_3} & \frac{A_{34}}{\rho_4} \\ \frac{A_{41}}{\rho_1} & \frac{A_{42}}{\rho_2} & \frac{A_{43}}{\rho_3} & \frac{A_{44}}{\rho_4} \end{bmatrix}_{be,e} + \sum_{be} \frac{A_{be} C_{F1} \hat{n}_y \delta_{j\alpha}}{18V_e} \begin{bmatrix} \frac{A_{21}}{\rho_1} & \frac{A_{22}}{\rho_2} & \frac{A_{23}}{\rho_3} & \frac{A_{24}}{\rho_4} \\ \frac{A_{31}}{\rho_1} & \frac{A_{32}}{\rho_2} & \frac{A_{33}}{\rho_3} & \frac{A_{34}}{\rho_4} \\ \frac{A_{41}}{\rho_1} & \frac{A_{42}}{\rho_2} & \frac{A_{43}}{\rho_3} & \frac{A_{44}}{\rho_4} \\ \frac{A_{51}}{\rho_1} & \frac{A_{52}}{\rho_2} & \frac{A_{53}}{\rho_3} & \frac{A_{54}}{\rho_4} \end{bmatrix}_{be,e} \\
& + \sum_{be} \frac{A_{be} C_{F1} \hat{n}_z \delta_{j\alpha}}{18V_e} \begin{bmatrix} \frac{A_{31}}{\rho_1} & \frac{A_{32}}{\rho_2} & \frac{A_{33}}{\rho_3} & \frac{A_{34}}{\rho_4} \\ \frac{A_{41}}{\rho_1} & \frac{A_{42}}{\rho_2} & \frac{A_{43}}{\rho_3} & \frac{A_{44}}{\rho_4} \\ \frac{A_{51}}{\rho_1} & \frac{A_{52}}{\rho_2} & \frac{A_{53}}{\rho_3} & \frac{A_{54}}{\rho_4} \\ \frac{A_{61}}{\rho_1} & \frac{A_{62}}{\rho_2} & \frac{A_{63}}{\rho_3} & \frac{A_{64}}{\rho_4} \end{bmatrix}_{be,e} + \sum_{be} \frac{A_{be} C_{F1} \epsilon \hat{n}_\alpha}{18V_e} \begin{bmatrix} \frac{A_{j1}}{\rho_1} & \frac{A_{j2}}{\rho_2} & \frac{A_{j3}}{\rho_3} & \frac{A_{j4}}{\rho_4} \\ \frac{A_{j1}}{\rho_1} & \frac{A_{j2}}{\rho_2} & \frac{A_{j3}}{\rho_3} & \frac{A_{j4}}{\rho_4} \\ \frac{A_{j1}}{\rho_1} & \frac{A_{j2}}{\rho_2} & \frac{A_{j3}}{\rho_3} & \frac{A_{j4}}{\rho_4} \\ \frac{A_{j1}}{\rho_1} & \frac{A_{j2}}{\rho_2} & \frac{A_{j3}}{\rho_3} & \frac{A_{j4}}{\rho_4} \end{bmatrix}_{be,e} \\
& + \sum_{be} \frac{A_{be} C_{F2} \hat{n}_j}{18V_e} \begin{bmatrix} \frac{A_{\alpha 1}}{\rho_1} & \frac{A_{\alpha 2}}{\rho_2} & \frac{A_{\alpha 3}}{\rho_3} & \frac{A_{\alpha 4}}{\rho_4} \\ \frac{A_{\alpha 1}}{\rho_1} & \frac{A_{\alpha 2}}{\rho_2} & \frac{A_{\alpha 3}}{\rho_3} & \frac{A_{\alpha 4}}{\rho_4} \\ \frac{A_{\alpha 1}}{\rho_1} & \frac{A_{\alpha 2}}{\rho_2} & \frac{A_{\alpha 3}}{\rho_3} & \frac{A_{\alpha 4}}{\rho_4} \\ \frac{A_{\alpha 1}}{\rho_1} & \frac{A_{\alpha 2}}{\rho_2} & \frac{A_{\alpha 3}}{\rho_3} & \frac{A_{\alpha 4}}{\rho_4} \end{bmatrix}_{be,e}
\end{aligned} \tag{4.309}$$

$$\begin{aligned}
[\mathbf{M}_L] = & \sum_e \frac{V_e}{4} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_e \quad \Delta t_{vis}^{(k)} = \frac{1}{2\mathbf{A}_{vis,ii}^{(k)}} = \frac{\mathbf{M}_{L,ii}}{2\mathbf{K}_{vis,ii}^{(k)}} \tag{4.310}
\end{aligned}$$

For efficiency, only the diagonal terms ($i = j$) are added to the global matrix. The stability is tested on the heat transfer ($k = 5$), momentum ($k = 2, 3, 4; j = \alpha = 1, 2, 3$), and turbulence model ($k = 6$ or 8). The values of C_{F1} , C_{F2} , and $C_{S,i}$ change with the test. The five tests have been listed in order of likelihood of instability and the order of implementation in *itime*.

Testing. Local viscous time stepping was used to stabilize several of the cases demonstrated in later chapters of this work. Without viscous time stepping, the solution could only be stabilized with very small CFL numbers (on the order of 10^{-4}) or global time steps. NS3D was also tested with only two-dimensional stability. The resulting solution was stable in the 2D plane but still unstable in the cross-flow direction. When three-dimensional stability was implemented, the entire solution was stabilized. The 1D stability equations (Eqs. 4.295 through 4.297) were shown to be stable to orthogonal meshes (Cartesian aligned) over a flat plate. The 1D equations need to be tested for a generic mesh with viscous wall packing.

CHAPTER V

IMPLEMENTATION

The previous chapter developed the discretized equations necessary for this work. This chapter discusses how those equations were implemented in this work and the implications of those equations. The development is expanded here when necessary. This chapter outlines the development of the in-house codes and CFDsol during this work. The implementation of propulsion and turbulence models is discussed. Memory requirements and run time comparisons are discussed along with the support software developed and used during this work.

5.1 In-House Codes

Much work was done in the in-house OSU codes during the course of this work. This work was done in parallel with Sukraw (2008), Brown (2009), Walters (2009), Pinkerman (2010), O'Neill (2011), Hassett, and others. To begin, Euler2D and Euler3D were adapted to be more user-friendly when editing the code and reusing the routines that already existed in the codes. Euler2D was updated with Euler3D, since the work after Cowan (2003) had only occurred in Euler3D. Propulsion models and acoustic outputs were added to Euler2D and Euler3D. The rigid body model in Euler3D was upgraded to include all of the terms read from its input files. Viscous terms were added to Euler2D and Euler3D to create NS2D and

NS3D, respectively. The viscous terms were added simultaneously with propulsion and acoustic testing. As routines were upgraded in the inviscid codes, these routines were mirrored into their viscous counterparts. Finally, SA and SST turbulence models were implemented and tested, first in NS2D and then in NS3D. Artificial dissipation needed to be added to both models, and the routines already used in the codes were adapted to handle turbulence variables. Other problems arose along the way, like energy balance, plotting, and initial conditions. Viscous local time stepping was developed and implemented. All of these topics will be discussed in brief. The input and output file formats for the most current versions of the codes are shown in Appendix C (Euler2D), D (NS2D), E (Euler3D), and F (NS3D).

5.1.1 General Changes

Many changes were made to Euler2D and Euler3D before the bulk of this work occurred. Euler2D was updated with Euler3D. This included pushing back features, such as run time print outs to the screen, residual studies, restart options and files, etc. Convergence and divergence checking were added to both Euler2D and Euler3D using *rsdtol* and *rsdmax*. These features are available to exit the cycles and iterations early. The solvers also exit early if the residual becomes equal to NAN or INFINITY (*test_NAN* and *test_infinity* in *utils*).

The constants were updated to be double precision. Rational numbers were left in place, while the irrational numbers were turned into parameters. For example, the number 1/3 was stored as the parameter “c13” so that the compiler calculates 1/3 in double precision and compiles it in place in the code like the rational numbers.

Cowan implemented the lumped mass matrix scaled by all of the constants seen in all contributions to the residual vector. Cowan did this to minimize computations, but this left the code ambiguous to those adapting the residual. The lumped mass matrix is now stored in its true form and the constants are applied to the appropriate components of the residual.

Locations where the constants could not be applied to match the development are marked with comments.

Cowan created a single routine (*solve2d_1pt* and *solve3d_1pt*) in each solver to assemble the residual vectors and update the unknowns during each iteration. This same routine also calculated the local time steps and called for artificial dissipation. These routines were split into 8 subroutine calls:

- *dt_calc* :: local time step calculations
- *flux_src_1pt* :: unsteady, flux integrals, and source terms
- *inv_wall_1pt* :: inviscid wall boundary integrals
- *sym_plane_1pt* :: symmetry plane boundary integrals
- *far_field_1pt* :: far field boundary integrals
- *rhs_flow_tang* :: flow tangency applied to RHS vector
- *pc_update* :: predictor-correction update of unknowns
- *rsd_calc* :: RMS of residuals

Further, the correction of boundary fluxes using Riemann invariants was also moved from *far_field_1pt* to *riem_inv*, which would be used to construct later boundary integrals. The previous list of subroutines were used to pull apart *solve2d_3pt* and *solve2d_4pt*. Only the flux integrals and source terms were integrated using second order quadrature. In Euler2D, higher order quadrature was adapted into *solve2d_npt* that is capable of integrating with one, three, or four Gauss points. The domain integrals, source terms, and inviscid boundary integrals are adapted to the number of Gauss points: *flux_src_npt* and *inv_wall_npt*. The other boundary integrals are evaluated using the single point routines. *solve2d_cont* was also

created with analytical integrals that are developed in Appendix B and implemented in *flux_src_cont* and *inv_wall_cont*. Analytical integrals are used when *ipnt* = 0.

Time step tracking was added in two forms: The number of cycles required to converge the energy residual to *rsdtol* is written to *case.cyc*. The ratio of local time steps is written to *case.time*. For the inviscid codes, the ratio of local to global time steps is written to the file. This ratio *utime* can be used by the user along with the number of cycles in *case.cyc* to determine an appropriate global time step and number of cycles per step. Later the ratio of different viscous local time steps to the previous local time step is written so that the user can determine an appropriate viscous time stepping scheme.

5.1.2 Upgrade to Rigid Body Dynamics Model

Cowan (2003) developed the original rigid body model, and O'Neill (2005) adapted that model to use quaternions instead of Euler angles. The rigid body model read mass, stiffness, and damping matrices from the *case.dyn*. The mass matrix included the mass of the body along the diagonal of the translational DOFs and inertial matrix for the rotational DOFs. The mass matrix was not checked for consistency in body mass and assumed *y*-symmetry in the inertial matrix, although the matrix was inverted as though fully populated. The inverted matrix was only applied to part of the rotational motion equation.

The current implementation

- checks the mass matrix for consistency in vehicle mass;
- reads, inverts, and applies the entire inertial matrix without assumptions;
- reads the entire damping and stiffness matrices but applies only the diagonal when updating the rotational DOF; and,
- corrected the nonlinear rotation terms to include the full inertial matrix.

The stiffness and damping matrices are limited in their use. The two matrices use the global position and velocity to contribute to the forces and moments on the body. These terms are not appropriate for free flying vehicles. The damping matrix has been used to model a drogue chute on a reentry vehicle. The stiffness and damping terms are most useful in modeling semi-stationary cases, like wind tunnel models.

All of the components discussed here have been verified using analytical means. These new features need to be tested on complex (asymmetric) vehicles and motion.

5.1.3 Creation of NS2D/3D

Viscous terms were added to Euler2D and Euler3D to create NS2D and NS3D. Several new controls, sizes, and arrays were added, and some of the existing arrays were enlarged to contain necessary information. The viscous wall boundary conditions and viscous flux integrals are added to the existing routines. Viscous and propulsion momentum were added to *get_force*. One new routine was added: *read_temp_bc*, which reads and stores the heat transfer boundary conditions. The viscous terms utilize many common factors in their denominators. Special care was used to minimize the number of divisions in the final implementation. Finally, viscous local time stepping was added to the model. All of these features are described here.

Six viscous controls were added and one control was removed in the conversion. The obvious controls are Reynolds number Re and Prandtl number Pr , which are included in all of the viscous equations in Chapters 3 and 4. The Reynolds number is calculated using free-stream properties and the reference length *refdim*: $Re = \rho_{\infty} V_{\infty} L / \mu_{\infty}$. The viscosity is controlled through two values: *Smod* and *lamb*. *Smod* is the modified Sutherland's

coefficient, which is used to control how viscosity varies with temperature (described in Section 5.1.6). *lamb* is the ratio of second to first viscosity, which must be greater than $-2/3$. The lower limit is generally assumed in other CFD solvers, but *lamb* has been left for the user to control for further research into how *lamb* can be used in shock capturing. The zero dissipation length *dislen* is used to scale down the artificial dissipation near solid walls. *dislen* is discussed in Section 5.1.5. *itime* was expanded to include viscous local time stepping, which is discussed later in this section. The temperature boundary conditions are read from the case.tbc file when *itempbc* is set to true. Finally, NS2D and NS3D are evaluated using one point quadrature so *ipnt* is removed (more precisely restricted to 1).

Two new geometry arrays were added, and two original arrays were amended to handle the additional geometry required for viscous analysis. *IBEL* originally contained the nodes defining each boundary element and the surface index to which that surface belongs. The stress on the adjacent element is used to evaluate the stresses along the boundary; this index is stored as the last entry in *IBEL* as read from the case.g2d or case.g3d. Moffitt (2004) used the element index to calculate the gradients while evaluating the boundary element. A faster method is to evaluate the boundary elements attached to a domain element, just after the domain element has been evaluated. To accomplish this, *IELM* was expanded to include two pointers to possible boundary elements for NS2D and three pointers for NS3D. The minimum distance to the wall was needed for turbulence modeling. The wall distance *DWALL* was calculated as the shortest distance between the node of interest and a wall node. A characteristic length for each element was needed for viscous stability. The minimum distance between the face of an element and its opposing vertex was stored in XLE. This

distance was calculated using Eqs. 4.296 and 4.297 for NS2D and NS3D, respectively. All of this occurs in *geom2d* and *geom3d*.

Several viscous and propulsion pointers were stored to minimize the calculations during each step. The number of viscous wall nodes *nwlv* and number of viscous singular nodes *nsdv* are stored in the case.g2d and case.g3d along with the limits for the viscous wall elements *LBE(7:8)*. The limits for propulsion boundary elements were calculated within the routine and stored as *LBEpr1* for the lower limit and *LBEpr2* for the upper limit, using the number of rocket and engine boundary elements. (These propulsion limits will be explained later.)

$$LBEpr1 = LBE(6) + 1 \quad (5.1)$$

$$LBEpr2 = LBEpr1 + sum(NRKT(:)) + sum(NENGO(:)) \quad (5.2)$$

These limits are used to define the ranges for applying the no-slip condition and viscous contributions to wall elements. The nodes and boundary elements in the geometry files were organized by Cowan to minimize searching and pointers. Figure 5.1 shows the organization of nodes in the geometry files, and Figure 5.2 shows the organization of boundary elements. Cowan stacked the wall nodes at the beginning of the COOR array. This work moved the viscous wall nodes to the middle of that array. The wall nodes are organized so that viscous geometry files can be used in inviscid simulations without any adaptation. The number of singular viscous wall nodes is important only as a pointer to track the beginning of the viscous nodes. The limits on flow tangency and no-slip are given in Eqs. 5.3 and 5.4. These two conditions are applied using Eqs. 4.90 and 4.95, and then the total enthalpy is corrected using Eqs. 4.94 and 4.96. Similarly, the residual is reset to zero when Eq. 4.95 is applied:

$$1 \leq ind_{inv} \leq nwl - nsd + nsdv - nwlv \quad (5.3)$$

$$nwl - nsd + nsdv - nwlv + 1 \leq ind_{vis} \leq nwl - nsd + nsdv \quad (5.4)$$

Nodes:

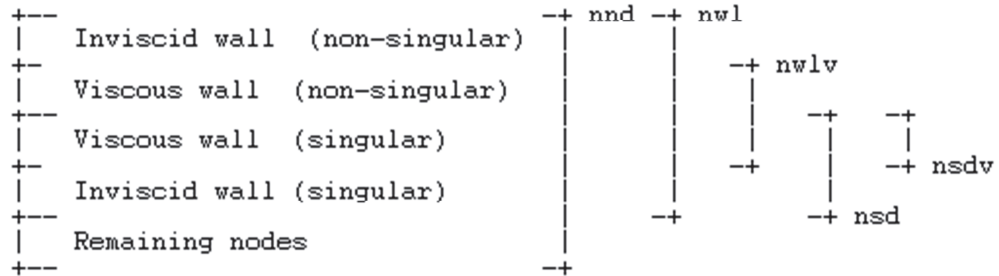


Figure 5.1: Organization of Nodes in case.g2d and case.g3d.

Boundary elements:

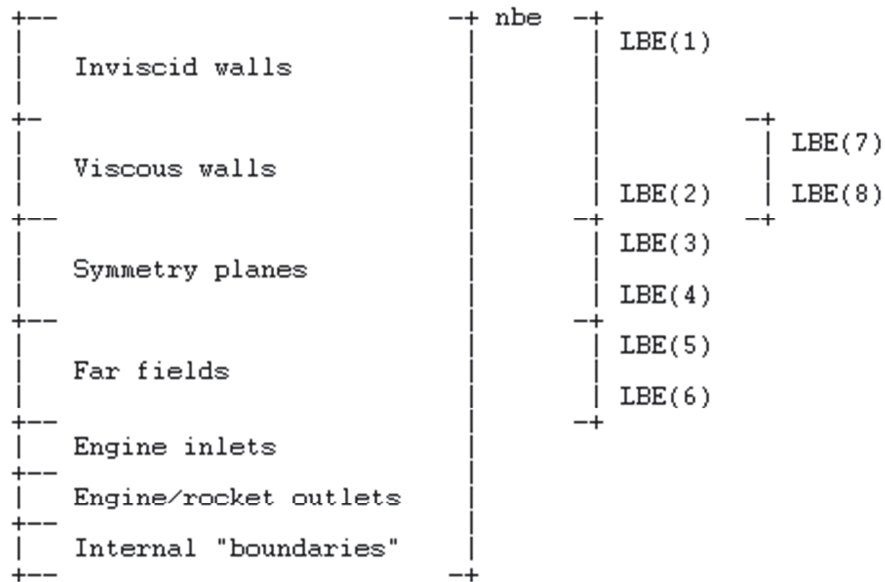


Figure 5.2: Organization of Boundary Elements in case.g2d and case.g3d.

After the contributions from a domain element have been added to the residual vector *RHS* in *flux_src_1pt*, the contributions along any viscous boundary elements attached to that domain element are also added to the residual. The pointers at the end of *IELM* are used to determine what boundary elements (if any) are attached to each element. If the pointer(s) are non-zero,

then the boundary element is identified with walls $LBE(1:2)$, far field $LBE(5:6)$, or propulsion planes $LBEpr1$ to $LBEpr2$. These elements are all handled by using the stresses and heat fluxes from the adjacent element to calculate the boundary normal fluxes (Eqs. 4.88, 4.89, 4.99, and 4.133). For symmetry planes $LBE(3:4)$, the normal velocity and normal derivatives are removed from the stresses and heat fluxes using Eq. 4.101 along with Eqs. 4.111 through 4.113 in NS2D and Eqs. 4.116, 4.119, and 4.128 in NS3D.

Heat Transfer Boundary Conditions. Heat transfer conditions have been applied in NS2D and NS3D, although the conditions in NS2D have only been loosely verified. Three boundary conditions are available:

- Adiabatic wall :: the heat flux is assumed to be zero for boundary integrals
- Specified heat flux :: the user specifies the heat flux by boundary element
- Specified temperature :: the user specifies the enthalpy as wall nodes

Adiabatic walls are the default condition and are assumed even if the other two conditions are assigned. The user can override the heat flux on $nbeq$ specific elements through their entry in the case.tbc file. The user can override both of these conditions by specifying the enthalpy at $nwlt$ wall nodes using the case.tbc file. This enthalpy was originally ramped over $ntemp$ iterations; but, after the application of viscous local time stepping, the wall enthalpy is applied in the first iteration. If the temperature boundary conditions flag $itempbc$ is set to true in the case.con, then the boundary conditions file case.tbc is read within $read_temp_bc$. The heat flux for all boundary elements is stored in $QWALL$, which is initialized to zero and replaced as specified in case.tbc. The treatment of wall nodes is stored $TWALL$ with pointers in $ITWL$. The heat fluxes are applied to wall elements in $flux_src_1pt$, and the enthalpy is

constrained at specified wall nodes in *set_bound* using Eq. 4.97. Because testing is required, a warning is issued to the screen when these conditions are being applied.

Minimization of Division. Division is very costly so efforts have been taken to reduce the occurrence of division while maintaining readability and accuracy of the codes. Irrational numbers like $1/3$, $1/6$, $1/12$, and $1/24$ have been assigned to parameters, which are inserted into the code during compile. Several controls are repeated as division: Cowan converted $1/dt$ to *dtI* and $gam-1$ to *gmI*. Similarly, this work has converted gam/gmI to *ggI*. Similarly the viscous controls *Re*, *Pr*, *PrT*, and *dislen* have inverses *Re_1*, *Pr_1*, *PrT_1*, and *dislen_1*. Any repetition a particular division is stored as its inverse in *denom* and then applied through multiplication. The original code has been kept as a comment above the adapted version.

Viscous Local Time Stepping. Viscous local time stepping has been applied in a series of similar subroutines using the equations in Section 4.5.4. If *itime* = -1 (or the default is used), then the path diverts to *dt_calc_visc_le*, which assembles the viscous local time steps using the minimum of Eq. 4.295. If *itime* ≥ 0 , then Eq. 4.290 is used to evaluate the time steps through the viscous stiffness matrix. For *itime* ≥ 0 , the heat transfer matrix is assembled in *dt_calc_visc_heat* using Eqs. 4.298 through 4.304, where $j = \alpha = 1$. For *itime* ≥ 1 , the momentum matrices are assembled in *dt_calc_visc_mom* using the same equations, where $j = \alpha$ is the component of momentum being tested. For *itime* = 2, the appropriate turbulence matrix is assembled in *dt_calc_visc_SA* or *dt_calc_visc_SST*, using the same equations and $j = \alpha = 1$. The nodal local time step is calculated using Eq. 4.291. As the different tests are made, the minimum ratio of time steps is evaluated and written to the case.time. The organization of this file is described in Appendix D and F.

5.1.4 Proper Tracking of Total Energy / Pressure

The velocity along symmetry and solid wall boundaries is often reduced to maintain the appropriate conditions along the boundary. If the total energy and enthalpy is not maintained properly, pressure and temperature gradients are artificially created along these boundaries. Eqs. 4.94 and 4.96 are used to adapt the total properties and maintain consistent static properties as velocity constraints are applied to walls and symmetry planes.

5.1.5 Zero Dissipation Length

Artificial dissipation affects the solution in a similar manner to viscous dissipation. This effect is most noticeable near the wall, where the shear stress is changed by the presence of artificial dissipation. Figure 5.3 shows the velocity profiles from a laminar boundary layer ($M = 0.3$, $800 \leq Re_x \leq 3200$) with and without artificial dissipation for the same mesh. The solution without artificial dissipation matches the theoretical velocity profile, skin friction, and thickness very well. Artificial dissipation attempts to reduce gradients in the flow, including those near the wall. The result is a reduced shear stress and velocity throughout the boundary layer. The thickness of the boundary layer also increases.

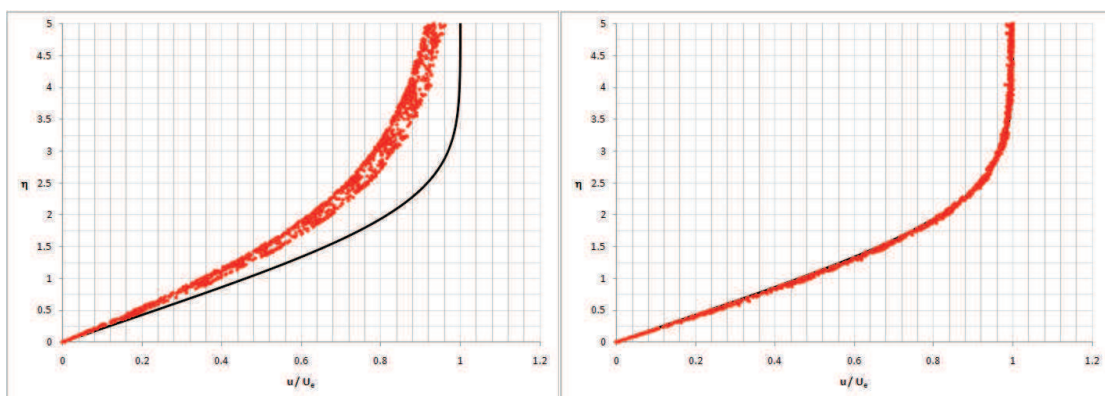


Figure 5.3: Laminar Velocity Profiles from NS2D with $diss = 1.0$ (Left) and 0.0 (Right).

The artificial dissipation is necessary in the external flow for capturing shocks and other flow features. Within the boundary layer, the opposite is true. Artificial dissipation reduces the skin friction, thickens the boundary layer, and adds error to the near wall solution. We desire to have artificial dissipation in the external flow but no extra dissipation in the boundary layer, especially near the wall.

The artificial dissipation in NS2D and NS3D was scaled

by the function shown in Figure 5.4. The effective

dissipation $diss_{eff}$ is zero over a distance $dislen$ near the wall. A cosine function is used to spline to the external flow over four $dislen$'s. The scaled model does not provide the necessary stability at the onset of a new solution, so the function is ramped down from unity to that shown in Figure 5.4 over 100 iterations. $diss_{eff}$ is calculated in the routine `diss_scalar`.

The new model works well in NS2D for $dislen$ equal to half of the maximum boundary layer thickness, giving results similar to Figure 5.3 without any artificial dissipation. $dislen$ is always active as a control, but the effect can be removed by setting $dislen$ equal to 10^{-20} .

5.1.6 Sutherland's

The viscosity μ is calculated from the freestream Reynolds number and scaled according to Sutherland's law (Eq. 3.223). Sutherland's coefficient S has been modified to remove its dimensions (Eq. 3.226). S_{mod} can be set to zero to avoid the use to Sutherland's equation. The viscosity ratio is set to unity when S_{mod} is zero. If S_{mod} is positive, then Sutherland's

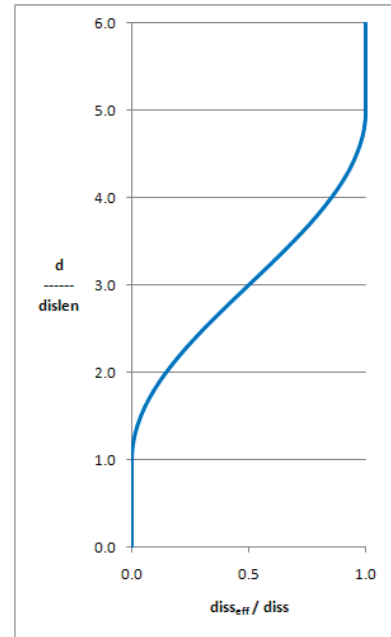


Fig. 5.4: Effective Dissipation.

equation is applied in *suther*, using Eq. 3.30 to calculate the local and freestream enthalpy. The thermal conductivity k is scaled from viscosity using the Prandtl number ($Pr = \mu c_p / k$), which is constant for a given problem.

5.1.7 Acoustic Output Files

The acoustic output files report density, velocity, and pressure data at a series of points defined by points and lines in space. The points exist in an element in the domain and are therefore interpolated from its points. The lines cross one or more elements in the domain and are interpolated where the line enters and leaves each element, along the segments between its nodes. The mathematics for calculating the acoustic points and lines in 2D and 3D can be found in Appendix G and H, respectively. The mathematics are setup in *read_acoustics* and utilized between iterations in *write_acoustics*.

The acoustic data is stored in four new arrays and two new sizes: The number of acoustic points is tracked in *nacp*, and the number of lines is kept in *nacl*. The maximum number of pointers per acoustic line is stored in *mxac*, which is hard coded just become the acoustic arrays are sized. The files to be written out are stored in the array *IOUT*, which corresponds to the files: case.rac, case.pac, case.uac, case.vac, and (for the 3D codes) case.wac. These files store data for density, pressure, and velocity, respectively, at all locations specified in the acoustics input file case.acst. The data is written to the output files after every iteration. The number of points in each acoustic line is stored in *NLN*. *IACST* holds all of the element indices for *nacp* acoustic points followed the elements crossing the acoustic line. *RACST* gives the local coordinates that are used to interpolate within the element specified by *IACST*.

5.2 NASA-CFDsol

Several adaptations were made to CFDsol during the NASA contract. Changes to the far field boundary, artificial dissipation, and update equation were discussed in Sections 4.2.3.1, 4.2.6.1, and 4.2.7.2 of Chapter 4, respectively. The other adaptations are discussed here. As the kinetic energy along solid walls and symmetry planes is adapted to constrain the velocity, the total energy has been adapted to maintain constant pressure. Rigid body and elastic deformations were added to the solver using routines from Euler3D. The viscous contributions, Sutherland's equation, and artificial dissipation were adapted to increase the accuracy of viscous simulations. Propulsion models were also added, and the SA turbulence model was corrected. These features are discussed in later sections.

5.2.1 Proper Tracking of Total Energy / Pressure

Similar to the OSU in-house codes, the total energy along solid walls and symmetry planes must be adapted as flow tangency (Eq. 4.90) and the no-slip (Eq. 4.95) conditions are applied. The total energy is stored in CFDsol and used to calculate the pressure at specific times. If the total energy is corrected to exclude the lost kinetic energy, then the pressure will not change. The total energy is corrected using Eqs. 4.94 and 4.96.

5.2.2 Non-Inertial Frame and Rigid Body Dynamics Model

The source terms applied with one Gauss point (Eq. 4.216) can be found in *flux_src_1pt* in the OSU in-house codes. Likewise, the rigid body model (Eqs. 3.254 through 3.268) comprises half of *asedrv*. The other half of the routine is the modal elastics model. These routines were transferred from Euler3D into CFDsol and then adapted to the different array structures used in CFDsol. The model was then verified by two simple test cases.

Steady Conditions over Airfoil. The non-inertial source terms were divided into three categories for testing: (1) Rotation of the frame; (2) constant velocity on the frame with freestream velocity; and (3) constant velocity of the frame alone. The NACA 0012 airfoil was tested at Mach 0.3, 5-degrees angle-of-attack as a baseline case. Each of the three cases was tested to generate a motion at the far field equivalent to the baseline.

In the first case, the freestream velocity was applied using *ifree* and then the frame was rotated by 5 degrees, simulating the angle of attack. (The mesh is shown in Figure 6.1 in the Chapter 6.) The results of the rotated frame (as “Non-Inertial, 5 deg”) are shown in Figure 5.5 in comparison to the baseline case (as “Inertial”). The two distributions compare very well, as expected since the far field velocity in the non-inertial frame has been rotated using **B** to have the same effects as applying an angle-of-attack in the inertial frame.

In the second case, the freestream velocity was applied using *ifree* and the frame was given a vertical velocity to create an equivalent angle-of-attack. The vertical velocity was calculated:

$$w_i = -V_\infty \tan 5^\circ \approx -0.087489$$

This mesh velocity creates a far field with an equivalent angle-of-attack but larger velocity magnitude (approx. 0.4%). The magnitude change is small so that its effects will be negligible. The results of the vertical mesh velocity (as “Non-Inertial, vel w/ free”) are shown in Figure 5.5. The distribution compares well to the baseline as expected since the far field velocity has an equivalent angle of incidence and only a slight change in magnitude.

In the final case, the freestream velocity was not used (*ifree* was turned off). Instead the velocity was created using the mesh velocity alone:

$$\vec{V}_t = \begin{Bmatrix} u_t \\ 0 \\ w_t \end{Bmatrix} = -V_\infty \begin{Bmatrix} 1 \\ 0 \\ \tan 5^\circ \end{Bmatrix} \approx \begin{Bmatrix} -1 \\ 0 \\ -0.087489 \end{Bmatrix}$$

The far field velocity is equivalent to the previous case. The results of the total mesh velocity (as “Non-Inertial, vel no free”) are shown in Figure 5.5. The distribution compares well to the baseline as expected since the far field velocity has an equivalent angle of incidence and only a slight change in magnitude. The solution is approaching the baseline solution but takes longer to converge. Stability is a particular concern because of the convergence rate of the solution, seen in the noise in the final solution. These three cases demonstrate the effectiveness of the non-inertial source terms and show that the implementation is correct.

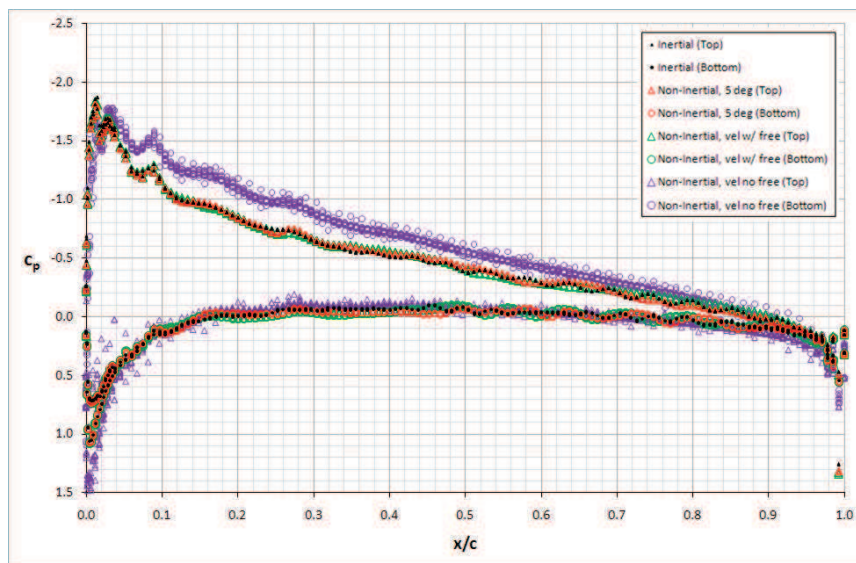


Figure 5.5: Pressure over NACA 0012 Airfoil (CFDsol, Inviscid, Non-Inertial).

Translating and Rotating Domain. The non-inertial source terms were also tested under constant velocity, acceleration, and system dynamic scenarios. The non-inertial solution was

shown to be stable under translation and rotation. Figure 5.6 shows a snapshot of the velocity solution for an empty spherical domain under translation and rotation. Time accuracy was also tested. The system model was tested by isolating the terms: Mass inversion, stiffness, and damping. The translational and rotational terms were tested separately and in conjunction. Particular attention was paid to the inversion of the inertial matrix. Exact analytical solutions were used for comparison.

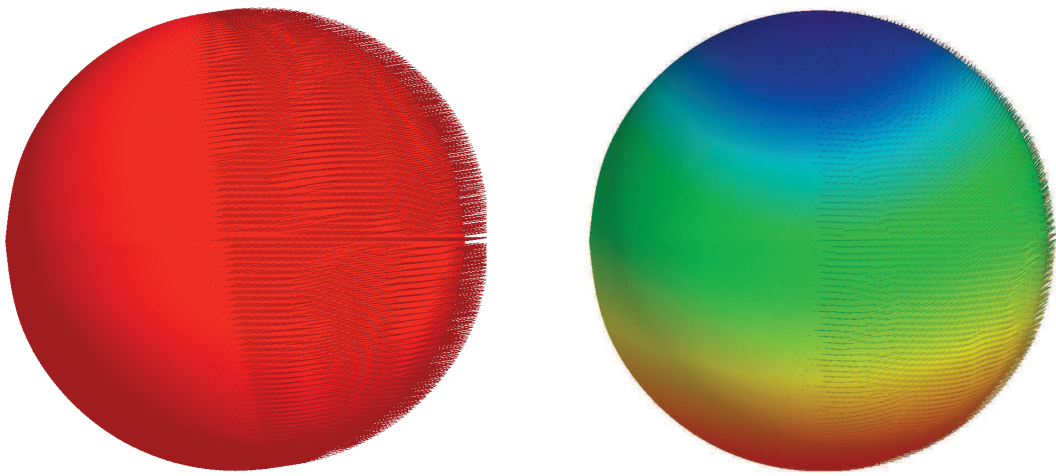


Figure 5.6: Empty Domain under Non-Inertial Translation (Left) and Rotation (Right).

5.2.3 Structural Dynamics Model

Transpiration (Eq. 4.90) are applied in *set_bound* in Euler3D using the rotated normal vector (as in Figure 3.2). The boundary velocity *BVEL* is calculated in *transpir3d*. The modal elastics model is the second half of *asedrv*. *asedrv* was transferred into CFDsol along with the rigid body model. The boundary velocity and transpired flow tangency were applied in CFDsol, using Euler3D as the standard. The airfoil model was then repeated for verification.

The NACA 0012 airfoil at Mach 0.3 was tested at 5 degree angle-of-attack. The results were then compared to the same mesh at zero angle-of-attack with a mode shape deflection representing a 5-degree rotation of the airfoil. The surface pressure for the 5-degree AOA is shown as black dots in Figure 5.7 (as “Inertial, AOA”); the surface pressure for the transpired rotation is shown as red dots (as “Transpired Angle”). The two distributions compare very well. The transpired rotation models the actual displacement very well.

The results were then compared to the same mesh at zero angle-of-attack with a mode shape velocity representing an airfoil plunging at an equivalent 5-degrees AOA. In other words, the velocity everywhere on the airfoil was specified to be:

$$\left(\vec{V}_b\right)_i = -V_\infty \tan 5^\circ \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix} \approx -0.087489 \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix}$$

The surface pressure for the plunging airfoil is shown in Figure 5.7 (as “Transpired Vel”).

The distribution is similar to the stationary airfoil and contains the differences expected since the airfoil is plunging instead of pitched. The two cases demonstrate the effectiveness of the transpired boundary condition to model structural or controls deflections and wall velocities.

5.2.4 Viscous Terms

The viscous contributions defined by Eqs. 4.44, 4.50, and 4.178 were already present in CFDsol. The viscous equations were checked for accuracy. The boundary integrals \mathbf{f}_σ were applied using one-quarter the element volume V_e , like the domain integrals. This was changed to one-third of the boundary element area A_{be} . This change does not affect the velocity along no-slip walls or any property along far field boundaries because these

properties are reset each iteration according to the boundary conditions. The boundary contributions are now correct for all other properties, especially along symmetry plane.

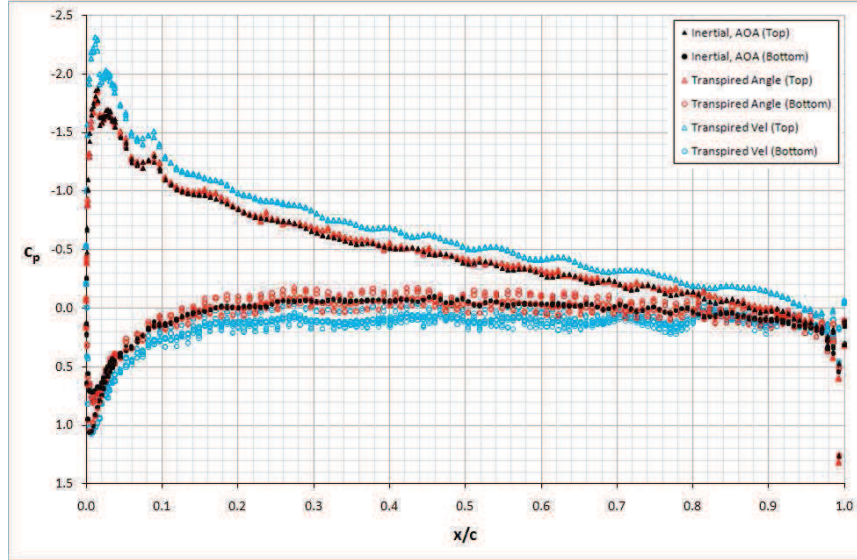


Figure 5.7: Pressure over NACA 0012 Airfoil (CFDsol, Inviscid, Transpiration).

5.2.5 Sutherland's Equation

CFDsol uses Sutherland's equation (Eq. 3.35) to model the variation of viscosity with respect to temperature. The other properties are setup to handle air ($S = 198.6$ °R). Eq. 3.35 is non-dimensionalized in a different manner in CFDsol than in the OSU in-house codes:

$$\mu^* = \left(\frac{h}{c_p T_\infty} \right)^{1.5} \frac{c_p T_\infty + c_p S}{h + c_p S} = \left(\frac{h^*}{C_1} \right)^{1.5} \frac{C_1 + C_2}{h^* + C_2} \quad (5.5)$$

$$C_1 = h_\infty^* = \frac{c_p T_\infty}{U_\infty^2} = \frac{\gamma R T_\infty}{\gamma - 1} \frac{1}{U_\infty^2} = \frac{1}{\gamma - 1} \frac{a_\infty^2}{U_\infty^2} = \frac{1}{(\gamma - 1) M_\infty^2} \quad (5.6)$$

$$C_2 = S^* = \frac{c_p S}{U_\infty^2} = \frac{c_p T_\infty}{U_\infty^2} \frac{S}{T_\infty} = \frac{198.6}{(\gamma - 1) T_\infty M_\infty^2} \quad (5.7)$$

The freestream temperature T_∞ is specified through $TINF$ in the case.data file in degrees Rankine ($^{\circ}R$). $TINF$ was later overwritten with freestream enthalpy (Eq. 5.6) and used to update the far field boundary conditions. To correct this problem, $TTINF$ was assigned the freestream enthalpy in the far field calculations, so that $TINF$ retains its assigned value.

A new option $IVIS$ was added to the solver so that constant viscosity solutions can be calculated. If $IVIS$ is set to 1, then the viscosity throughout the domain is equal to the freestream viscosity, set through the freestream Reynolds number RE . For any other values of $IVIS$, viscosity is calculated according to Sutherland's law, as shown in Eqs. 5.5 through 5.7.

5.2.6 Zero Dissipation Length – Instability

The artificial dissipation in CFDsol decreases the effective Reynolds number near the wall, beyond acceptable levels. The zero dissipation length $dislen$ was implemented in CFDsol in the same manner as NS3D. This lead to the detection of many modes of instability.

5.2.6.1 Crossflow Instability

The cross-flow instability in CFDsol was first discovered with the laminar boundary layer cases. These cases are very thin cross-sections bound by two symmetry planes. Figure 5.8 shows this geometry looking down from above, where the cross-flow velocity vectors become very apparent. (The vectors are best seen with the color spectrum off on a black background.) The symmetry planes are very well behaved in this situation; CFDsol removes any normal velocity along the symmetry planes. The velocity vectors protruding through the symmetry planes in Figure 5.8 all originate within the domain.

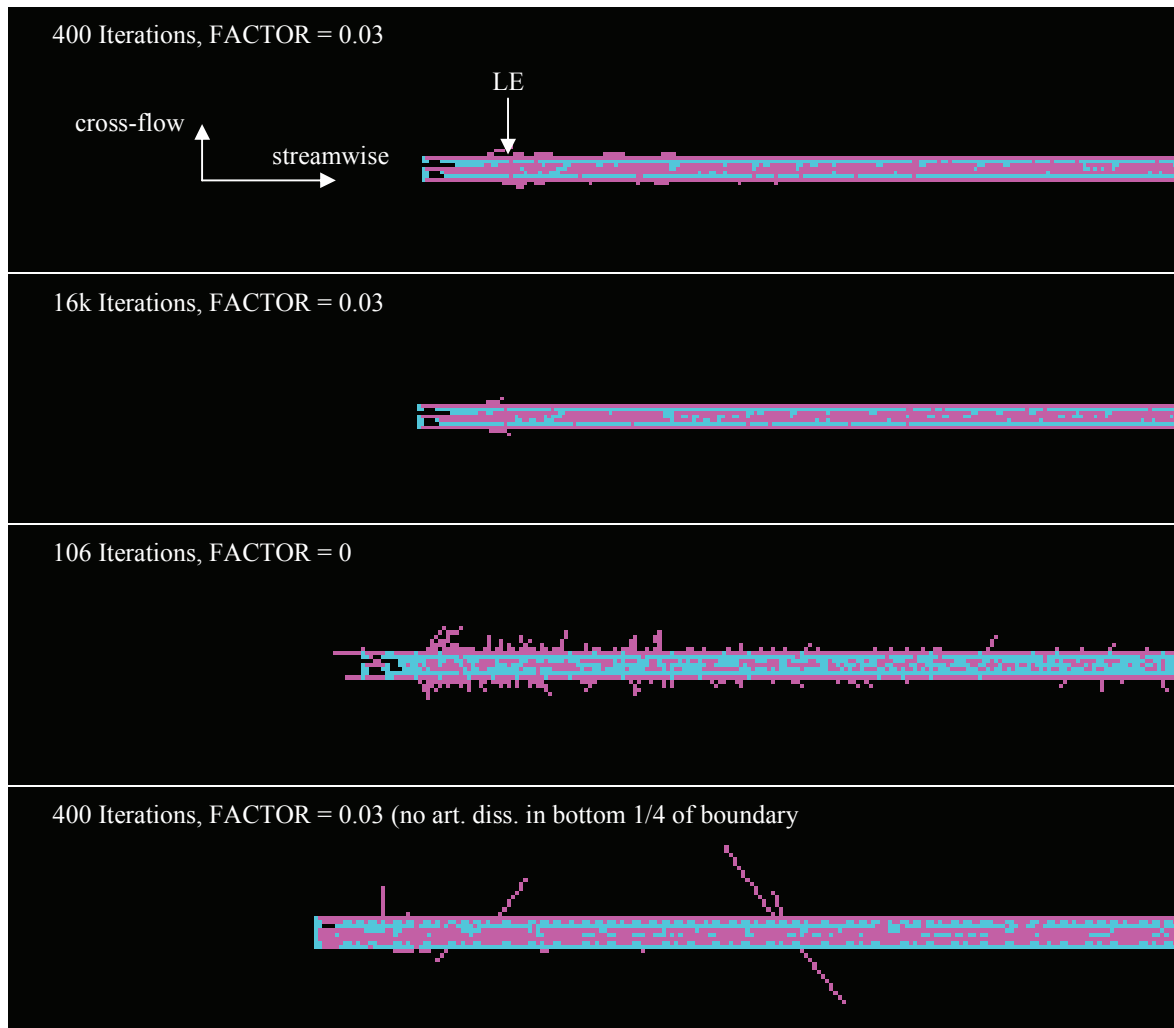


Figure 5.8: Cross-Flow Velocity Vectors (Looking Down on Plate, Sym on Top & Btm).

Figure 5.8 shows four cases that demonstrate the cross-flow instability and the effectiveness of artificial dissipation in controlling the cross-flow velocities. The cross-flow velocities perturb early in the solution (400 iterations). Many more iterations are required for artificial dissipation to bring the velocity back under control (16k iterations). If no artificial dissipation is used, then the cross-flow velocity diverges everywhere, much earlier in the run (106 iterations). The problem is not limited to the higher velocity flow at the top of the boundary layer. When artificial dissipation was removed from the bottom quarter of the boundary

layer using *dislen*, the near-wall velocities still diverged. The boundary layer case was also tested in several different orientations (*xy*-, *xz*-, and *yz*-planes and their obliques) to ensure that the instability was not a coding error. The problem is invariant to the orientation angle. Eight nodes were selected for the speed and magnitude of their cross-flow instability. The contributions from the inviscid, viscous, and artificial dissipation terms in the update equation were tracked in time. A plot of one of these nodes is shown in Figure 5.9. The left portion of the figure illustrates how the inviscid, viscous, and artificial dissipation terms contribute to the cross-flow velocity as the three terms balance each other out. The right portion highlights the first 100 iterations. The inviscid contributions jump across the first iteration and continue to strongly influence the velocity. The viscous and artificial dissipation terms grow to balance the inviscid contributions, keeping the solution from diverging completely. Artificial dissipation is much more powerful in the demonstrated case, explaining why the solution diverged much more quickly when no artificial dissipation was used in the solution. The cross-flow velocity takes on a very similar shape to the inviscid contributions, which shows that the inviscid terms are contributing to the cross-flow instability.

The inviscid contributions can be decomposed into the unsteady, momentum flux, and pressure terms:

$$RSD = \underbrace{\mathbf{M}_c(\tilde{\mathbf{v}}^{i+1} - \tilde{\mathbf{v}}^i)}_{\text{unsteady term}} + \underbrace{\Delta t \left(\frac{\partial u_i}{\partial x_i} \mathbf{M}_c + \mathbf{K} \right) \frac{1}{2} (\tilde{\mathbf{v}}^{i+1} + \tilde{\mathbf{v}}^i)}_{\text{momentum flux term}} + \underbrace{\Delta t \mathbf{R}}_{\text{pressure term}} - \underbrace{\Delta t (\mathbf{K}_\sigma + \mathbf{f}_\sigma)}_{\text{viscous terms}} - \underbrace{\Delta t \hat{\mathbf{R}}}_{\text{artificial dissipation}} \quad (5.8)$$

The contributions to the cross-flow velocity from the unsteady, momentum flux, and pressure terms are plotted in Figure 5.10. Pressure contributes the largest contributions to the velocity

and abruptly changes in the first iteration. The pressure contributions represent the velocity changes seen in Figure 5.9.

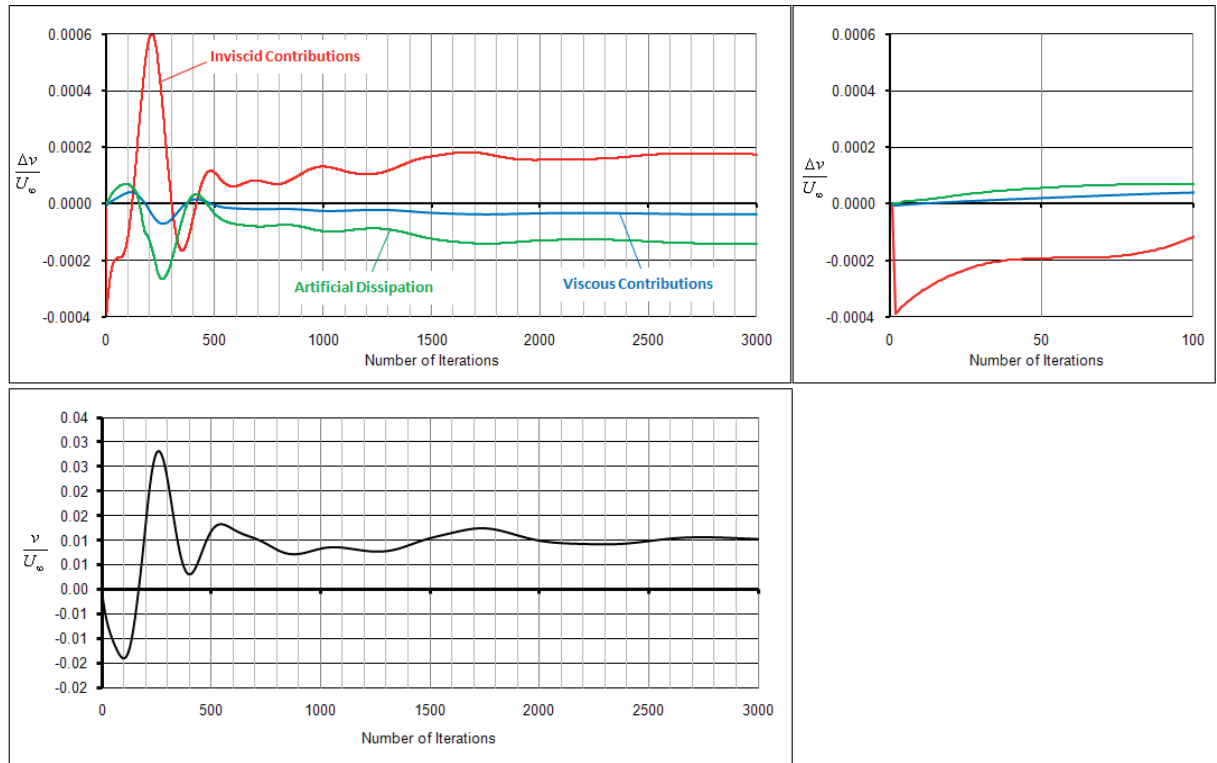


Figure 5.9: Contributions and Cross-Flow Velocity at Single Node.

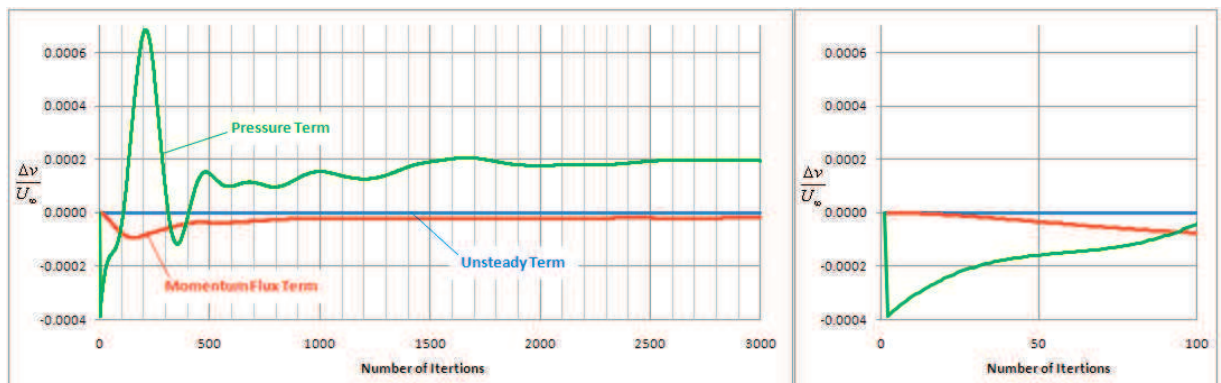


Figure 5.10: Inviscid Contributions to Crossflow Velocity at Single Node.

Several attempts were made to correct the pressure problem. First, the cross-flow velocity was set equal to zero for all iterations, and artificial dissipation was removed from the

solution. The solution diverged in plane with the boundary layer gradients. The divergence now moved to the vertical direction (normal to the plate). Velocity vectors from this solution are shown in Figure 5.11. The pressure gradients affect all three momentum equations, and the instability has now been demonstrated in more than the cross-flow direction.

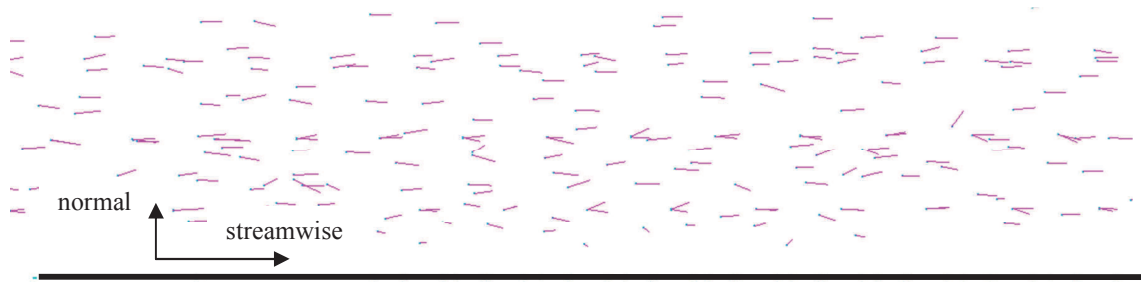


Figure 5.11: Large Vertical Oscillation near Bottom of Boundary Layer.

A similar attempt was made to neglect changes in density, cross-flow velocity, and total energy, desiring to minimize changes in pressure. The pressure changes were then proportional to the kinetic energy with no density or total energy feedback. The solution diverged slowly. Next, the pressure terms were neglected in the update equation, retaining freedom in cross-flow and other properties. The solution converged slowly. Artificial dissipation was removed, and the solution slowly diverged. Pressure is a necessary feedback for stability.

The pressure is calculated from a nonlinear equation: $p = (\gamma-1)(\rho E - \rho u_j u_j / 2)$. The pressure is represented in CFDsol using the same piecewise linear representation as the other properties. A correction factor was formulated in hopes of stabilizing the solution through a more accurate pressure gradient:

$$\frac{\partial p}{\partial x_i} = A_{ij} p_j + \varepsilon_{p,i} \quad \varepsilon_{p,i} = \frac{\gamma-1}{2} A_{ij} \left(\rho_j u_{kj} u_{kj} - \rho_j \bar{u}_k \bar{u}_k - 2 \bar{\rho} \bar{u}_k u_{kj} \right) \quad (5.9)$$

The higher order pressure gradient did not add any stability nor, for that matter, change the solution at all. A solution similar to Figure 5.10 was created using the higher order gradient.

5.2.6.2 Stability Window

The discussion above results in a mesh stability window. If the mesh is too coarse, the discretization is either inaccurate or unstable. If the mesh is too fine, the numerical error propagates through the derivatives that are used to estimate the fluxes on the domain. The result is often a mesh that requires such a small time step (or relaxation factor) that the problem becomes inefficient to solve. Sometimes, the mesh is unconditionally unstable and much time is wasted trying to find a means of stabilizing the solution. Instead the mesh must be regenerated from scratch. The search for mesh convergence often settles on a solution or mesh that is “close enough” and never as good as the user desires.

Examples. CFDsol uses explicit property derivatives that are susceptible to numerical errors, and this susceptibility grows inversely with the size of the elements. The problem was first identified with inviscid cases that could not be mesh converged. The problem reoccurred for an empty rectangular domain that was being used to verify the advection of turbulent quantities. The domain was filled with tetrahedra of equal size. The turbulence model was decoupled from the Navier-Stokes equations so the velocity profile was laminar. The case should have been inherently stable for small enough time steps. The opposite was true. The case was *unconditionally unstable*. The mesh was tested in NS2D and NS3D under laminar conditions; both were found to be stable.

The instability greatly affected the five transonic cases, which were examined in CFDsol. Pave3D was created to implement tetrahedral meshes, where the span between the symmetry

planes was only spanned by two elements. Pave3D wraps airfoil domain in a U-shape, discretizing the domain into hexahedron specified by the x -, y -, and z -spacings (or θ -spacing around the leading edge). Each hexahedron is then broken apart into five tetrahedral elements. The mesh is reported to the CFD solvers in unstructured format. The transonic meshes from Pave3D were stable for all spacings, even through the transonic shocks.

The stability problem became even more noticeable when small elements were tested in the near wall region of a laminar boundary layer. The stability of these elements was only maintained by artificial dissipation, which decreases the effective Reynolds number by at least one order of magnitude. The viscous routines in CFDsol were verified in NS3Dsol, or NS3D with the viscous routines from CFDsol. This problem gets worse for the turbulent boundary layer where the smallest element is on the order of 0.0015δ , compared to 0.1 for a laminar boundary layers.

Stability tests have been ran in Euler2D, Euler3D, and CFDsol for an empty domain, circular cylinder, and other cases. Euler2D and Euler3D were found to be unconditionally stable in the freestream case, whereas CFDsol exhibits a perfect Courant–Friedrichs–Lewy (CFL) condition (Δt proportional to Δx) for a thin domain and unconditionally unstable when multiple elements span each direction. Euler2D and Euler3D exhibit a CFL condition for subsonic and supersonic solid wall geometries. The instabilities in CFDsol are nonlinear and relax at supersonic speeds.

Ellipse. Persistent problems were found with simple cases, like the circular cylinder and ellipse. For one mesh, the inviscid ellipse diverged at a relaxed iteration of $niter \times tau = 958$, where $niter$ is the number of iterations and tau is the relaxation factor. The mesh was scaled

by 0.01, 0.1, 10, and 100 and diverged at the same relaxed iteration. A picture of the divergence is shown in Figure 5.12. The problem was circumvented by starting a fresh mesh convergence.

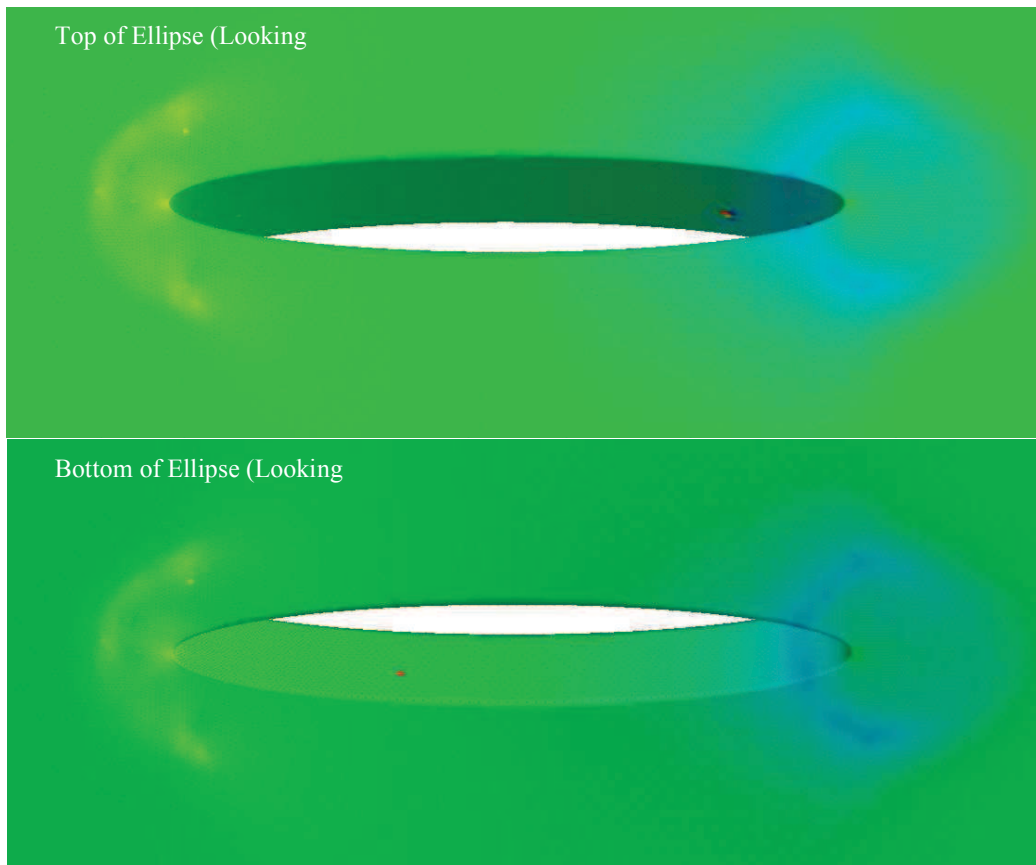


Figure 5.12: Divergence in Pressure on Surface of Inviscid Ellipse ($niter \times tau = 958$).

The stable mesh was then refined to model a boundary layer and wake region with Reynolds number of 4000 (based on ellipse length). As the boundary layer began to form, the density and pressure along the ellipse began to oscillate at one node. The oscillations were confined to one symmetry plane, so the other symmetry plane was used in the verification. The density and velocity oscillations (shown in Figure 5.14) were self-perpetuating and had a wave number equal to the inverse of $12 \Delta x$.

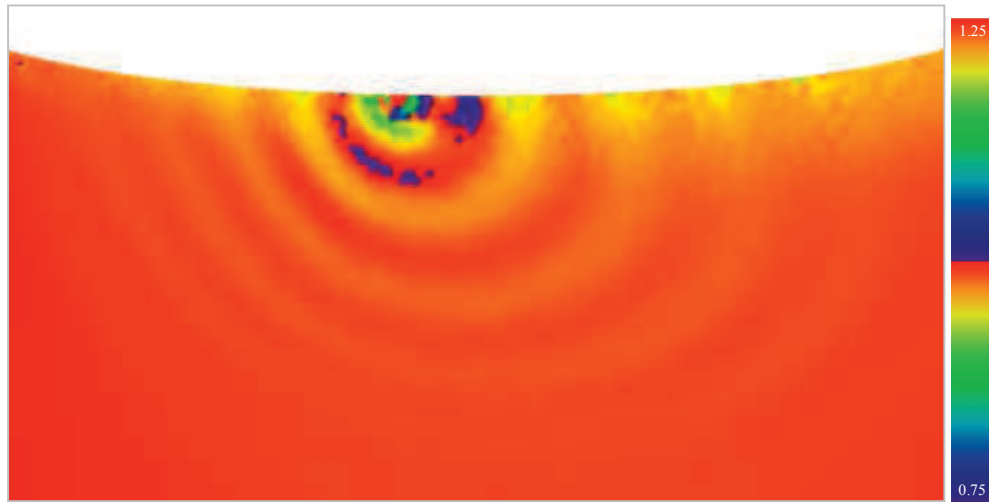


Figure 5.13: Oscillations in Density along Wall of Ellipse ($Re = 4000$).

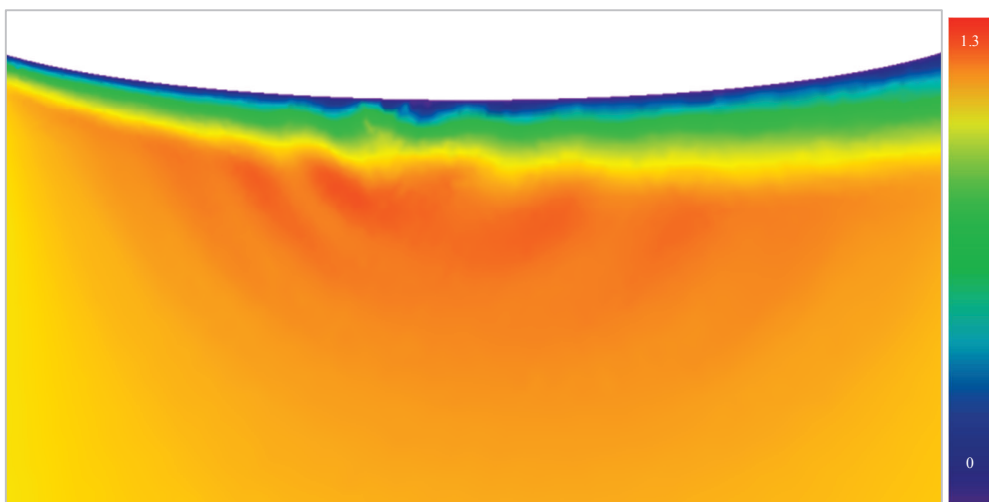


Figure 5.14: Oscillations in Velocity along Wall of Ellipse ($Re = 4000$).

Several problems were encountered with a circular cylinder. During inviscid mesh convergence, the solution on the finest mesh oscillates repetitively, as shown in Figure 5.15. The oscillations are best seen in the minimum pressure, top center of distribution. The figure is color coded so that the reader can see the oscillations more easily.

The circular cylinder was tested under several viscous conditions. Problems persisted for two particular Reynolds numbers: 26 and 41. A time step study for $Re = 41$ is shown in

Figure 5.16. The plot on the left shows the number of iterations before divergence, and the right plot shows the total CFD time (iterations times Δt) before the crash. For $\Delta t > 5 \times 10^{-6}$, the number of iterations increases for smaller time steps but the CFD time decreases. The decreasing CFD time can be discouraging. Below $\Delta t < 5 \times 10^{-6}$, stability increases and the CFD time increases quickly. At $\Delta t = 10^{-6}$, the solution showed early signs of divergence at 40k iterations (2.3 days). The solution used in the verification required a time step of 10^{-7} (3.3M iterations, 20.6 days).

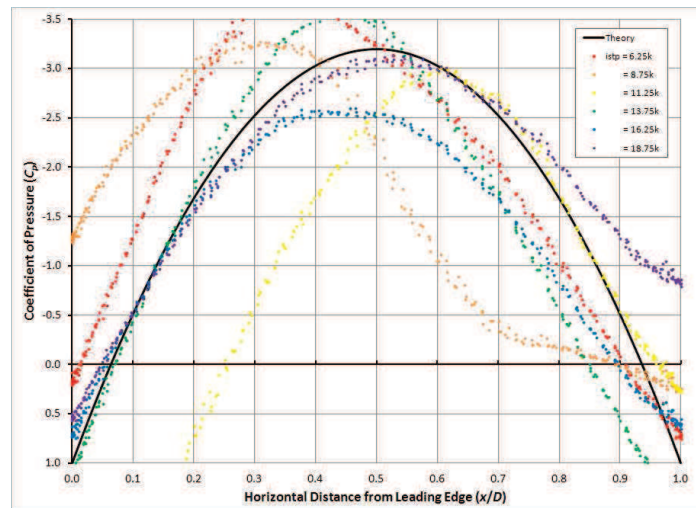


Figure 5.15: Oscillations in Pressure Distribution around Inviscid Cylinder.

The $Re = 26$ case is more interesting. Figure 5.17 shows a dual-failure mode in the time step study for $Re = 26$. For time steps below 6×10^{-7} , the velocity diverges in the near-wall region, shown in the upper left of Figure 5.17. Around $\Delta t = 5.5 \times 10^{-7}$, the failure switches erratically between the two modes. For $\Delta t < 5 \times 10^{-7}$, stability begins increasing again and divergence occurs in pressure on the wall. The CFD time for both failure modes is constant; the time step triggers the stability of one mode over the other. The divergence locations are weakly related. The initial conditions (scratch or inviscid solution) had no effect on the problem. At

$\Delta t = 10^{-8}$, the solver diverged at 230k iterations (36 hrs). The problem was solved by starting from scratch. The geometry was redrawn, meshed, and refined without taking previous “lessons” into account.

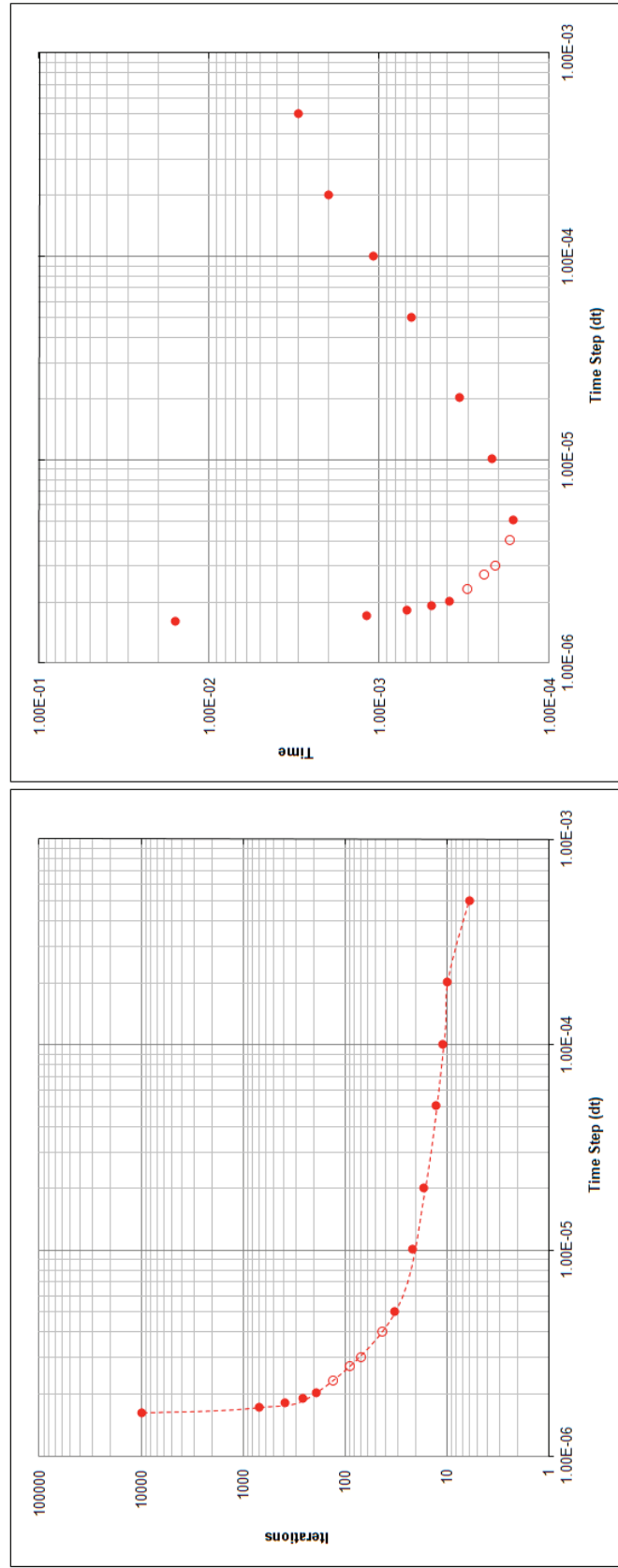


Figure 5.16: Time Step Study for Circular Cylinder ($Re = 41.0$).

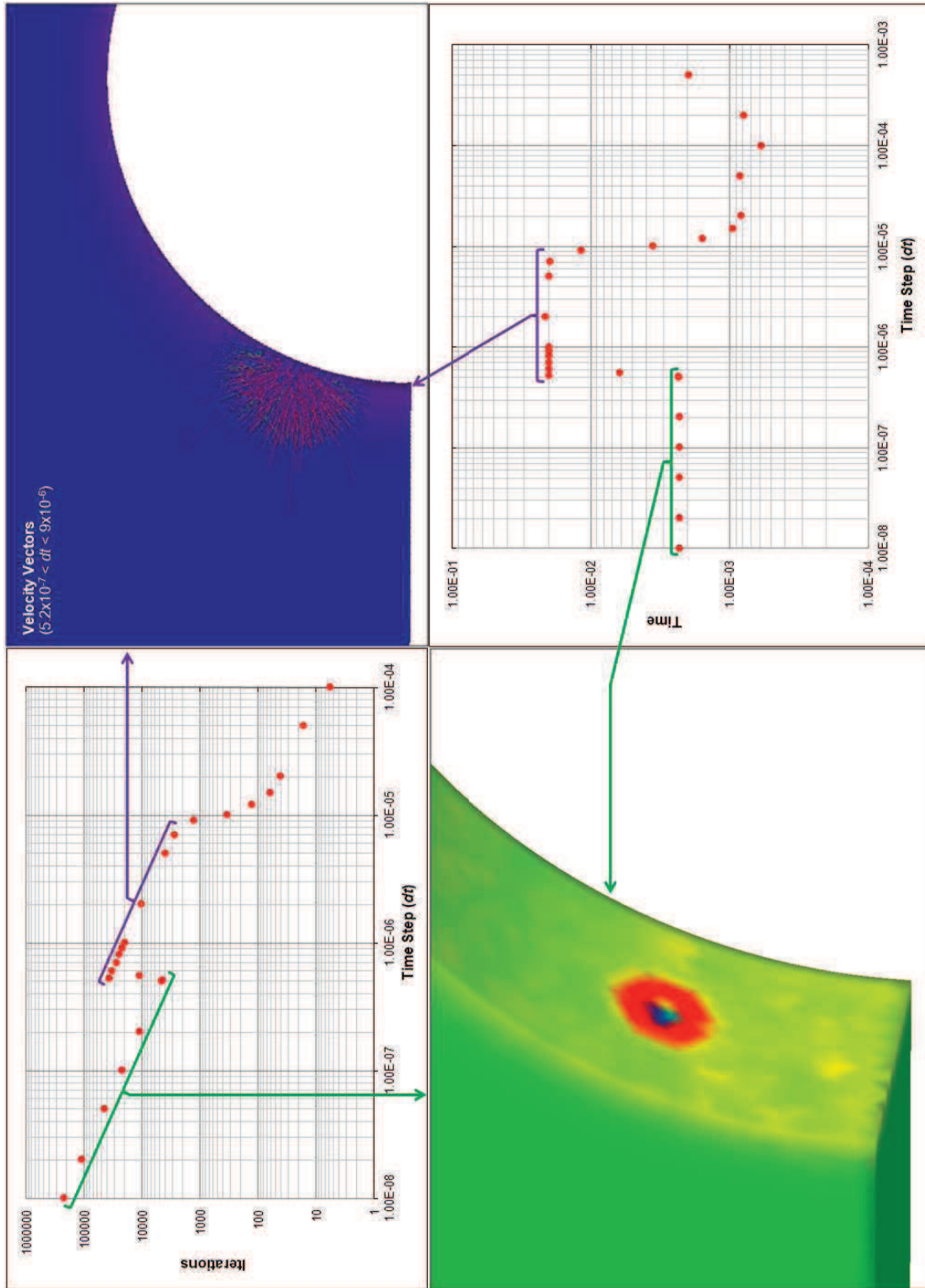


Figure 5.17: Time Step Study for Circular Cylinder ($Re = 26.0$).

Explanation. Explicit derivatives create a numerical error that feeds back into the properties.

The problem is mainly concentrated in the velocity components but spreads throughout the properties. Reorienting the solution so that all three velocity components are non-zero does not affect stability. The equations below show the inviscid residual equation for CFDsol. Property gradients are present six times in those equations (emphasized by red and green boxes). The red box is updated every iteration of a given step forward. The effects of this term *might* be improved by inverting M_+ for each solution step. The green boxes are still present when inverting the matrix.

$$\overrightarrow{RSD}_{CFDsol} = \mathbf{M}_+ \mathbf{U}_{n+1} - \mathbf{M}_- \mathbf{U}_n + \vec{F}$$

$$\mathbf{M}_+ \mathbf{U}_{n+1} = \sum_e \left[\left(1 + \frac{\Delta t_e}{2} \frac{\partial u_i}{\partial x_i} \right) [\mathbf{M}_C]_e^{(4)} + \frac{\Delta t_e}{2} \frac{V_e}{4} \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix} \bar{u}_i \frac{\partial \Phi_e}{\partial x_i} \mathbf{U}_{n+1} \right]$$

$$\mathbf{M}_- \mathbf{U}_n = \sum_e \left[\left(1 - \frac{\Delta t_e}{2} \frac{\partial u_i}{\partial x_i} \right) [\mathbf{M}_C]_e^{(4)} - \frac{\Delta t_e}{2} \frac{V_e}{4} \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix} \bar{u}_i \frac{\partial \Phi_e}{\partial x_i} \mathbf{U}_n \right]$$

$$\vec{F} = \sum_e \left[\Delta t_e \frac{V_e}{4} \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix} \left(\frac{\partial p}{\partial x_i} \bar{e}_i^k + \frac{\partial u_i}{\partial x_i} [\mathbf{M}_C]_e^{(4)} p_e \xi \right) \right]$$

Cowan (2003) built inherent stability into Euler3D by shifting the derivative from the flux to the shape function using Gauss' theorem. The residual equation for Euler3D (first-order in time) is shown below. The Euler3D residual is broken into three parts: The unsteady and

boundary integral \mathbf{B} terms are averaged over their respective elements using the mass matrix. The element flux term \mathbf{A} is averaged using one-point Gauss quadrature. The effects of numerical errors are minimized by averaging the properties applied in the residual.

$$\overline{RSD}_{Euler3D} = \Delta t_n \left(\sum_{be} \mathbf{M}_{c,e} \frac{\mathbf{U}_{n+1} - \mathbf{U}_n}{\Delta t} - \mathbf{A}(\mathbf{U}_{n+1}) + \mathbf{B}(\mathbf{U}_{n+1}) \right)$$

$$\mathbf{A}(\mathbf{U}_{n+1}) = \sum_e V_e \frac{\partial \Phi_e^T}{\partial x_i} (\bar{u}_i \bar{\mathbf{U}}_{n+1} + \bar{p}_{n+1} \bar{\mathbf{e}}_i)$$

$$\mathbf{B}(\mathbf{U}_{n+1}) = \sum_{be} \mathbf{M}_{c,e} \left(V_n \mathbf{U}_{n+1} + p_{n+1} \begin{Bmatrix} \mathbf{0} \\ \hat{n}_j \\ V_n \end{Bmatrix} \right)$$

5.3 Propulsion Models

Three modes of propulsion have been implemented in the OSU in-house codes: Rocket, turbojet, and scramjet (quasi-combustion) engines. The first two modes are applied through boundary integrals; the quasi-combustion terms are modeled through source terms. The propulsion models add two files, six new routines, and nine new arrays. Only the rocket and quasi-combustion terms were added to CFDsol because of time constraints on the contract.

5.3.1 Quasi-Combustion Terms

The quasi-combustion conditions are read from the case.cmb file and constructed in *read_combustion* using Eqs. 4.222 and 4.225. The type of distribution used to construct the quasi-combustion terms is specified through *icomb*, which is found in the case.con file. If *icomb*=0, the combustion terms are not used. If *icomb*=1, the data in case.cmb is treated as nodal and constructed from a piece-wise linear source distribution. If *icomb*=2, the data

in case.cmb is treated as elemental and constructed from a piece-wise constant source distribution. The size of the combustion arrays *ncmb* depends on the type of model used. For *icomb*=1, *ncmb* is the number of nodes. For *icomb*=2, the data is read into the arrays as elemental data and converted to nodal contributions to the residuals, which is stored back in the combustion arrays. These contributions are added to the residual in *solve2d_1pt* and *solve3d_1pt*. The combustion arrays *MDOT* and *MDOTE* are defined:

- *MDOT* (:) :: Contributions to residual from mass generation term
- *MDOTE* (:) :: Contributions to residual from heat generation term

5.3.2 Rocket and Engine BCs

Rocket BCs. The rocket boundary conditions were implemented in *rocket_bc_1pt* using Eqs. 4.129 through 4.134. The rocket outflow routine calls *riem_inv* for the Riemann correction to the boundary fluxes. The properties and controls for all rockets are read from case.eng using *read_engine*. The initial total properties across the rocket planes are calculated in *init_engine_press* and then recalled later during ramp up.

The number of iterations to ramp-up the properties at the outflow plane is specified through *nrstp*, which is read from case.eng. The file also contains the number of rocket boundaries *nrbc*. The number of boundary elements *npbc* used to model the rocket and engine boundary types is used to size the propulsion arrays. The number of boundary elements is calculated from the upper limit on the wall, symmetry, and far field boundary elements:

$$npbc = nbe - MAX(LBE(2), LBE(4), LBE(6)) \quad (5.10)$$

The number of boundary elements on each rocket boundary is tracked in *NRKT*. *IRKT* contains the pointers for each rocket boundary element. *ARKT* contains the four properties that define each rocket boundary plane:

- *ARKT*(: ,1) :: User-specified total pressure at rocket boundary
- *ARKT*(: ,2) :: User-specified total enthalpy at rocket boundary
- *ARKT*(: ,3) :: Initial total pressure at rocket boundary, used for ramp up
- *ARKT*(: ,4) :: Initial total enthalpy at rocket boundary, used for ramp up

Coupled Turbojet Engine BCs. The turbojet engine inflow and outflow conditions were constructed and coupled in *engine_bc_1pt* using Eqs. 4.135 through 4.138. The flux to properties conversion (Eqs. 4.142 through 4.145) was implemented in *flux_props* and called by the engine outflow boundary. The engine outflow routine calls *riem_inv* for the Riemann correction to the boundary fluxes. The properties and controls for all turbojet engines are read from case.eng using *read_engine*. In the case of a scratch start, the static pressure along the engine inflow planes is set equal to the freestream total pressure. For a restart, the most current pressure distribution is used. The initial properties across the engine planes are calculated in *init_engine_press* and then recalled later during ramp up.

The number of iterations to ramp-up the properties at the outflow plane is specified through *nrstp*. The gain *k* in the mass flow rate controller is specified through *gain*. These controls are read from case.eng. The file also contains the number of coupled engine boundaries *nebc*.

The coupled engine boundaries are similarly defined by the number of boundary elements on the inflow plane *NENGI* and total boundary elements used by the engine. For convenience, the number of elements on the outflow plane is calculated as $NENGO - NENGI$, so that $NENGI + 1$ and *NENGO* become the lower and upper limits to their loops. *IENG* contains

the pointers for each engine boundary element. *AENG* contains the four properties that define each rocket boundary plane:

- *AENG*(: ,1) :: User-defined mass flow rate at inflow plane
- *AENG*(: ,2) :: Static pressure at inflow plane, from previous iteration
- *AENG*(: ,3) :: User-defined mass flow rate of fuel
- *AENG*(: ,4) :: User-defined uninstalled thrust
- *AENG*(: ,5) :: User-defined enthalpy flow rate
- *AENG*(: ,6) :: Area of inflow plane
- *AENG*(: ,7) :: Area of outflow plane
- *AENG*(: ,8:) :: Area-weighted normal at outflow plane

5.4 Turbulence Modeling

Two turbulence models were implemented in NS2D and NS3D: SA and SST. CFDsol already contained a form of the SA model, although badly realized; this model was adapted to be similar to the implementation in NS3D. The SA model was added into the OSU codes in six routines; the SST model was added through five routines:

- *spalart2d_1pt* or *spalart3d_1pt* :: SA differential and closure equations
- *set_bound_SA* :: Applies wall boundary condition to SA variable
- *dt_calc_visc_SA* :: Viscous time stepping with SA stiffness matrix
- *smooth1_SA_turb* :: Lower order dissipation model for SA variable
- *smooth2_SA_turb* :: Higher order dissipation model for SA variable
- *calc_rnu* :: Conversion between $\rho\nu$ and μ_T

- *sst2d_1pt* or *sst3d_1pt* :: SA differential and closure equations
- *set_bound_SST* :: Applies wall boundary condition to SST variables
- *dt_calc_visc_SST* :: Viscous time stepping with SA stiffness matrix
- *smooth1_2var_turb* :: Lower order dissipation model for SA variable
- *smooth2_2var_turb* :: Higher order dissipation model for SA variable

The subroutines *set_bound_SA* and *set_bound_SST* are included with their base models *spalart2d_1pt* and *sst2d_1pt* (or *spalart3d_1pt* and *sst3d_1pt*). These routines are applied

between iterations in the base models and to the initial conditions in *set_init_2d* and *set_init_3d* and when restarting in *convert_unt*.

Turbulent contributions were added to the momentum and energy equations in *flux_src_1pt*. Reynolds stresses ρT_{ij} (Eq. 3.221) and turbulent transport of heat Q_j (Eq. 3.222) were constructed using the strain tensor to mirror their molecular counterparts, viscous stress τ_{ij} and conduction q_j'' . The Reynolds stresses were dotted with velocity to create the turbulent dissipation terms $\rho T_{ij} u_j$ (Eq. 4.31), which are constructed to mirror viscous dissipation $\tau_{ij} u_j$. When turbulent kinetic energy K is present from the SST model, the gradient of K is calculated and applied to the energy equation (Eq. 4.31). The Reynolds terms were calculated on each element and applied to its nodes (Eqs. 4.43 and 4.44) and any boundary elements attached to the element (Eqs. 4.49 and 4.50) just like their molecular counterparts. No Reynolds stresses and turbulent transport of heat are applied to boundary integrals along walls because the eddy viscosity is identically zero along solid walls. Symmetry planes are treated in the same manner as molecular stresses, where gradients are calculated in terms of tangential and binormal velocities and gradients. The turbulent transport of heat and normal gradient of K are assumed to be zero along symmetry planes.

5.4.1 Turbulent properties

Six new arrays were created to hold the turbulent properties, residuals, and dissipation. The nodal strain is calculated in the SST model and used to calculate C_α (Eq. 3.237):

- UNT1 (: ,1) :: Most current eddy viscosity μ_T
- UNT1 (: ,2) :: Most current turbulent kinetic energy ρK
- UNT1 (: ,3) :: Most current dissipation turbulent energy $\rho \epsilon$
- UNT1 (: ,4) :: Most current value of model variable: $\rho \nu$ or $\rho \omega$

- UNT (: ,1:4) :: Turbulent unknowns from previous step
- UNTO (: ,1:4) :: Turbulent unknowns from two previous steps
- RHST (: ,1:2) :: Residual or RHS vector for turbulent differential equations
- RDIST (: ,1:2) :: Artificial dissipation vectors for turbulent differential equations
- STRN (:) :: Nodal strain (calculated with hybrid element or area-weighting)

For the SA model, only the first and last values are stored in UNT1. For the SST model, all four properties are stored at once. These properties are written to the case.un# file for tracking in *write_unk*. The properties are also written to caset.un# for plotting (*write_unt*). The turbulent properties are plotted in Gplot2D and Gplot3D through different property names:

- Eddy viscosity μ_T is stored and plotted as density
- Turbulent kinetic energy ρK is stored in *u* and plotted as velocity magnitude
- Dissipation turbulent energy $\rho \epsilon$ is stored and plotted as total energy
- The model variable ($\rho \nu$ or $\rho \omega$) is stored in pressure and plotted as C_p

The caset.un# file cannot be used to restart the solution. Instead, the case.un# file contains all flow properties, including the turbulent properties. The case.un# file can be renamed case.unk and used to restart a solution. The turbulent properties are also written to the safe mode restart files (case.rst and case.rs2) for smooth restart. When the solver is restarted using initial conditions, those conditions are tested for model consistency. If the SST model (*iturb* = 2) was used to produce the case.unk, then the turbulent properties are converted to the new model format, including laminar (*iturb* = 0). If the SA model (*iturb* = 1) is used to produce the case.unk, the model can be converted to a laminar solution, but the SST model has more unknowns, which cannot be pieced together from the SA variables. When converting from SA to SST solutions, the turbulent properties are reset to laminar to start the SST solution. All turbulent property conversions occur in *convert_unt* (found along with *read_unk*). Scratch starts are calculated using the freestream conditions in the case.con.

Turbulent residuals are collected into RMS values and written to case.rsd and case.rsd2.

The turbulent freestream conditions are defined in the case.con as:

- *PrT* :: Turbulent Prandtl number, used for turbulent transport of heat
- *muTinf* :: Freestream eddy viscosity μ_T
- *turbI* :: Turbulent intensity in freestream I
- *rnuinf* :: Freestream $\rho\nu$ for SA model, calculated using *calc_rnu* if zero
- *rhoKinf* :: Freestream ρK for SST model, calculated from Eq. 3.248 if zero
- *rhoWinf* :: Freestream $\rho\omega$ for SST model, calculated from Eq. 3.250 if zero

muTinf and *turbI* are included for user convenience. *rnuinf*, *rhoKinf*, and *rhoWinf* are used throughout the code. If any of *rnuinf*, *rhoKinf*, and *rhoWinf* is zero in the controls file, then that value is calculated from *muTinf* and/or *turbI*. The new values have been added to the freestream properties array *cinf*. The last value of *cinf* represents the model variable, either *rnuinf* or *rhoWinf*. The next to the last value of *cinf* is assigned the *rhoKinf* if an SST model is present. For laminar solutions, the last two values of *cinf* are zero.

5.4.2 Energy Balances

The addition of turbulent kinetic energy K from the SST model must be tracked in the total energy and enthalpy (Eqs. 3.120 and 3.121). The total energy or enthalpy are used to calculate pressure (Eqs. 3.130 and 3.131), ratio of total to static temperature (Eq. 3.133), and other areas. Any instant addressing total energy or enthalpy must also correct that calculation with K when present in the solution. The energy balance was adjusted in the following routines:

- *flux_props* :: Corrected total enthalpy term in pressure (Eq. 3.131)
- *flux_src_lpt* :: Corrected static pressure with turb KE (Eq. 3.130)
- *init_engine_press* :: Corrected ratio of total to static temperature (Eq. 3.133)
- *pc_update* :: Corrected Δp with *RHST(1)* and total enthalpy (Eq. 3.131)
- *read_unk* :: Corrected total enthalpy when recalculated (Eq. 3.131)

- *riem_inv* :: Corrected acoustic speed and *allx* (Δp)
- *rocket_bc_1pt* :: Corrected ratio of total to static temperature (Eq. 3.133)
- *set_init2d & set_init3d* :: Corrected ratio of total to static temperature (Eq. 3.133)

5.4.3 SA Model

The Spalart-Allmarus SA model was presented in compressible form in Chapter 3 (Deck, 2002) along with its rotation-curvature correction (Shur, 2000) and Oliver's (2008) adaptation to \hat{S} . (Note that the DS_{ij}/Dt term was not included in the rotation-curvature correction because the unsteady term is very costly to track and the advection term requires second derivatives.) These methods were assembled into a single model and discretized according to Chapter 4. A series of splines are used to construct an approximate interpolation of $\rho\hat{v}$ as a function of eddy viscosity μ_T in the subroutine *calc_rnu*. So that the freestream conditions can be calculated from an eddy viscosity.

The majority of the SA model is housed in *spalart2d_1pt* and *spalart3d_1pt*, which is organized like *solve2d_1pt* and *solve3d_1pt* without the use of additional subroutines. Domain integral contributions are created and boundary integrals are calculated per their attachment to the domain elements. The far field boundary condition restricts the flow to $\rho\hat{v}_\infty$ when the characteristic points into the domain. A predictor-corrector was setup similar to the RANS equations, but $\rho\hat{v}$ was limited to $\rho\hat{v}_\infty$ as a lower limit, except along solid walls. The solid wall boundary condition ($\rho\hat{v} = 0$) is applied within *set_bound_SA*. The components were assembled in order unsteady plus advection, diffusion, artificial dissipation, and source terms.

Artificial dissipation was created using the RANS dissipation models as a basis. The new models were called *smooth1_SA_turb* and *smooth2_SA_turb* to show their heritage. These models can be used to add artificial dissipation to any further one-equation turbulence models. The SA artificial dissipation is scaled by $disst*diss$, where both values are read from the case.con file. The value of *disst* can be used to scale up or back the artificial dissipation in the turbulence model without adversely affecting the RANS.

Previous Implementation in CFDsol. CFDsol initially contained a model resembling an incompressible Spalart-Allmaras model with a “trip transition” model. Four of the terms in the model were either implemented incorrectly or did not correspond to any terms in the actual SA model. The equations were reformulated in compressible form, and the trip transition model was eliminated to simplify the model.

Initial Conditions. An initial non-zero value of $\rho\hat{v}$ allows the source terms to produce turbulence in the near-wall region. Higher values will create turbulence faster, so the magnitude of the initial condition is important. Dissipation will attempt to minimize the turbulent viscosity in the solution, so a minimum value must be maintained, here equal to the initial condition. Four initial conditions were tested on a flat plate boundary layer: $\rho\hat{v}_\infty = 10^{-1}$, 10^{-2} , 10^{-4} , and 10^{-6} . For CFDsol, the first value was found to be too large, creating excessive turbulence in the laminar region; and, the latter two values were too small, taking too long to generate turbulence throughout the domain. The initial condition is suggested: $\rho\hat{v}_\infty = 10^{-2}$. Cebecci (2005) suggests using $\rho\hat{v}_\infty = 5$ to create a fully turbulent boundary layer when the transition location is unimportant.

Component Testing. The governing equations were implemented and tested in components (unsteady, advection, diffusion, source, and artificial dissipation) so that each component could be isolated. A discontinuity in a straight tube was used to test the unsteady and advection terms. The domain was initially seeded with a high value (0.2), while the inflow boundary was set to a lower value (0.1). The side boundaries maintained these values as the wave front moved down the tube at the bulk velocity of the fluid. Figure 5.18 shows six snapshots of the ρv -wave advecting through the straight tube. The wave in the straight tube showed the advection terms to be accurate and stable.



Figure 5.18: Six Snapshots of Turbulent Advection in a Straight Domain.

NS2D Turbulent Bubble. A second test was completed to be a bit more complicated. A bubble of turbulence ($\rho \hat{v} = 0.2$) was generated on the upstream side of a large freestream domain ($\rho \hat{v}_\infty = 0.1$). The bubble was allowed to naturally advect downstream. Figure 5.24 shows the initial and advected bubble. As the bubble advects downstream, striations appear within the bubble and upstream of the bubble. These striations are not natural but numerical. The initial discontinuity in the solution creates a gradient in the properties. For a non-zero velocity, the property gradient creates a non-zero residual that changes the shape of the

profile. The bubble advects downstream according to the bulk velocity of the flow, leaving the effects of the gradient as a shadow. The shadow advects with the bubble, creating additional shadows. Figure 5.25 shows centerline slices as the bubble advects downstream.



Figure 5.19: Turbulent Bubble Initial Conditions (left) and Advected Downstream (right).

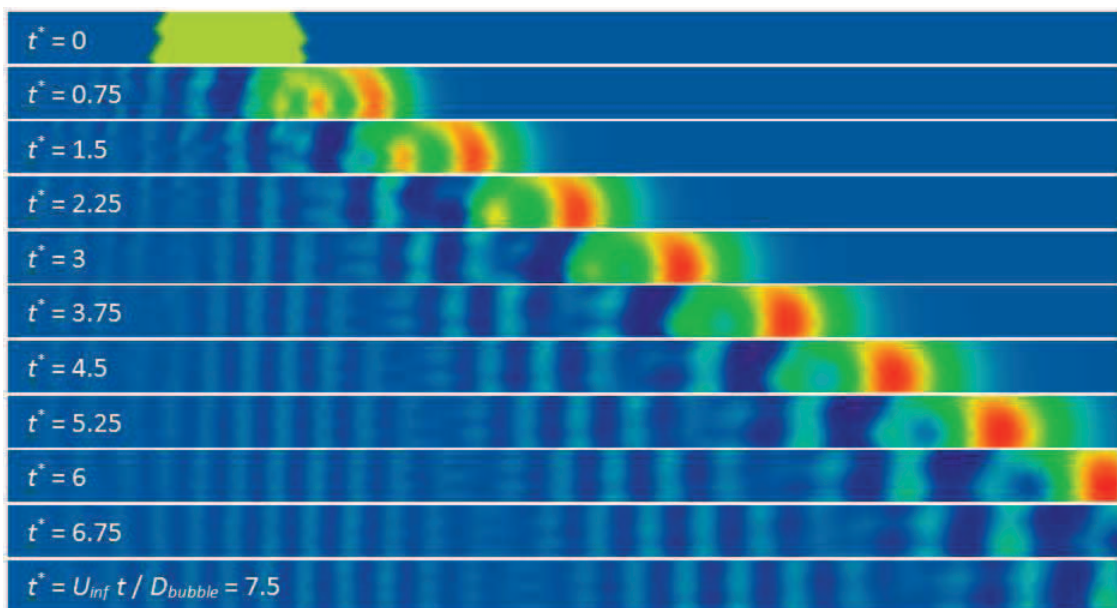


Figure 5.20: Slices of Turbulent Bubble being Advected Downstream.

The model shows added stability when the viscous diffusion terms are added to the model, but only at lower Reynolds numbers. Figure 5.21 shows the effects of Reynolds number on the bubble. Larger Reynolds numbers ($Re > 10^4$, $refdim = 1$) show little difference from the advection only solution. Smaller Reynolds numbers ($Re < 10^3$) increase numerical stability and diffuse the bubble. The source terms were added to the model with no change. The source terms showed to be stable in the absence of strains or walls.

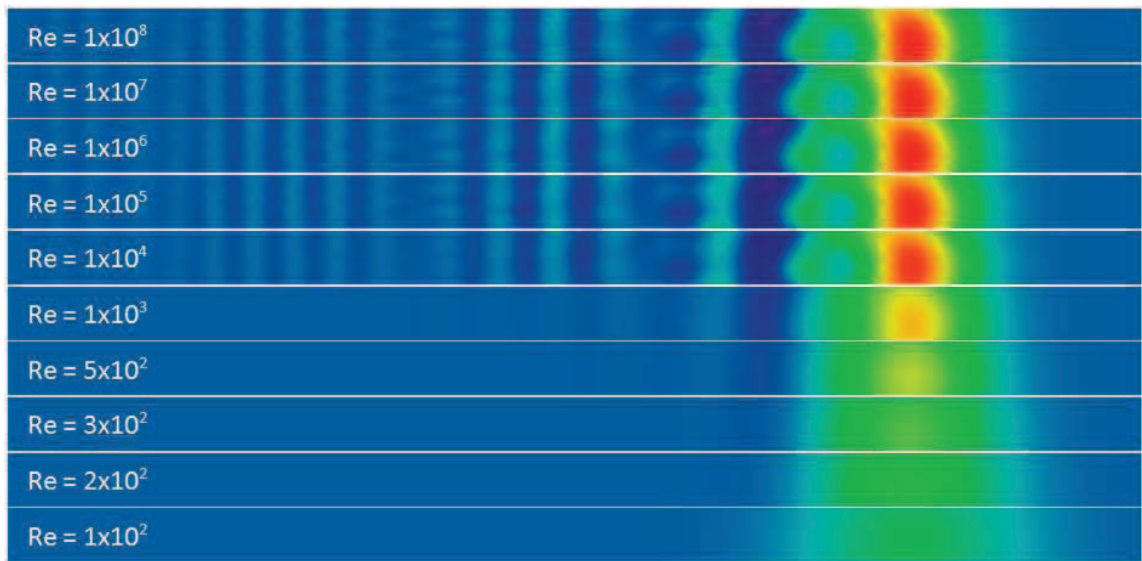


Figure 5.21: Slices of Turbulent Bubble with Different Levels of Viscous Diffusion.

NS2D & NS3D Artificial Dissipation. Cowan's artificial dissipation routines were used as a pattern to add dissipation to the SA model:

$$\rho \hat{v}_{n+1} = \rho \hat{v}_n - \Delta t \mathbf{M}_L^{-1}(\mathbf{RHS}_T + \mathbf{D}(\rho \hat{v}_n)) \quad (5.11)$$

$$\mathbf{D}(\rho \hat{v}) = \sum_{nsg} \frac{1}{3} dis_{s_T} \left(f_1 \left| \vec{\nabla}_t \right| a \Delta \rho \hat{v} \right) \quad (5.12)$$

where $disst_T$ is the dissipation scaling factor (= $disst * diss$ in case.con), $\vec{\nabla}_l$ is the gradient along the segment, a is the local speed of sound, and $\Delta\rho v$ is the difference in the SA variable along the segment. The function f_l is calculated based on the local Mach number:

$$f_l = \begin{cases} \frac{1}{2}(1 + M^2) & \text{if } |M| < 1 \\ |M| & \text{if } |M| \geq 1 \end{cases} \quad (5.13)$$

The lower order model is suggested for supersonic solutions. The higher order model is calculated using the same equations, where the small differences $\Delta\rho v$ are neglected. The differences are removed so that the model can be used in rotating and subsonic domains.

The dissipation scalar was tested to find an appropriate level of artificial dissipation. Figure 5.22 shows a bubble of turbulence at various levels of viscous and artificial dissipation. At $Re = 300$, the bubble shows little diffusion near the edges and no numerical oscillations. At $Re = 10^5$, diffusion has little effect, and numerical oscillations persist in the flow. The solution is also shown at the higher Reynolds number with various levels of artificial dissipation. The scalar $diss$ used in the Navier-Stokes equations is also appropriate for the SA model and shows an equivalent diffusion to $Re = 300$. (The SA model is stable for $disst_T$ up to 10.)

Figure 5.23 shows the performance of the complete model for some of the Reynolds numbers shown in Figure 5.21.

The model was extended one dimension and incorporated into NS3D. Neither component or artificial dissipation testing was performed in NS3D.

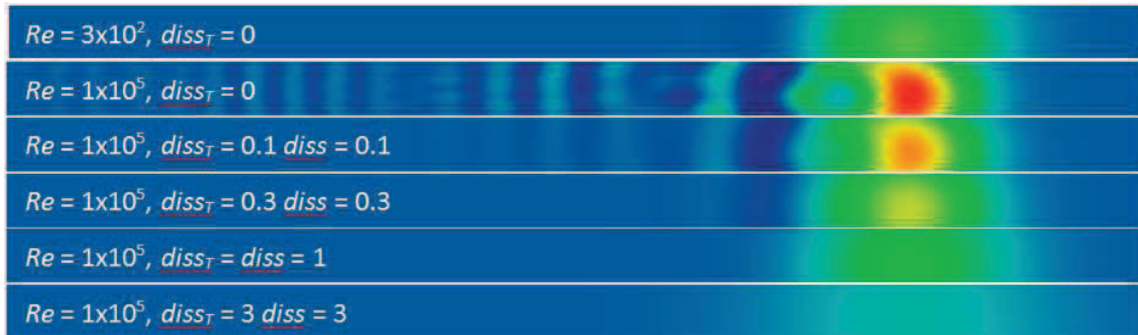


Figure 5.22: Advected Bubble at Various Levels of Molecular & Artificial Dissipation.

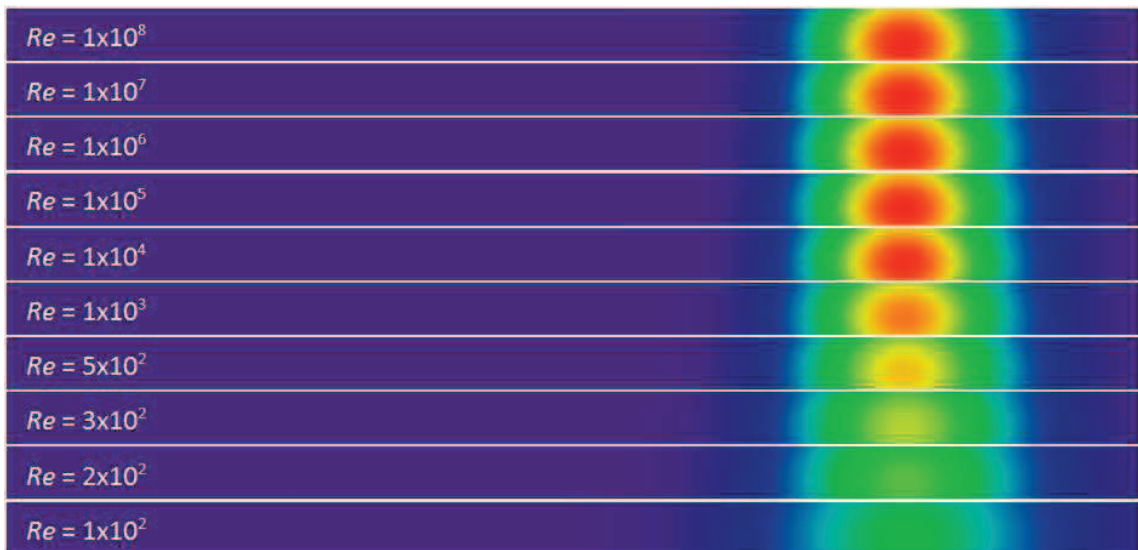


Figure 5.23: Slices of Turbulent Bubble Solution from NS2D-SA (Various Re).

CFDsol Turbulent Bubble. The turbulent bubble test was repeated in CFDsol to test its components and tune artificial dissipation. Figure 5.24 shows the initial and advected bubble. The advected bubble looks different because of the different numerics in CFDsol.

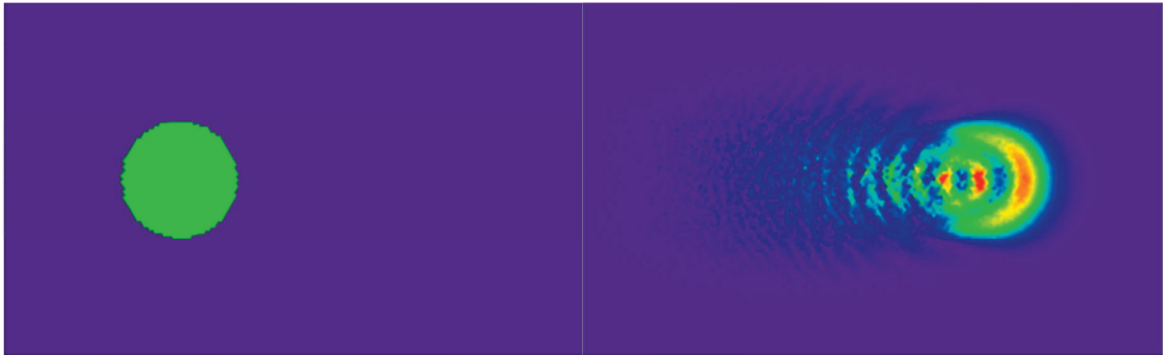


Figure 5.24: Turbulent Bubble Initial Conditions (left) and Advected Downstream (right).

Figure 5.25 shows centerline slices of the domain as the bubble advects downstream. The model shows added stability when the viscous diffusion terms are added, but only at lower Reynolds numbers. Figure 5.26 shows the effects of Reynolds number on the bubble.

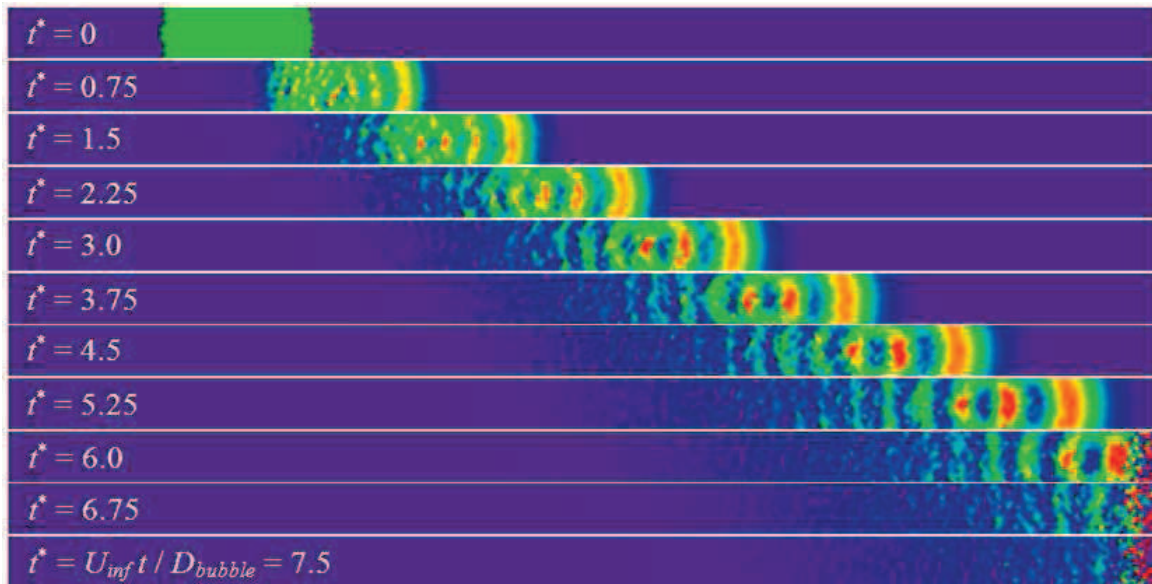


Figure 5.25: Slices of Turbulent Bubble being Advected Downstream.

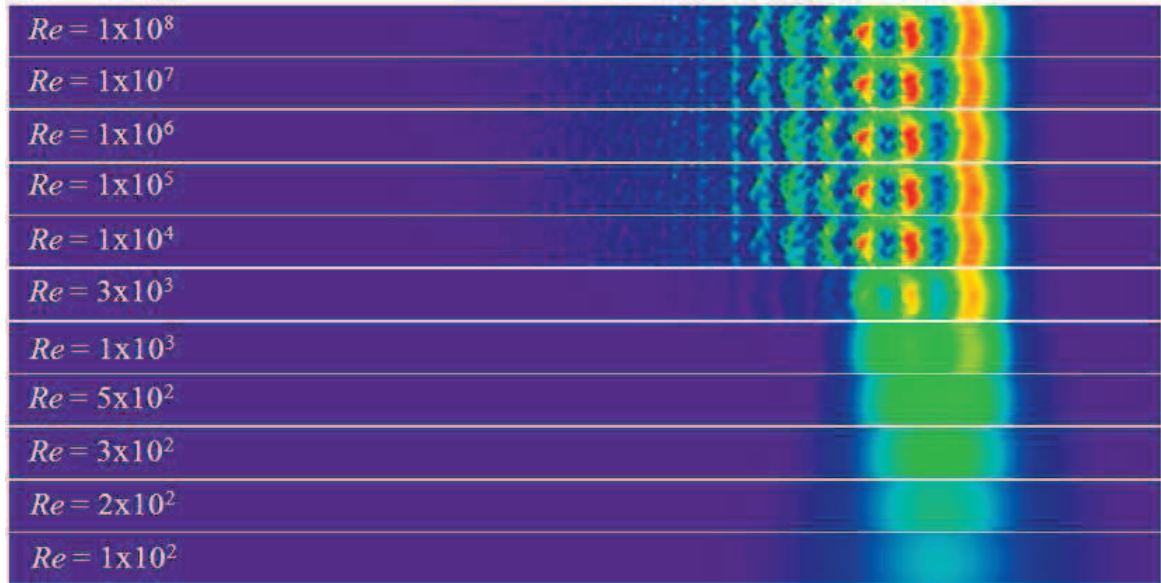


Figure 5.26: Slices of Turbulent Bubble with Different Levels of Viscous Diffusion.

CFDsol Artificial Dissipation. The SA model showed a need for artificial dissipation, especially at high Reynolds numbers. The artificial dissipation used by the other governing equations in CFDsol was adapted to the SA variable and implemented in the model:

$$\rho \hat{v}_{k+1} = \rho \hat{v}_k + \Delta t \mathbf{M}_L^{-1} (\mathbf{K}_v \rho \hat{v}_k + \mathbf{K}_{\sigma,v} + \mathbf{f}_{\sigma,v} + \mathbf{S}_v + \hat{\mathbf{R}}_v) \quad (5.14)$$

$$\hat{\mathbf{R}}_v = \frac{C_s \tau}{\Delta t} \mathbf{M}_L^{-1} (\mathbf{M}_c - \mathbf{M}_L) \rho \hat{v}_k \quad (5.15)$$

The pressure switch S_e was removed from the SA dissipation model so that dissipation is applied to the SA model throughout the domain. The model was tested with only the advection and artificial dissipation terms in effect. The dissipation scalar was tested to find an appropriate level of artificial dissipation. Figure 5.27 shows a bubble of turbulence at various levels of viscous and artificial dissipation. The scalar *FACTOR* used in the RANS equations is also appropriate for the SA model and shows an equivalent diffusion to $Re = 10^3$.

(The SA model is unstable for C_s larger than 1.2 *FACTOR*.) Figure 5.28 shows the performance of the complete model for the Reynolds numbers shown in Figure 5.26.

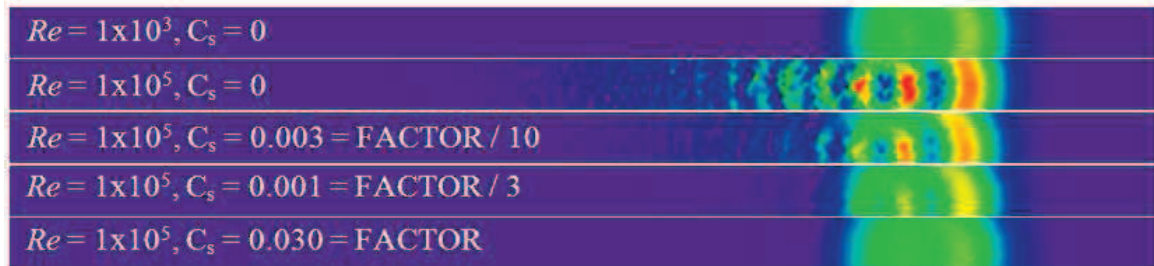


Figure 5.27: Advected Bubble at Various Levels of Molecular & Artificial Dissipation.

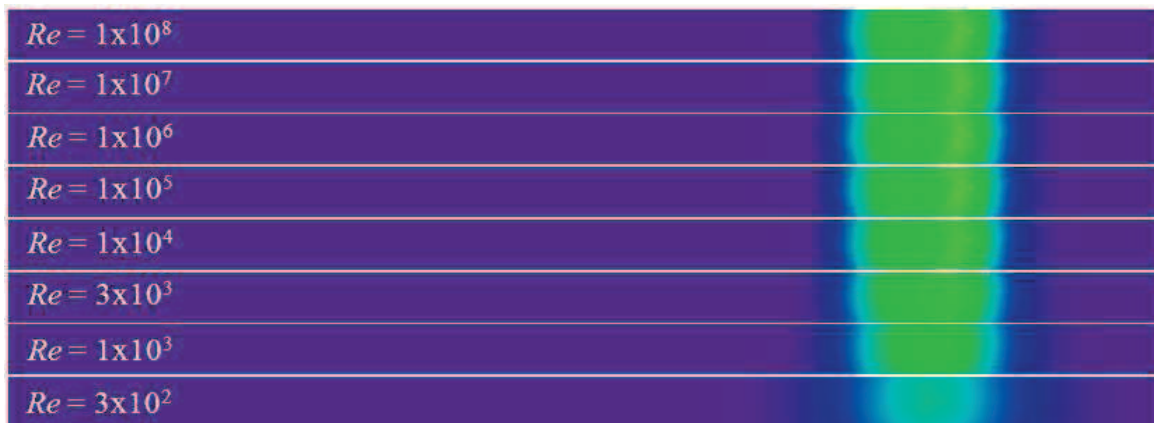


Figure 5.28: Slices of Turbulent Bubble from Complete SA Model (Various Re).

5.4.4 SST Model

Menter's SST model was presented in compressible form in Chapter 3 (Rumsey, 2009) along with its rotation-curvature correction (Shur, 2000) and Spalart & Rumsey's (2007) method of maintaining the freestream values in the external flow. These methods were assembled into a single model and discretized according to Chapter 4. The rotation-curvature correction was copied directly from the SA model.

The majority of the SST model is housed in *sst2d_1pt* and *sst3d_1pt*, which is organized like *solve2d_1pt* and *solve3d_1pt* without the use of additional subroutines. Domain integral contributions are created and boundary integrals are calculated per their attachment to the domain elements. The far field boundary condition restricts the flow to ρK_∞ and $\rho \omega_\infty$ when the characteristic points into the domain. A predictor-corrector was setup similar to the RANS equations, but ρK and $\rho \omega$ were limited to ρK_∞ and $\rho \omega_\infty$ as a lower limit, except along solid walls. $\rho \epsilon$ is calculated and stored in *UNT* when the other properties are updated (Eq. 3.244). The solid wall boundary condition ($\rho K = 0$; Eq. 3.253) is applied within *set_bound_SST*. The components were assembled in order unsteady plus advection, diffusion, artificial dissipation, and source terms.

Artificial dissipation was created using the RANS dissipation models as a basis. The new models were called *smooth1_2var_turb* and *smooth2_2var_turb* to show their heritage. These models can be used to add artificial dissipation to any further two-equation turbulence models. The two-variable dissipation models point to two locations in *UNT*. One location is assumed to contain K and the other is specified by a pointer *ivar*. The SST artificial dissipation is scaled by $disst * diss$ like the SA model; both *disst* and *diss* are read from the case.con file. The value of *disst* can be used to scale up or back the artificial dissipation in the turbulence model without adversely affecting the RANS equations.

Nodal Strain. The calculation of eddy viscosity in the update subsection requires the magnitude of the strain tensor at all nodes in the field (Eq. 3.237). The nodal strain is calculated and stored in *STRN*. Such strain tensor is known in a piece-wise constant manner. The strain tensor is pushed to the nodes using a hybrid element technique:

$$\sum_e \int_{\Omega_e} \Phi_e^T (\Phi_e S_n - \bar{S}_e) d\Omega = 0 \quad (5.16)$$

$$\mathbf{M}_C S_n = \sum_e \int_{\Omega_e} \Phi_e^T \bar{S}_e d\Omega = \sum_e \int_{0 \leq \xi \leq 1} \Phi_e^T \bar{S}_e |J_e| d\xi = \sum_e \frac{\Omega_e}{d+1} \{1\} \bar{S}_e \quad (5.17)$$

$$S_n = \mathbf{M}_C^{-1} \sum_e \frac{\Omega_e}{d+1} \{1\} \bar{S}_e \approx \mathbf{M}_L^{-1} \sum_e \frac{\Omega_e}{d+1} \{1\} \bar{S}_e \quad (5.18)$$

where \bar{S}_e is the strain on the element, S_n is the strain at the nodes (piece-wise linear on elements). The lumped mass matrix is used as an approximate and efficient means of inverting the mass matrix. For NS2D and NS3D, the hybrid equations are written:

$$S_n \approx \mathbf{M}_L^{-1} \sum_e \frac{A_e}{3} \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix} \bar{S}_e \quad (5.19)$$

$$S_n \approx \mathbf{M}_L^{-1} \sum_e \frac{V_e}{4} \begin{Bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{Bmatrix} \bar{S}_e \quad (5.20)$$

Artificial Dissipation. Cowan's artificial dissipation routines were used as a pattern to add dissipation to the SST model:

$$\mathbf{U}_{T,n+1} = \mathbf{U}_{T,n} - \Delta t \mathbf{M}_L^{-1} (\mathbf{RHS}_T + \mathbf{D}(\mathbf{U}_{T,n})) \quad (5.21)$$

$$\mathbf{D}(\mathbf{U}_T) = \sum_{nsg} \frac{1}{3} diss_T (f_1 |\vec{\nabla}_t| c \Delta \mathbf{U}_T) \quad (5.22)$$

Experiments were run on components and artificial dissipation as with the SA model. These tests showed similar results to the NS2D-SA results shown above. The same limits and effectiveness can be assumed on $diss_T$.

5.5 Memory

Before this work began, Euler2D required the following integer and real allocations:

$$INT_{E2D-03} = 3nel + 2nsg + 3nbe + nr(1 + 12nr)$$

$$REAL_{E2D-03} = 37nnd + 5nel + 2nsg + 3nbe + 4nwl + nr(6 + 22nr + 2nwl)$$

After adding propulsion models and acoustic outputs, Euler2D requires:

$$INT_{Euler2D} = INT_{E2D-03} + nnd + nrbc(1 + npbc) + nebc(2 + npbc) + 20$$

$$REAL_{Euler2D} = REAL_{E2D-03} + 4nnd + 4nrbc + 9nebc$$

After viscous terms and turbulence models to Euler2D, NS2D requires:

$$INT_{NS2D} = INT_{Euler2D} + 2nel + nbe$$

$$REAL_{NS2D} = REAL_{Euler2D} + 18nnd + nel + 2nwl + nbeq$$

Before this work began, Euler3D required the following integer and real allocations:

$$INT_{E3D-03} = 4nel + 2nsg + 4nbe + nr$$

$$REAL_{E3D-03} = 51nnd + 10nel + 3nsg + 4nbe + 6nwl + nr(8 + 34nr + 3nwl)$$

After adding propulsion models and acoustic outputs, Euler3D requires:

$$INT_{Euler3D} = INT_{E3D-03} + nnd + nbe + nrbc(1 + npbc) + nebc(2 + npbc) + 20$$

$$REAL_{Euler3D} = REAL_{E3D-03} + 6nnd + 4nrbc + 10nebc$$

After viscous terms and turbulence models to Euler3D, NS3D requires:

$$INT_{NS3D} = INT_{Euler3D} + 3nel$$

$$REAL_{NS3D} = REAL_{Euler3D} + 18nnd + nel + 2nwlt + nbeq$$

5.6 Run Times

Time comparisons were ran to break down the operational cost of each component of the OSU in-house codes. Figure 5.29 compares the run times for inertial solutions in Euler2D and NS2D. The first two columns illustrate how costly the artificial dissipation models are in Euler2D. These same models are used in NS2D. If the viscous dissipation is sufficient for stability ($Re < 500$), then artificial dissipation should be avoided for efficiency. The viscous terms were added to Euler2D to create NS2D; these terms add approximately 50% to the base level run time. Incorporating the SA and SST models adds roughly 60 to 100% more to the run time, without including the artificial dissipation and local time stepping required to stabilize the coupled models. Figure 5.29 also shows that the most complex viscous model (NS2D – SST – High Diss) requires twice the resources compared to the baseline inviscid model (Euler2D – High Diss). Figure 5.30 shows similar comparisons for Euler3D and NS3D.

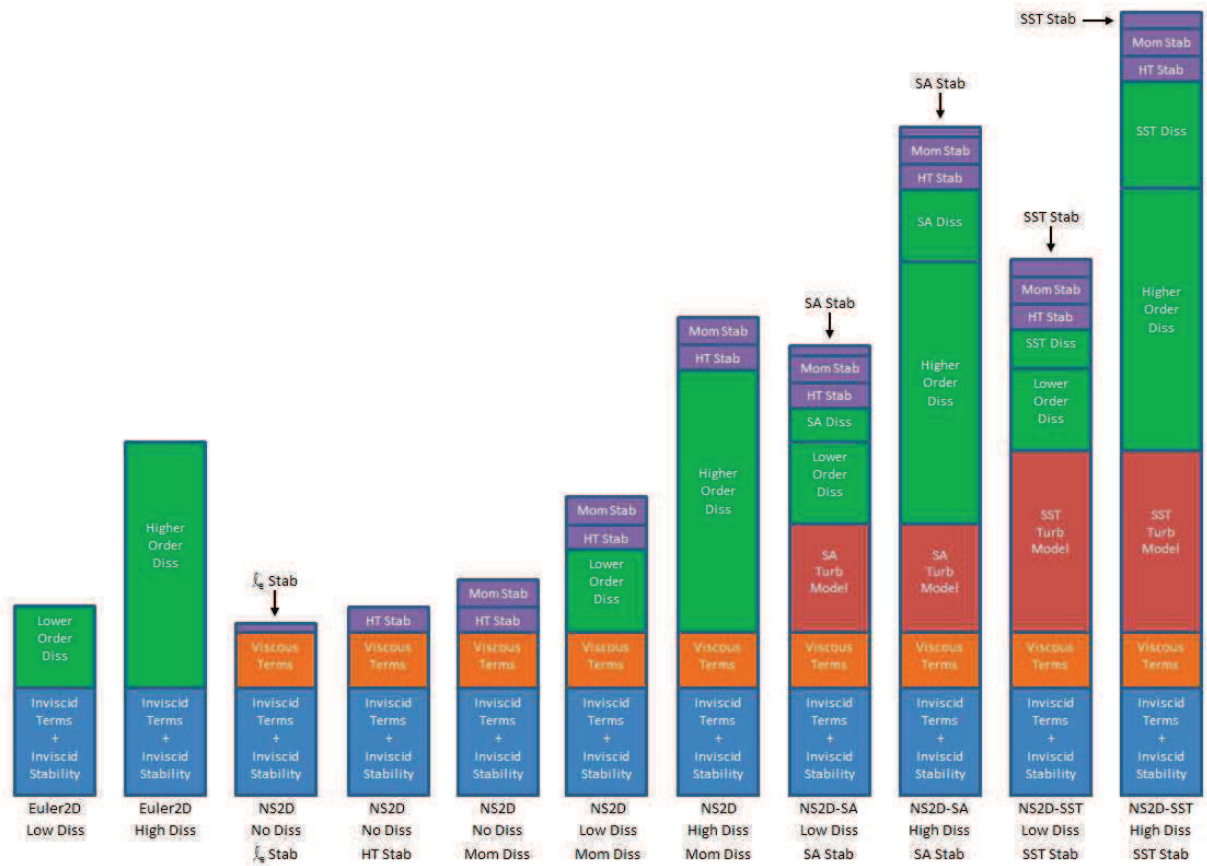


Figure 5.29: Comparison of Run Times for Euler2D and NS2D.

Figure 5.31 and Figure 5.32 shows comparisons between non-inertial solutions from the 2D and 3D codes, respectively. Inertial run times have been presented alongside each non-inertial solution. The non-inertial source terms require a small fraction of the overall run time for any of the given model combinations in 2D or 3D. The non-inertial source terms are more expensive in 3D because there are more vectors, cross products, and components to evaluate.

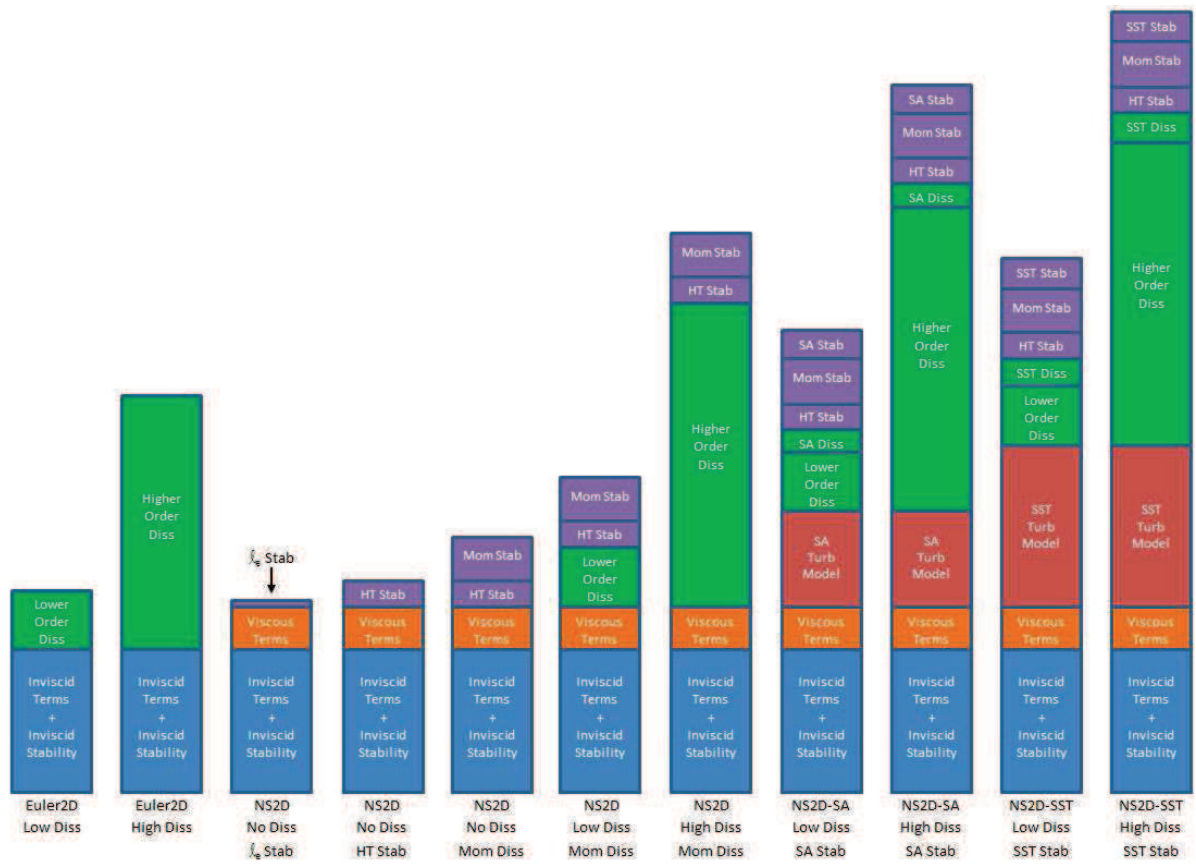


Figure 5.30: Comparison of Run Times for Euler3D and NS3D.

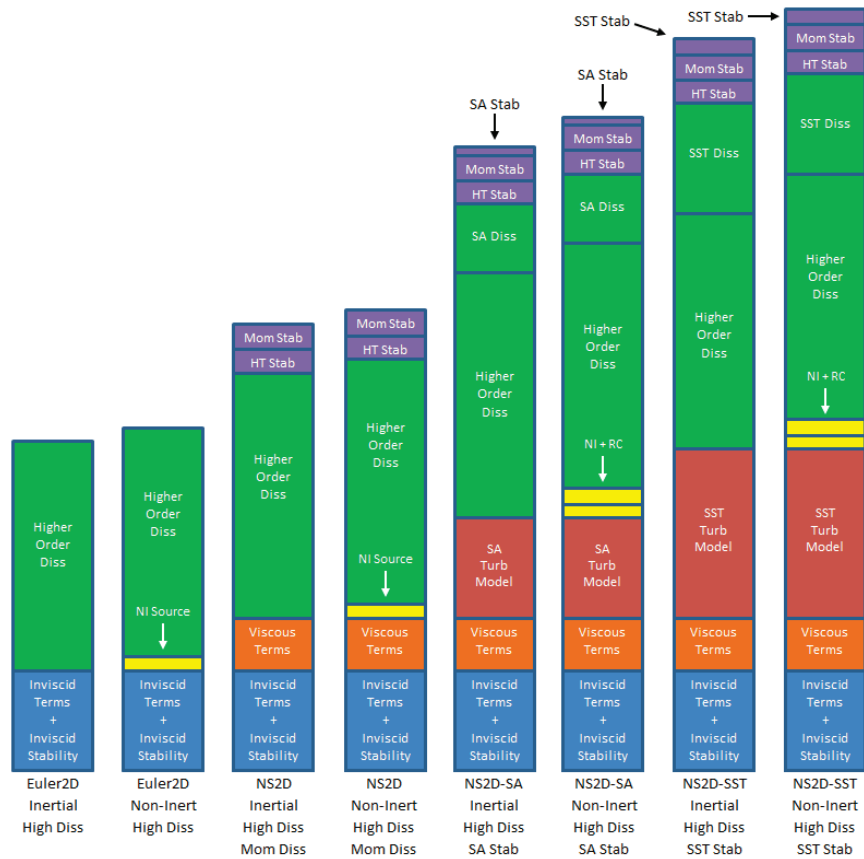


Figure 5.31: Comparison of Run Times in 2D Inertial and Non-Inertial Frames.

5.7 Support Software

Several support codes have been written by various developers in the CASElab. This section outlines the support software used in this research and gives credit where it is known to the original author. Additions and uses (current or future) are outlined.

5.7.1 Geometry Codes

Several codes were written to create, adapt, or adjust the geometry and mesh.

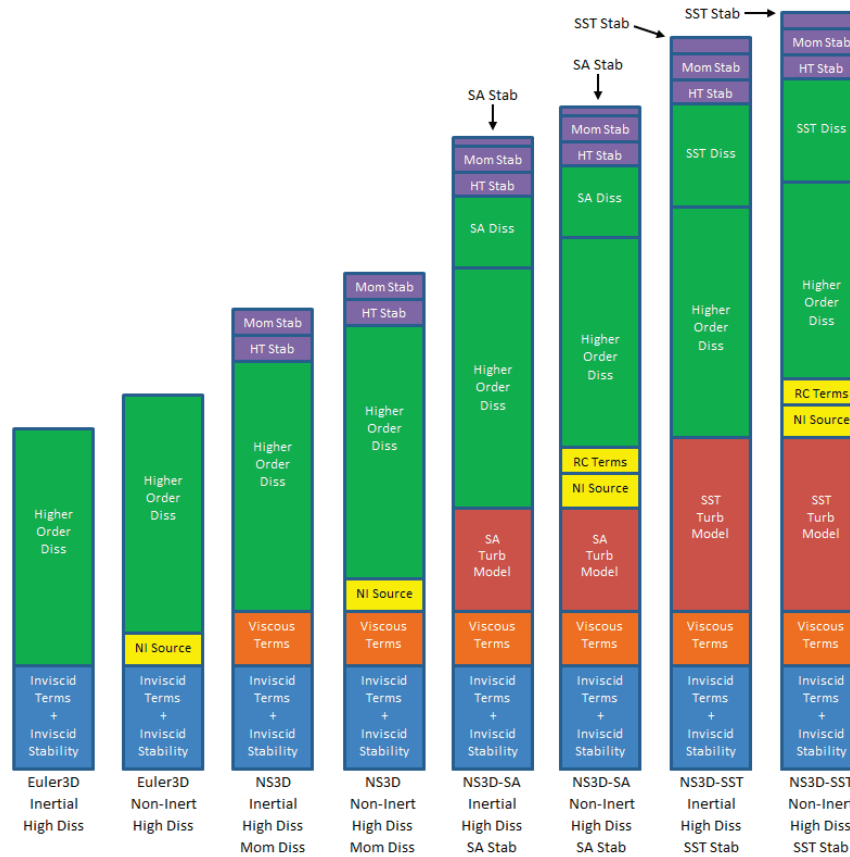


Figure 5.32: Comparison of Run Times in 3D Inertial and Non-Inertial Frames.

Surface and SurfaceOld. SurfaceOld was part of the FELISA software (Peiro, 1993).

SurfaceOld defines all of the surfaces and their cutting curves in the case.sur. A background mesh is defined in case.bac along with sources to refine the spacing. SurfaceOld uses a Delauney marching front method to generate a triangulation over all surfaces in case.sur. The final triangulation is written to the case.fro. SurfaceOld uses a water-tight tolerance when checking surfaces and curves and their intersections. Surface is a revised version of SurfaceOld with a tighter tolerance. 2D meshes are generated with SurfaceOld.

Volume and VolumeOld. VolumeOld was part of the FELISA software (Peiro, 1993).

VolumeOld starts with the discretized surfaces in the case.fro. A background mesh is defined in case.bac along with sources to refine the spacing. VolumeOld uses a Delauney marching front method to generate a tetrahedral mesh between all surfaces in case.fro. The final mesh is written to the case.gri. VolumeOld uses a water-tight tolerance when checking surfaces and their intersections. Volume is a revised version of VolumeOld with a tighter tolerance. 3D meshes are generated using VolumeOld.

Pave2D and Pave3D. Pave2D were written as an alternative to generating specific cases with SurfaceOld. Pave2D create mathematically regular meshes over many domains and distributions: rectangular; radial around a central surface; H- and U-meshes around airfoils; laminar and generic (transitioning) boundary layer meshes over flat plates; turbulent sections (far downstream from leading edge) over flat plate. Pave3D were written as an alternative to generating specific cases with VolumeOld and creates 3D extrusions of the previous cases described for Pave2D.

Convert_Plate and Convert_Bump. Convert_Plate and Convert_Bump are used to convert the geometry files from NASA Langley Turbulence Modeling Resource website (Rumsey, 2012). Convert_Plate converts structured grids over flat plate into case.g2d files, and Convert_Bump converts 2D bump geometries to case.g2d format. The geometries can be converted using rising, falling, or alternating diagonals. Rising or falling are suggested.

MakeG2D. MakeG2D was written by Cowan (2003) and adapted to include viscous information. MakeG2D converts the mesh in case.fro and boundary conditions in case.bco into the 2D geometry file case.g2d. The geometry is sorted according to Figure 5.1 and Figure

5.2. The case.g2d defines the geometry in Euler2D and NS2D. If the case.g2d is viscous, a connectivity is created between boundary elements and adjacent elements. The nodes and elements along viscous walls are written to the case.vwl in preparation for temperature boundary conditions.

MakeG3D and MakeNC3D. MakeG3D was written by Cowan (2003) and adapted to include viscous information. MakeG3D converts the surface mesh in case.fro, volume mesh in case.gri, and boundary conditions in case.bco into the 3D geometry file case.g3d (binary). The geometry is sorted according to Figure 5.1 and Figure 5.2. The case.g3d defines the geometry in Euler3D and NS3D. If the case.g3d is viscous, a connectivity is created between boundary elements and adjacent elements. The nodes and elements along viscous walls are written to the case.vwl in preparation for temperature boundary conditions. Since the case.g3d file is a binary file, an ASCII version (ascii.g3d) is written if the file is already present upon run time. MakeNC3D was adapted from MakeG3D by O'Neill (2011). MakeNC3D writes a case.nc3d file in the NetCDF format. (See Appendix E for specifics.)

MakeCFS. MakeCFS was written by Gupta (2007) and adapted by AES to include singular nodes. MakeCFS converts the surface mesh in case.fro and volume mesh in case.gri into the 3D geometry file case.cfs. MakeCFS does not sort the nodes or boundary elements. The case.cfs defines the geometry in CFDsol.

g3d2cfs and cfs2g3d. These are converters between case.g3d and case.cfs files. Since the case.g3d is sorted in a particular order, a case.cfs cannot be converted to case.g3d and used in Euler3D or NS3D. A converted case.g3d (using cfs2g3d) can be viewed using Gplot3D,

which is the method of choice for viewing the geometry mesh and CFDsol solution. `g3d2cfs` can be used to create a quick ASCII version of a `case.g3d`.

`g3d2nc3d` and `nc3d2g3d`. These are converters between `case.g3d` and `case.nc3d` files. Both formats contain the same information and sorting, so the mesh is equivalent and conserved in the processing. Both files are binary files, but the `case.nc3d` is written in the NetCFD format.

Remesh3D. Remesh3D was created by O'Neill (2011) by converting Remesh to read `case.g3d` files. Remesh3D is used to analyze the solution for a given `case.g3d`. The first or second derivative of the solution is used to define a new distribution of nodes. A `caseR.bac` file is written containing all of the elements and nodes in the `case.g3d`. A spacing and orthogonal vector set is defined at each nodes in the domain. The `caseR.bac` can be renamed to `case.bac` and used to mesh the solution using `SurfaceOld` and `VolumeOld`.

If first derivatives are used, the gradient vector is used to determine the spacing at each node, and the direction vectors are equal to the Cartesian basis $(\hat{i}, \hat{j}, \hat{k})$. If second derivatives are used, the Hessian matrix is estimated at each node in the domain. The eigenvalues of the Hessian are used to estimate a new spacing at each node, and the eigenvectors of the Hessian are used to define the orthogonal direction vectors.

The new mesh spacing is controlled through five parameters: S_max is the maximum spacing (ratio of spacing between directions) on an element. F_spa determines whether to keep the current spacing, new spacing, or mixture of the two ($0 \leq F_spa \leq 1$). D_min and D_max are the minimum and maximum spacing on the new mesh. D_ref is the reference spacing used to relate the old and new meshes.

Remesh3D has been used in conjunction with SurfaceOld, VolumeOld, MakeG3D, and Euler3D to create the meshes shown in Figure 5.33 and Figure 5.34

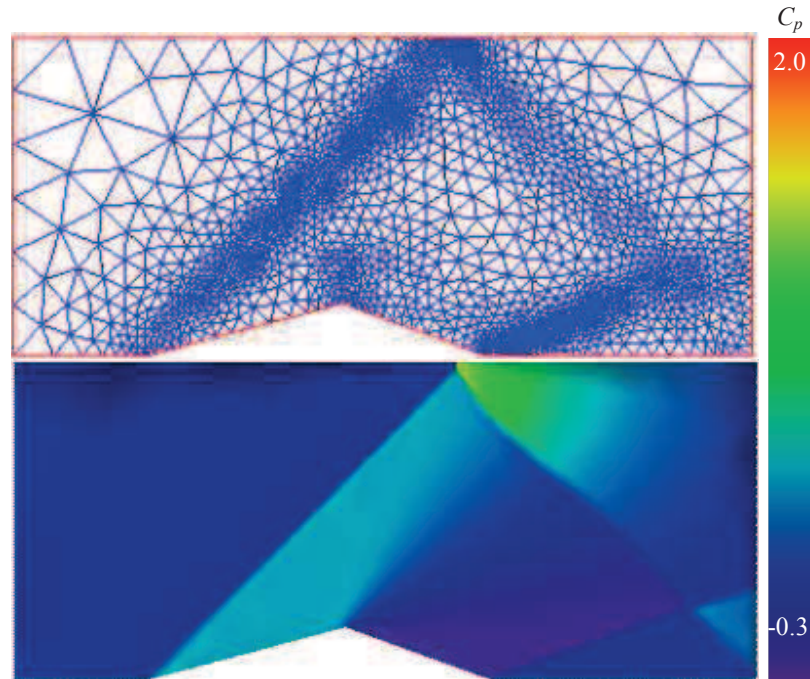


Figure 5.33: Remeshed Solutions for Supersonic Wedge.

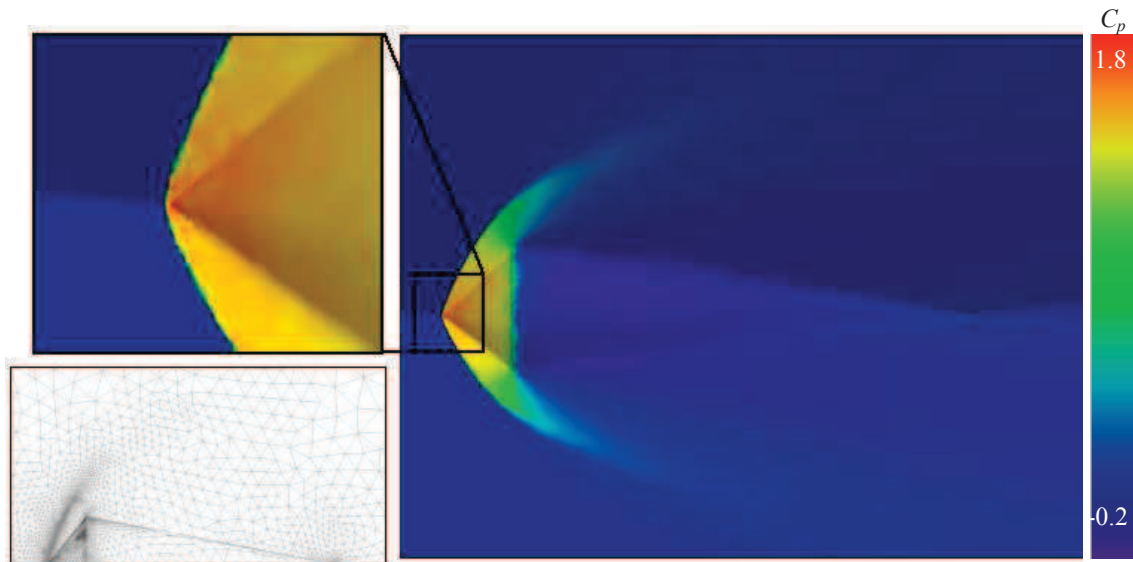


Figure 5.34: Remeshed Solutions for Supersonic Cones.

5.7.2 File Preparation Codes

Several codes have been written to prepare files for the OSU codes.

EditCon. EditCon was created to edit the case.con and case.data controls files from the prompt or within batch files. EditCon writes a table to the screen for the user to see what the current control scheme means and allows the user to change entries before writing back to the file. A controls file can be read in the formats of any of the five solvers used in this work, edited, and written back to the same or to a different format. EditCon can also be used to create a case.con full of default values by inserting the first “&control” and last “/” lines into the case.con. EditCon can be used to read the empty file and immediately write the default controls back to the same file. If a default case.data is desired, the empty case.con is created, converted to defaults, and then converted to case.data format.

SetInit2D and SetInit3D. SetInit2D and SetInit3D were created by Cowan (2003) to see domains with desired distributions for testing. The following initial conditions can be created using SetInit2D and SetInit3D:

- Double-shock shock tube (not available in SetInit3D)
- Single-shock shock tube
- Steady shock
- Centrifuge
- Constant gradient or constant curvature
- Sine functions
- Random perturbations, nodal (random)
- Random perturbations, elemental (all nodes of any element are different)
- Blasius or generic boundary layer solutions
- Velocity reduction by number of nodes off wall, generic boundary layer that is not a boundary layer solution (not available in SetInit3D)
- Rocket or afterburner region (like ABPrep2D)
- Cavity, no-flow region (CavPrep2D)
- Bluff body regions (no-flow downstream of body)
- Flat plate and engine solutions for moment integral testing

UnkDist. UnkDist is a generic and more user-friendly version of SetInit3D. UnkDist must be hard-coded with a distribution, but the programmer can define any distribution using any property(s) in *user_dist*.

UnkCorrect3D. UnkCorrect3D reads a case.unk and assumes either pressure, total energy, or total enthalpy is correct. The other two properties are calculated for consistency and written back to the case.unk.

ABPrep2D. ABPrep2D was adapted from SetInit2D to create a stable initial condition for rocket and/or afterburner problems. ABPrep2D creates linear ramps between high and low pressure regions to create a smooth startup. This program was made obsolete by SetInit2D and the ramp-up iterations for rocket and engine outflow boundaries.

CavPrep2D. CavPrep2D was adapted from SetInit2D to create a more accurate initial condition for open cavities. CavPrep2D creates a no-flow region within the bounds of the cavity. This eliminates acoustic waves created by the freestream slamming into the downstream wall just after startup. This program was made obsolete by SetInit2D.

Frstrm2D. Frstrm2D reads a case.unk file and resets the properties within a rectangular region to freestream properties, as defined in case.con. The new properties are written to the case.unk.new. This program assumes inviscid case.unk file. A viscous case.unk can be converted to inviscid using UnkAdapt2D and vice versa.

Combust2D and Combust3D. Combust2D and Combust3D define the combustion case.cmb file from user inputs. The user defines a series of Cartesian rectangular regions with mass and heat production. The transitions can be discontinuous or cosine splined, and the transitions can be different in each direction (discontinuous at wall and smooth transition in flow).

A case.unc is written plotting like a case.unk in GlPlot2D/3D, where mass generation is plotted as density and heat generation is plotted as C_p . Viewing is direct for nodal distributions. Elemental distributions are pushed to the nodes using area- or volume-weighted averages. The data can also be written to case.txt file with format: coordinates, mass gen, heat gen.

MakeVec2D. MakeVec2D used to create 2D case.vec file. MakeVec2D can be used to create rigid body rotations or translations of solid walls and/or produces flap deflections by rotating a section of a solid wall.

MakeVec3D. MakeVec3D uses mode shape vectors and frequencies from STARS case.arrays file to create a 3D case.vec file. Rigid body and controls deflections can also be created.

5.7.3 Solution / Geometry Conversion Codes

Several codes have been written to convert between solutions and their geometries.

2Dto3D. 2Dto3D converts a case.g2d to case.g3d for plotting. The case.g3d created with 2Dto3D cannot be used in Euler3D or NS3D. Each triangle in the 2D mesh is converted to a tetrahedral with one node off of the 2D plane. When viewing in Glplot3D, the case.g3d must be loaded alone and then reload with the case.unk or multiple case.un# files. If the two operations are done in one step or with steps between them, Glplot3D will fault and terminate. This program was made obsolete by 2Dextrude.

2Dextrude. 2Dextrude converts every triangle into a prism and then subdivides that prism into tetrahedral. The 3D mesh is only one prism thick. 2Dextrude can convert geometry alone, with a case.unk, or with case.un# files. The original 2D plane becomes a symmetry

plane. Another symmetry is created to mirror the first, one unit away. Nodes are created in the middle of each prism and each prism face. An ASCII file of geometry is written for debugging.

Geom2Dto3D. Geom2Dto3D converts the case.sur, case.bac, and case.bco files designed for a 2D mesh into corresponding files meant for a 3D mesh.

Rotate2D and Rotate3D. Rotate2D and Rotate3D rotates geometry and velocity vectors through Euler rotations of an original case.g3d and case.unk. All rotations are made in fixed frame of original x,y,z, in the order z, y, and then x. Rotate2D and Rotate3d can read inviscid and viscous versions of case.g2d and caseIg3d and inviscid, laminar, and turbulent versions of case.unk or case.un# files.

UnkASCII. UnkASCII was created from UnkAdapt2D and later incorporated back into UnkAdapt2D. UnkASCII converts a 2D case.unk file from binary to ASCII format. This program was made obsolete by UnkAdapt2D.

Unk2Out. Unk2Out converts a case.unk (OSU restart file) to a case.out (CFDsol restart file).

UnkAdapt2D and UnkAdapt3D. UnkAdapt2D and UnkAdapt3D were created from UnkAdapt. UnkAdapt2D and UnkAdapt3D read the case.unk file in inviscid, laminar, or turbulent formats. The user is allowed to view and adapt the header. The adapted file is written to case.unkA and an optional ASCII format (case.unk.ascii) for debugging and/or post-processing. UnkAdapt2D and UnkAdapt3D can be used to add *Re* to inviscid files or *iturb* to laminar files. UnkAdapt3D can handle case.g3d or case.nc3d.

UnkInterp2D and UnkInterp3D. UnkInterp2D and UnkInterp3D interpolates a case.unk from one mesh (“name1”, case.g2d or case.g3d) to another mesh (“name2”, case.g2d or case.g3d). Values outside of the domain are extrapolated to nearest boundary element. The geometry must be the same. The user is prompted to adapt header before writing the unknowns file. The case.unk is assumed to already contain *iturb*. For older unknowns files, the turbulent flag can be added by running through UnkAdapt2D or UnkAdapt3D.

5.7.4 Solution Analysis

Several codes were written to parse or interpret data files.

MakeCut2D. MakeCut2D extracts data from the case.g2d and case.unk files. Properties can be extracted along lines or boundary curves. This is similar to acoustic lines but limited to post-processing a case.unk file that has already been written. MakeCut2D does not recognize viscous or turbulent controls or turbulent unknowns.

MakeCut3D. MakeCut3D extracts data from the case.g3d and case.unk files. Properties can be extracted along lines or averaged over boundary surfaces. All of the properties along a specific surface can also be written. The properties or forces along an intersection of a plane and boundary surface can also be exported. MakeCut3D does not recognize viscous or turbulent controls or turbulent unknowns.

ParseDat. ParseDat parses the xd.dat, xn.dat, or case.lds files to pull plottable data. The minimum time step returns the original data. The maximum time step returns data points for each case.un# file. The data can also be parsed at discrete intervals between these two limits.

Acst_FFT. Acst_FFT reads the acoustic output files (case.rac, case.uac, case.vac, case.wac, and case.pac) and calculates a probability spectral density using FFT.

Lift_Hist. List_Hist calculates the forces and moments for a set of case.un# files. This program was been made obsolete when the case.lds was added to outputs of CFDsol.

Slices. Slices produces property cuts along specified plane and rotations for each case.un# file in a solution (similar to MakeCuts3D).

5.7.5 Validation Codes

SmHess. SmHess is a Smith-Hess panel method code written by Dr. Arena.

PrepBlasius2D and PrepBlasius3D. PrepBlasius2D and PrepBlasius3D pull profiles near the wall and compares to Blasius solution.

Turb2D. Turb2D pulls profiles near the wall and compares to turbulent boundary layer profiles. Turb2D was used to extract meaningful data from Rumsey's (2012) flat plate, and 2D bump cases. Turb2D was also used for grid convergence.

CHAPTER VI

DEMONSTRATION OF IN-HOUSE CODES

Verification is the process of comparing solutions from a method with those produced by other analytical or numerical methods. Validation is a similar process, where experimental data is used as a means of comparison (Roache, 1998a and 1998b). All verifications and validations are demonstrations of capability, but demonstrations are not verifications or validations without comparison. Verification proves that the simulation is properly representing the desired physics model, and the CFD solution is verified when the solution matches the desired physics model (analytical or numerical) within a reasonable limit or in the limit as the mesh approaches a continuous discretization. Validation is performed on a verified model to demonstrate how *valid* (accurate) that model is for a particular purpose, which the validation data encompasses.

The OSU solvers used in this research have been demonstrated on various test cases across multiple regimes. The physics within the solvers are first verified with theoretical and numerical predictions and then validated with experimental and empirical data. The cases presented here do not represent an exhaustive verification or validation but rather demonstrate the capabilities of the OSU in-house codes. The test cases are separated into three

main divisions: Inviscid aerodynamics (Euler2D and Euler3D), propulsion modeling (Euler2D and Euler3D), and viscous aerodynamics (NS2D and NS3D).

6.1 Inviscid Aerodynamics

Inviscid aerodynamics tests are used to compare theoretical, numerical, and experimental data to those created from Euler2D and Euler3D. Some tests are created as a baseline comparison and will later be expanded to include viscous effects using NS2D and NS3D; other tests were generated as a parallel comparison with CFDsol, using the same mesh spacing and distributions. The inviscid tests have been divided into subsonic, transonic, supersonic, and time-accurate. The first three divisions test the steady-state accuracy of Euler2D and Euler3D operating throughout the compressible regimes. The time-accuracy was compared with CFDsol.

6.1.1 Subsonic

Six geometries were tested at subsonic speeds. An NACA 0012 airfoil was tested at two speeds. The lower speed Mach 0.3 case is compared to a numerical solution produced using a Smith-Hess panel method. The higher speed Mach 0.5 case is compared to experimental data. An RAE 2822 airfoil was also tested at Mach 0.6 and represents one of the most accuracy comparisons. Four simple shapes were tested: A double-arc airfoil, ellipse, circular cylinder, and sphere.

6.1.1.1 NACA 0012 Airfoil (Mach 0.3, 5 deg)

The NACA 0012 airfoil is a simple subsonic case that can be compared to a numerical solution using a Smith-Hess panel method (Katz, 2001). The airfoil was held at 5 degrees

angle of attack at Mach 0.3, to create different pressure distributions on the top and bottom surfaces. The Smith-Hess solution (Arena, unpublished) breaks the surfaces of the airfoil into a series of vortex-panels and predicts the velocity and pressure distributions on the top and bottom surfaces. These pressures are converted to C_p for easier comparison and corrected for compressibility using the Prandtl-Glauert (Anderson, 2001):

$$C_p = (C_p)_{M=0} / \sqrt{1 - M_\infty^2}$$

The distribution of pressure over the airfoil is shown in the 2D test case description and used to calculate lift and moment (LE) coefficients: $c_l = 0.6217$ and $c_{m,LE} = -0.1567$. The pressure drag should be zero.

The mesh was generated with a constant background spacing on an elliptical far field. Regions around the leading and trailing edges were refined to capture the curvature of the nose radius and trailing edge flow. The wake was refined for the steady and later unsteady solutions. The final mesh is shown in Figure 6.1.

The Euler3D solution converged after 3000 iterations, with 4 inner cycles per iteration and a *CFL* of 0.5. Higher order dissipation was used with a dissipation constant *diss* of 0.6. The distributions of four of the properties are plotted in Figure 6.2 and again in Figure 6.3 in reference to their far field distributions. The pressure distribution shows the highest pressure at the stagnation point, just under the leading edge. The highest velocity corresponds to the lowest pressure, located over the top of the forward section of the airfoil. Entropy production is confined to the airfoil because no shocks or viscous stresses are present. The most entropy is generated along the nose of the airfoil, signifying a possible need to refine the mesh.

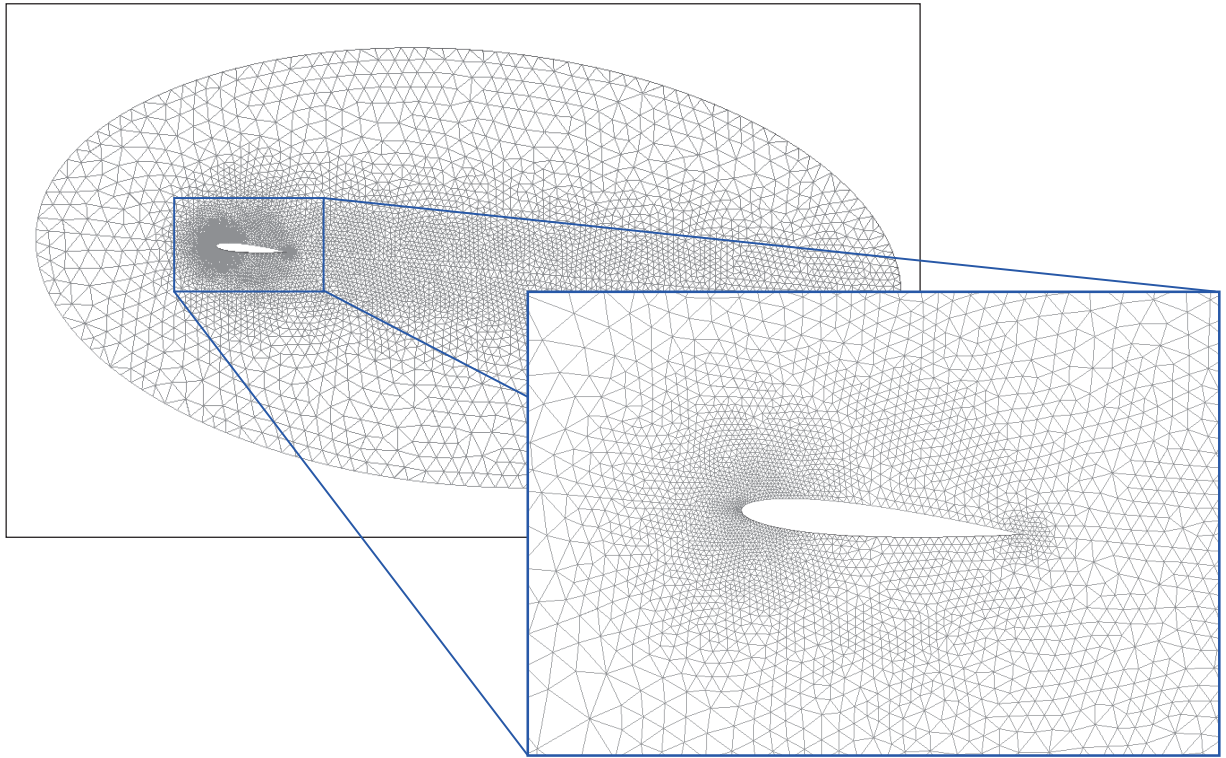


Figure 6.1: NACA 0012 Airfoil Mesh, Elliptical Far Field.

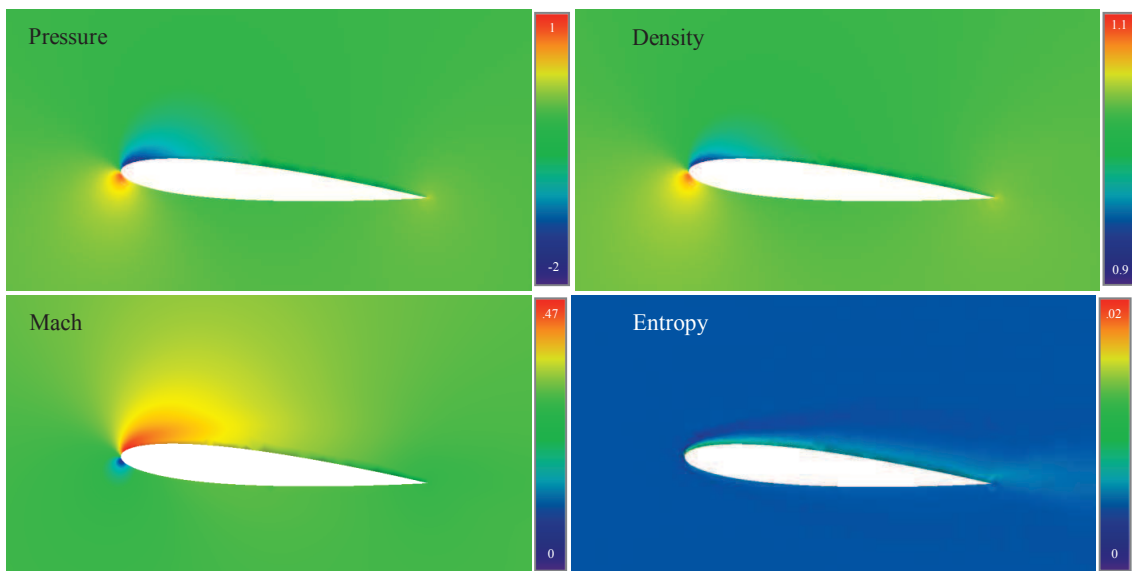


Figure 6.2: Pressure, Density, Mach, and Entropy Distrib. for NACA 0012 (Mach 0.3).

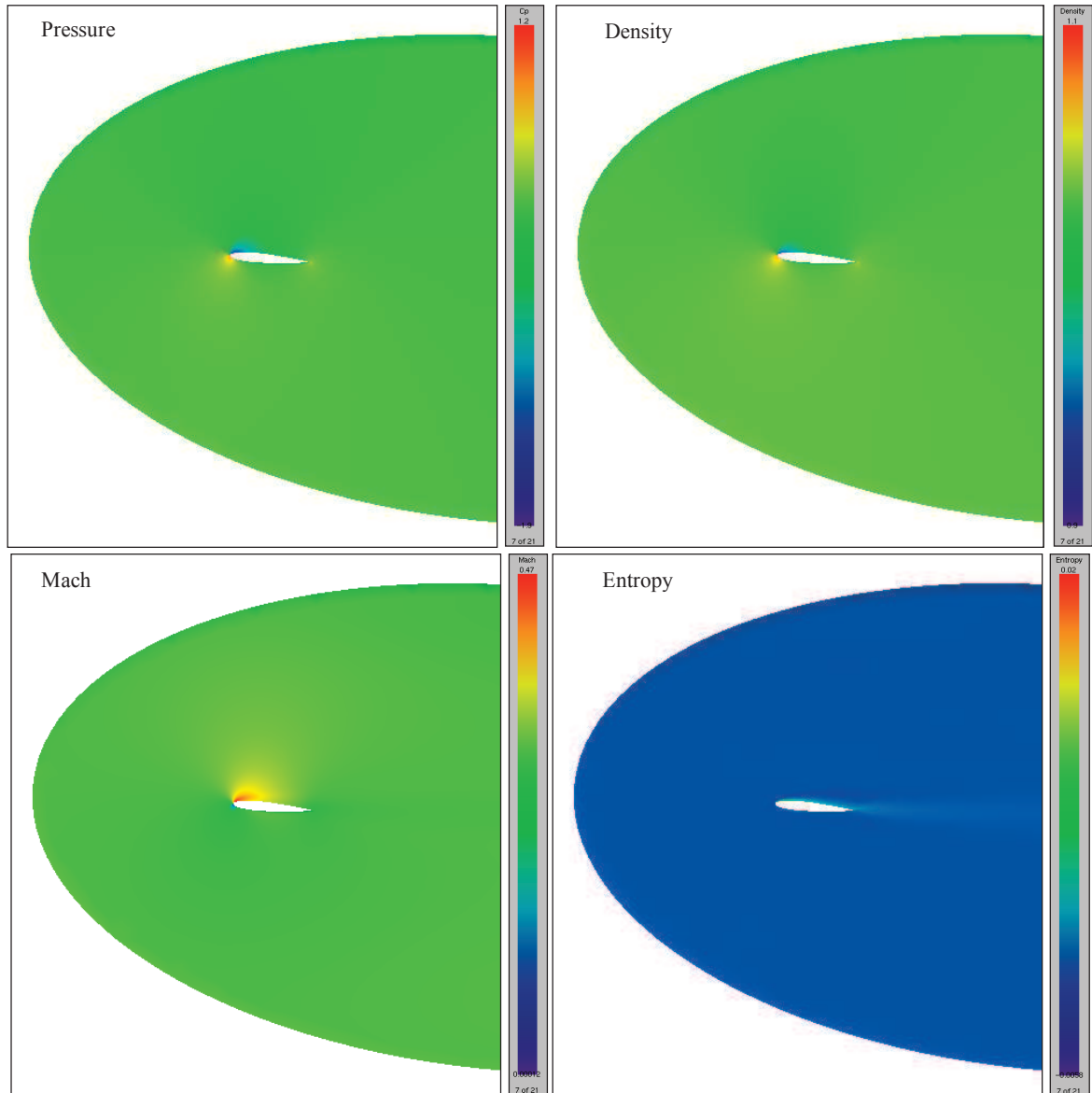


Figure 6.3: Pressure, Density, Mach, and Entropy Distrib. for NACA 0012 (Mach 0.3).

The Euler2D and Euler3D solutions are shown in comparison with the Smith-Hess solution in Figure 6.4 and Figure 6.5, respectively. The pressure distribution found using Euler2D and Euler3D matches the Smith-Hess solution very well. Euler3D predicts less suction than desired at the leading edge. Small errors are present near the leading edge, so any mesh

refinement would be necessary at the LE, where the greatest changes in geometry and the flow field occur.

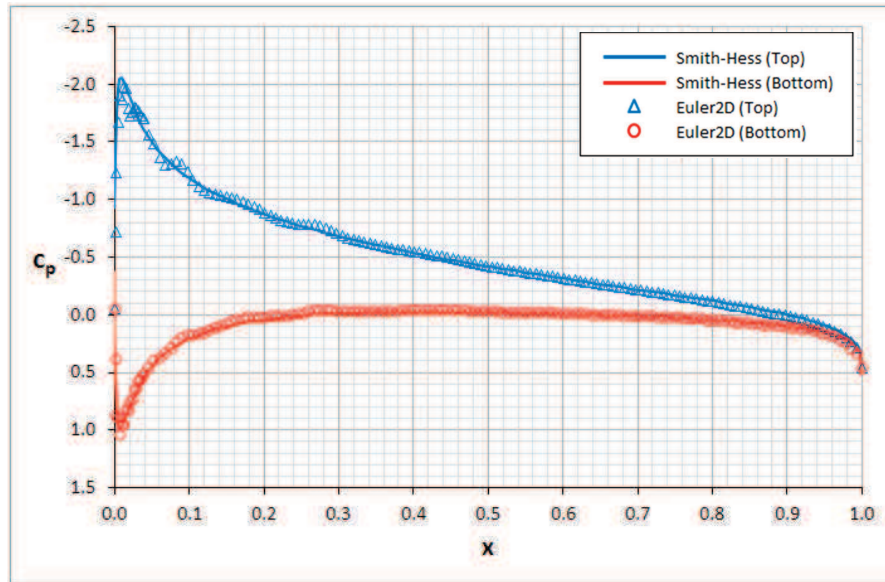


Figure 6.4: Coefficient of Pressure from Euler2D Compared to Smith-Hess Solution.

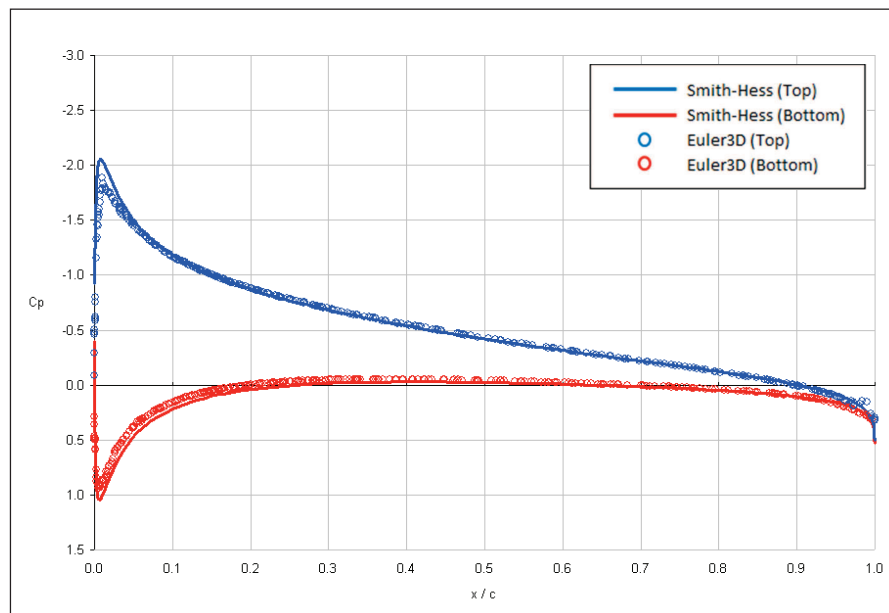


Figure 6.5: Coefficient of Pressure from Euler3D Compared to Smith-Hess Solution.

The pressure everywhere in the Euler3D solution is plotted versus the x' -distance (see Figure 6.6). x' is a measure of the distance downstream of the airfoil leading edge. The airfoil perturbs the flow about a freestream value: $C_p = 0$. The pressure begins to change upstream of the airfoil and returns to the freestream pressure downstream. The pressure over the top and bottom surface of the airfoil in Figure 6.6 is equal to that in Figure 6.4, where the pressure coefficient has been inverted in Figure 6.4. Looking more closely at Figure 6.6, the pressure on the “far field” within the bounds of the airfoil ($0 < x' < 1$) is not equal to that at the up and downstream far fields. The coefficient of pressure has been perturbed away from the freestream value because of the presence of the airfoil.

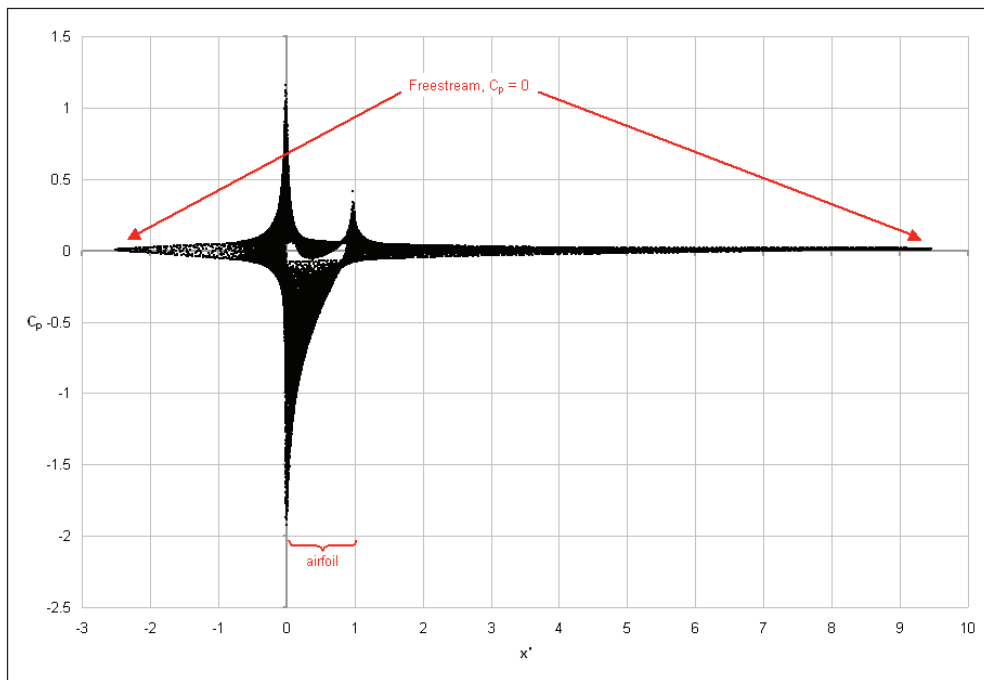


Figure 6.6: Coefficient of Pressure through Euler3D Solution Domain.

Anderson (2001) shows that the pressure above and below an airfoil experiences a change due to the presence of the airfoil. As the distance away from the airfoil grows, the properties become more *like* the freestream because the influence of the airfoil is spread out over a larger area. Any “far field” put in proximity of the airfoil should show a perturbation from the freestream values. Similar distributions are shown for density and velocity in Figure 6.7.

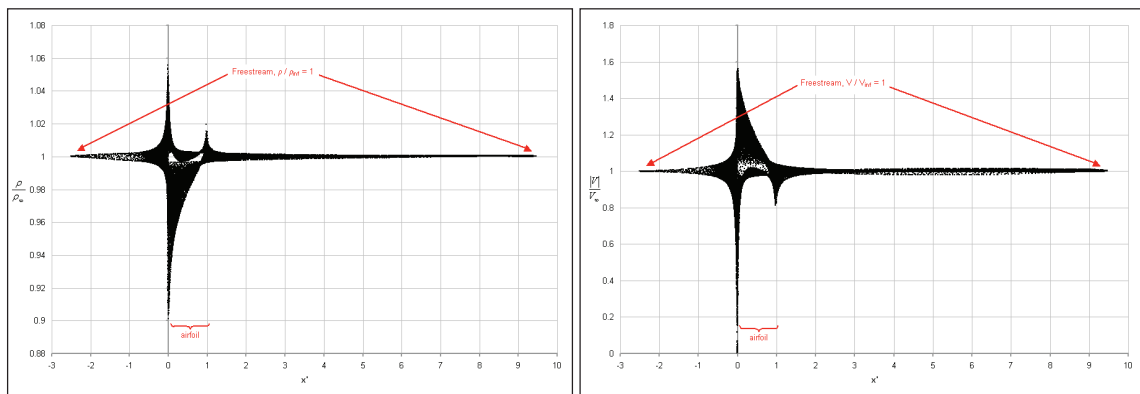


Figure 6.7: Distribution of Density (Left) and Velocity (Right) from Euler3D Solution.

6.1.1.2 NACA 0012 Airfoil (Mach 0.502, 1.77 deg)

An NACA 0012 airfoil was repeated in Euler3D at Mach 0.502 and 1.77-degrees angle of attack. The CFD solution is compared below with data from Barche (1979). The mesh is shown in Figure 6.8. The meshes were generated using Pave2D and Pave3D instead of the Delaunay method used in Surface and Volume. Pave3D was used to produce a mathematical mesh for CFDsol, where the three-dimensional mesh was only one element thick.

Figure 6.9 shows the pressure and Mach distributions predicted by Euler3D, and Figure 6.10 compares the surface pressure from Euler2D and Euler3D to experimental data from Barche

(1979). The mesh spacing is the same, so Euler2D and Euler3D predict the same pressure distributions. The pressure on the lower surfaces matches very well, but the pressure on the top surface differs at the leading edge. The loss in suction was thought to come from a lack of refinement near the leading edge but testing in Euler2D points elsewhere. Further testing was done by Hassett (not yet published), who improve the solution over the top of the airfoil.

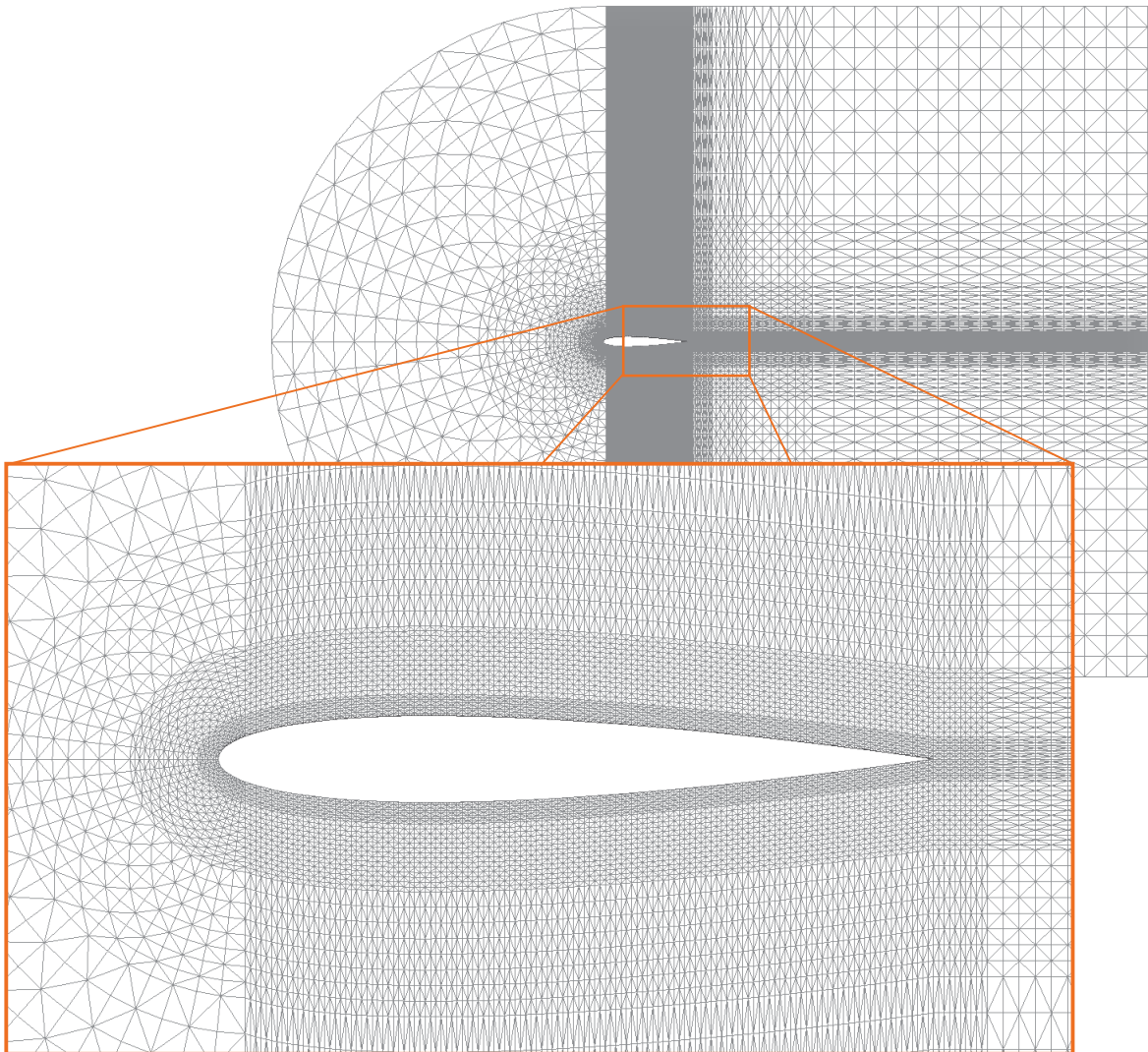


Figure 6.8: Mesh for NACA 0012 Airfoil (Mach 0.502, 1.77° AOA).

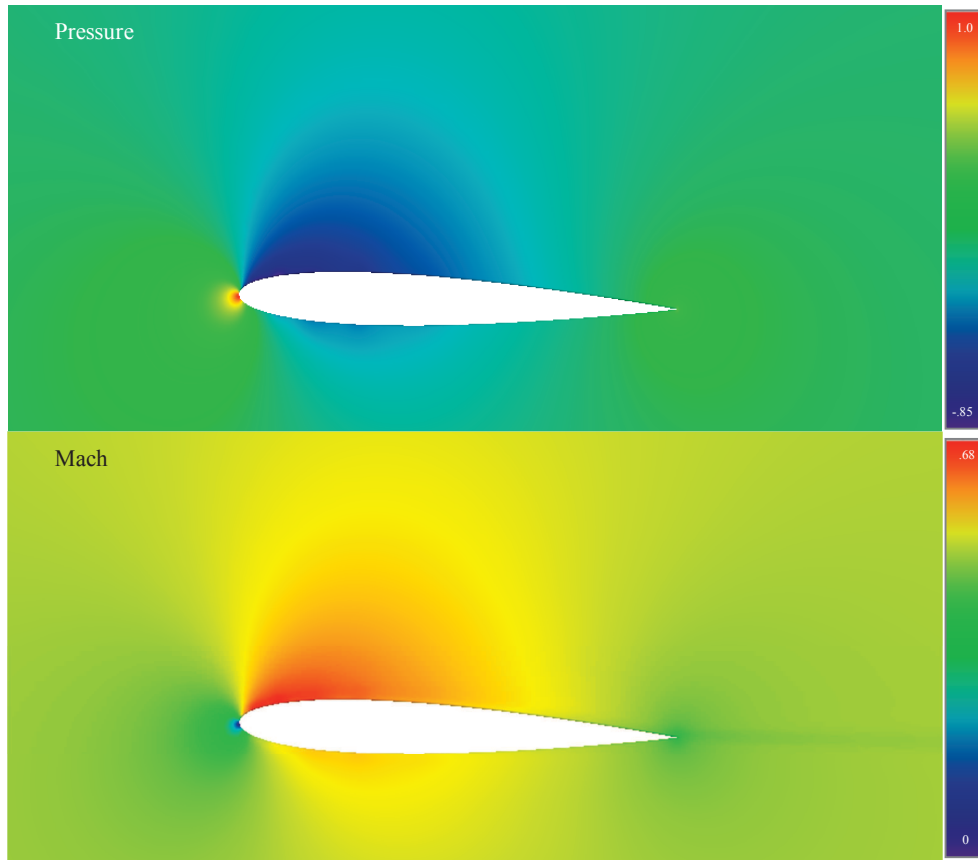


Figure 6.9: Pressure and Mach around NACA 0012 Airfoil (Mach 0.502, 1.77° AOA).

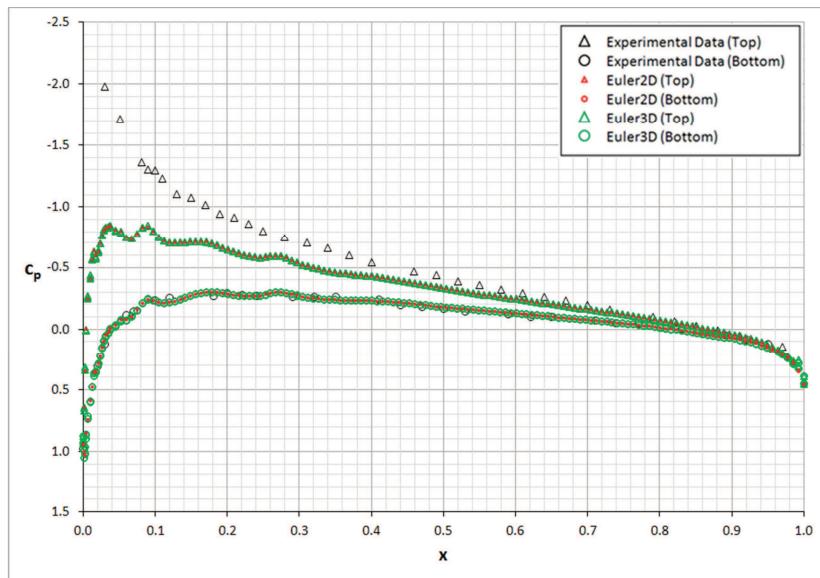


Figure 6.10: Surface Pressure over NACA 0012 Airfoil (Mach 0.502, 1.77° AOA).

6.1.1.3 RAE 2822 Airfoil (Mach 0.6, 2.57 deg)

An RAE 2822 airfoil was modeled in Euler2D and Euler3D at Mach 0.6 and 2.57-degrees angle of attack and compared below with data from Barche (1979). The mesh produced using Pave2D and Pave3D is shown in Figure 6.11:

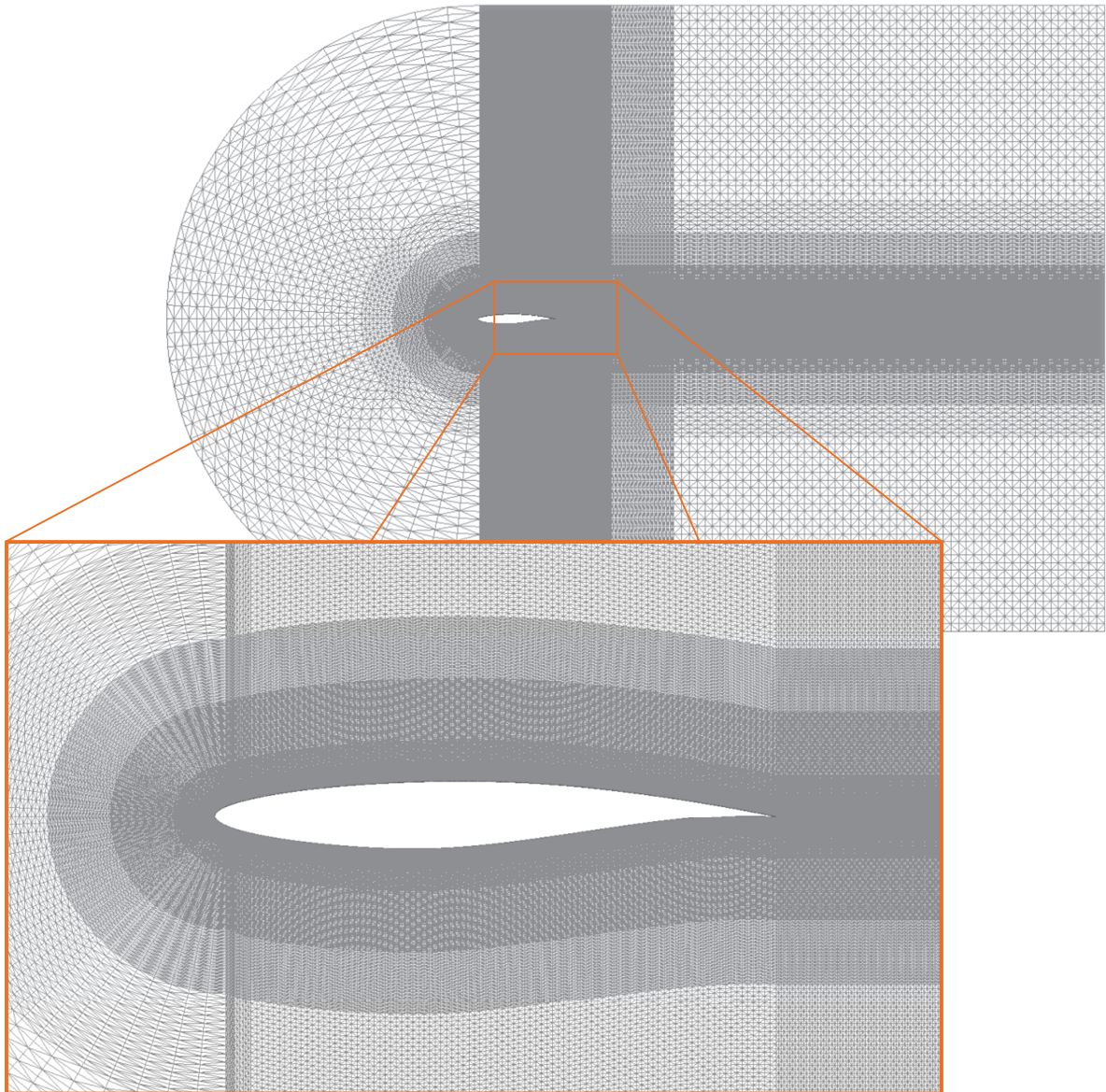


Figure 6.11: Mesh for RAE 2822 Airfoil (Mach 0.6, 2.57° AOA).

Results are shown in Figure 6.12 and Figure 6.13. Figure 6.12 shows the pressure, density, Mach, and entropy distributions predicted by Euler3D, and Figure 6.13 compares the surface pressure from Euler2D and Euler3D to experimental data from Barche (1979). The pressure on both surfaces matches very well with the experiment; the largest variations occur near the leading (top) and trailing edge (bottom). The two CFD solutions match up, except near the leading edge along the bottom surface.

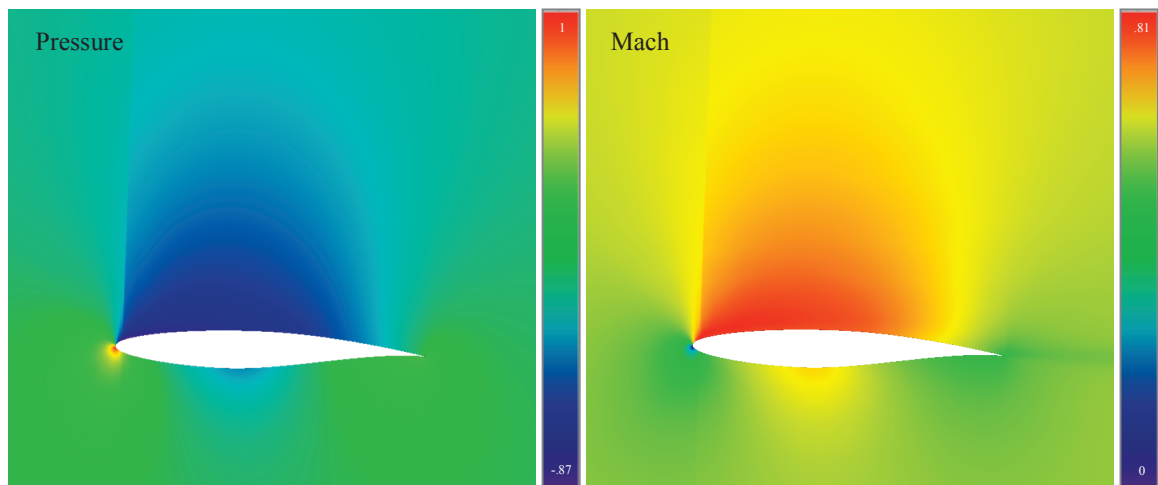


Figure 6.12: Pressure and Mach around RAE 2822 Airfoil (Mach 0.6, 2.57° AOA).

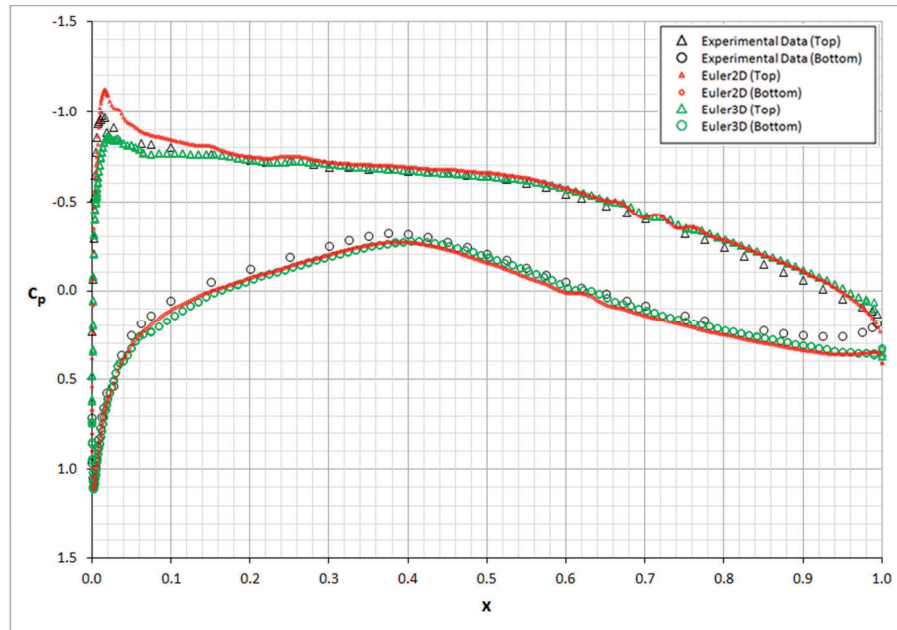


Figure 6.13: Surface Pressure over RAE 2822 Airfoil (Mach 0.6, 2.57° AOA).

6.1.1.4 Double-Arc Airfoil

A double-arc airfoil was demonstrated at Mach 0.3. The pressure, Mach, and entropy distributions are shown in Figure 6.14 along with the mesh used in Euler2D.

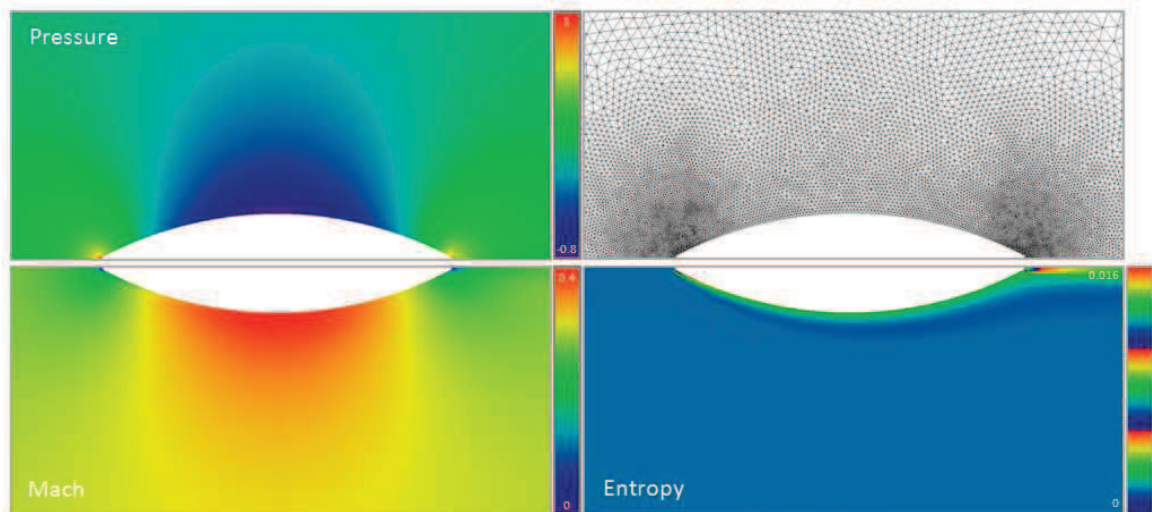


Figure 6.14: Pressure, Mach Number, and Entropy Distrib. near Cat's Eye (Mach 0.3).

6.1.1.5 Ellipse (Mach 0.3)

An ellipse with 6:1 ratio was tested under inviscid conditions, in preparation for a viscous boundary layer and in comparison to CFDsol. The inviscid pressure and velocity distributions are shown in Figure 6.16 for the mesh shown in Figure 6.15. The mesh was generated using three mesh sources so that the spacing was finer at the leading and trailing edges. The spacing was chosen to resemble the radius of curvature of the ellipse surface. (A viscous mesh was added to the near-wall region in preparation for viscous testing.) The inviscid surface pressure distribution from Euler3D for the ellipse is shown in Figure 6.17.

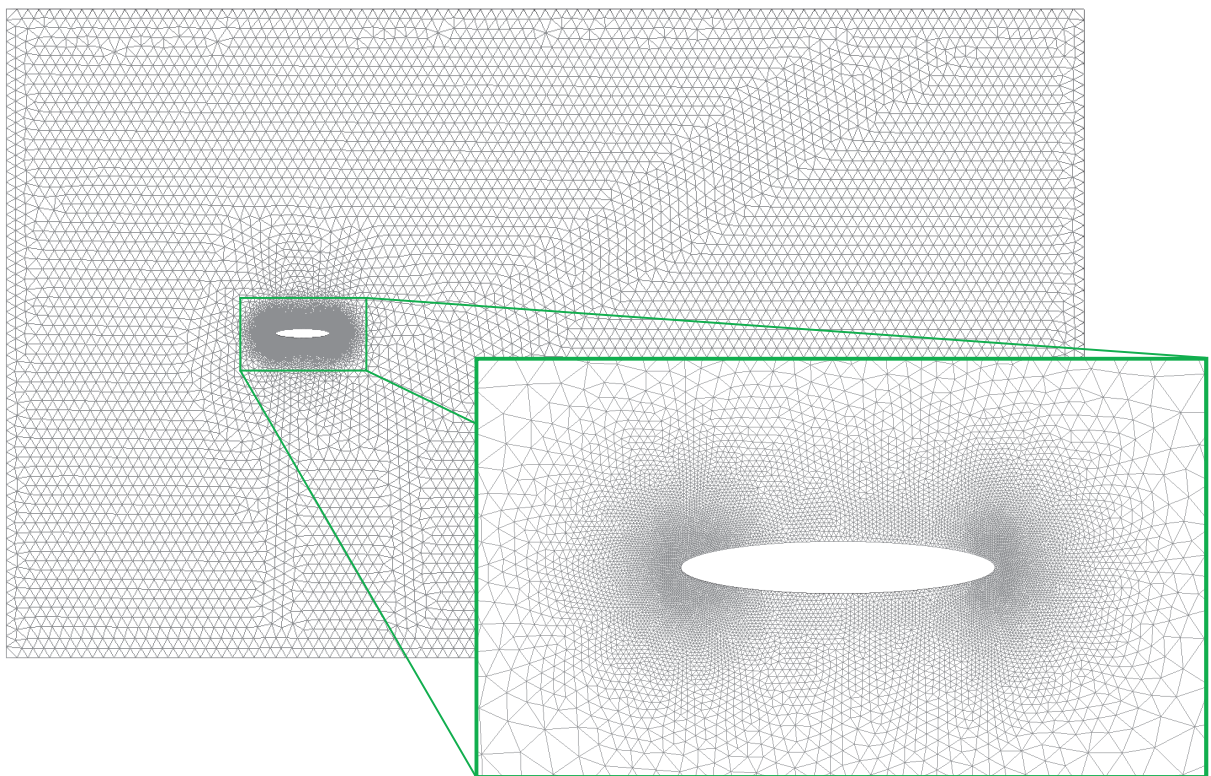


Figure 6.15: Mesh for 6:1 Ellipse (Mach 0.3).

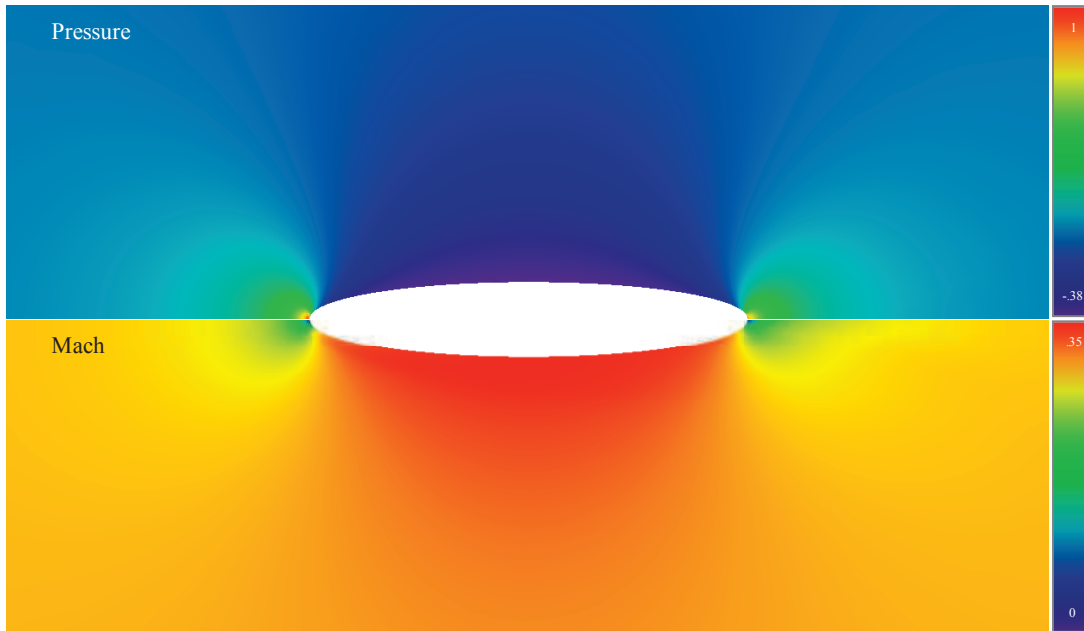


Figure 6.16: Pressure and Mach Distributions around Ellipse (Mach 0.3).

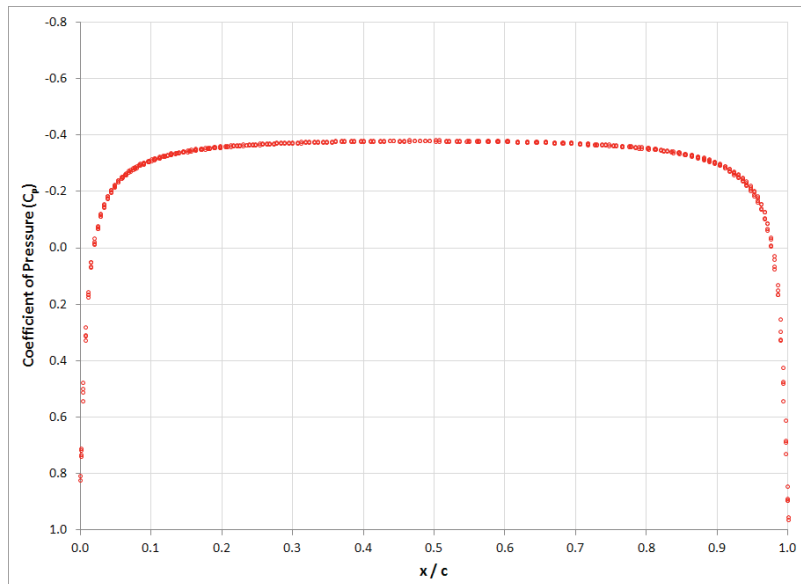


Figure 6.17: Surface Pressure over Ellipse (Mach 0.3).

6.1.1.6 Cylinder

A circular cylinder was tested under inviscid conditions to demonstrate a simple, symmetric two-dimensional body. (If only half of the cylinder was modeled, the run time and effects of artificial dissipation would have been reduced. Brown (2009) found that the Second Law is no longer satisfied when modeling a circular cylinder with minimal artificial dissipation. The symmetry plane enforces a flow pattern. This solution was found in preparation for viscous solutions, which require the full cylinder.) The mesh was generated so that the spacing on the cylinder surface is 6% of the radius of the cylinder, doubling at 3.75 radii from the center of cylinder. The pressure and Mach distributions around the cylinder are shown in Figure 6.19. The mesh used to produce that solution is shown in Figure 6.18.

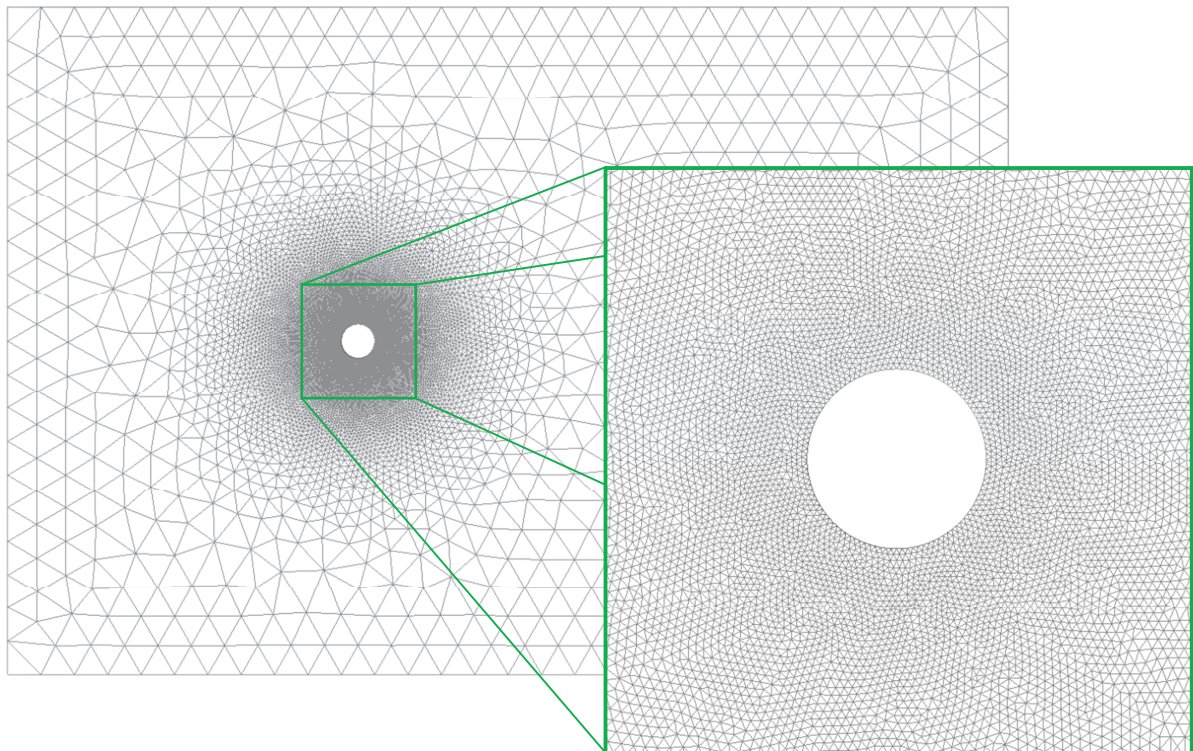


Figure 6.18: Mesh for Circular Cylinder (Mach 0.3).

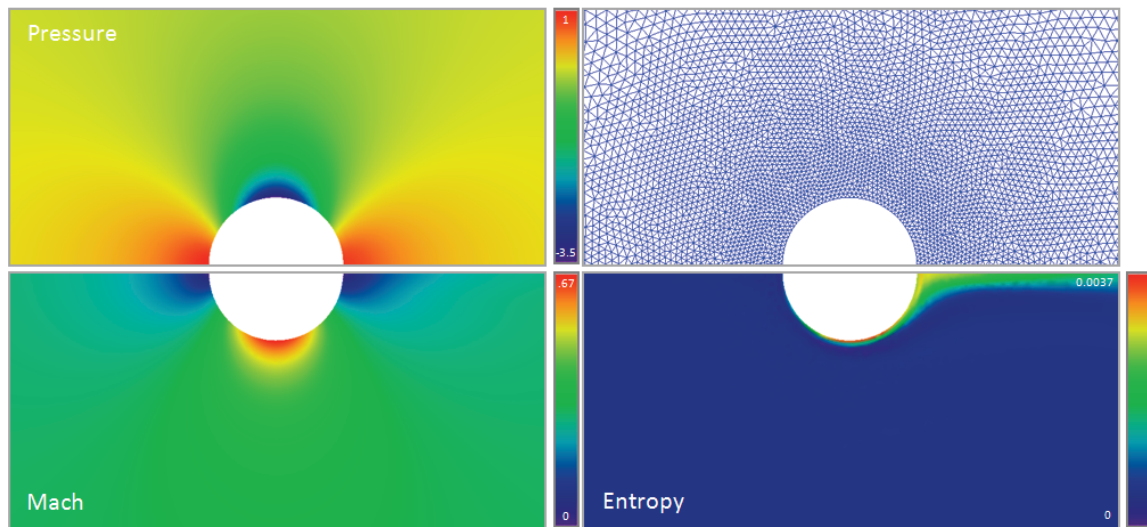


Figure 6.19: Pressure, Mach Number, and Entropy Distributions near Cylinder (Mach 0.3).

The surface pressure distributions from Euler2D and Euler3D are shown in Figure 6.20 (and corresponding surface velocity in Figure 6.21) compared with potential theory corrected for compressibility. The pressure distribution is asymmetric across the centerline because artificial dissipation is an *a priori* means of enforcing the Second Law. The artificial dissipation produces positive entropy production on the field, but the stagnation point is still allowed to float. Otherwise the distribution matches theory very well. Distributions predicted using Euler3D are compared with those modeled in Euler2D. Euler2D shows the rotated stagnation point more clearly for low amounts of artificial dissipation (low entropy production). Without enough artificial dissipation to produce positive entropy production, the CFD model cannot lock in on one solution. The Euler2D solution was repeated with a symmetry plane and with more dissipation. The symmetry plane improves the solution greatly. Adding dissipation causes the flow to “separate” at the trailing edge, but the solution over 90% of the cylinder surface is very accurate.

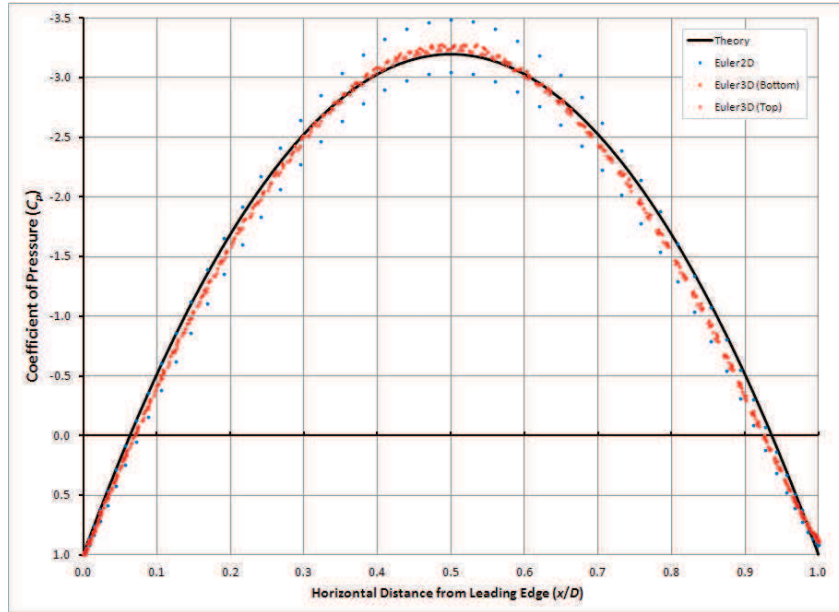


Figure 6.20: Surface Pressure Distributions over Circular Cylinder (Mach 0.3).

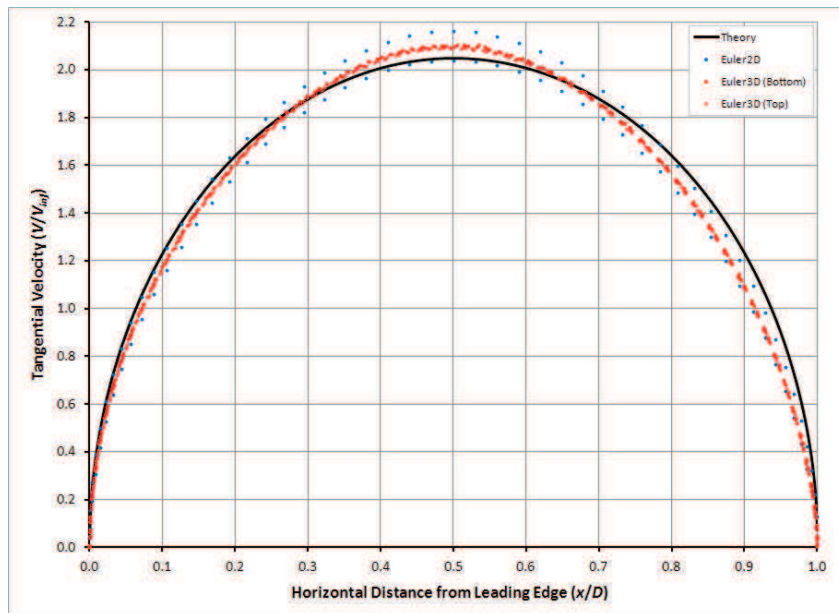


Figure 6.21: Velocity Distributions over Surface of Circular Cylinder (Mach 0.3).

6.1.1.7 Sphere

A sphere was tested under inviscid conditions to demonstrate a simple three-dimensional body. Only one quarter of the sphere was modeled to minimize the run time. The mesh was generated so that the spacing at the sphere surface is 6% of its radius, doubling at 3.75 radii from the center of sphere. The final mesh is shown in Figure 6.22.

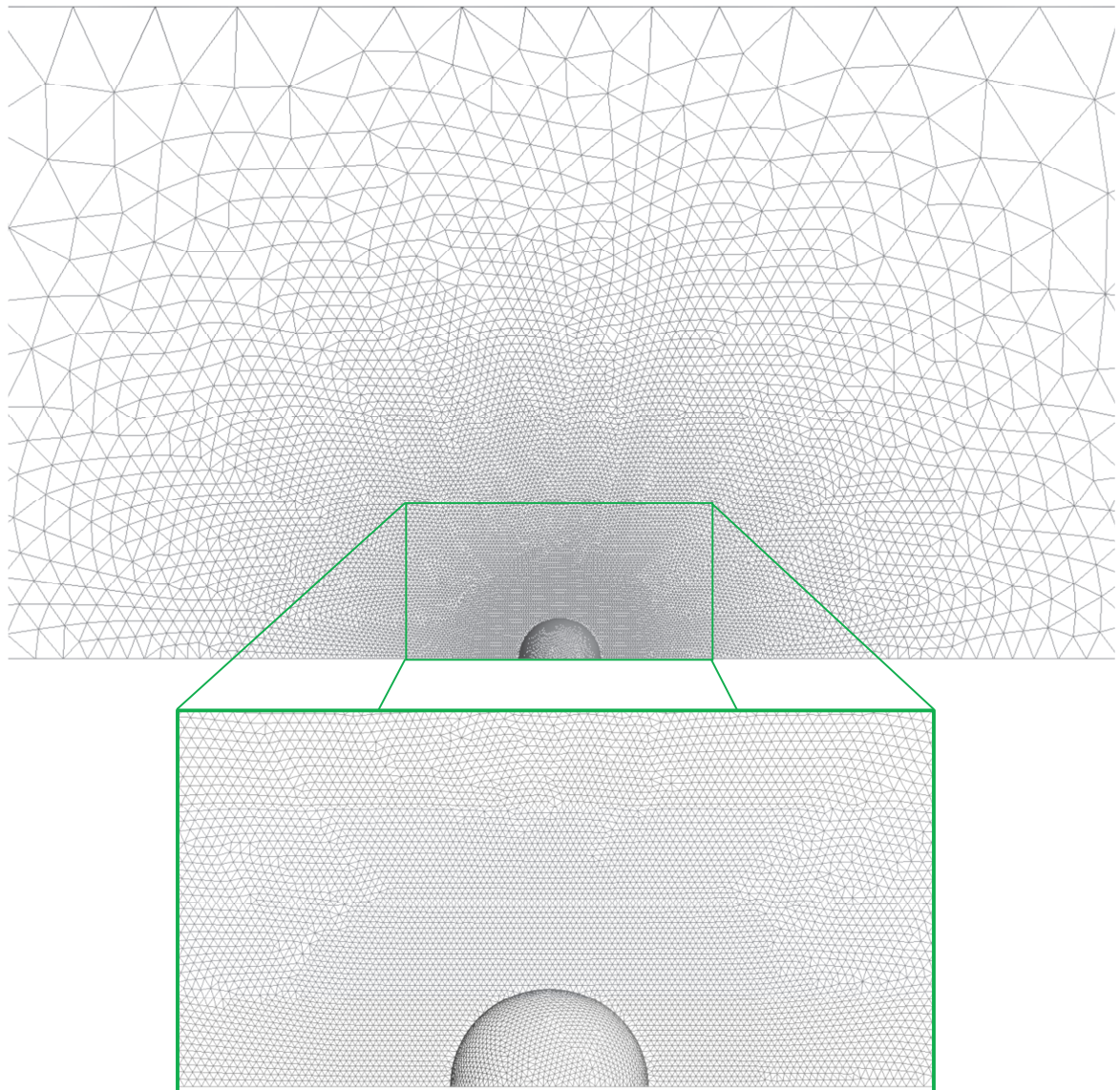


Figure 6.22: Mesh for Sphere (Mach 0.3).

The pressure and Mach distributions around the sphere are shown in Figure 6.23. The surface pressure distribution is shown in Figure 6.24 compared with potential theory with and without correction for compressibility. The Euler3D solution matches the incompressible theory very well. The compressible correction lowers the pressure by 5%, so the solution is 5% off at the mid-plane, decreasing to the stagnation points. The pressure distribution is asymmetric across the centerline because artificial dissipation causes the inviscid field to separate off of the wall near the aft stagnation point.

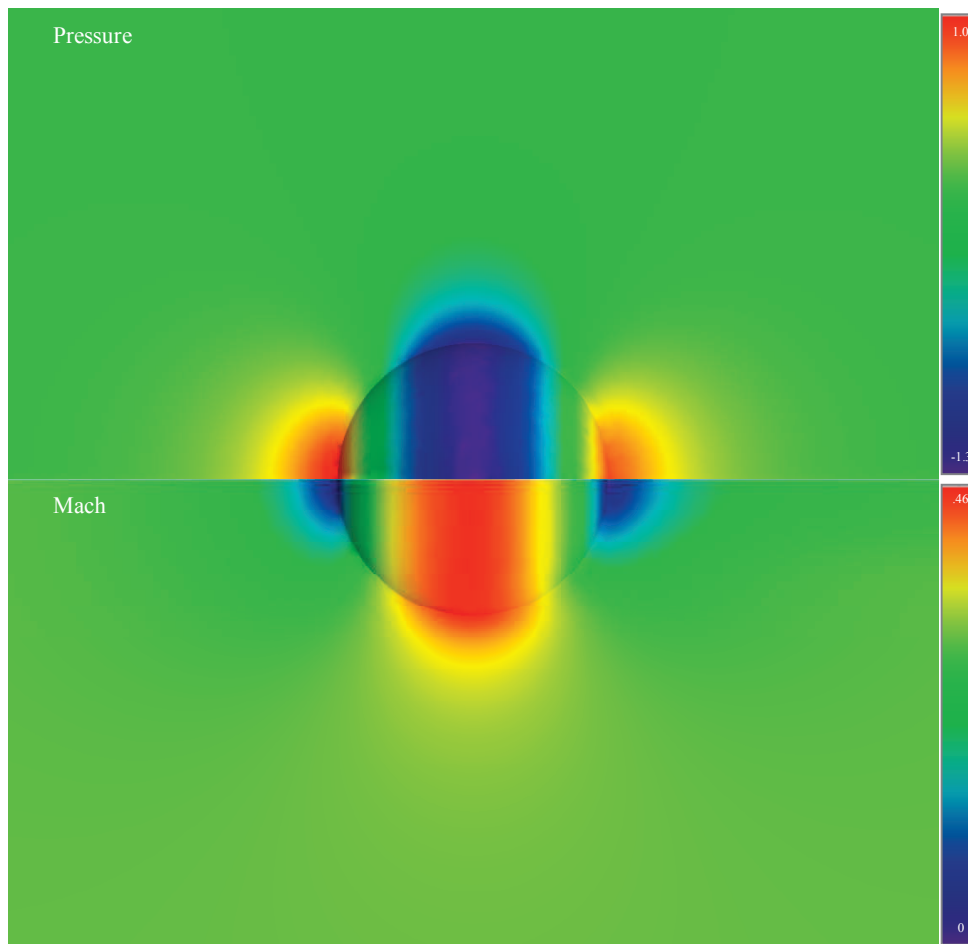


Figure 6.23: Pressure and Mach around Sphere (Mach 0.3).

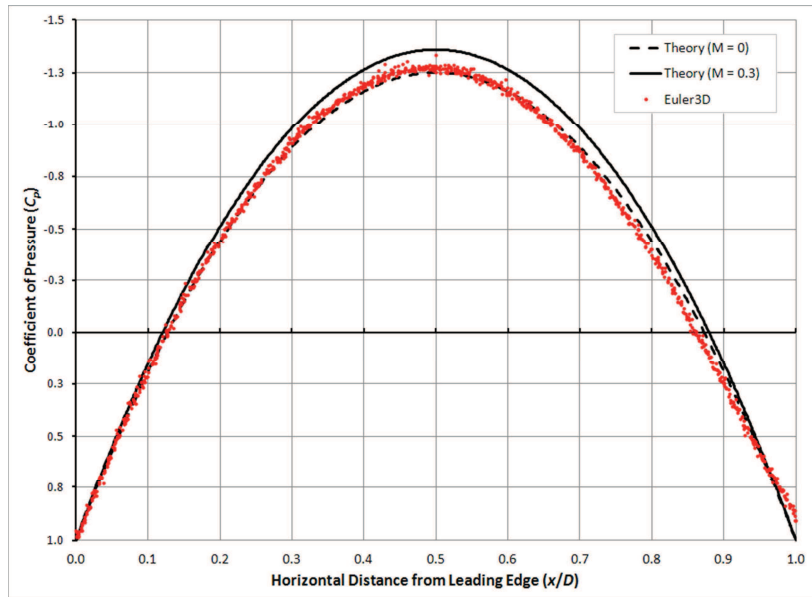


Figure 6.24: Surface Pressure over Sphere (Mach 0.3).

6.1.2 Transonic

Four airfoils are demonstrated at five transonic conditions and compared to experimental data. The airfoils were tested at freestream Mach numbers from 0.73 to 0.835 and angles of attack from -0.13 to 2.8 degrees. The NACA 0012 airfoil was tested in an asymmetric flow condition, and the RAE 2822 is repeated here in a lifting condition. The CAST 7 airfoil and NASA 10% supercritical airfoil add variety to the geometries tested.

6.1.2.1 NACA 0012 Airfoil (Mach 0.835, -0.13 deg)

The NACA 0012 airfoil is repeated here at transonic conditions. The symmetric airfoil was held at -0.13 degrees to create different distributions on the two surfaces. The shock lower surface should occur further aft on the airfoil because of the negative angle of attack. Figure 6.25 shows the location of both transonic shocks in pressure and Mach number. Figure 6.26

shows the mesh used to generate those solutions. The mesh was refined near the transonic shocks by iterating the CFD solution with tighter meshes.

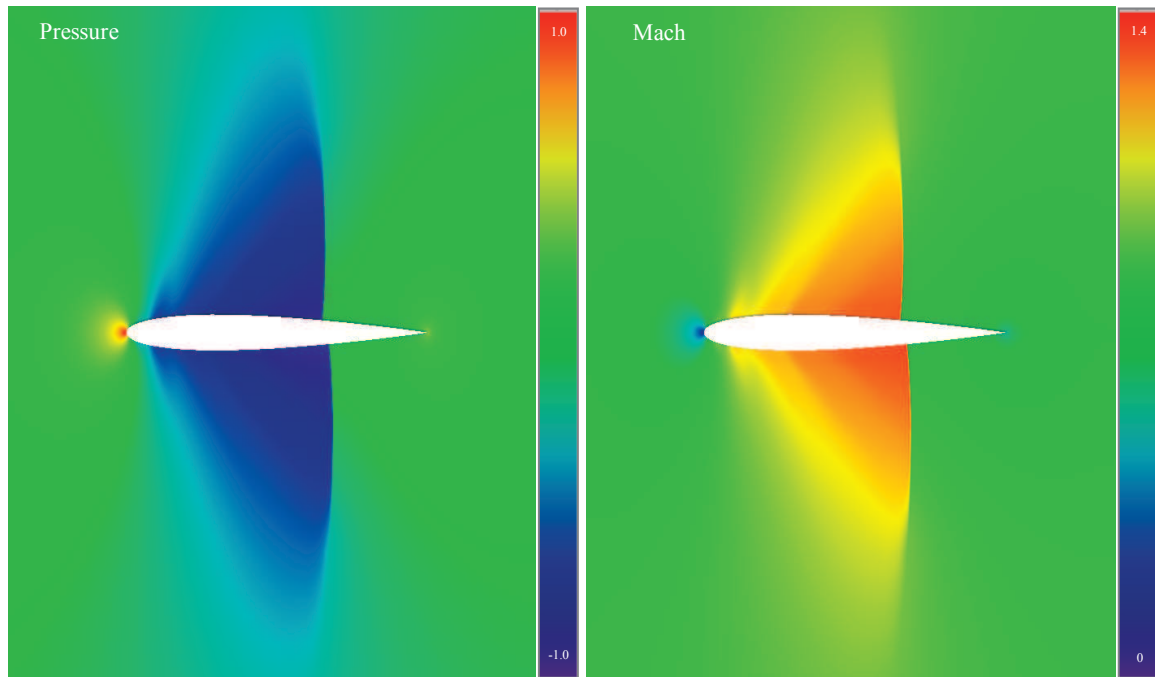


Figure 6.25: Pressure and Mach around NACA 0012 Airfoil (Mach 0.835, -0.13°).

Figure 6.27 shows the pressure distribution on both surfaces compared with data from Barche (1979). The CFD solution shows the location of both shocks to be further aft than seen in the experimental data. The location of the shocks is thought to be affected by the boundary layer thickness, not present in the Euler2D or Euler3D solution. (These solutions were not repeated in NS2D or NS3D because of time. This should be revisited in the future.) The CFD solutions in Figure 6.27 also shows oscillations near the leading edge that should be eliminated by a tighter leading edge mesh. Hassett (not yet published) has improved the leading edge accuracy but were unable to improve the shock locations in the inviscid solvers.

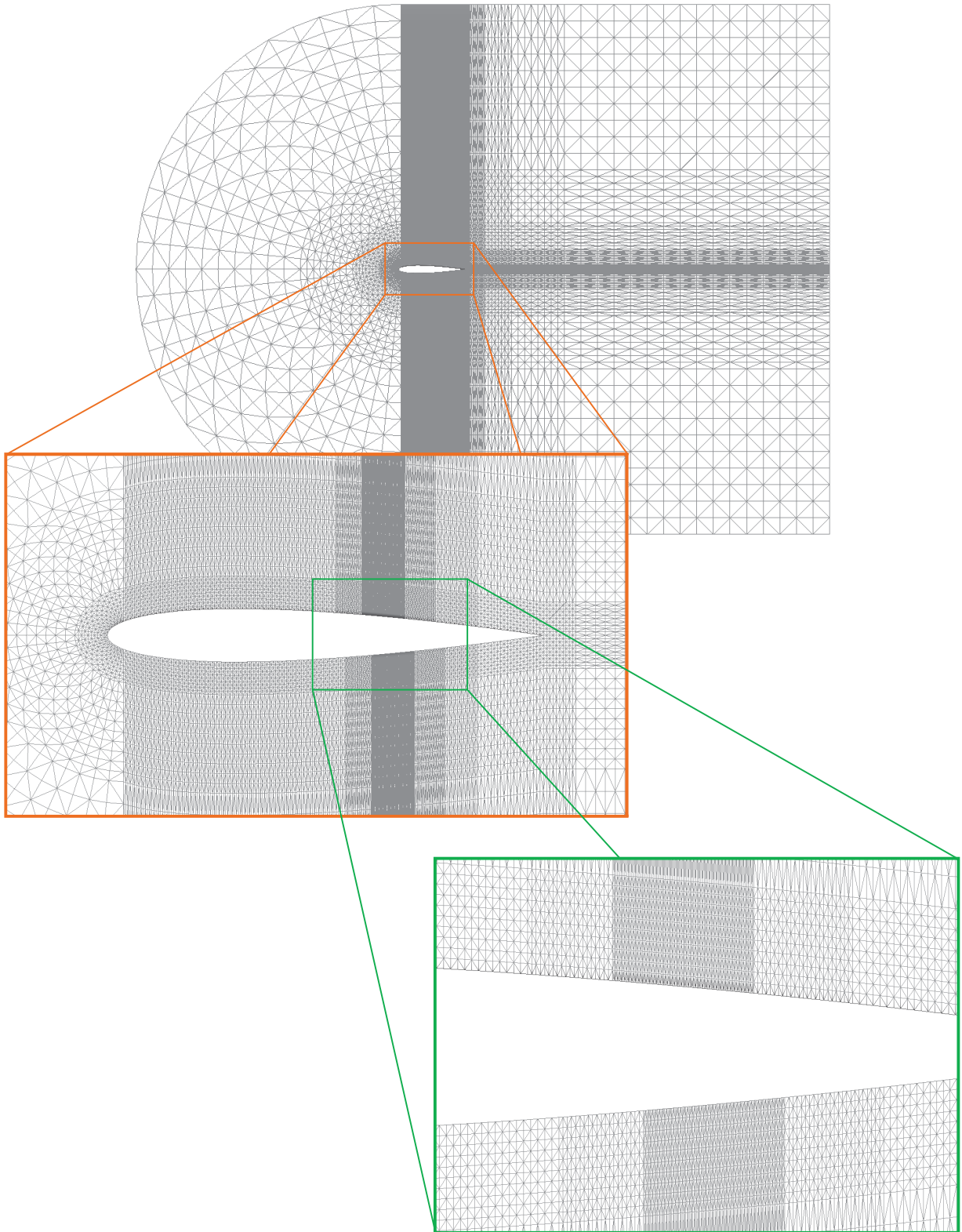


Figure 6.26: Mesh for NACA 0012 Airfoil (Mach 0.835, -0.13° AOA).

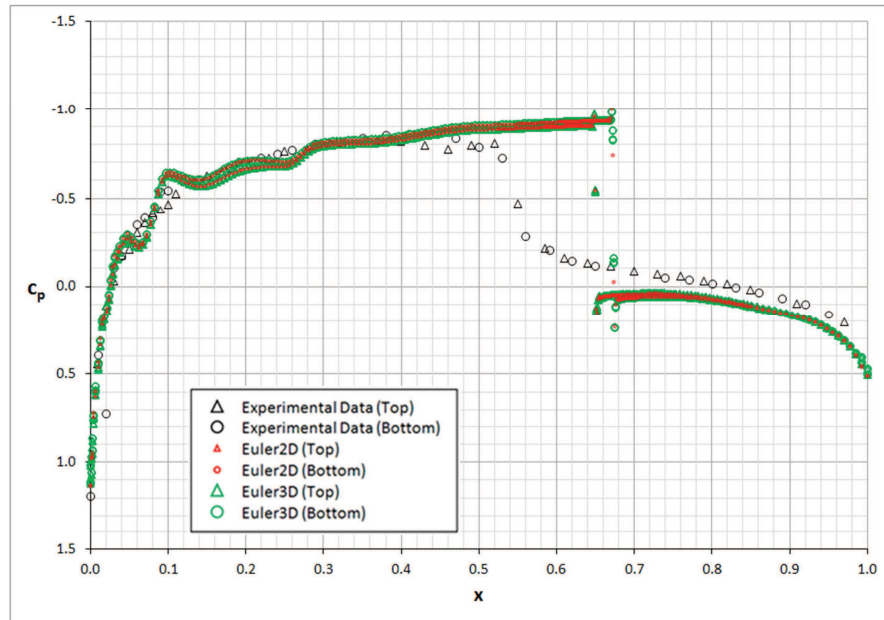


Figure 6.27: Surface Pressure over NACA 0012 Airfoil (Mach 0.835, -0.13° AOA).

6.1.2.2 RAE 2822 Airfoil (Mach 0.73, 2.8 deg)

The RAE 2822 airfoil was tested at a freestream Mach number of 0.73 and angle of attack of 2.8 degrees. The mesh used by Euler2D and Euler3D was produced in Pave2D and Pave3D, respectively (shown in

Figure 6.29). The pressure and Mach distributions are shown in Figure 6.28. The pressure distribution on the airfoil is shown in Figure 6.30 compared to data from AGARD (1988).

The experimental data shows a large pressure bubble near the leading edge. The CFD solutions do not predict as much suction near the leading edge. Both CFD solutions shock just aft of the experimental data. Euler2D predicts a very sharp shock, while the

experimental data (while sparse in data across the shock region) shows a longer shock region. The rest of the pressure distribution is very similar between the CFD and experimental data.

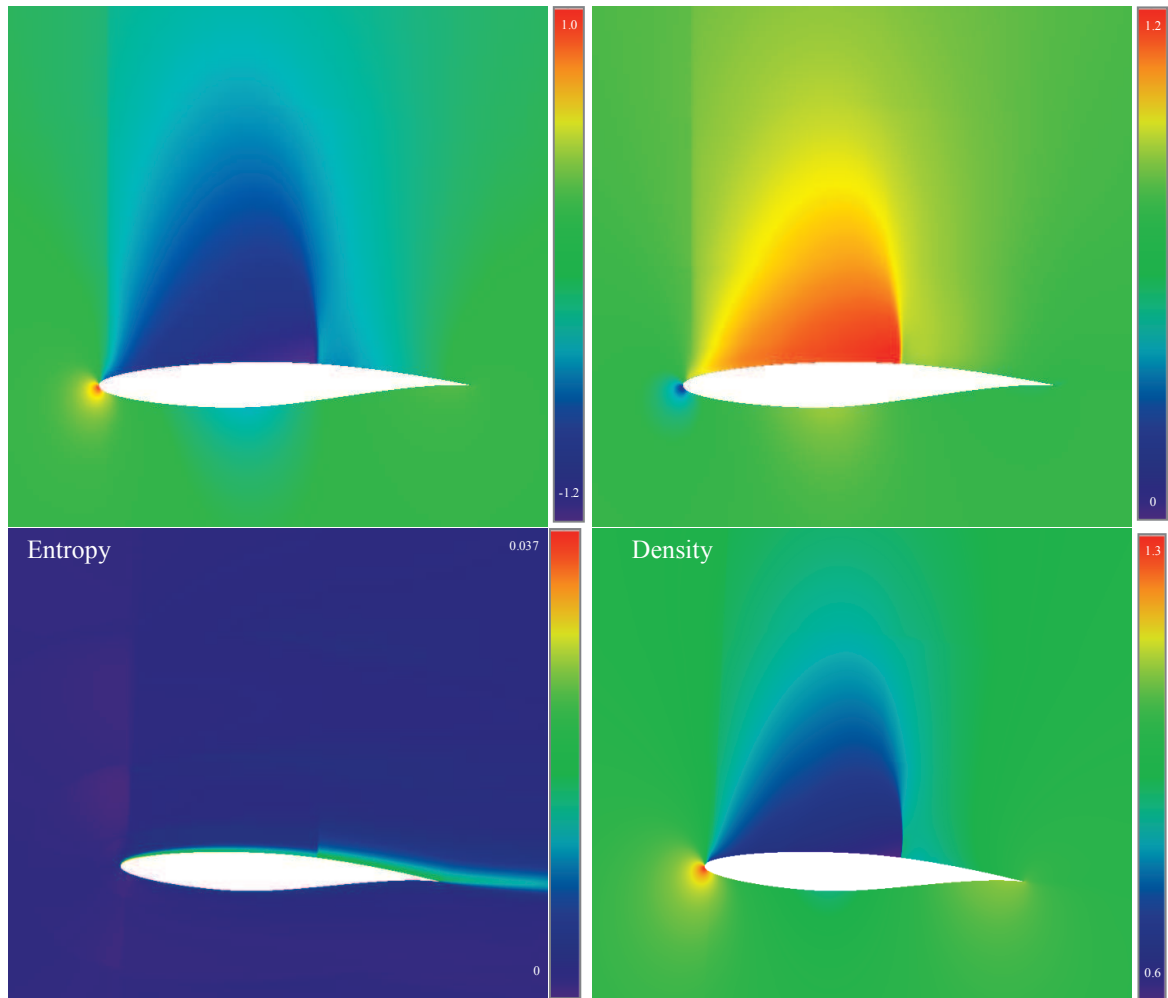


Figure 6.28: Pressure and Mach around RAE 2822 Airfoil (Mach 0.73, 2.8° AOA).

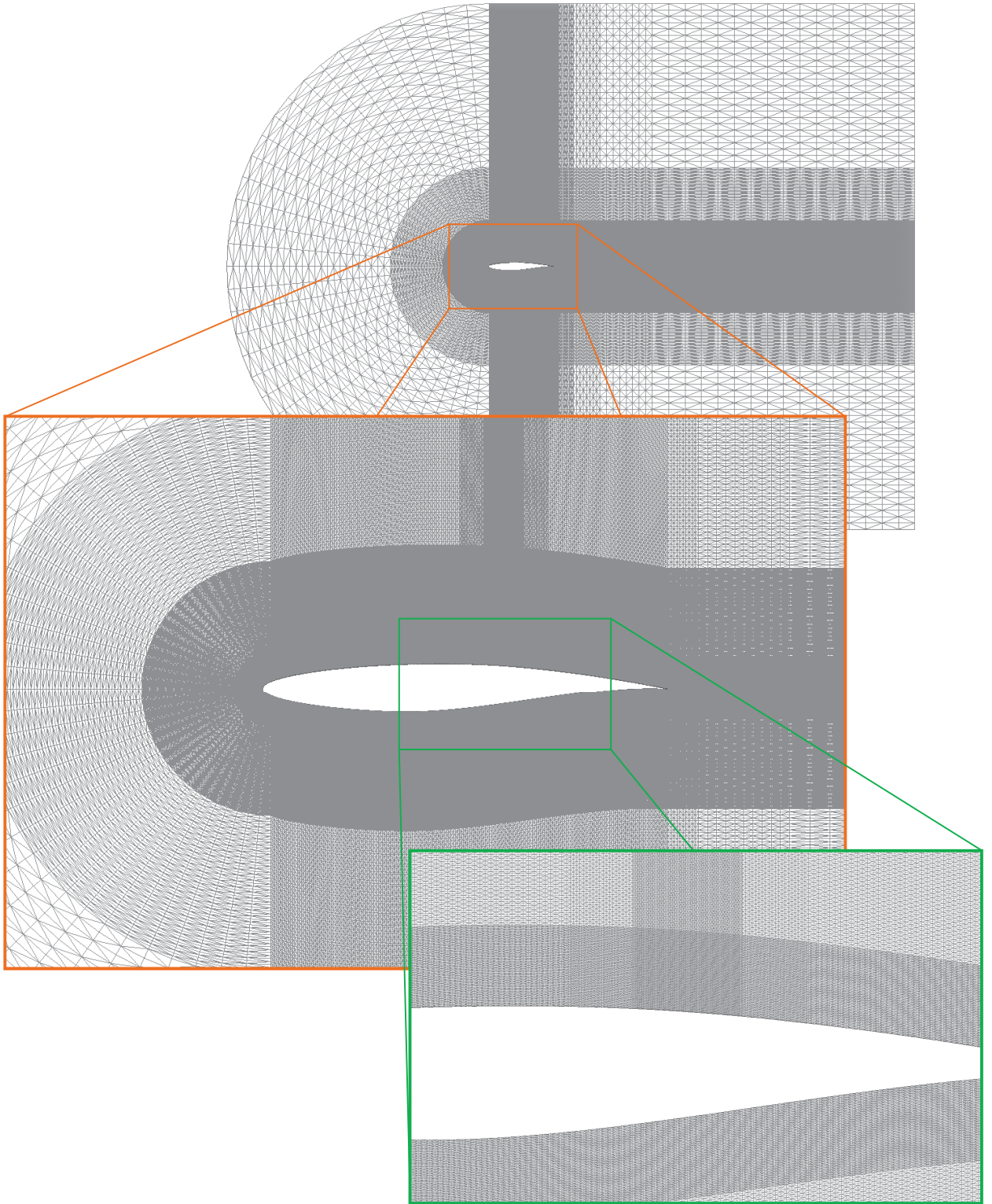


Figure 6.29: Mesh for RAE 2822 Airfoil (Mach 0.73, 2.8° AOA).

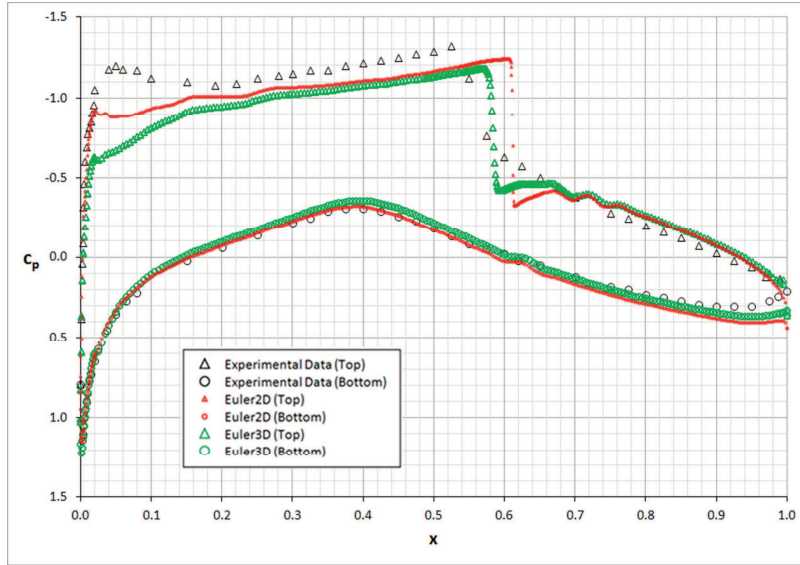


Figure 6.30: Surface Pressure over RAE 2822 Airfoil (Mach 0.73, 2.8° AOA).

6.1.2.3 CAST 7 Airfoil (Mach 0.765, 2.52 deg)

The CAST 7 airfoil was tested at two different transonic speeds and the same angle of attack: Mach 0.765 and Mach 0.785 at an angle of attack of 2.52 degrees. The two flight speeds create two different transonic regions over the top surface. The shock shifts aft for the higher speed case. The meshes used for the Mach 0.765 and 0.785 cases are shown in Figure 6.31 and Figure 6.34, respectively. The meshes were generated in Pave2D and Pave3D and refined through successive iterations through Euler2D. The shock refinements were placed using the first iteration, and the refined region was much broader than seen in the final meshes. The solution was updated, and the refined region was narrowed on the updated shock location. The mesh spacing was also decreased to sharpen the shock response.

Figure 6.32 and Figure 6.35 show the pressure distributions in the near field of each case. Figure 6.33 and Figure 6.36 compare the predicted pressure distributions on the airfoil at each condition to experimental data from Barche (1979). The solution from Euler2D and Euler3D shock 30 to 35% of the chord aft of the experimental data. The region between the CFD shock location and that measured in experiments represents the largest differences seen. The remaining pressure distribution matches the experimental data very well. Possible explanations include boundary layer thickness (not modeled in the inviscid codes), mesh refinement upstream of the shock region, or experimental uncertainties. Further tests are required to determine the cause of the discrepancies.

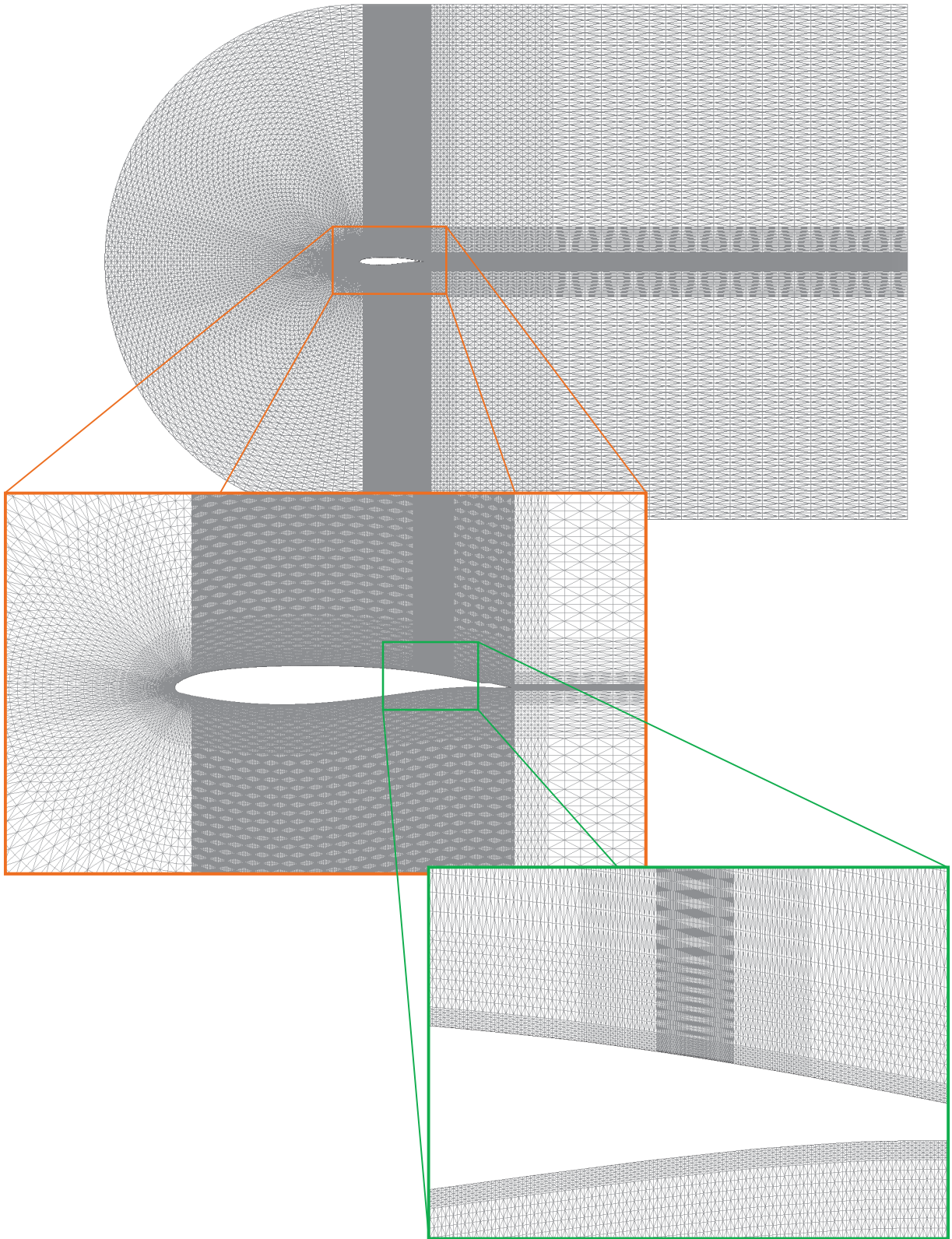


Figure 6.31: Cast 7 Airfoil Mesh (Mach 0.765, 2.52° AOA).

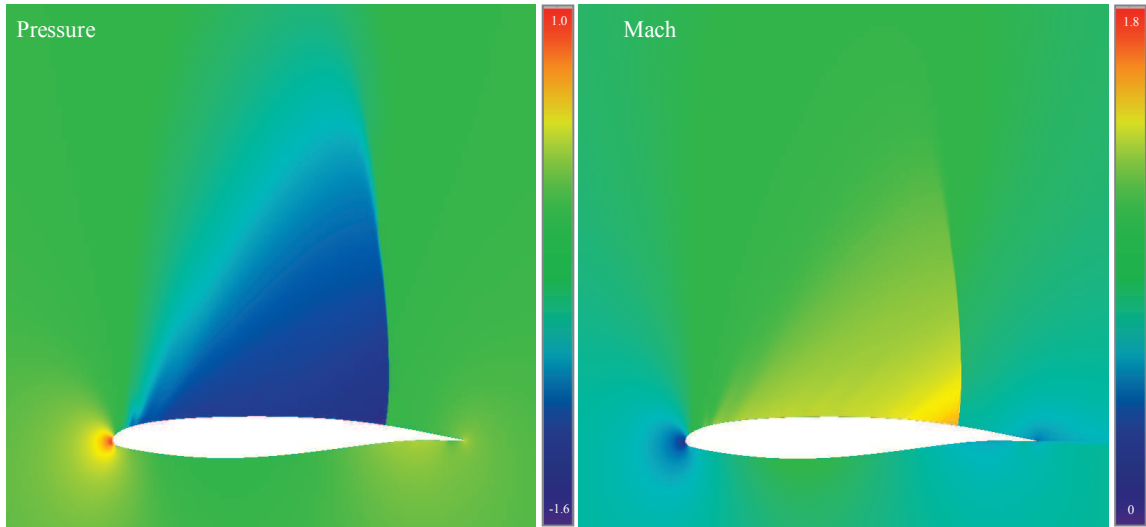


Figure 6.32: Pressure and Mach Distrib. around CAST 7 (Mach 0.765, 2.52° AOA).

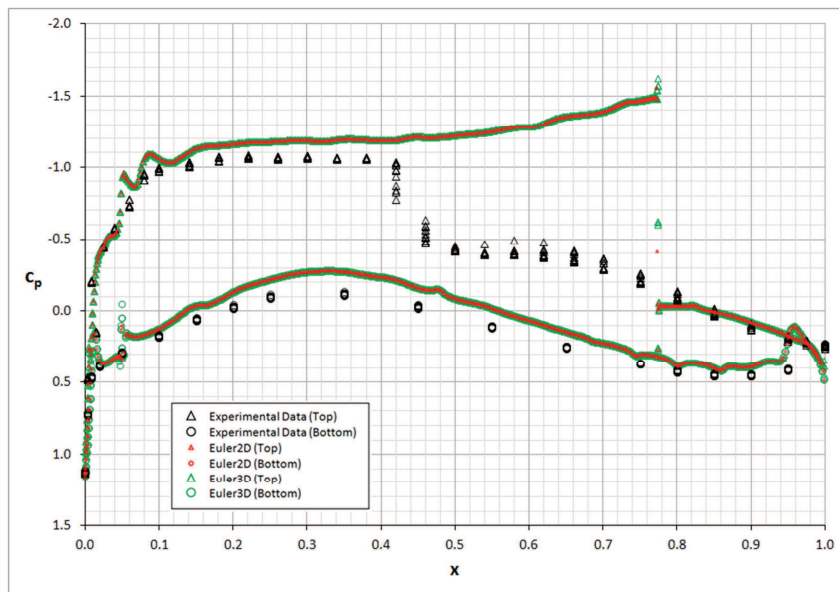


Figure 6.33: Surface Pressure Distrib. over CAST 7 Airfoil (Mach 0.765, 2.52° AOA).

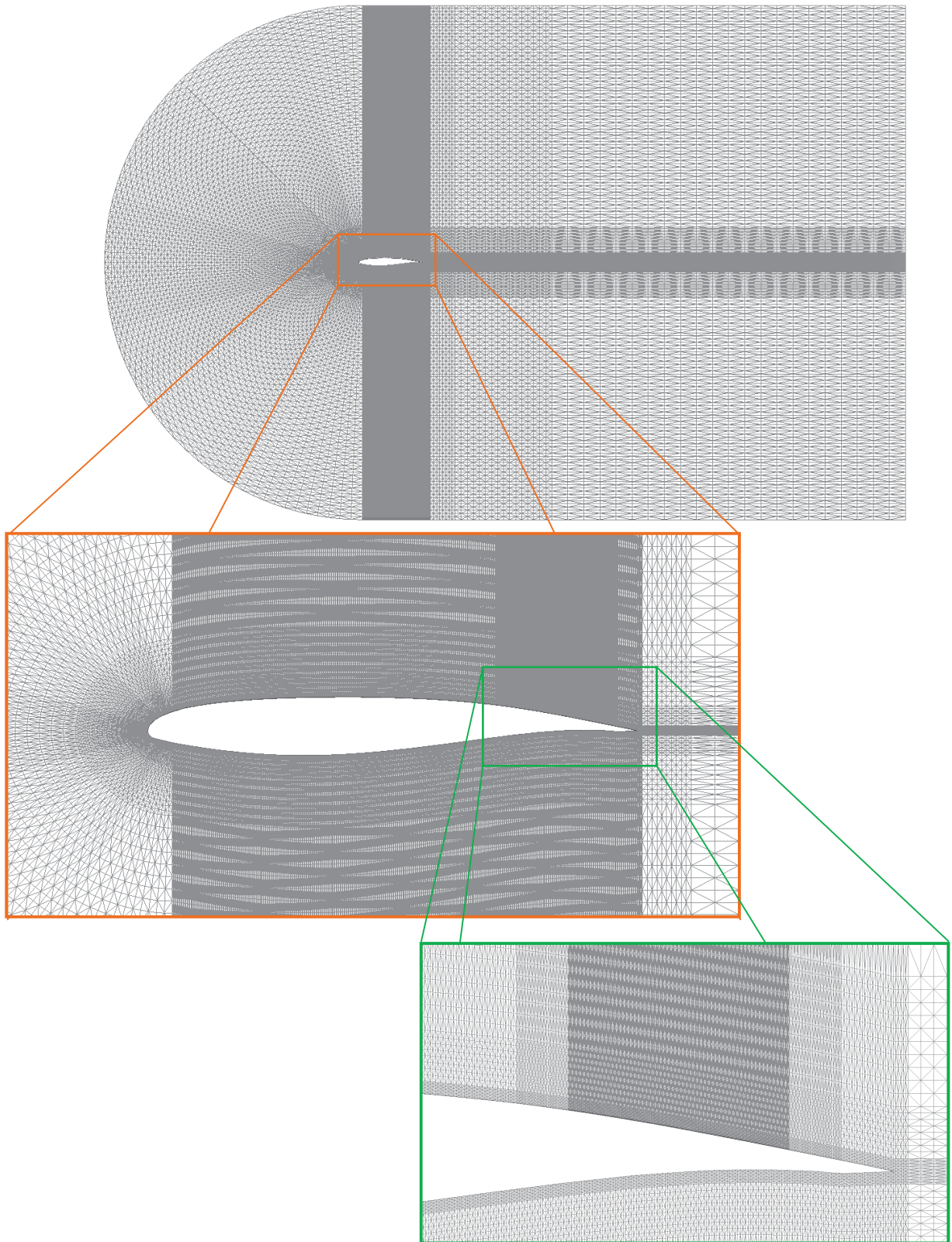


Figure 6.34: Cast 7 Airfoil Mesh (Mach 0.785, 2.52° AOA).

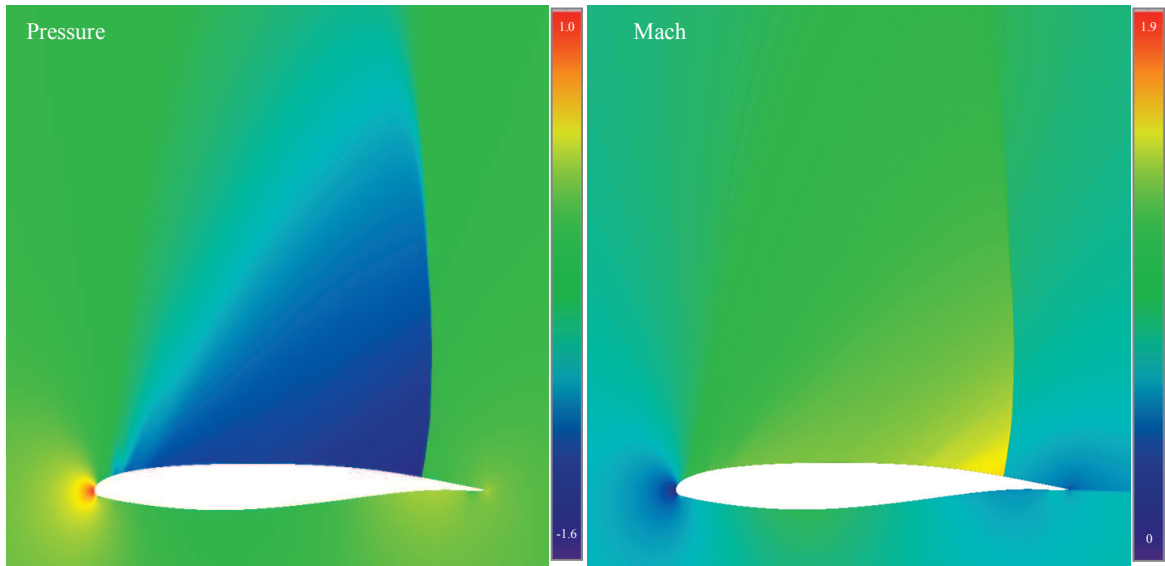


Figure 6.35: Pressure and Mach Distributions around CAST 7 (Mach 0.785, 2.52°).

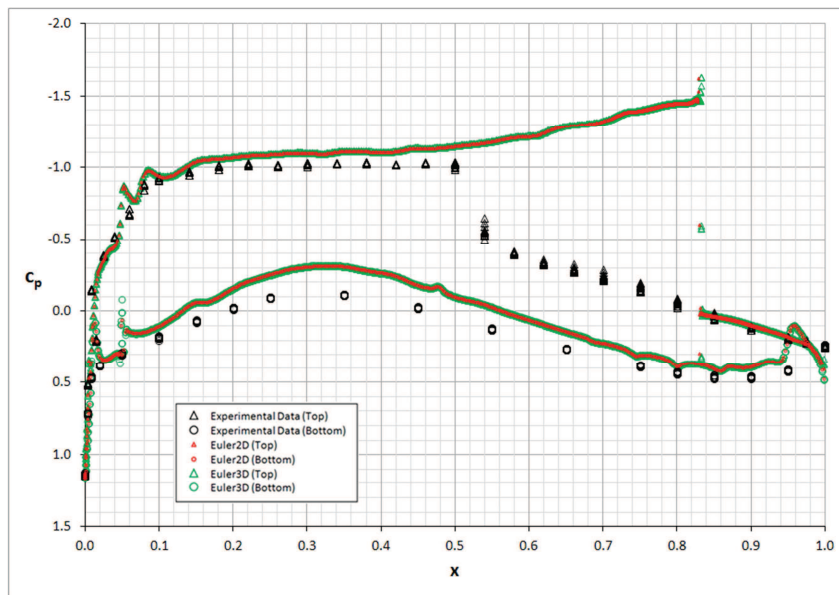


Figure 6.36: Surface Pressure Distribution over CAST 7 Airfoil (Mach 0.785, 2.52°).

6.1.2.4 NASA 10% Supersonic Airfoil (Mach 0.79, 2 deg)

The NASA 10% supercritical airfoil is a strongly cusped airfoil with a large region with nearly constant thickness. The NASA airfoil was tested at Mach 0.79 and compared to experimental data from Barche (1979). The mesh used for this comparison is shown in Figure 6.38. The pressure and Mach distributions are shown in Figure 6.37. The CFD solution predicts the shock very near the trailing edge. The experimental data predicts a shock near 75% of the chord. Figure 6.39 shows a comparison between the CFD and experimental distributions.

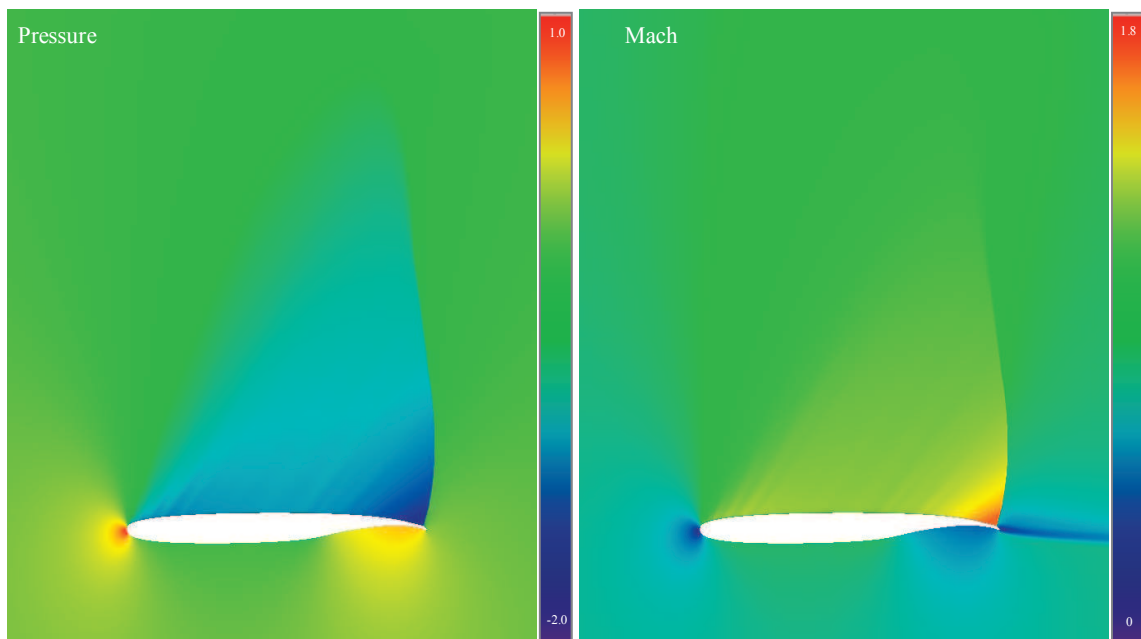


Figure 6.37: Pressure and Mach around NASA 10% Supercritical (Mach 0.79, 2°).

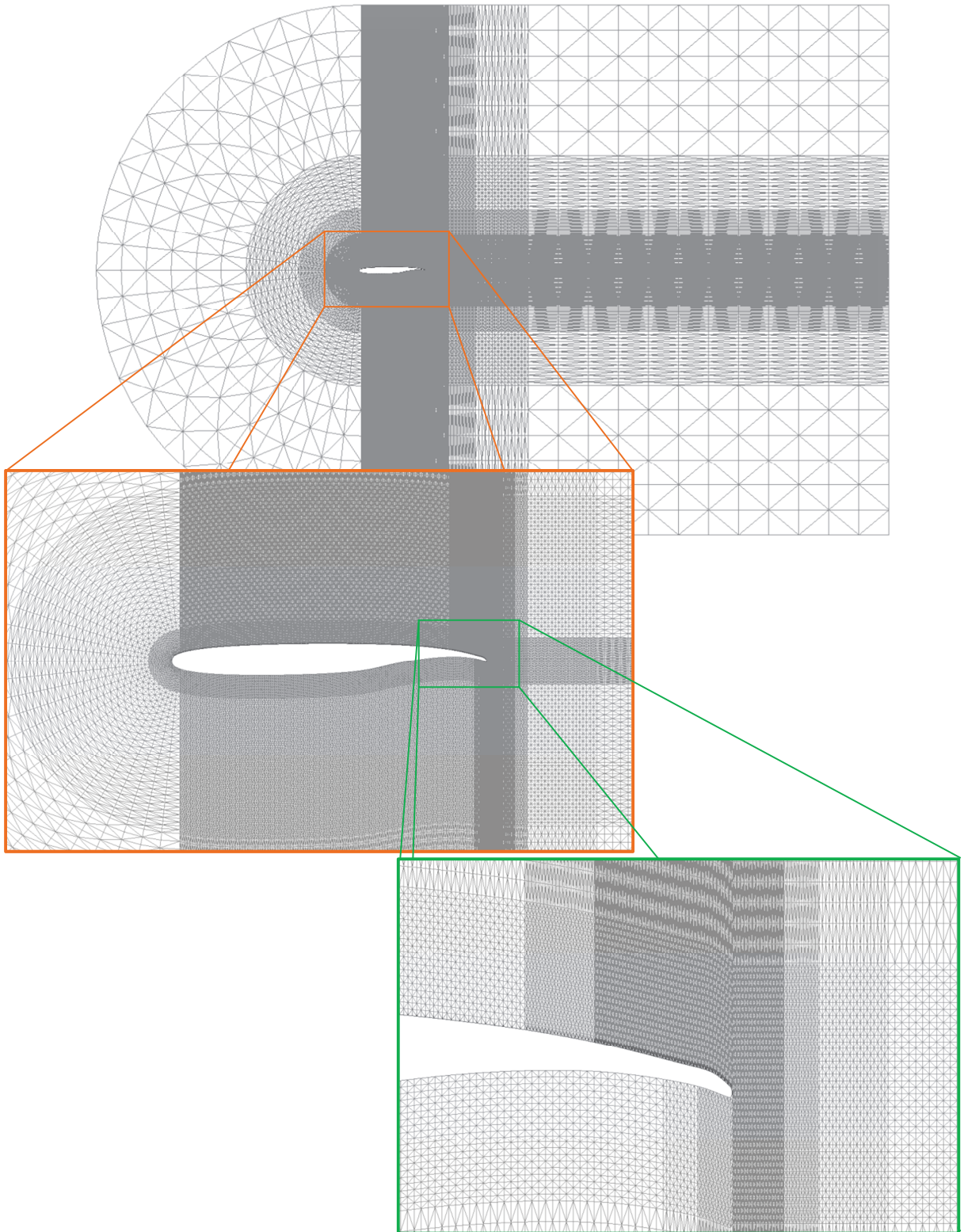


Figure 6.38: Mesh for NASA 10% Supercritical Airfoil (Mach 0.79, 2° AOA).

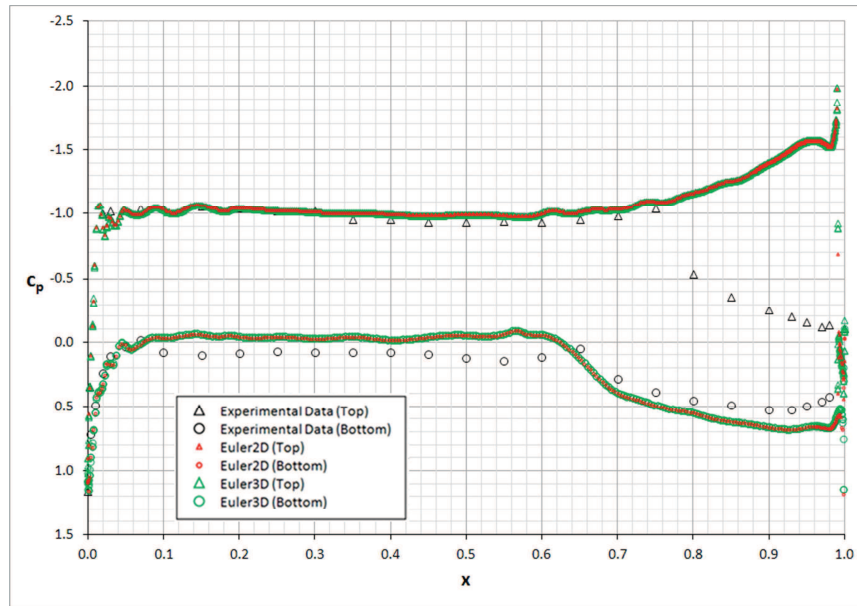


Figure 6.39: Surface Pressure over NASA 10% Supercritical Airfoil (Mach 0.79, 2°).

Euler2D has matched the distribution on many of the transonic cases except between the two shock locations. Hassett (not yet published) tested the uncertainty in angle of attack and has ruled this out as an explanation. This idea is still explored here to alert the reader to the effects of angle of attack on shock location and the rest of the pressure distribution.

Figure 6.40 and Figure 6.41 show pressure distributions created by varying the angle of attack. Both figures show the advancement of the transonic region across the top surface of the airfoil as the angle grows. The pressure on the bottom surface of the airfoil does not change appreciably, and the pressure on the top surface changes due to the transonic shock and its location alone. The pressure distribution varies by approximately 5% upstream and downstream of the shock region.

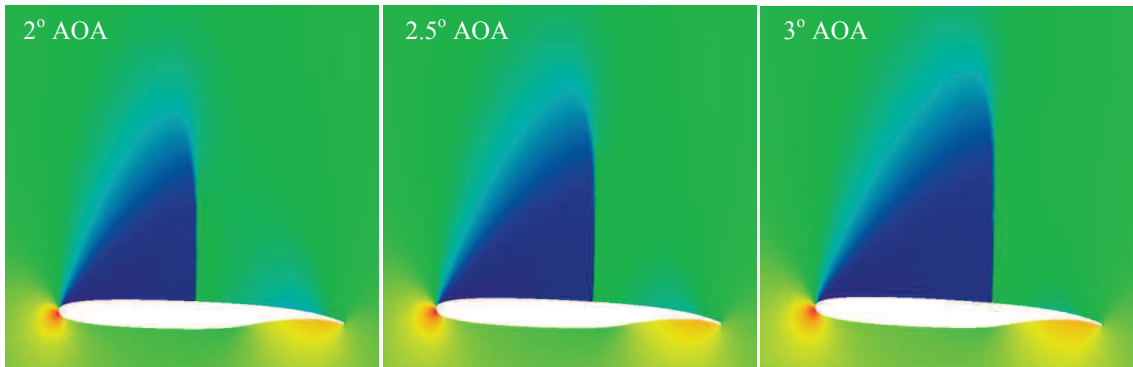


Figure 6.40: Pressure Distribution over NASA 10% Supercritical Airfoil (Mach 0.7).

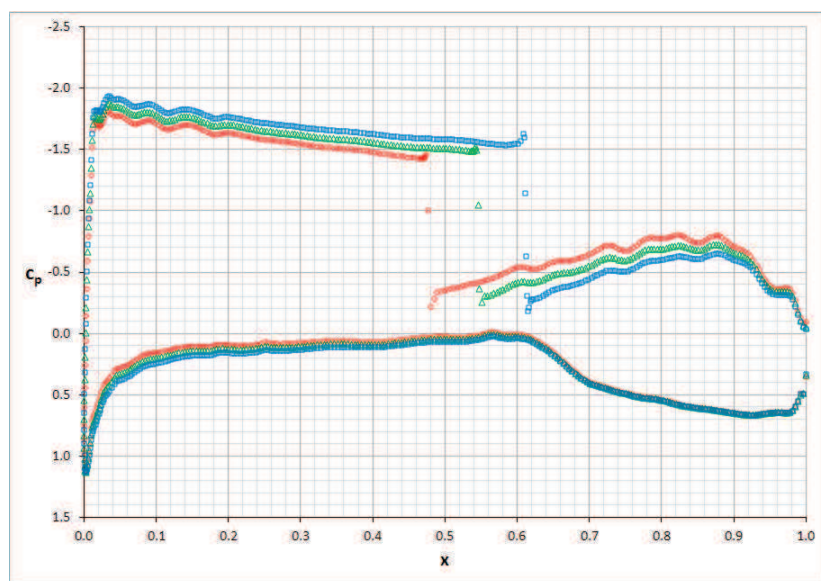


Figure 6.41: Pressure Distribution over NASA 10% Supercritical Airfoil (Mach 0.7)

6.1.3 Supersonic

The supersonic regime is represented here by four test cases with various conditions on each of those cases. A shock tube was modeled with a three-part initial condition. Two traveling waves of equal magnitude move across the shock tube in opposite directions. Pressure, density, and velocity distributions are given as time slices. All cases tested matched shock expansion theory with less than 0.2% error far downstream. Tighter meshes improved the transition through shocks and expansions. A series of compression and expansion corners

were investigated meshing guidelines for oblique shock waves and expansion fans and also worked as verification cases. When the appropriate mesh spacing is used, the error across the shock or expansion is reduced. Finally, a double wedge airfoil was tested at an angle of attack, creating four unique shocks and two expansions.

6.1.3.1 Double Shock Tube

A shock tube with two membrane partitions was modeled in Euler2D. The shock tube was divided into three regions of equal length L . The pressure and density in the outer regions was set to unity, and the pressure and density in the inner region was set to one tenth. The velocity everywhere is zero. These properties are the initial conditions for the solution to model the membrane being broken at $t = 0$. Pave2D was used to mesh the domain into equal steps in the horizontal and vertical dimensions: $\Delta x = 0.002 L$ and $\Delta y = 0.005 L$ (300,000 elements total). The boundaries of the domain were all modeled as solid walls. The three full-page figures that follow illustrate the progression of the solution through 45,000 steps at a time step of $\Delta t = 4.75 \times 10^{-5}$. A slice is shown every 3000 steps.

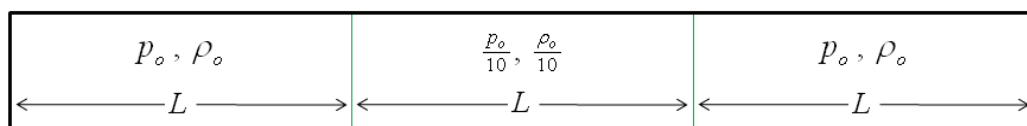


Figure 6.42: Shock Tube Geometry

Figure 6.43 below shows 16 pressure slices taken at regular intervals. You can see the compression waves (crisp normal shocks, white lines) moving inward while the expansion waves (fuzzy transitions bound by dashed white lines) move out toward the ends. The compression waves cross in the center and keep moving toward the opposing walls. At the

instance the two compression waves cross, they cancel each other making a homogeneous center. But the waves continue on, releasing double-traveling shock waves. The trailing wave in each doublet is faster, overtaking its counterpart. The compression waves coalesce back into one traveling wave on each half of the domain. The slices show the compression waves until just after they reflect off of the end plates. As the compression waves pass through the expansion region (bound by dashed lines), the compression waves slow down because of the change in properties.

Figure 6.44 shows 16 density slices at the same intervals. The progression of compression and expansion waves in density is very similar to those in pressure. The trailing wave is faster than the leading. The pressure and density waves coalesce at the same time. The density solution also leaves an artifact within the inner region. The artifact is created when the trailing waves contact the leading waves from the opposite membrane. The result is a standing wave artifact, which is thought to be numerical in nature, not real to the physics.

Figure 6.45 shows 16 slices of the velocity magnitude, non-dimensionalized to the initial compression wave speed. The pattern of compression and expansion waves is very similar to pressure and density. Trailing compression waves are created along the centerline and coalesce with their leading counterparts. The velocity between the compression and expansion waves is the highest and nearly constant; the velocity is lower and changes continuously across the expansion region (bounded by dashed lines). The other regions have no velocity until they are visited by one of the shocks or expansion fronts.

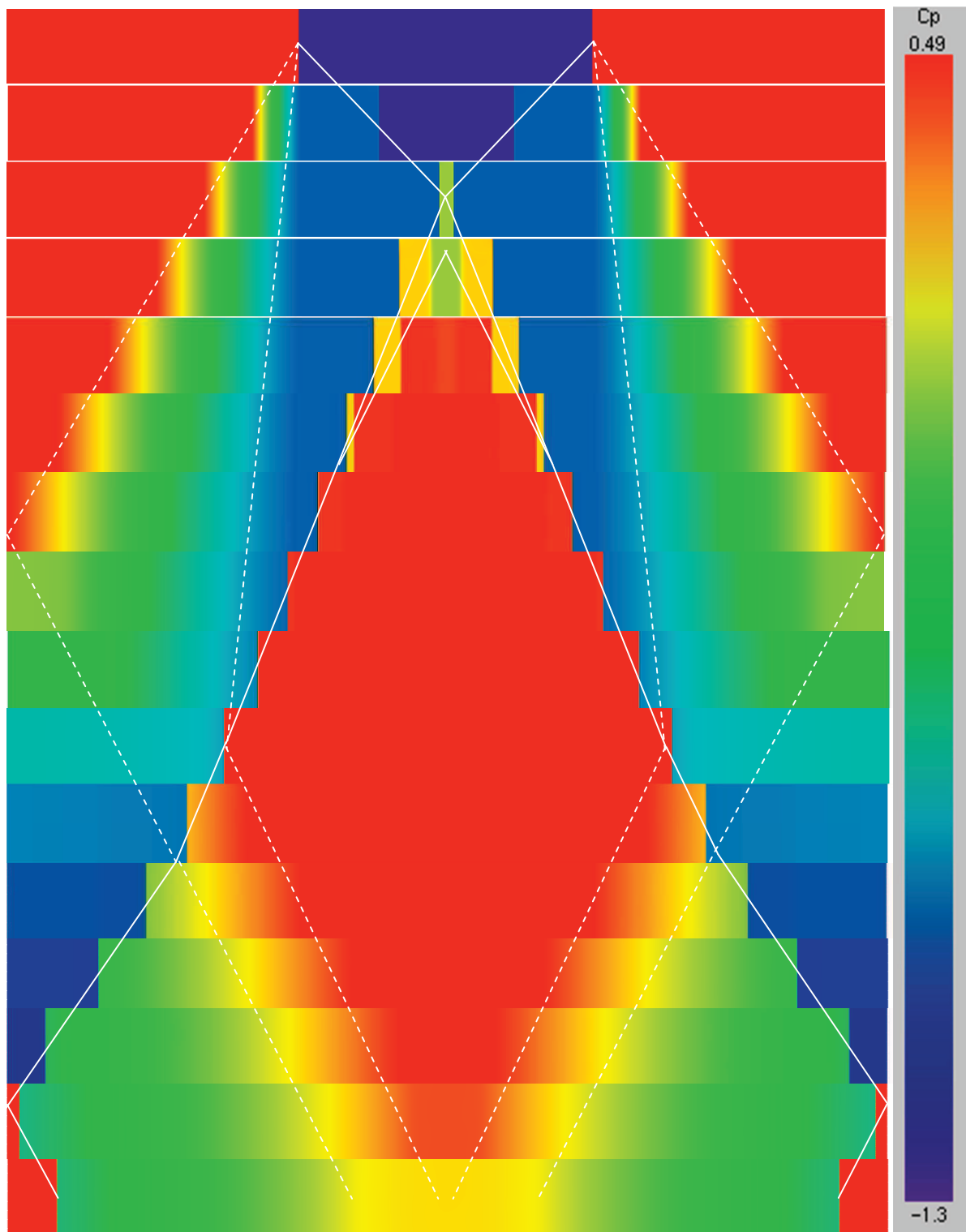


Figure 6.43: Pressure Slices through Shock Tube at Regular Intervals.

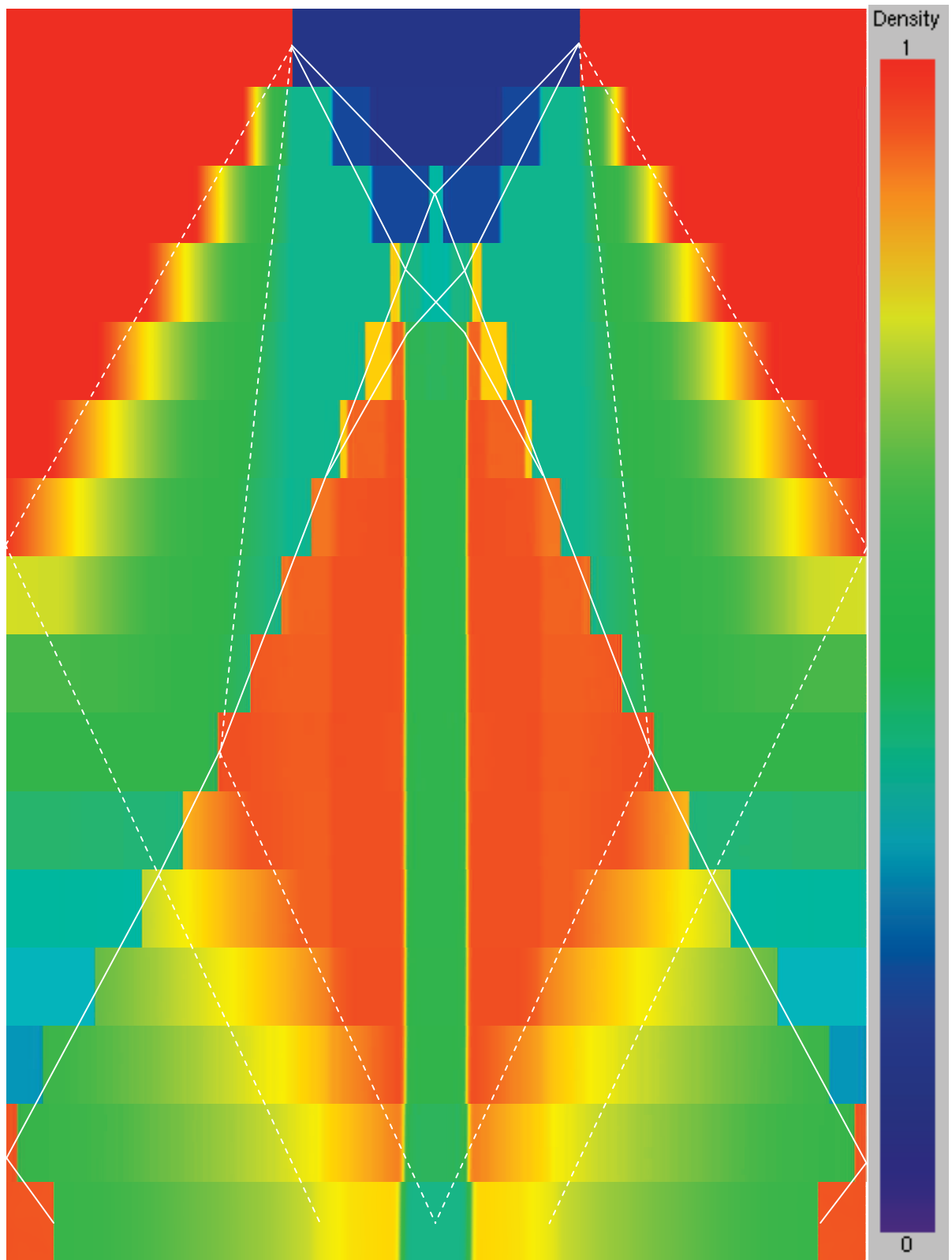


Figure 6.44: Density Slices through Shock Tube at Regular Intervals

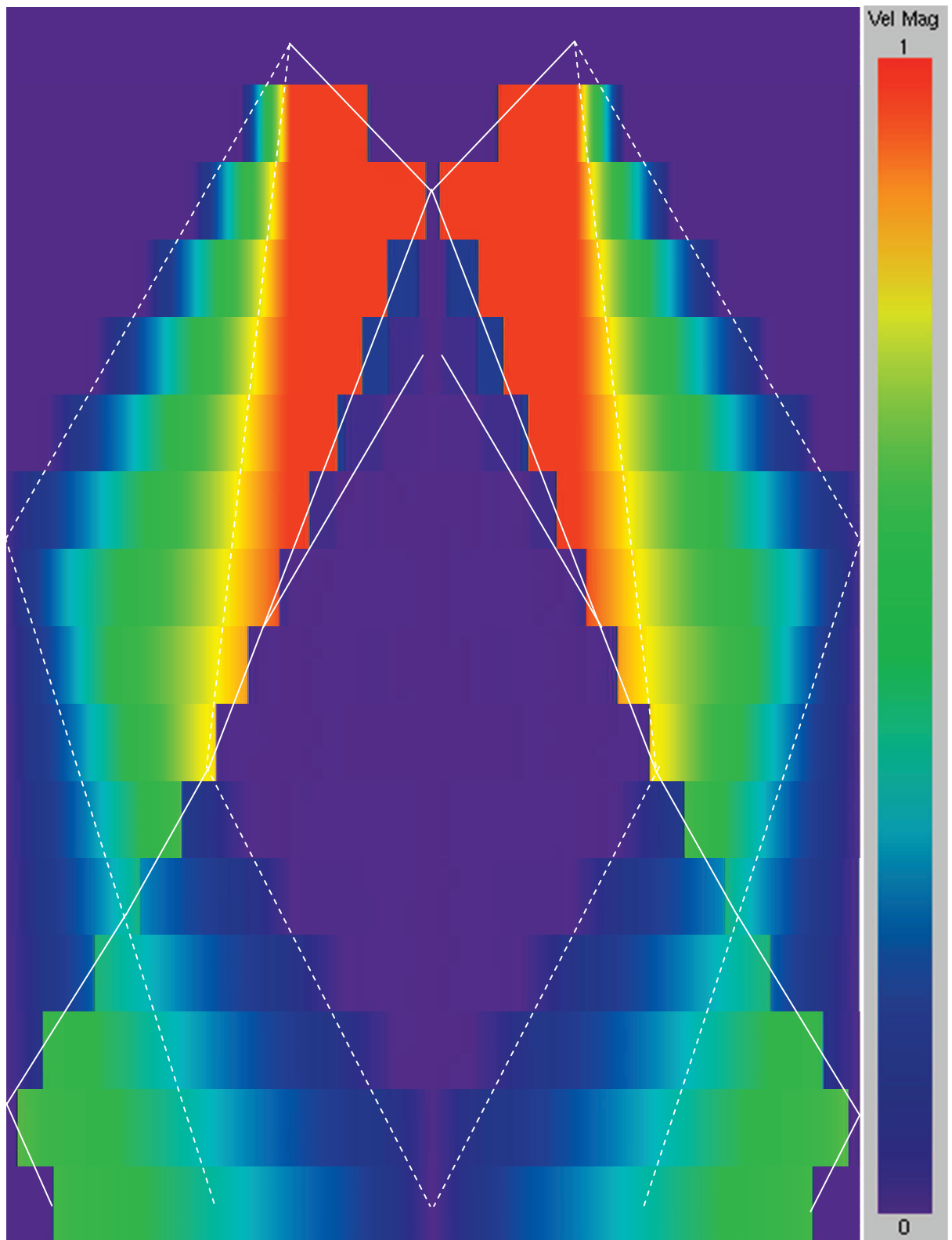


Figure 6.45: Velocity Slices through Shock Tube at Regular Intervals.

6.1.3.2 Supersonic Compression Corner

When supersonic flow passes a compression corner, shown in Figure 6.46, an oblique shock wave forms that slows down the flow behind the wave. The slower flow also turns to align with the downstream geometry. Figure 6.46 shows such a shock wave and binary solution from upstream to downstream. Theoretically, the transition from upstream to downstream occurs over an infinitesimally thin distance. Realistically, losses due to viscous dissipation and heating increase this distance to the cover of millimeters. In CFD models, the viscous and heating effects are often second to the artificial dissipation required to capture the shock discontinuity. The effects of artificial dissipation are explored here.

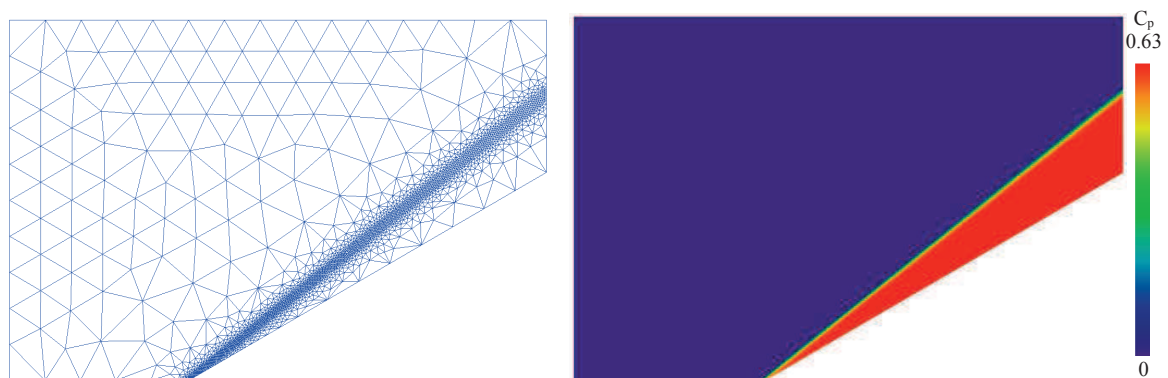


Figure 6.46: Mesh (left) and Pressure (right) for a 30° Compression Corner at $M = 10$.

Figure 6.47 shows the pressure distribution along the wall of the 30° compression corner (shown in Figure 6.46). This solution is typical for a compression corner. The region upstream of the corner is constant at freestream conditions. At the corner, possibly one element upstream, the properties change quickly. Downstream of the corner, the artificial dissipation smoothes out the sharp transition. Hopefully, any violation of the Second Law is

avoided, and the flow transitions quickly to the downstream “jump-condition” properties.

The “jump-condition” represents the analytical solution to the Euler equations for a particular shock case (John, 1984). The numerical solution to the Euler equations also arrives at this same solution, far downstream from the shock. (In all cases, the solution matches within 0.1% error, far downstream.) The word “far” is used because artificial dissipation smears the solution near the shock. If this region is appropriately meshed and a low dissipation is used, this distance can be minimized and the solution seems to “jump” across the shock.

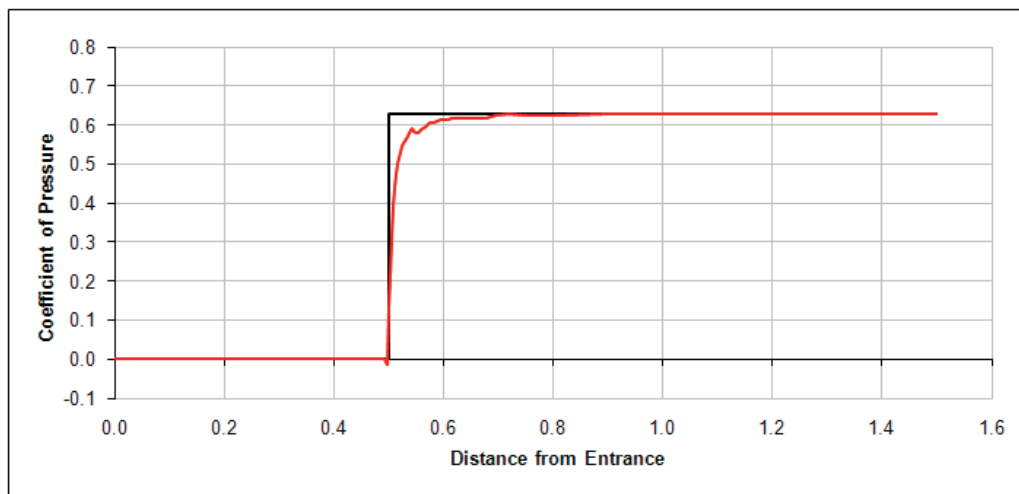


Figure 6.47: Pressure along Wall of 30° Compression Corner, $M = 10$, $\Delta x = 0.032 L$.

Figure 6.48 shows the smeared region just downstream of the shock wave for several mesh spacings ranging from 0.0004 to 0.1 times the length of the downstream plate, which has a length of unity. The curves on the left of Figure 6.48 can be collapsed into the curves on the right of Figure 6.48 by scaling the distance along the plate by the inverse of the mesh spacing. In other words, this artificial dissipation model requires a set number of elements to “smear” the solution from upstream to downstream “jump” conditions. The number of

elements is changed by the upstream Mach number, corner angle (shock strength), and amount of artificial dissipation. These effects are further explored.

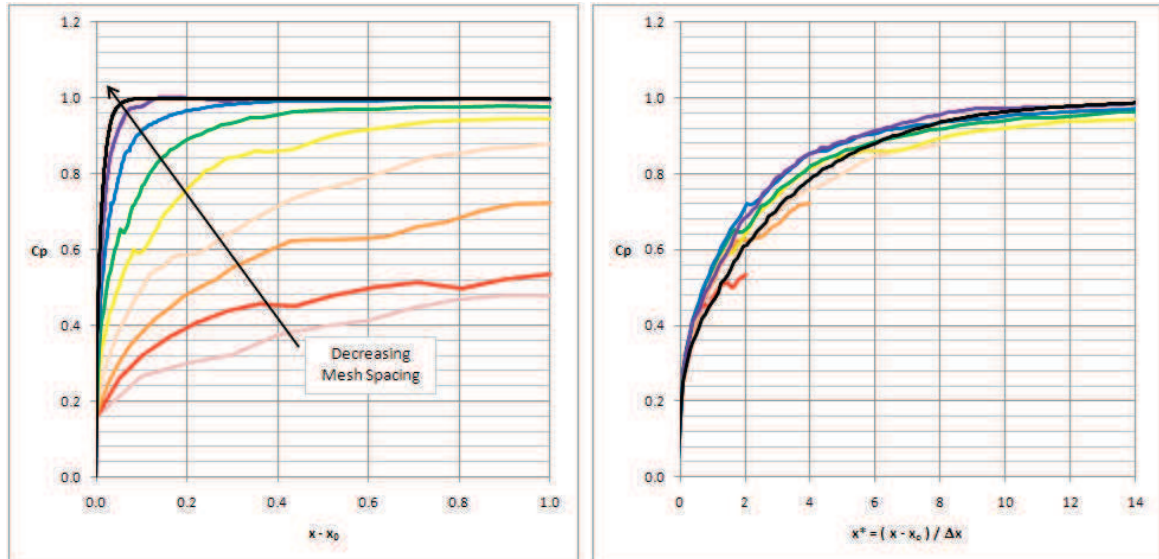


Figure 6.48: Uncollapsed (Left) and Collapsed (Right) Pressure Profiles Downstream of a 15° Compression Corner ($M = 10$) versus Mesh Spacing.

Figure 6.49 shows region just downstream of the shock for various amounts of dissipation ($0.8 \leq diss \leq 2$). More dissipation smears the solution further downstream, no matter how many elements are used, so less dissipation is more desirable. This trend, of course, has a lower limit, where the solution becomes unstable. The curves on the left of Figure 6.49 for various dissipation coefficients are collapsed into the curve on the right of Figure 6.49 by dividing by the dissipation scalar $diss$. This shows the number of elements can be scaled by the amount of dissipation used.

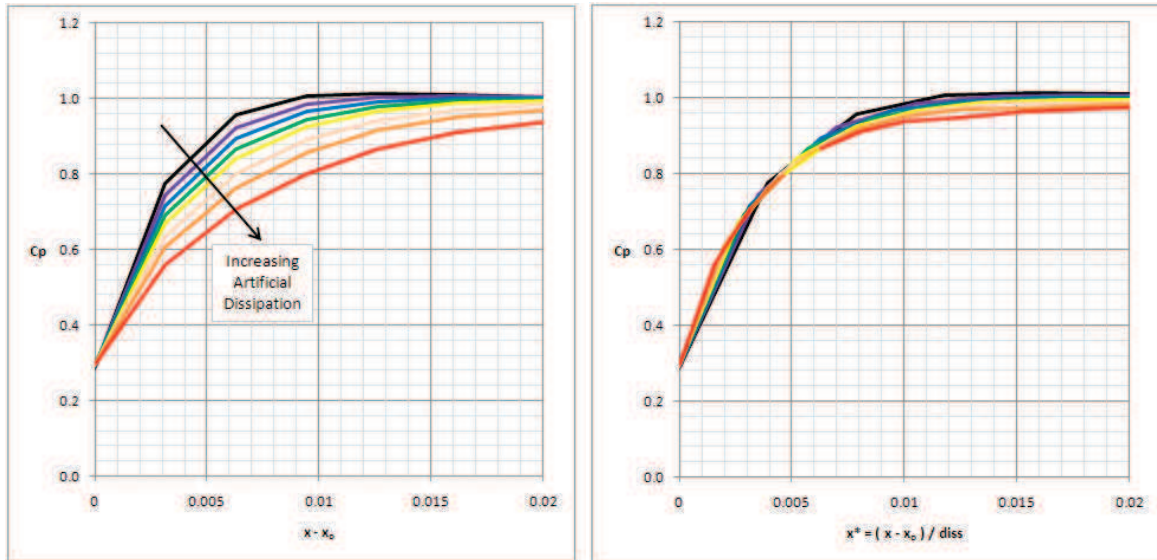


Figure 6.49: Uncollapsed (Left) and Collapsed (Right) Pressure Profiles Downstream of a 15° Compression Corner (M = 2) versus Artificial Dissipation Scalar.

Meshes from 0.004 to 0.1 times the downstream plate length were tested in the region of the shock wave for compression corners ranging from 5 to 40 degrees at Mach numbers of 2, 3, 5, and 10. The results were collapsed using the method shown in Figure 6.48 and Figure 6.49. The results ($diss = 1.0$) are shown in Table 6.1 and graphically in Figure 6.50. Figure 6.50 plots the number of elements versus the measure of the corner angle over the maximum angle δ_{max} , or angle at which the shock detaches into a bow shock. The table and plots show that the number of elements is most highly affected by the upstream Mach number and then by the angle turner (or strength of the compression). The number of elements increases with Mach number, which is intuitive. The number of elements increases for smaller angles, which is counter-intuitive.

Table 6.1: Number of Elements (N) versus Compression Angle and Mach Number.

Angle	Mach			
	2	3	5	10
5	8	30	65	400
10	7	20		
15	5	16	45	150
20	3	13		
30		4	18	35
37			6	
40				8

The error downstream of a compression corner (shock wave) can be minimized by placing an appropriate number of elements in that region, as suggested by Table 6.1 and Figure 6.50.

For instance, if one wishes for the solution to approach the jump properties within 5% of the downstream plate length (reasonable for engineering estimates), then N elements (as shown in the Table 6.1 or Figure 6.50) should be placed downstream of the shock over 5% of the plate length. Another example would be an incident shock near another surface. A reflected shock in the inflow plane of the combustor would necessitate matching the downstream jump condition. If the distance between the shock and point of interest (other wall, inflow plane, combustor, etc) is expressed as x , then at least N elements should be placed downstream of the shock over the distance x so that the solution can properly transition to the correct downstream properties. This gives a spacing of x / N along the shock center. More elements will improve the accuracy but also slow down the solution speed. If, for some unforeseen reason, the artificial dissipation is increased (stability) or decreased (accuracy), the number of elements within the “smeared” region needs to be adapted: $x / N / diss$

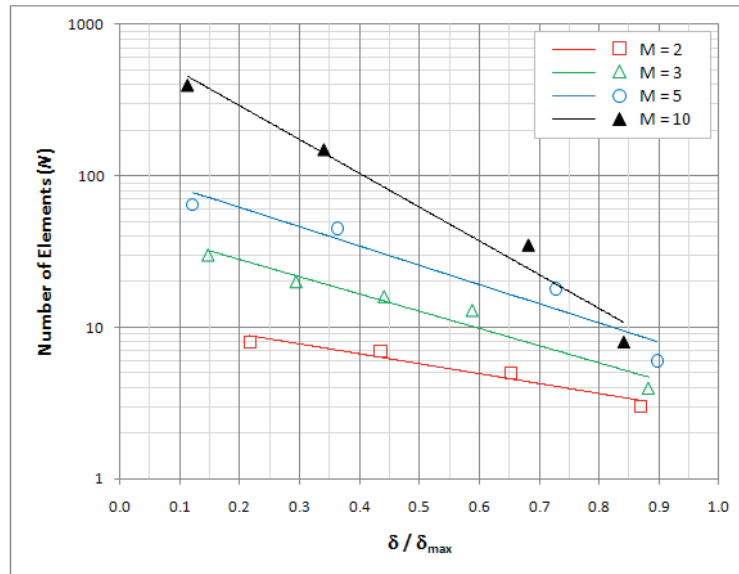


Figure 6.50: Number of Elements (N) versus Compression Angle and Mach Number.

6.1.3.3 Supersonic Expansion Corner

Similar to the compression corner, the expansion corner produces different conditions up and downstream of the corner, but the expansion occurs over an intentionally wider region – the expansion fan. Figure 6.51 shows this region on a very fine mesh. Figure 6.52 shows the pressure distribution along the wall up and downstream of the corner. This solution is typical: Upstream properties are constant, and the downstream properties transition to the desired theoretical values (within 0.1% of theory over far enough distances). The transition region is smoothed out by artificial dissipation. The smoothed (or smeared) region just downstream of the corner is enlarged by larger elements, higher Mach number, and larger dissipation scalars.

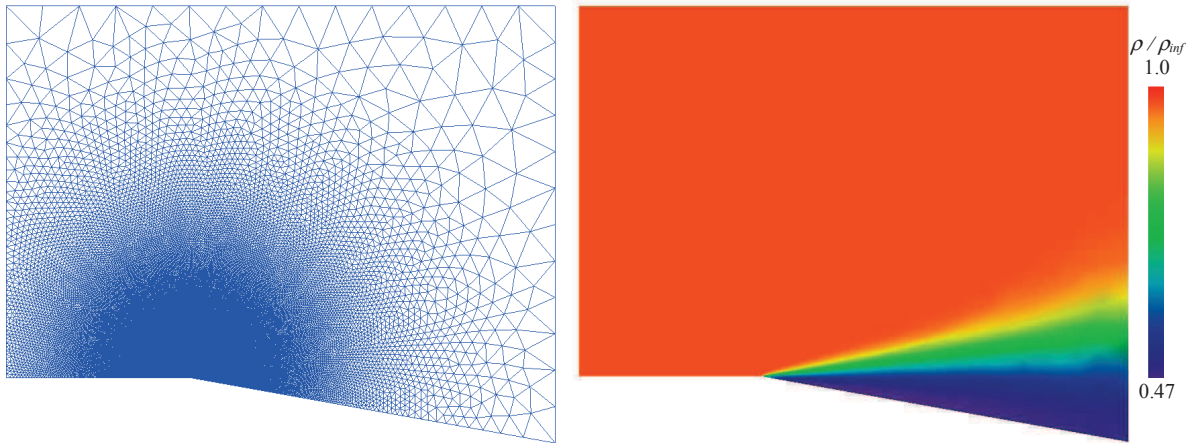


Figure 6.51: Mesh (left) and Density (right) for 10° Expansion Corner at $M = 5$.

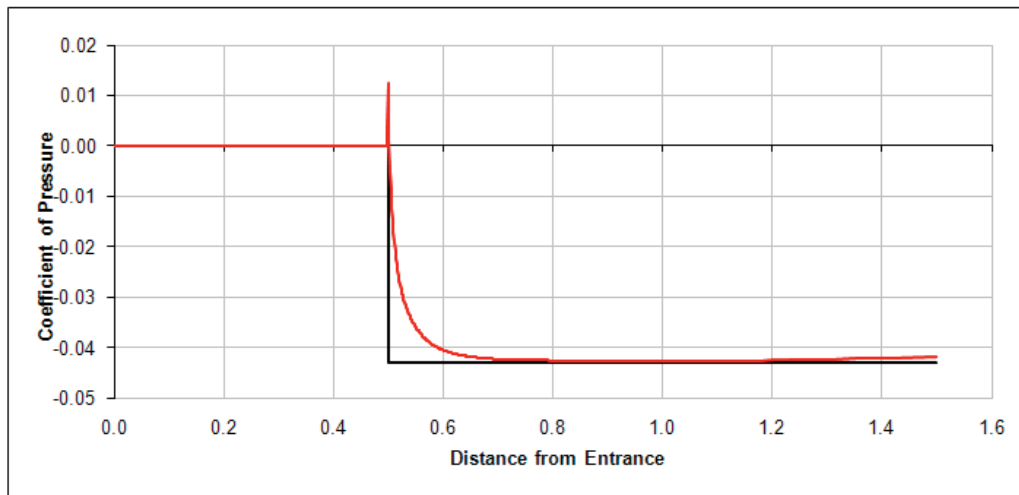


Figure 6.52: Pressure along Wall of 10° Expansion Corner, $M = 3$, $\Delta x = 0.016 L$.

Curves similar to Figure 6.48 and Figure 6.49 were produced for the expansion corner, where the solutions from different mesh spacings or dissipation scalars can be collapsed on top of each other by dividing by the mesh spacing or dissipation scalar. The expansion corners (2 to 30 degrees) were tested at Mach 2, 3, 5, and 10 along with the effects of higher and lower

dissipation scalars. The results ($diss = 1.0$) are shown in Table 6.2 and graphically in Figure 6.53. Figure 6.53 (left) plots the number of elements versus the measure of the corner angle. The expansion angle seems to have a minimal influence on the number of elements required to model the expansion process. Figure 6.53 (right) shows the number of elements versus the upstream Mach number M .

Table 6.2: Number of Elements (N) versus Expansion Angle and Mach Number.

Angle	Mach			
	2	3	5	10
2				280
5	12	27	75	300
10	13	30	80	300
20	17	36	80	
30	16	30		
AVG	15	30	80	300

The error downstream of an expansion corner (expansion fan) can be minimized by placing an appropriate number of elements in that region, as suggested by Table 6.2 and Figure 6.53 (right). This process is similar to that described for the compression corner, but the number of elements does not depend on the angle of the corner.

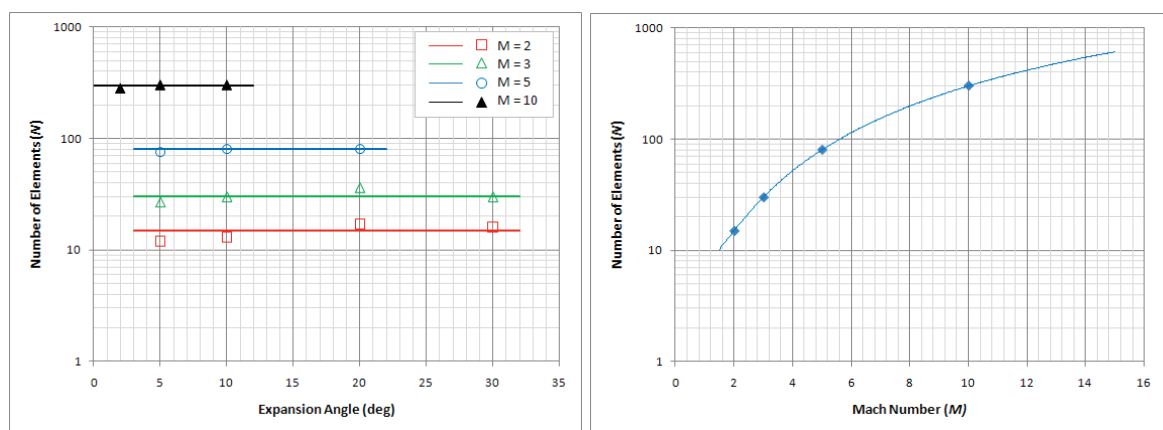


Figure 6.53: Number of Elements (N) versus Expansion Angle and Mach Number.

6.1.3.4 Supersonic Double Wedge (5 deg) Airfoil (Mach 2, 2 deg)

The supersonic double-wedge airfoil is a simple supersonic case that can be compared to a reliable analytical solution. The airfoil geometry is defined to be symmetric both top-to-bottom and front-to-back with half-wedge angle of 5 degrees. The airfoil was held at 2 degrees at Mach 2, to create four different oblique shocks and two different expansion fans for comparison to theory.

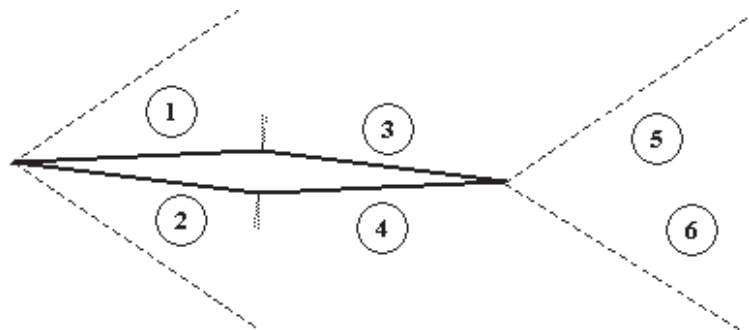


Figure 6.54: Double-Wedge Airfoil Geometry (to scale).

The analytical solution used here comes from shock-expansion theory (John, 1984). Shock-expansion (SE) analysis predicts the pressure and Mach number on each of the four surfaces with shock angles. The pressure coefficients are used to calculate the lift, wave drag, and moment (LE) coefficients. The shock angles are shown with respect to the centerline of the airfoil.

The mesh was generated in three sessions: The initial mesh was generated to refine the area around the airfoil with no refinement for the shocks or expansions. The solution from Euler2D was used to locate the shocks and expansions, instead of using analytical angles. The regions around the shocks and expansions were refined to capture the property changes in these areas. Euler2D was used to refine the solution, which showed a further need for

refinement at the corners of the airfoil. The final mesh contained 2.4M elements and can be seen in Figure 6.57. Each image in the Figure 6.57 represents one level of refinement, and the areas of refinement are illustrated with the darkest clusters of elements. The 2D sources and domain were extruded to create the 3D sources and domain. Such an extrusion can be done before meshing with *Geom2Dto3D* or after meshing with *2DExtrude*.

Euler3D converged on a solution for the double-wedge airfoil in 1,100 iterations, with 4 inner cycles per iteration and *CFL* of 0.5. Lower order dissipation was used with a dissipation factor *diss* of 1.0. The distributions of four of the properties are plotted in Figure 6.58. Pressure, density, and local Mach number all show the presents of shocks and expansion. The shocks are crisp, abrupt changes in flow properties, while the expansions are “fanned” changes in properties around the corner. The pressure and Mach number distributions over the top and bottom surfaces of the airfoil are shown in Figure 6.55 for Euler2D and Figure 6.56 for Euler3D.

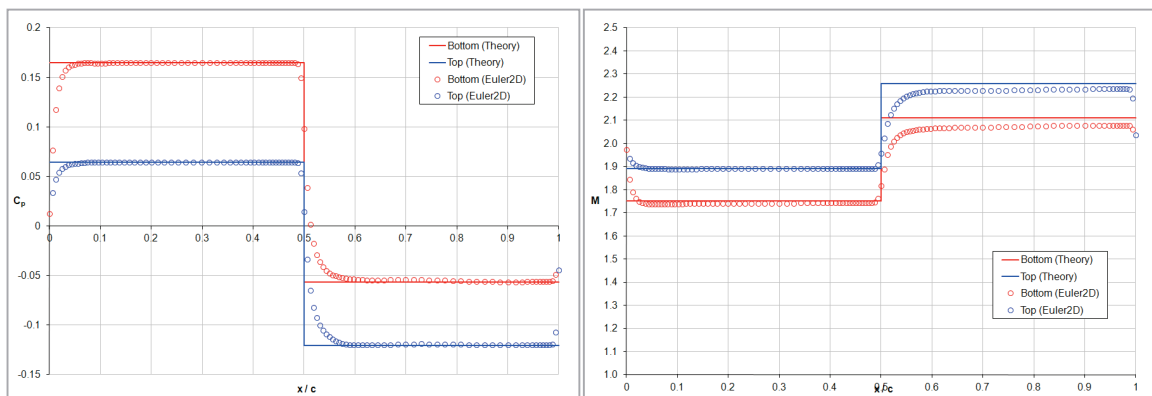


Figure 6.55: Pressure and Mach Distributions for Double-Wedge Airfoil (Euler2D).

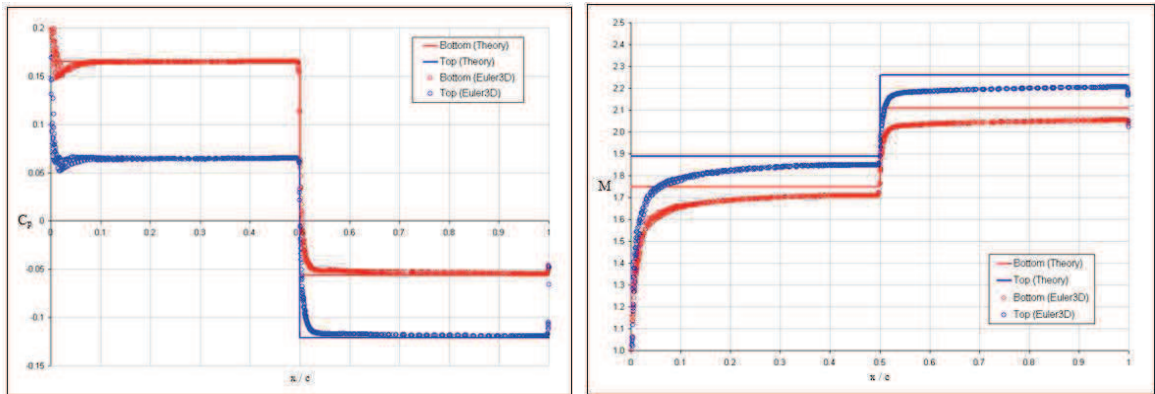


Figure 6.56: Pressure and Mach Distributions for Double-Wedge Airfoil (Euler3D).

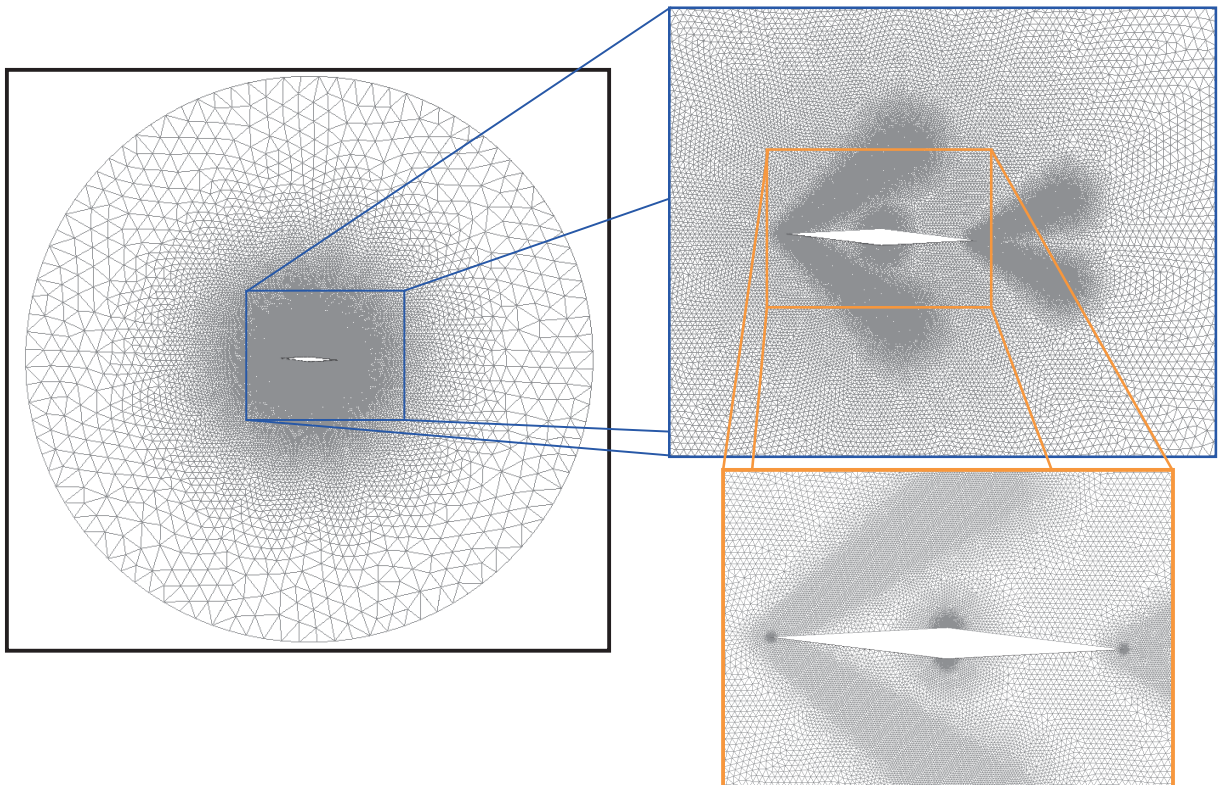


Figure 6.57: Double-Wedge Airfoil Mesh (Euler3D).

Table 6.3 and Table 6.4 shows the results from Euler2D and Euler3D in comparison with the shock-expansion theory. The CFD values for zones 1 and 2 (forward half) were averaged between 30 and 40% chord. Values for zones 3 and 4 (aft half) were averaged between 80

and 90% chord. These two locations were chosen because the Mach number approaches a zero derivative in the CFD plots in Figure 6.56.

The percent differences shown in Table 6.4 are very low for the pressure, Mach number, and shock angles. Shock angles were estimated using rough “eye-balled” lines and a protractor. This process is not always that reliable for the CFD results. The lift and wave drag coefficients were very reliable, but the coefficient of moment was 22% higher than that predicted by SE theory. Figure 6.56 shows the pressure behind both shocks and expansions lags due to artificial dissipation. The lag pushes the center of pressure further aft creating a larger moment. (This case was created before the meshing standards in Figure 6.50 and Figure 6.53 were defined. This case could be repeated with the standards to reduce the error in moment coefficient and possibly lift and wave drag.)

Table 6.3: Results from Euler2D Compared with Shock-Expansion Theory.

Coeff. of Pressure			Local Mach #			Shock Angle			Aero. Coeff.		
Zone	C_p	% diff	Zone	M	% diff	---	(deg)	% diff	---	---	---
1	0.0621	-4%	1	1.89	0%	LE, Top	36	4%	c_l	0.0803	-3%
2	0.1577	-4%	2	1.75	0%	LE, Bot'm	34	-1%	$c_{d,w}$	0.0201	-3%
3	-0.1099	-9%	3	2.21	-2%	TE, Top	30	-12%	$c_{m,LE}$	0.0442	20%
4	-0.0461	-18%	4	2.05	-3%	TE, Bot'm	30	4%			

Table 6.4: Results from Euler3D Compared with Shock-Expansion Theory.

Coeff. of Pressure			Local Mach #			Shock Angle			Aero. Coeff.		
Zone	C_p	% diff	Zone	M	% diff	---	(deg)	% diff	---	---	---
1	0.0646	0%	1	1.84	-2%	LE, Top	35	1%	c_l	0.0814	-1%
2	0.1649	0%	2	1.71	-3%	LE, Bot'm	35	2%	$c_{d,w}$	0.0203	-2%
3	-0.1189	-2%	3	2.20	-3%	TE, Top	28	-18%	$c_{m,LE}$	0.0449	22%
4	-0.0540	-4%	4	2.05	-3%	TE, Bot'm	27	-6%			

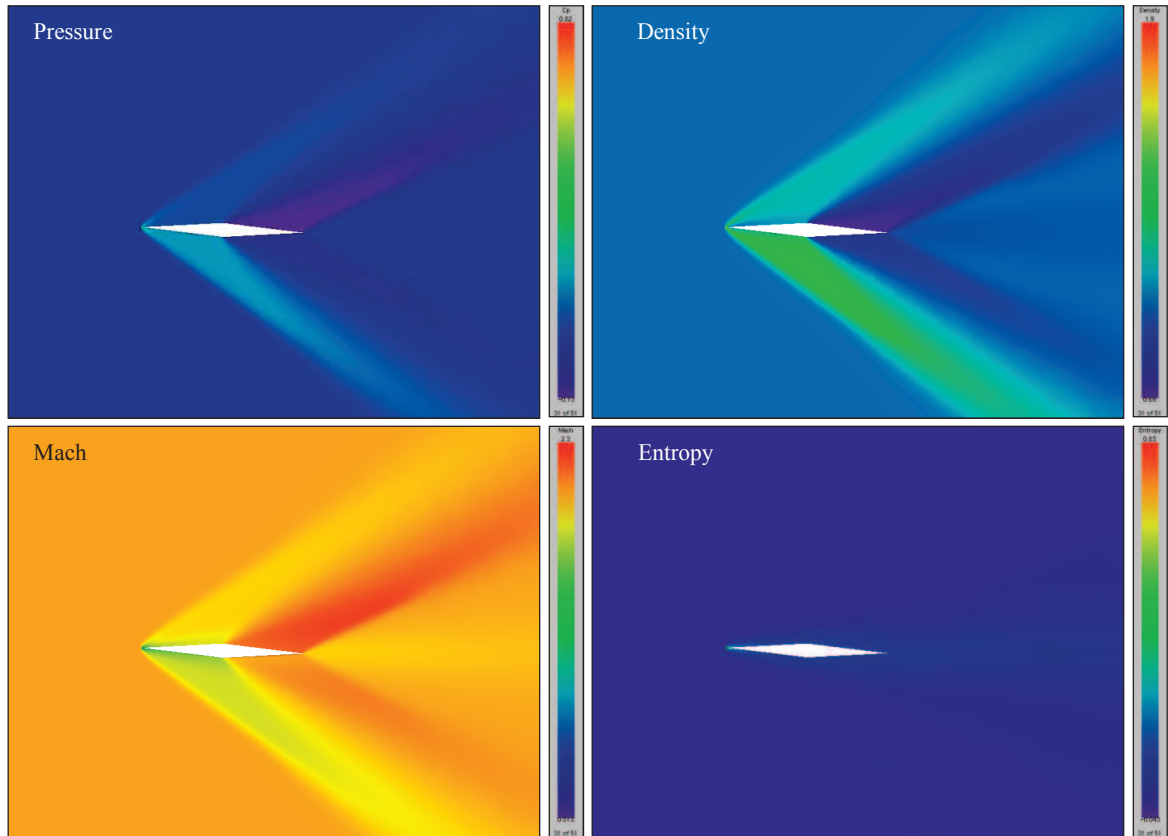


Figure 6.58: Pressure, Density, Mach, and Entropy Distrib. for Double-Wedge Airfoil.

Mesh convergence was shown by comparing the pressure coefficients, local Mach numbers, and aerodynamic coefficients, as seen in Table 6.4, for the three mesh refinements. The last mesh refinement changes the pressure coefficients less than 1%, Mach numbers 4%, and aerodynamic coefficients 2%. (Shock angles were not compared for their relative.)

The double-wedge airfoil was also tested at Mach 5 and 10. The meshes were generated using the same three-step process as the previous Mach 2 case. The Mach distributions for these two flight speeds are shown in Figure 6.59. The figure shows the shock angles decrease with flight speed, so that the shocks begin to encroach on the airfoil surface itself. The solution is also slurred over airfoil surface due to artificial dissipation. The results from Euler3D were again compared to SE theory. The results are shown in Table 6.5 and Table

6.6 for Mach 5 and 10, respectively. The Mach 5 solution compared reasonable well (within 5-7%) to theory, while the Mach 10 solution (within 11-16%) needs some further mesh refinement. These results show that Euler3D produces reasonable accuracy at supersonic and hypersonic speeds.

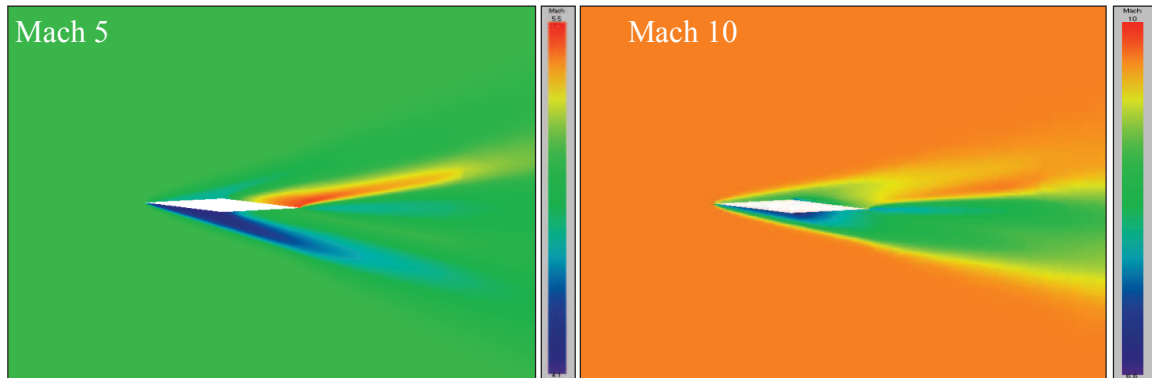


Figure 6.59: Mach Distrib. for Double-Wedge Airfoil at Mach 5 and 10 (Euler3D).

Table 6.5: Comparison of Mach 5 Solutions to SE Theory.

Coeff. of Pressure			Local Mach #			Shock Angle		
Zone	C_p	% diff	Zone	M	% diff	---	(deg)	% diff
1	0.0237	-5%	1	4.98	6%	LE, Top	16	3%
2	0.0700	-2%	2	4.10	-5%	LE, Bot'm	17	16%
3	-0.0330	-5%	3	5.50	-6%	TE, Top	15	-12%
4	-0.0175	-4%	4	4.90	-7%	TE, Bot'm	10	-8%

Table 6.6: Comparison of Mach 10 Solutions to SE Theory.

Coeff. of Pressure			Local Mach #			Shock Angle		
Zone	C_p	% diff	Zone	M	% diff	---	(deg)	% diff
1	0.0130	-9%	1	8.74	-2%	LE, Top	10	2%
2	0.0460	-5%	2	7.32	-4%	LE, Bot'm	11	17%
3	-0.0110	-11%	3	11.28	-15%	TE, Top	12	-2%
4	-0.0065	-13%	4	8.92	-16%	TE, Bot'm	6	9%

Figure 6.60 shows the convergence of the RMS error in the energy equation (or energy residual) from Euler3D. The residual converges with the inverse of the relaxation factor *CFL*; cutting the relaxation factor in half doubles the run time. The plot also shows that steady state solutions in Euler3D converge to a residual of 10^{-16} or smaller, given a long enough run time.

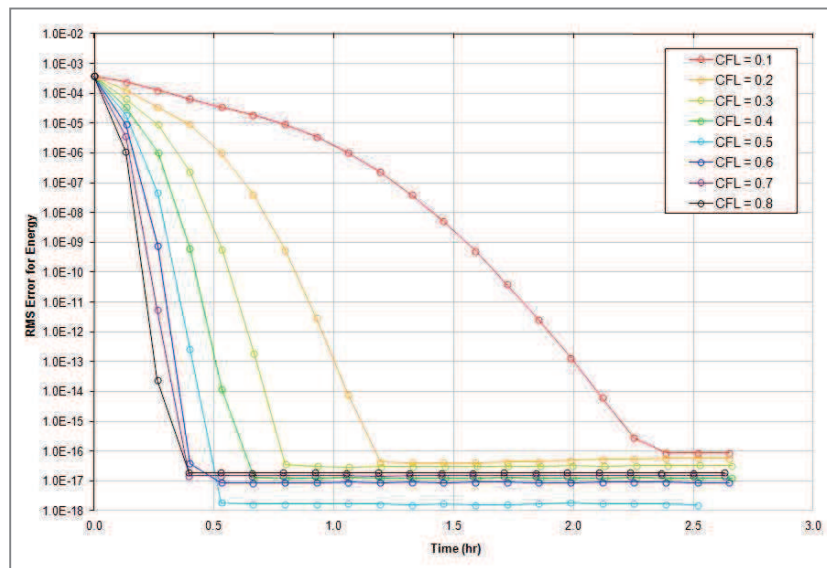


Figure 6.60: Convergence of Energy Residual vs. CFL.

6.1.4 Time-Accurate

The Wagner airfoil was selected as a simple unsteady case that can show time-accuracy on load calculations. Wagner releases acoustic waves when the flow suddenly sees the wall surfaces. These acoustic waves were also tracked to show accurate propagation of waves.

6.1.4.1 Wagner Airfoil (NACA 0012, Mach 0.3, 5 deg)

In 1925, Wagner analytically investigated the flow over a suddenly accelerated airfoil.

Wagner suggested the problem of an airfoil held at a specified angle of attack in still air. The

air is accelerated to a desired speed so abruptly that the velocity was modeled using a step impulse. Wagner (1925) found that the lift on the airfoil started at half of its steady state lift and asymptotically approached the steady lift as time progressed. The flow is dominated by a startup vortex that produces downwash on the airfoil. The downwash reduces the lift that the airfoil produces. As the flow progresses, the startup vortex moves further downstream, decreasing its influence on the airfoil.

Jones (1940) found a simplified equation that represents the lift history of a suddenly accelerated airfoil. Jones used Wagner's equations and assumed a shape for the lift history. Jones approximated the coefficients for his lift history:

$$c_l = c_{l\alpha}(\alpha - \alpha_{L=0})\left(1 - 0.165e^{-0.045s} - 0.335e^{-0.3s}\right)$$

where $c_{l,\alpha}$ is the lift slope, $\alpha_{L=0}$ is the zero lift angle, and s is the clearing-time, or time for the freestream to move one half-chord downstream: $s = 2V_\infty t / c$.

The Wagner problem was modeled in Euler3D using a NACA 0012 airfoil at Mach 0.3 and 5-degrees angle of attack. The solution was compared to that of Jones, which was corrected for compressibility at Mach 0.3 using Prandtl-Glauert (Anderson, 2001). Euler2D and Euler3D were both shown to match theory very well (see Figure 6.61 and Figure 6.62). The CFD lift approaches infinity as time goes to zero because the CFD solution requires a finite amount of time to grow the startup vortex and then release it into the wake, whereas Wagner assumes that the startup vortex appears and is released instantaneously into the flow.

Figure 6.63 and Figure 6.64 show snapshots of the unsteady pressure and velocity distributions near the airfoil. The snapshots show the growth of the startup vortex at the trailing edge

of the airfoil. The release of the vortex from the trailing edge coincides with the minimum lift at approximately $t^* = 0.5$, shown in Figure 6.61 and Figure 6.62. The vortex drifts downstream in the airfoil's wake, decreasing the downwash on the airfoil, increasing the lift created by the airfoil. The vortex drifts downstream at 95% of the freestream velocity. Figure 6.64 shows acoustic waves that emanate from the airfoil surface. These acoustic waves are created when the flow solution instantaneously “feels” the presence of the airfoil.

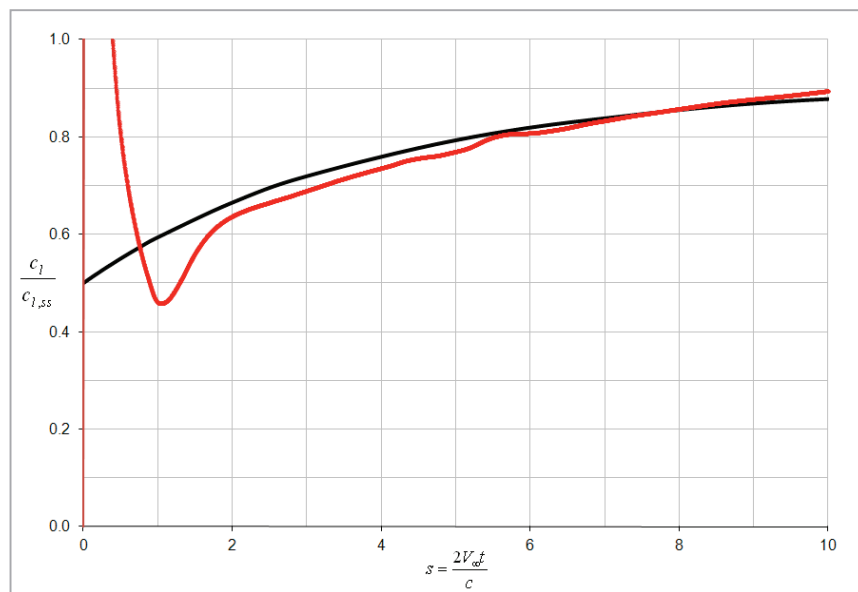


Figure 6.61: Wagner Solution from Euler2D Compared to Jones' Approximation.

Figure 6.65 shows the pressure distribution upstream and downstream of the airfoil (shown in black) at eight time steps. The eight steps correspond to later pictures shown in Figure 6.64. The data between the LE and TE of the airfoil has been omitted to emphasize the waves traveling away from the airfoil.

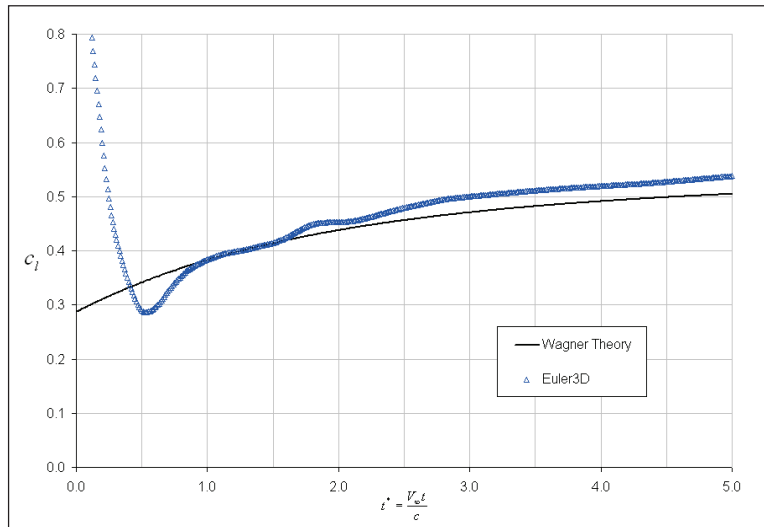


Figure 6.62: Wagner Solution from Euler3D Compared to Jones' Approximation.

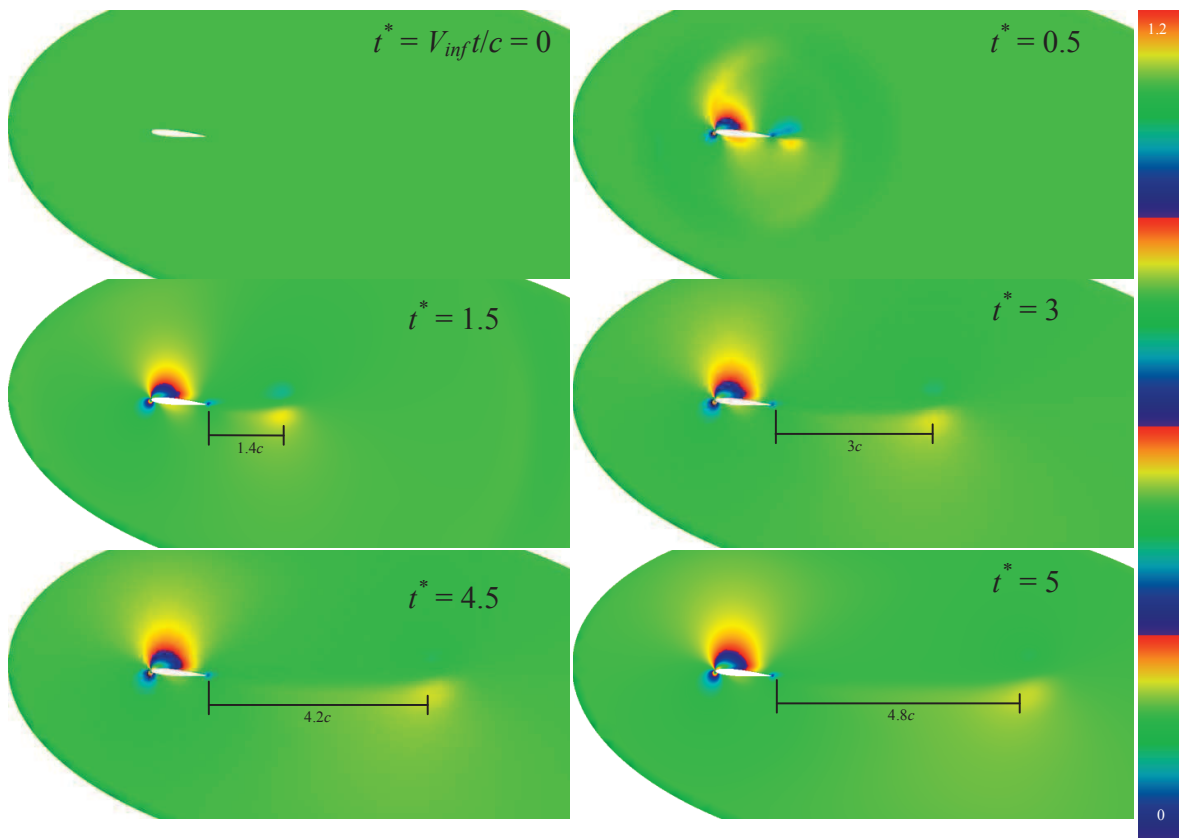


Figure 6.63: Snapshots of the Unsteady Velocity Distribution for Wagner Solution.

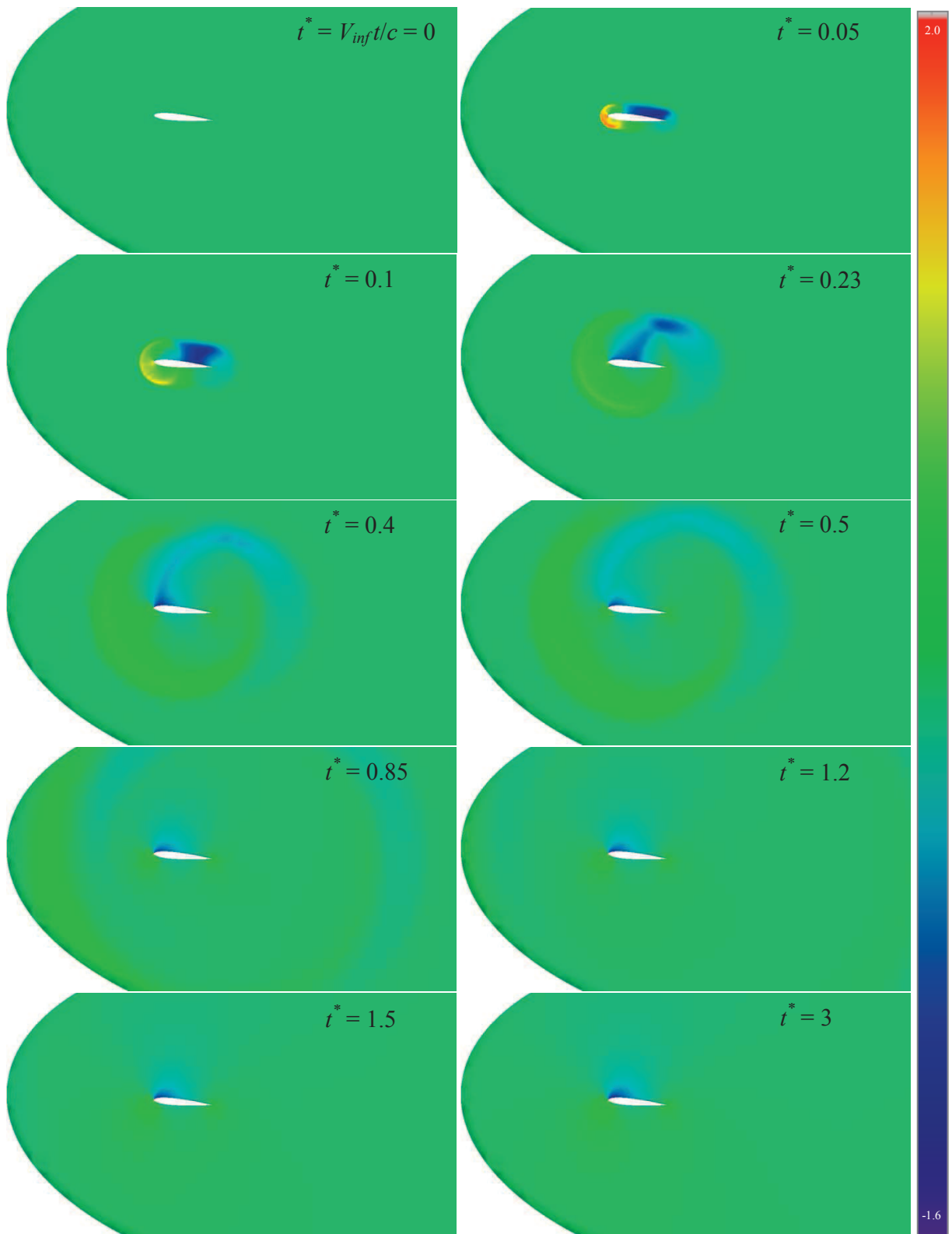


Figure 6.64: Snapshots of the Unsteady Pressure Distrib. (C_p) for Wagner Solution.

Figure 6.66 shows the position of both waves versus time. The forward traveling (LE) wave is shown at the first six time steps. The LE acoustic wave is very strong at the onset and drops (in theory) with $1/r^2$. The LE wave is a large overshoot in pressure followed by a small undershoot in pressure. The pressure behind the LE wave returns to approximately freestream (with influence from the airfoil local flow).

The aft traveling (TE) wave is also very strong at the onset and drops (in theory) with $1/r^2$. The TE wave is an undershoot in pressure (smaller than the LE wave) followed by a smaller overshoot in pressure. The pressure behind the TE wave is slightly higher than freestream for much of the domain because of the presence of the airfoil local flow. The shape of the TE waves is much clearer than the LE waves because the mesh is tighter in the wake region than upstream of the airfoil.

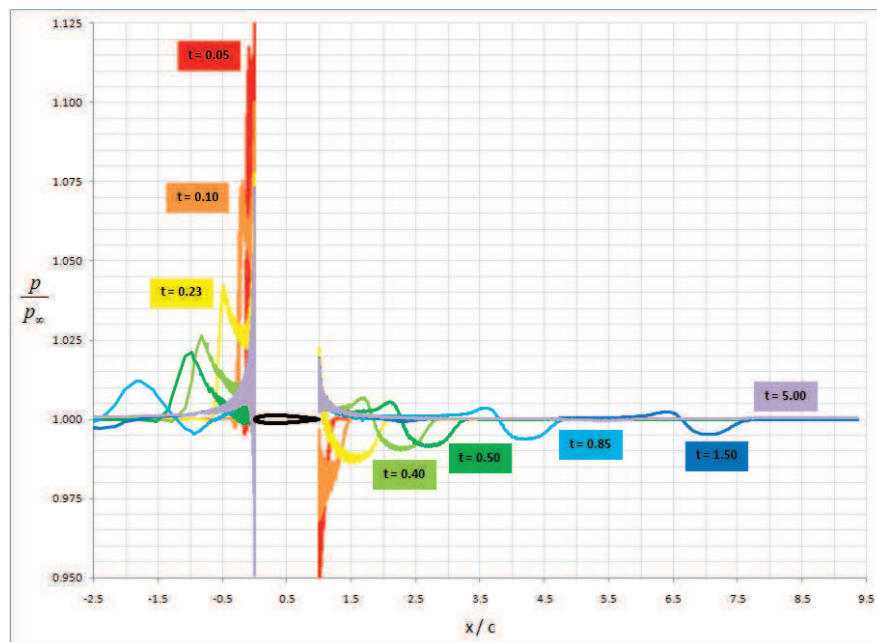


Figure 6.65: Wagner Pressure Slices Up and Downstream of Airfoil.

Figure 6.66 shows that the LE wave travels upstream at a velocity of 2.2037 and the TE wave travels downstream at a velocity of 4.3571. The average of these two speeds (as a vector) is 1.0767, which corresponds to the mean propagation speed (freestream velocity). The difference between the wave speeds and their mean gives an estimate of the acoustic speed in the flow: 3.2804. This acoustic speed is 1.6% lower than the user defined acoustic speed (dim'less: $a_{inf}/V_{inf} = M^{-1} = 1/0.3 = 3.33$), and the mean propagation speed is 7.7% higher than the freestream velocity (dim'less: 1.0). The differences in predicted and actual speeds are due to compressibility, which changes the local acoustic speeds within the wave.

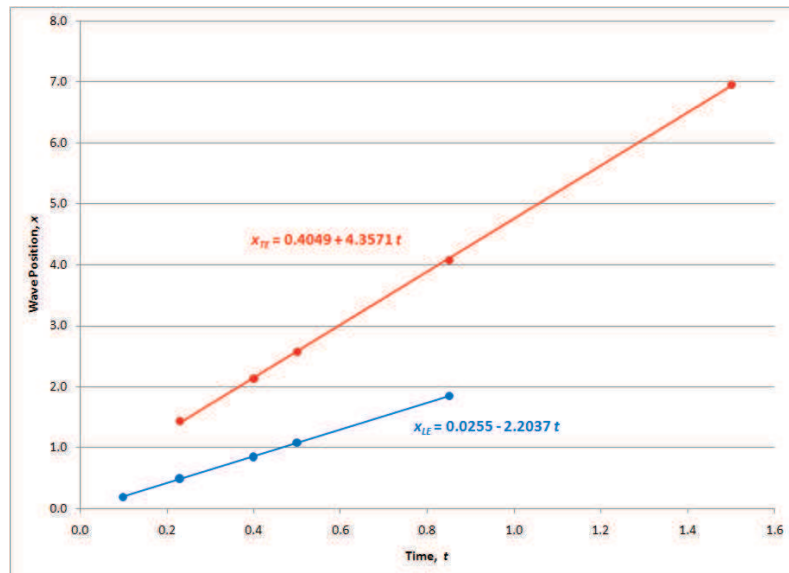


Figure 6.66: LE and TE Wave Position as a Function of Time.

6.2 Propulsion Modeling

The quasi-combustion terms and propulsion boundary conditions were tested in Euler2D and Euler3D using a linear combustor, GHV, converging-diverging rocket nozzle, and several turbojet engines. The quasi-combustion terms were tested in a linear afterburner with mass and enthalpy addition at subsonic, supersonic, and hypersonic speeds. A GHV was used to

demonstrate how the quasi-combustion terms can be applied to hypersonic (scramjet) vehicles. Stability derivatives and performance trade-offs are also estimated for the cross-section as an examples of quasi-combustion as an early design tool. The CD nozzle was tested at three total pressures at and below design pressure. One subsonic inlet and two supersonic inlets were demonstrated at on-design, off-design, and angles of attack.

6.2.1 Quasi-Combustion Terms

The quasi-combustion terms have been tested using enthalpy and mass generation. Three different Mach numbers were tested, representing subsonic, supersonic, and hypersonic regimes. The first set of tests adds enthalpy to the flow without mass addition. Two different amounts of energy were added to the supersonic and hypersonic cases. Two different distributions were tested in each situation: A constant hat-function and a cosine-smoothed function. The solutions from Euler2D and Euler3D were checked with Rayleigh line flow (John, 1984). Mass was then added to the subsonic and supersonic cases, followed by the combination of mass and enthalpy addition. The properties predicted by adding mass and enthalpy to the flow in Euler2D and Euler3D were compared to theoretical values created by modifying Rayleigh's theory to include mass flow (Bathie, 1996; Mattingly, 1996). These cases show that the quasi-combustion terms work properly and within reasonable accuracy for subsonic, supersonic, and hypersonic speeds.

6.2.1.1 Subsonic Linear Afterburner (Mach 0.4)

Heat was added to the flow in a subsonic combustor to simulate the effects of combustion. Figure 6.67 and Figure 6.68 were produced by modifying the outflow boundary. The boundary integrals along the far field BC in Euler3D were modified with the following code:

```

xc = ( COOR(n1,1) + COOR(n2,1) + COOR(n3,1) ) / 3.0d0 ! element midpoint
if ( xc .GT. 10.0d0 ) then ! midpoint btwn inflow & outflow
  rr = cinf(1) * 0.491d0 ! rho2 / rho1 = 0.491
  ur = cinf(2) * 2.037d0 ! u2 / u1 = 2.037
  vr = cinf(3) * 0.0d0
  wr = cinf(4) * 0.0d0
  pr = cinf(5) * 0.768d0 ! p2 / p1 = 0.768
  hr = cinf(6) * 0.491d0 * 1.6439d0 ! rH2 / rH1 = 0.491 * 1.6439
endif

```

Mass was added to the flow in a subsonic duct to demonstrate the mass addition terms.

Figure 6.69 and Figure 6.70 were produced by modifying the outflow boundary condition.

The boundary integrals along the far field BC in Euler3D were modified with the following:

```

xc = ( COOR(n1,1) + COOR(n2,1) + COOR(n3,1) ) / 3.0d0 ! element midpoint
if ( xc .GT. 10.0d0 ) then ! midpoint btwn inflow & outflow
  rr = cinf(1) * 1.038d0 ! rho2 / rho1 = 1.038
  ur = cinf(2) * 1.012d0 ! u2 / u1 = 1.012
  vr = cinf(3) * 0.0d0
  wr = cinf(4) * 0.0d0
  pr = cinf(5) * 0.986d0 ! p2 / p1 = 0.986
  hr = cinf(6) * 1.038d0 * 0.952d0 ! rH2 / rH1 = 1.038 * 0.952
endif

```

Mass and enthalpy were added to the flow in a subsonic afterburner as a final test of the subsonic capabilities of the quasi-combustion terms in Euler2D and Euler3D. Figure 6.71

and Figure 6.72 were also produced by modifying the outflow boundary condition. The

boundary integrals along the far field BC in Euler3D were modified with the following code:

```

xc = ( COOR(n1,1) + COOR(n2,1) + COOR(n3,1) ) / 3.0d0 ! element midpoint
if ( xc .GT. 10.0d0 ) then ! midpoint btwn inflow & outflow
  rr = cinf(1) * 0.534d0 ! rho2 / rho1 = 0.534
  ur = cinf(2) * 1.966d0 ! u2 / u1 = 1.966
  vr = cinf(3) * 0.0d0
  wr = cinf(4) * 0.0d0
  pr = cinf(5) * 0.761d0 ! p2 / p1 = 0.761
  hr = cinf(6) * 0.534d0 * 1.502d0 ! rH2 / rH1 = 0.534 * 1.502
endif

```

This modification was required because the Riemann conditions at both the inflow and outflow boundaries allow the solution to “float” slightly while gradients are created to balance the governing equations. The final solution does not match at the inflow or outflow boundaries, but the ratio of their properties is correct. The absolute properties are used to

calculate the local Mach, which is different at both planes. To match the solution, the outflow condition is modified to resemble the desired properties, trapping the solution where desired. The maximum error for the Mach 0.4 cases is 0.03%.

The ratio of mass, momentum, and energy flow rates have been calculated for these six cases below. Each ratio uses the ratio of properties represented graphically:

$$\frac{\dot{m}_2}{\dot{m}_1} = \frac{\rho_2 u_2}{\rho_1 u_1} \quad \frac{P_2}{P_1} = \frac{(\rho_2 u_2^2 + p_2)A}{(\rho_1 u_1^2 + p_1)A} = \frac{p_2}{p_1} \frac{1 + \gamma M_2^2}{1 + \gamma M_1^2} \quad \frac{\dot{m}_2 H_2}{\dot{m}_1 H_1} = \frac{\rho H_2 u_2}{\rho H_1 u_1}$$

The maximum error for the Mach 0.4 cases is 0.17%. The ratio of all three flow rates matches theory very well, with only numerical errors to account for the differences.

6.2.1.2 Supersonic Linear Afterburner (Mach 2)

Heat was added to the flow in a supersonic combustor to simulate the effects of combustion. The outflow boundary conditions do not need to be modified because the characteristics are taken from upstream (domain elements). The maximum error for the Mach 2 cases is 0.13%.

Mass (and enthalpy) was added to the flow in a supersonic duct. The ratio of mass, momentum, and energy flow rates were calculated using the equations in the previous section. The maximum error for the Mach 2 cases is 0.35%. The ratio of all three flow rates matches theory very well, with only numerical errors to account for the differences.

6.2.1.3 Hypersonic Linear Afterburner (Mach 7)

Heat was added to the flow in a hypersonic combustor to simulate the effects of combustion. The outflow boundary conditions do not need to be modified because the characteristics are taken from upstream (domain elements). The maximum error for the Mach 7 cases is 0.08%.

The cosine (smoothed transitions) work better at hypersonic speeds than sharp functions. Density fluctuations occur for sharp transitions (sudden changes in heat generation). The heat generation creates changes in enthalpy through the energy equation, which in turn creates changes in velocity through the momentum equation, which finally create changes in density through the continuity equation. Pressure is updated along with any change in properties through the equation of state. Density is the last to be updated and reflects all of the slight variations in the other properties. Smooth transitions should be used at high supersonic and hypersonic speeds.

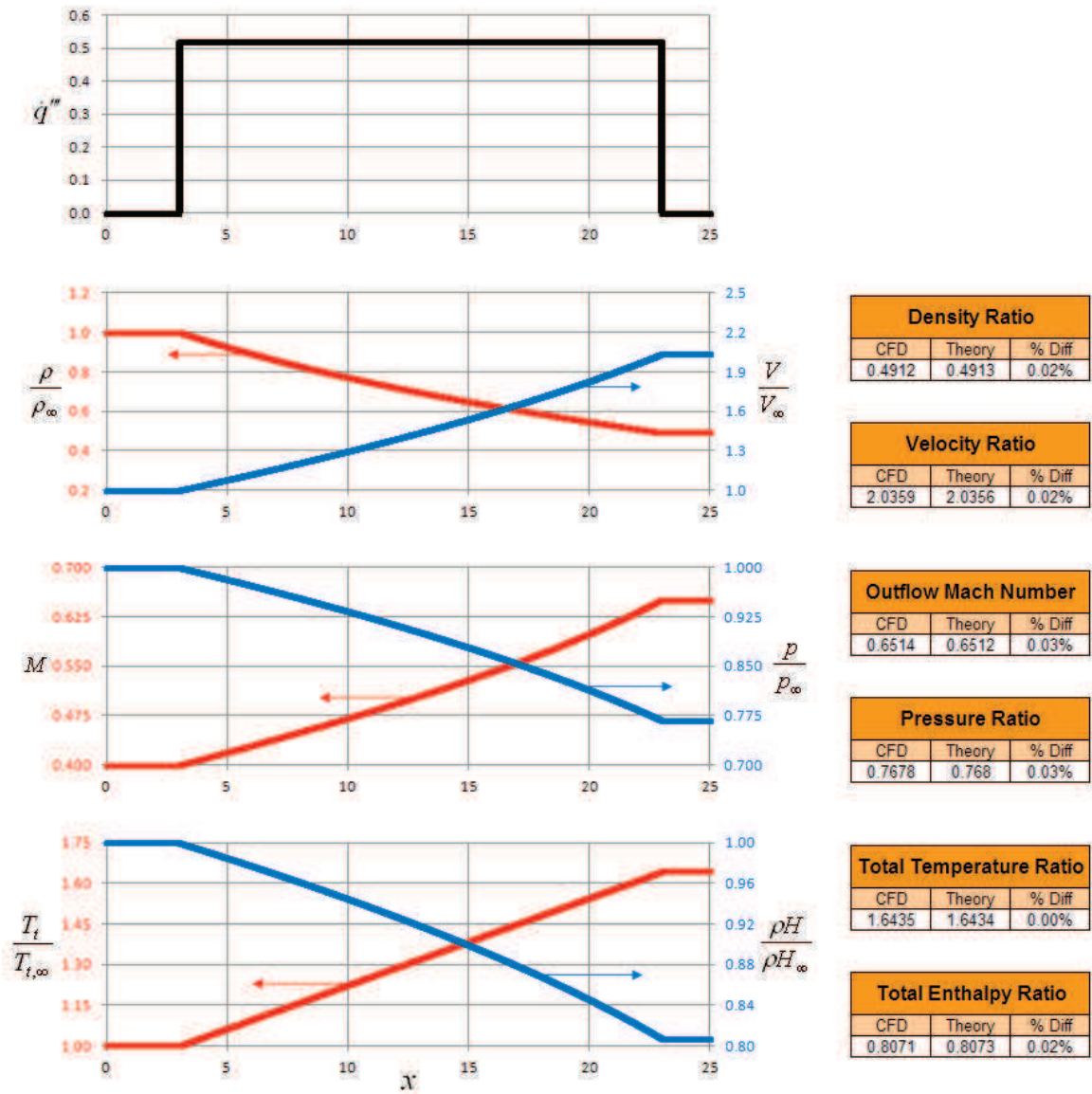


Figure 6.67: Subsonic (Mach 0.4) Constant Heat Generation.

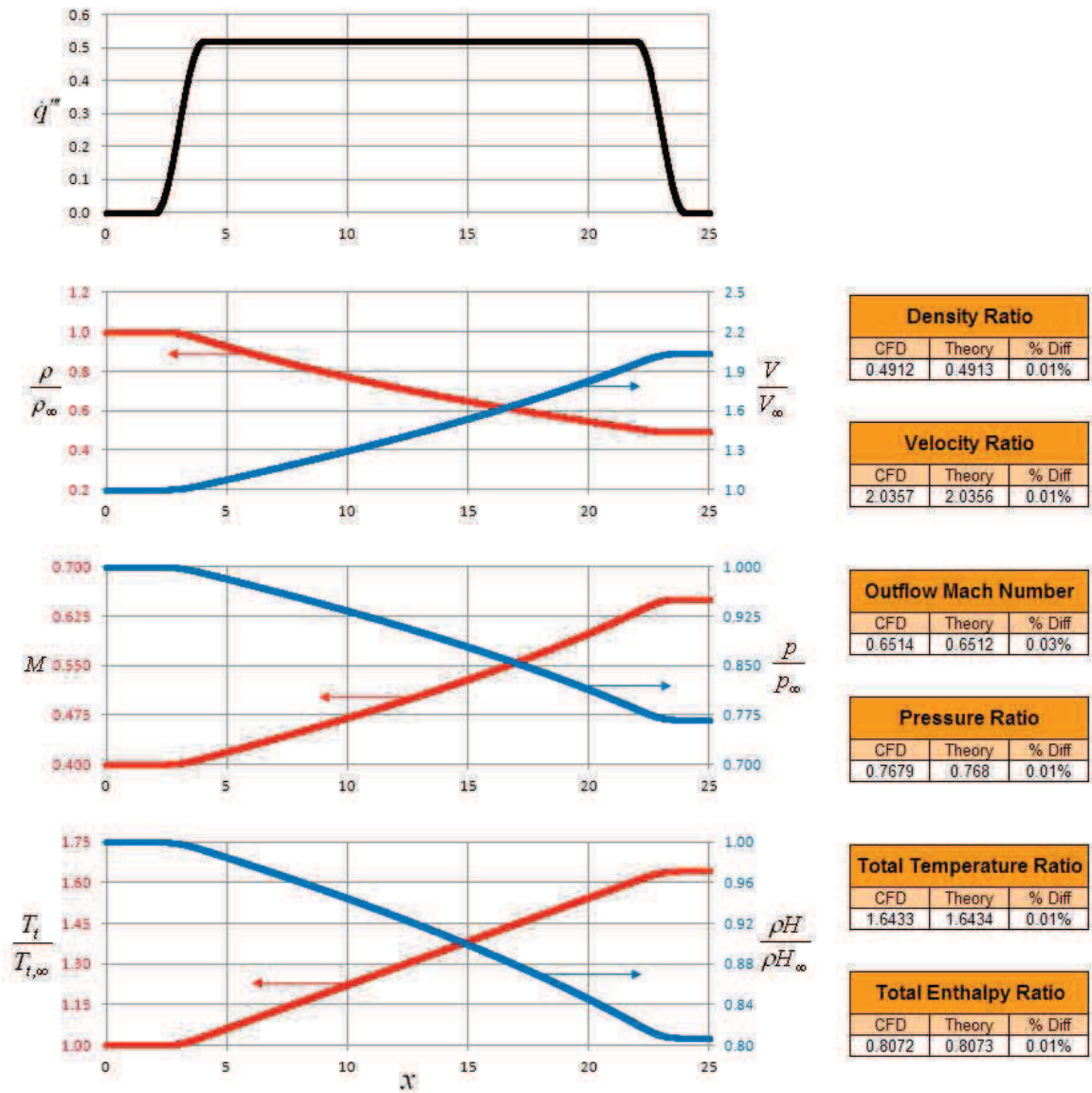


Figure 6.68: Subsonic (Mach 0.4) Cosine Heat Generation.

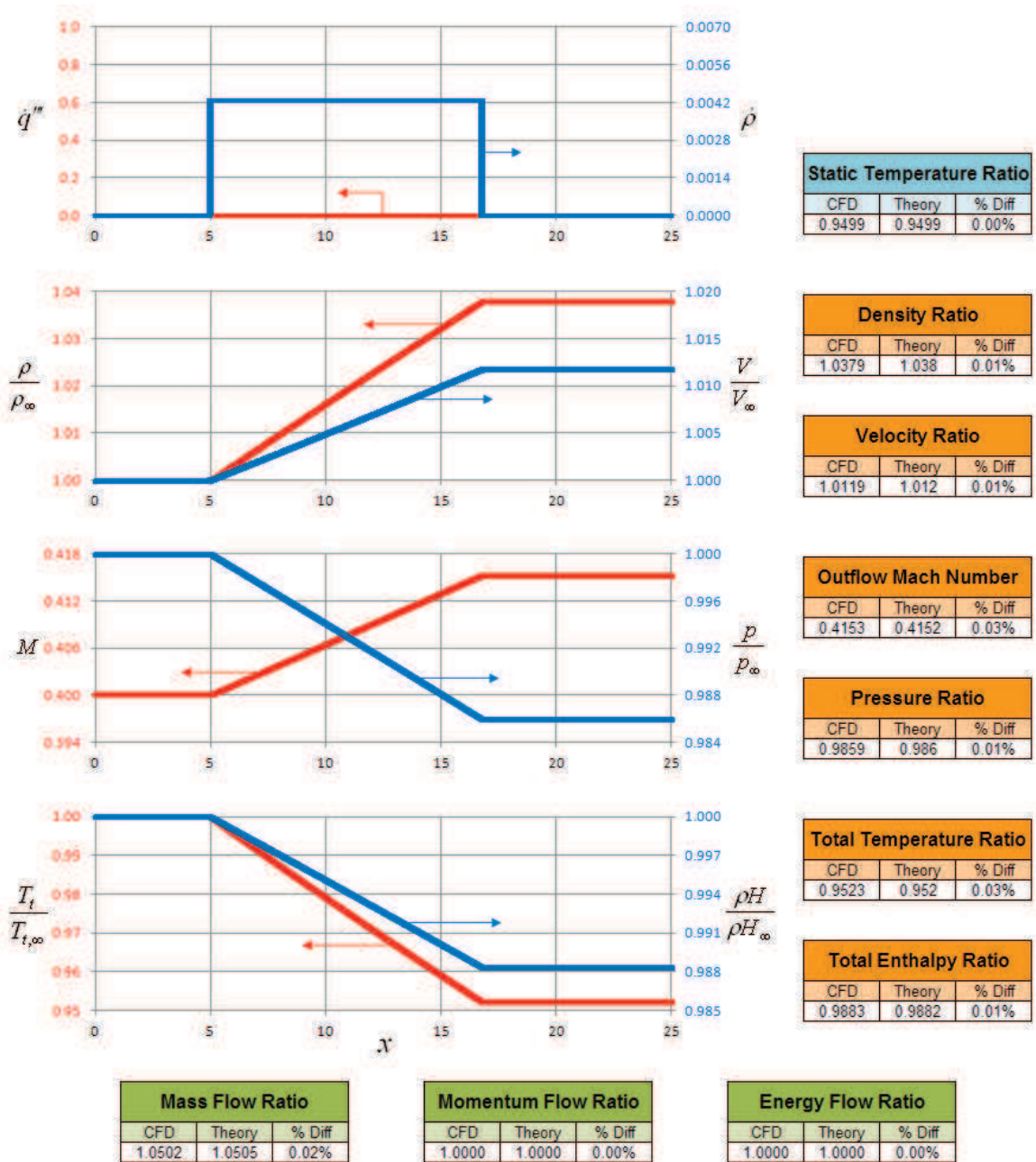


Figure 6.69: Subsonic (Mach 0.4) Constant Mass Generation.

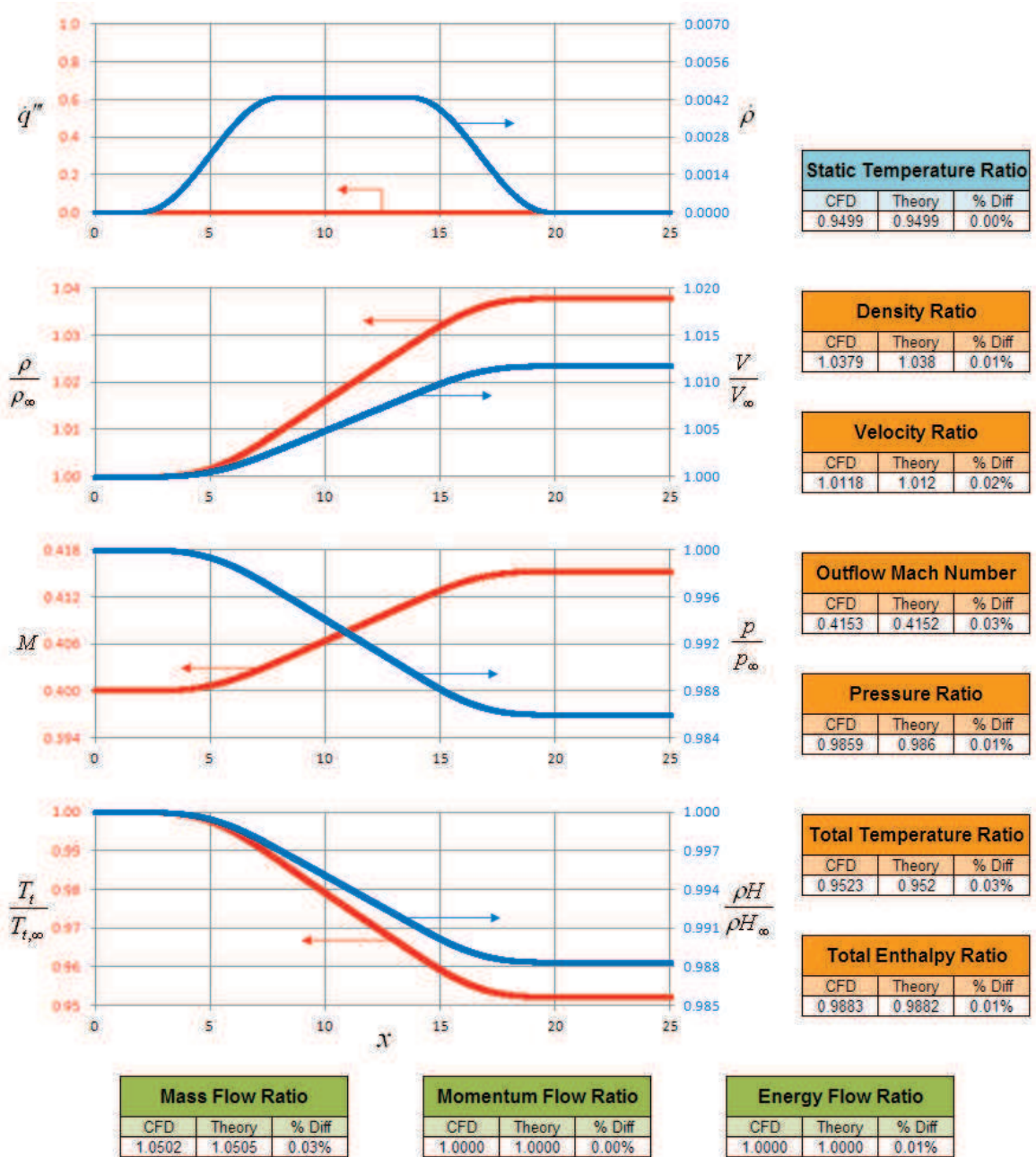


Figure 6.70: Subsonic (Mach 0.4) Cosine Mass Generation.

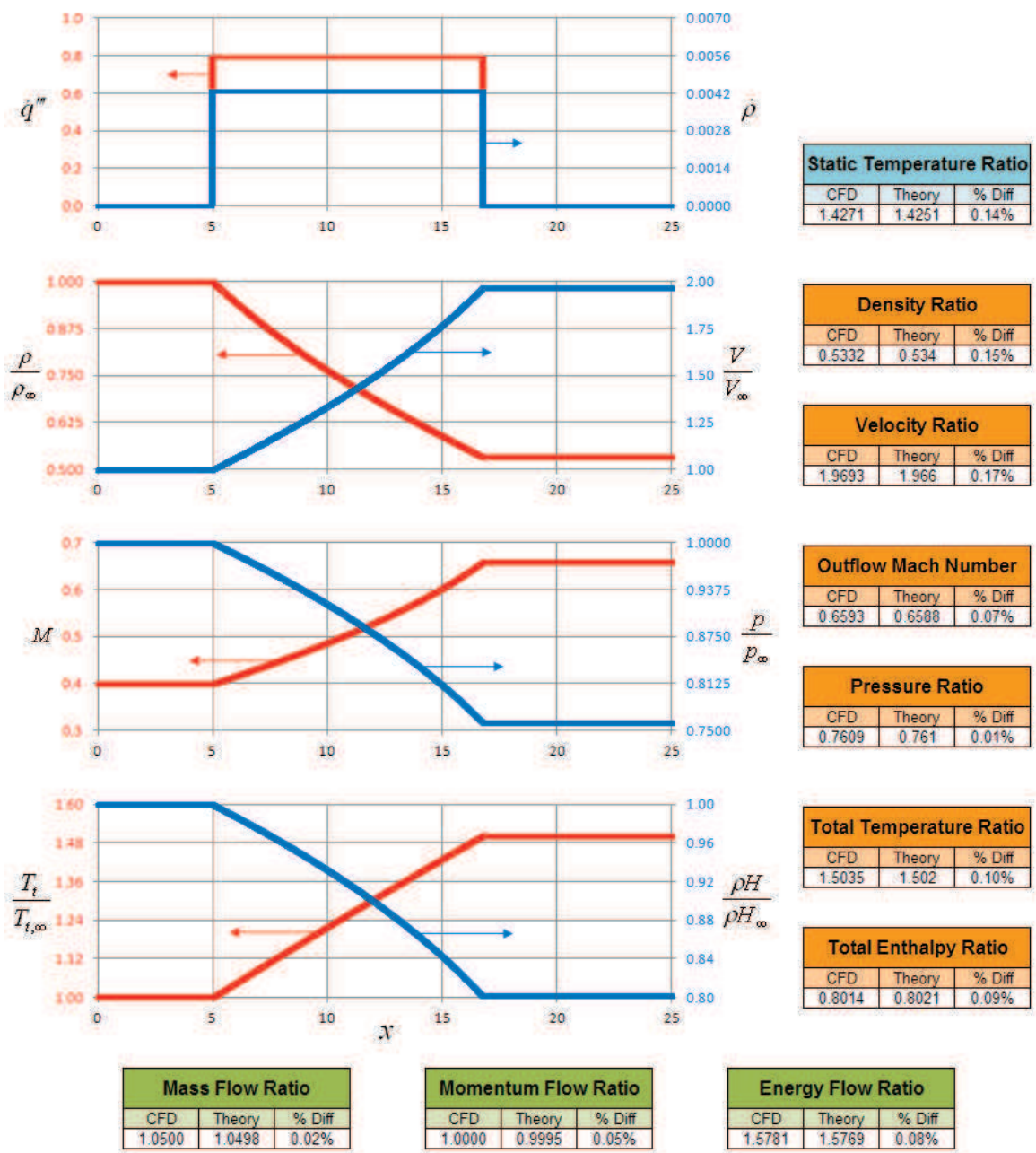


Figure 6.71: Subsonic (Mach 0.4) Constant Mass and Heat Generation.

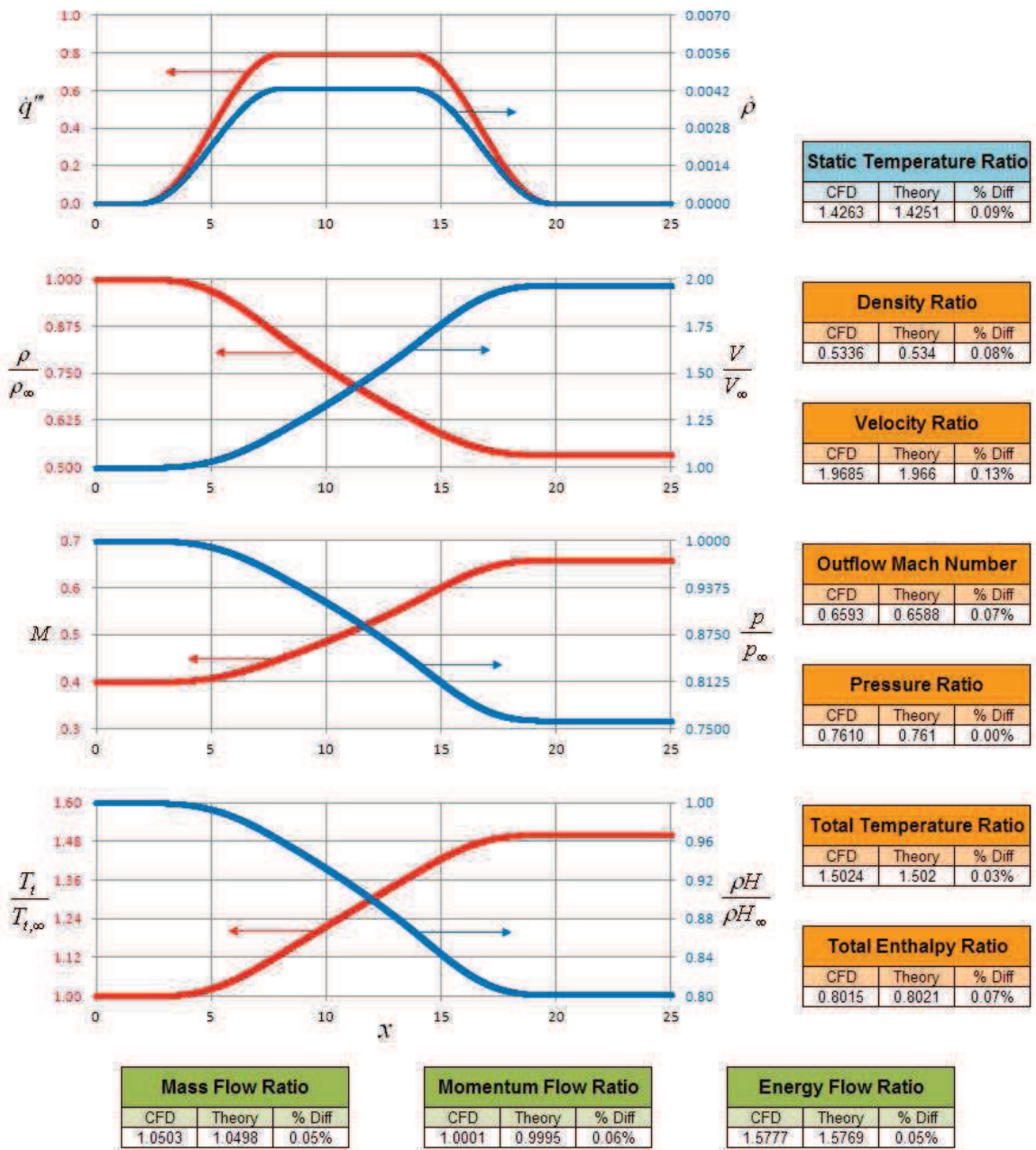


Figure 6.72: Subsonic (Mach 0.4) Cosine Mass and Heat Generation.

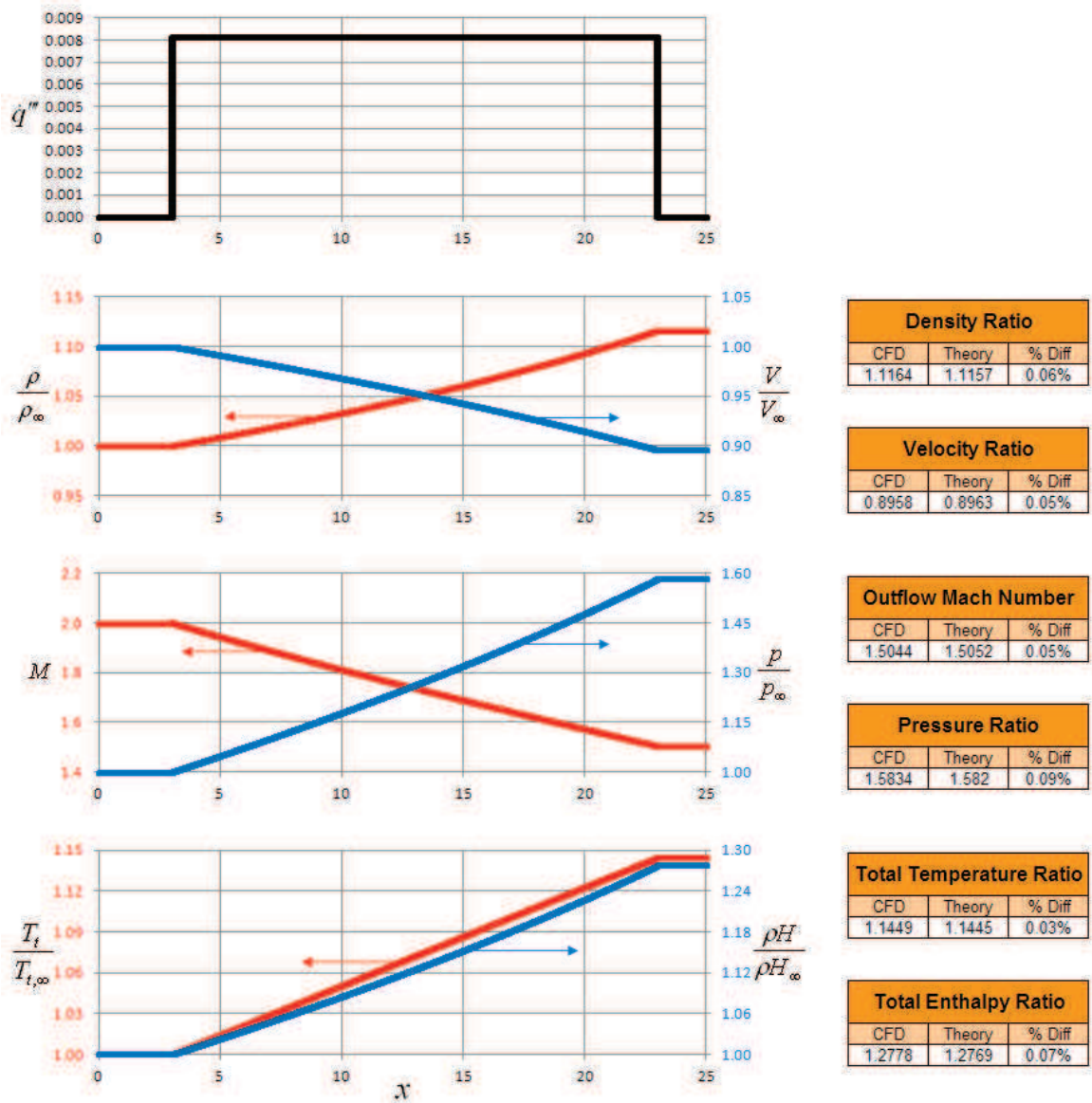


Figure 6.73: Supersonic (Mach 2.0) Constant Heat Generation.

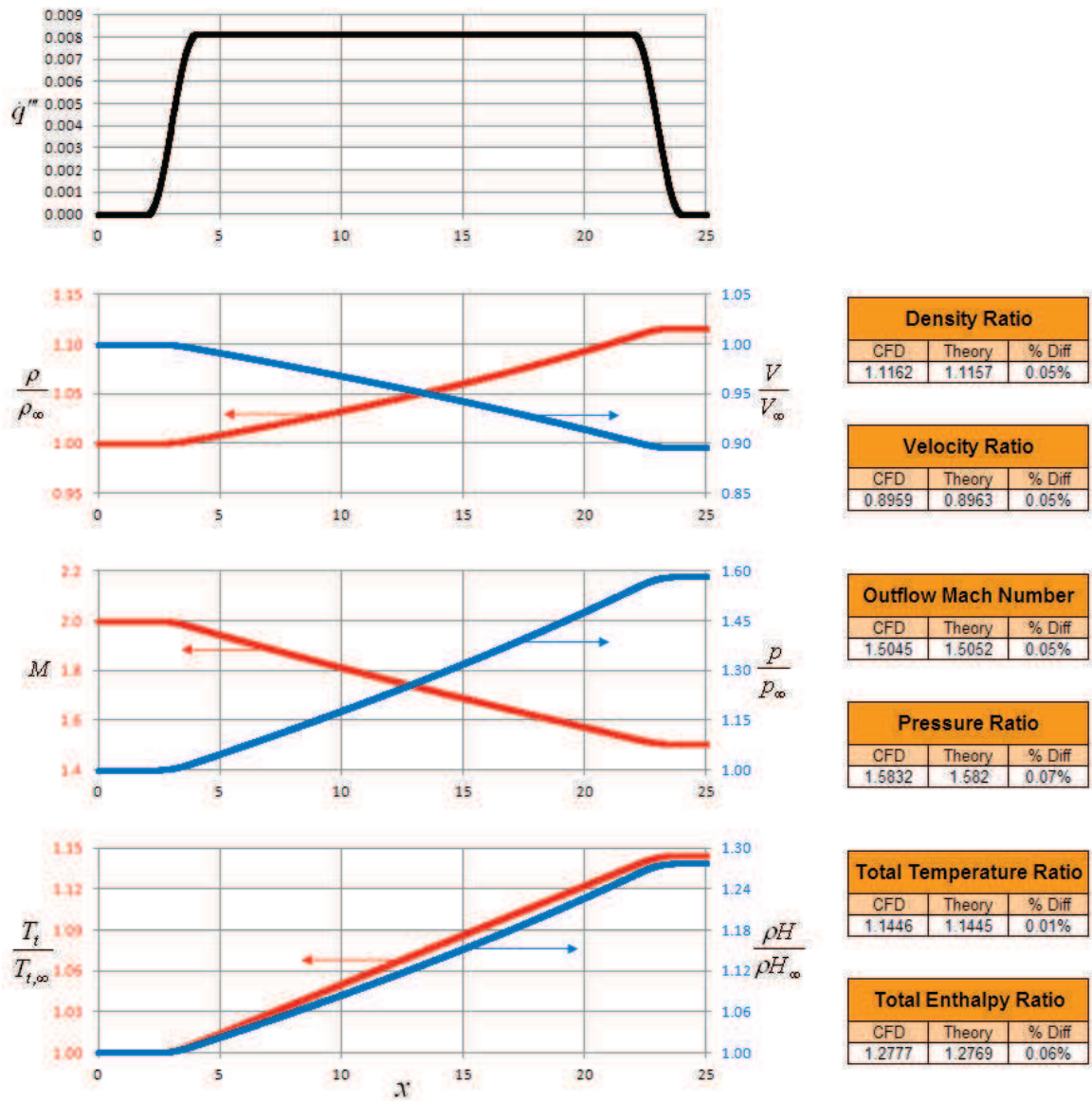


Figure 6.74: Supersonic (Mach 2.0) Cosine Heat Generation.

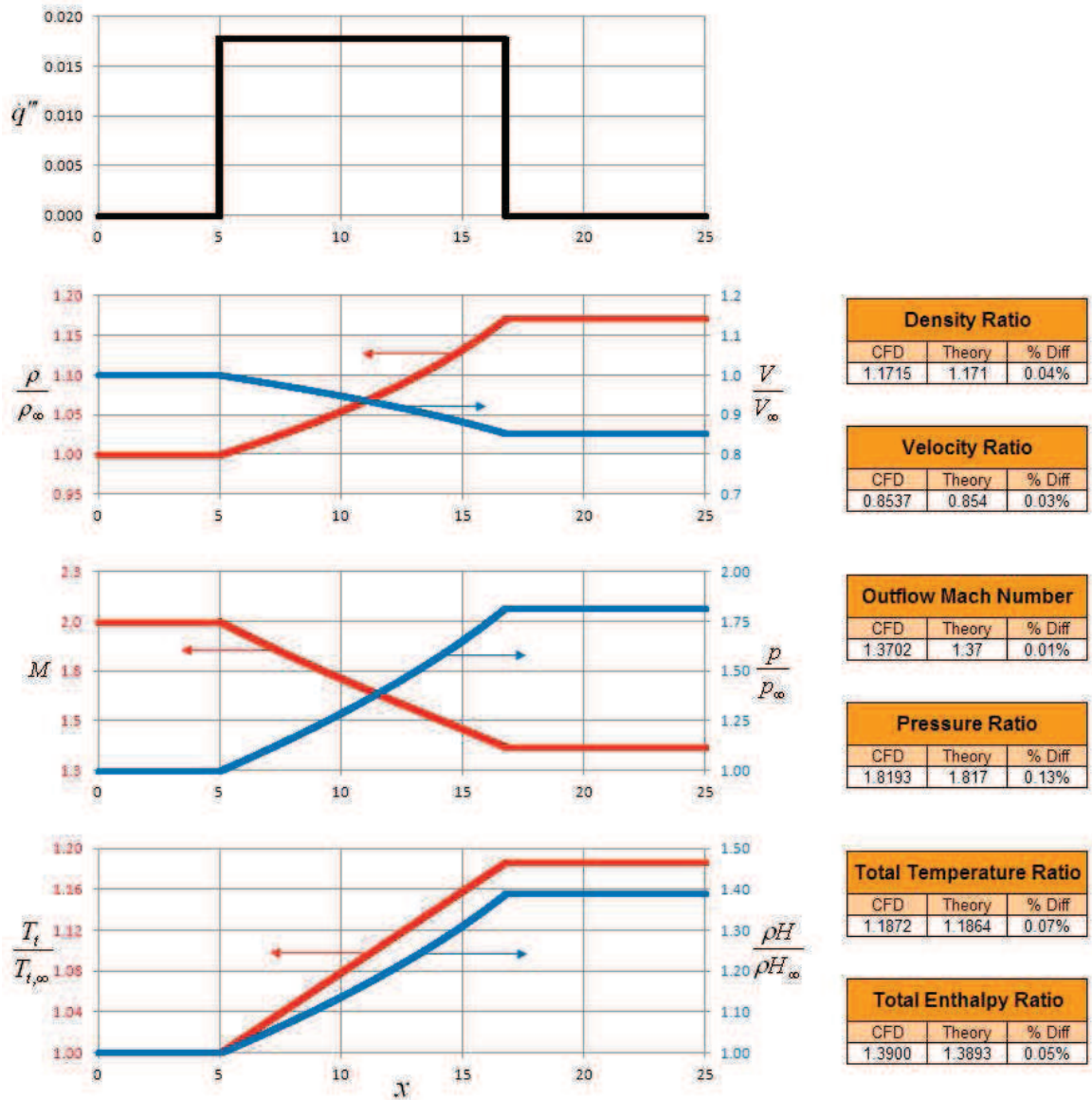


Figure 6.75: Supersonic (Mach 2.0) Constant Heat Generation.

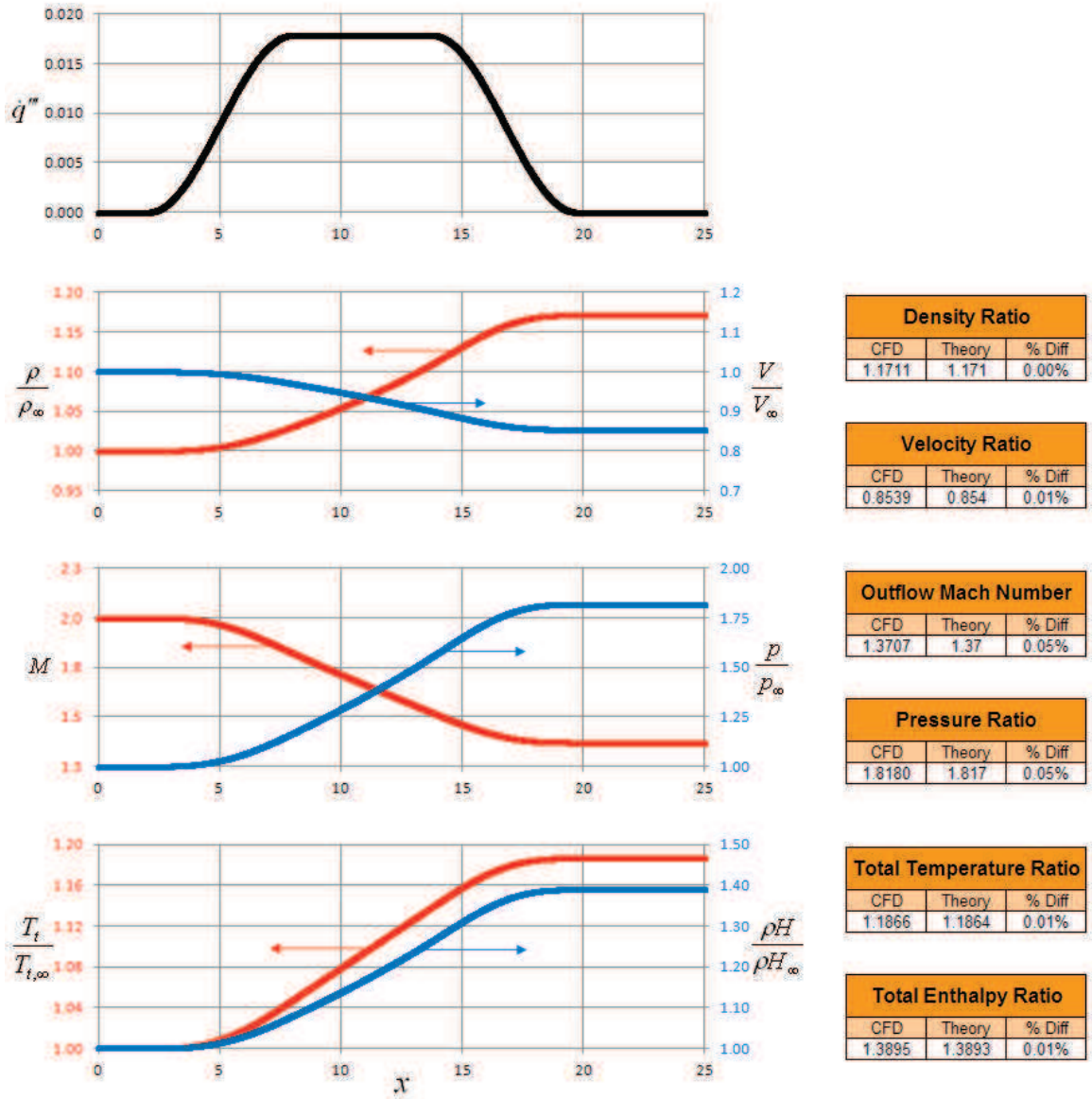


Figure 6.76: Supersonic (Mach 2.0) Cosine Heat Generation.



Figure 6.77: Supersonic (Mach 2.0) Constant Mass Generation.

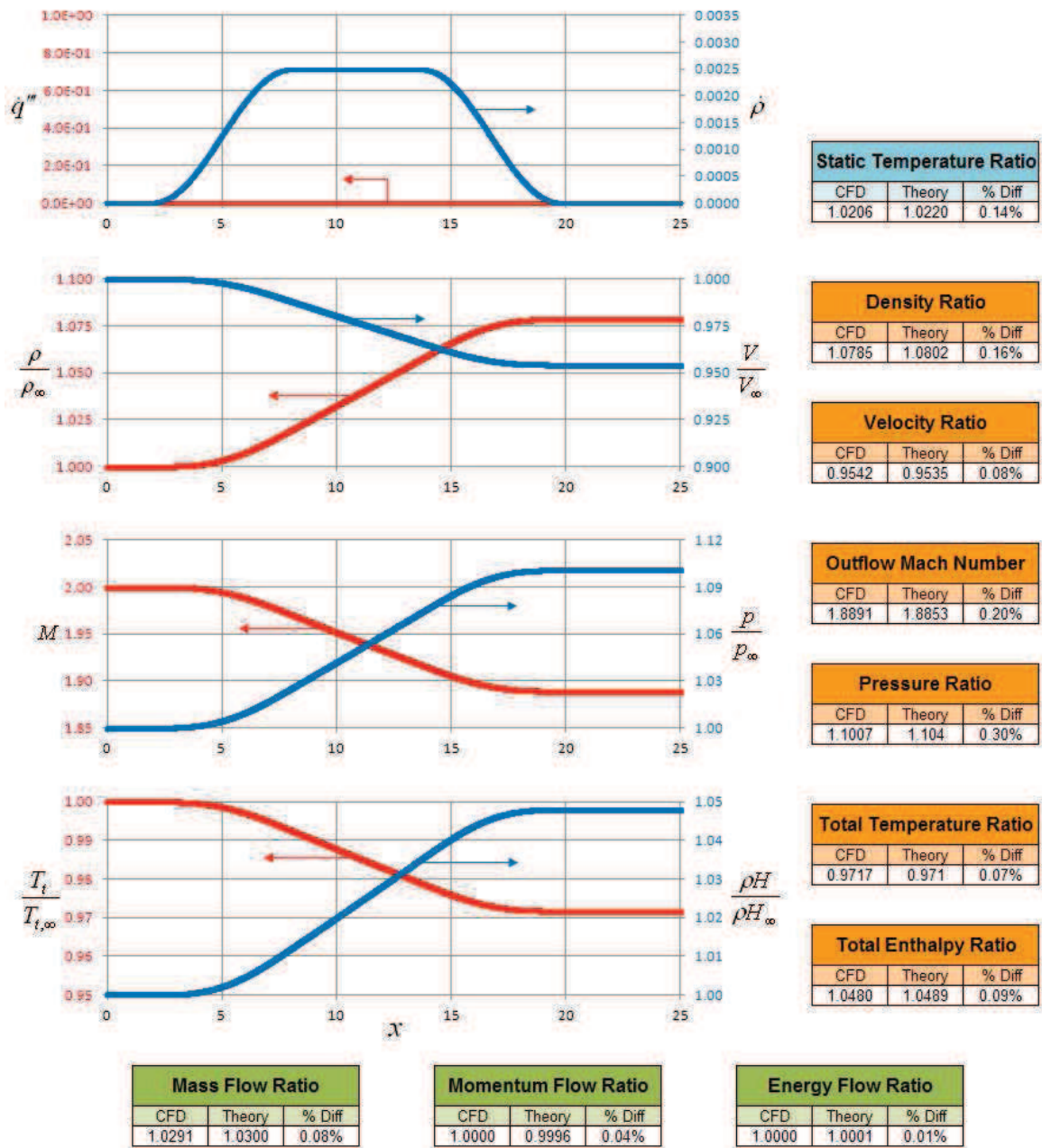


Figure 6.78: Supersonic (Mach 2.0) Cosine Mass Generation.



Figure 6.79: Supersonic (Mach 2.0) Constant Mass and Heat Generation.

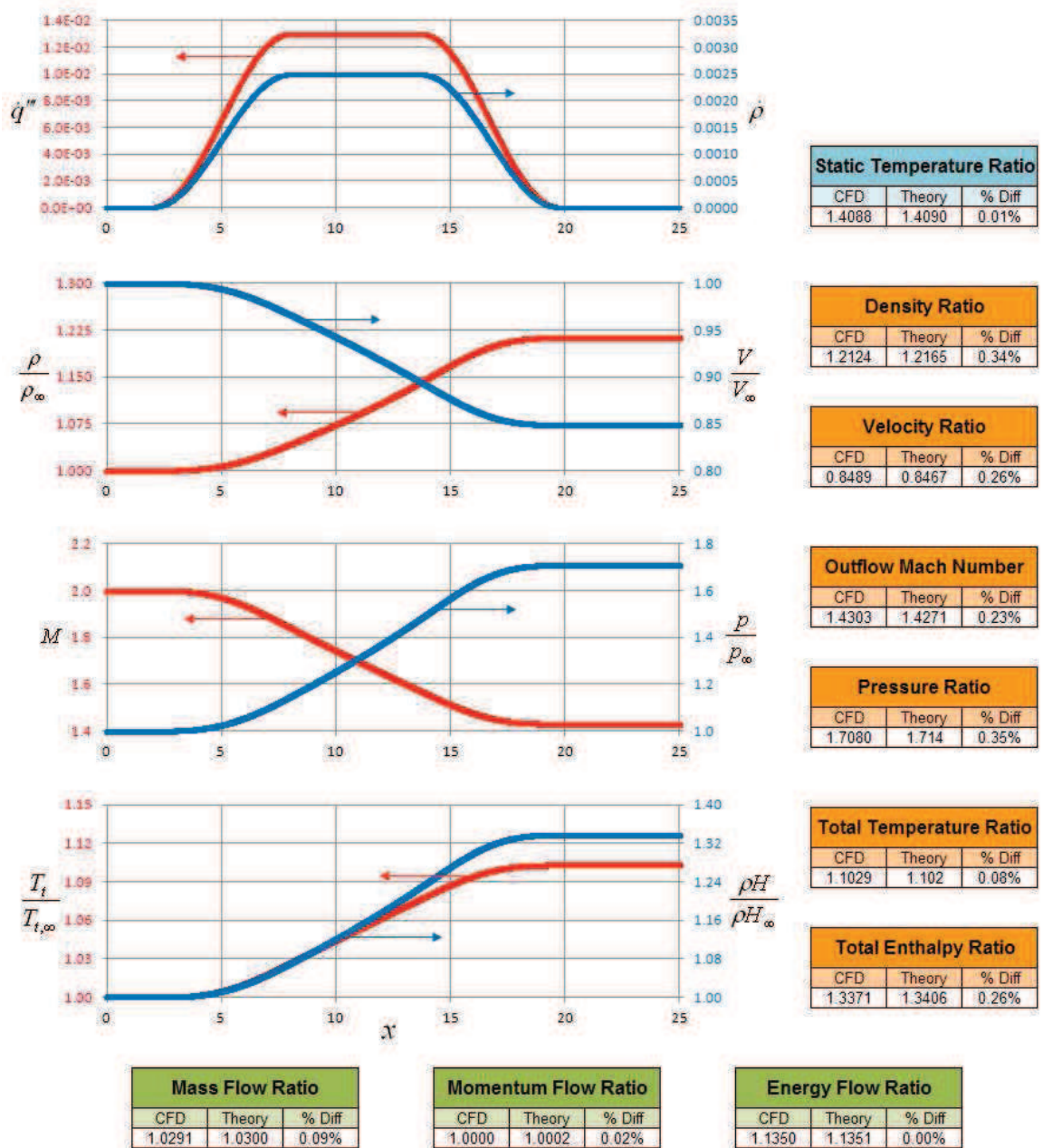


Figure 6.80: Supersonic (Mach 2.0) Cosine Mass and Heat Generation.

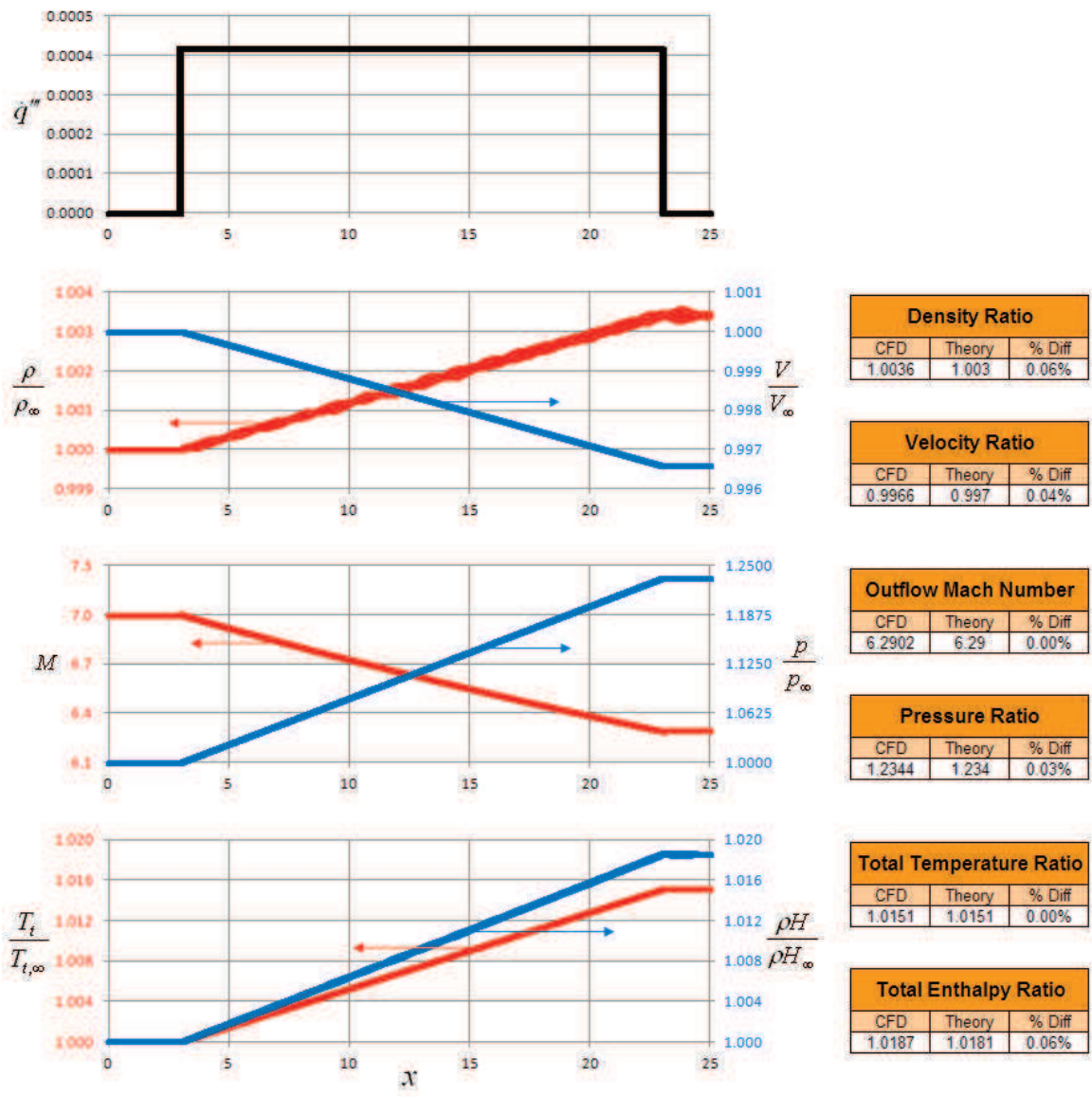


Figure 6.81: Supersonic (Mach 7.0) Constant Heat Generation.

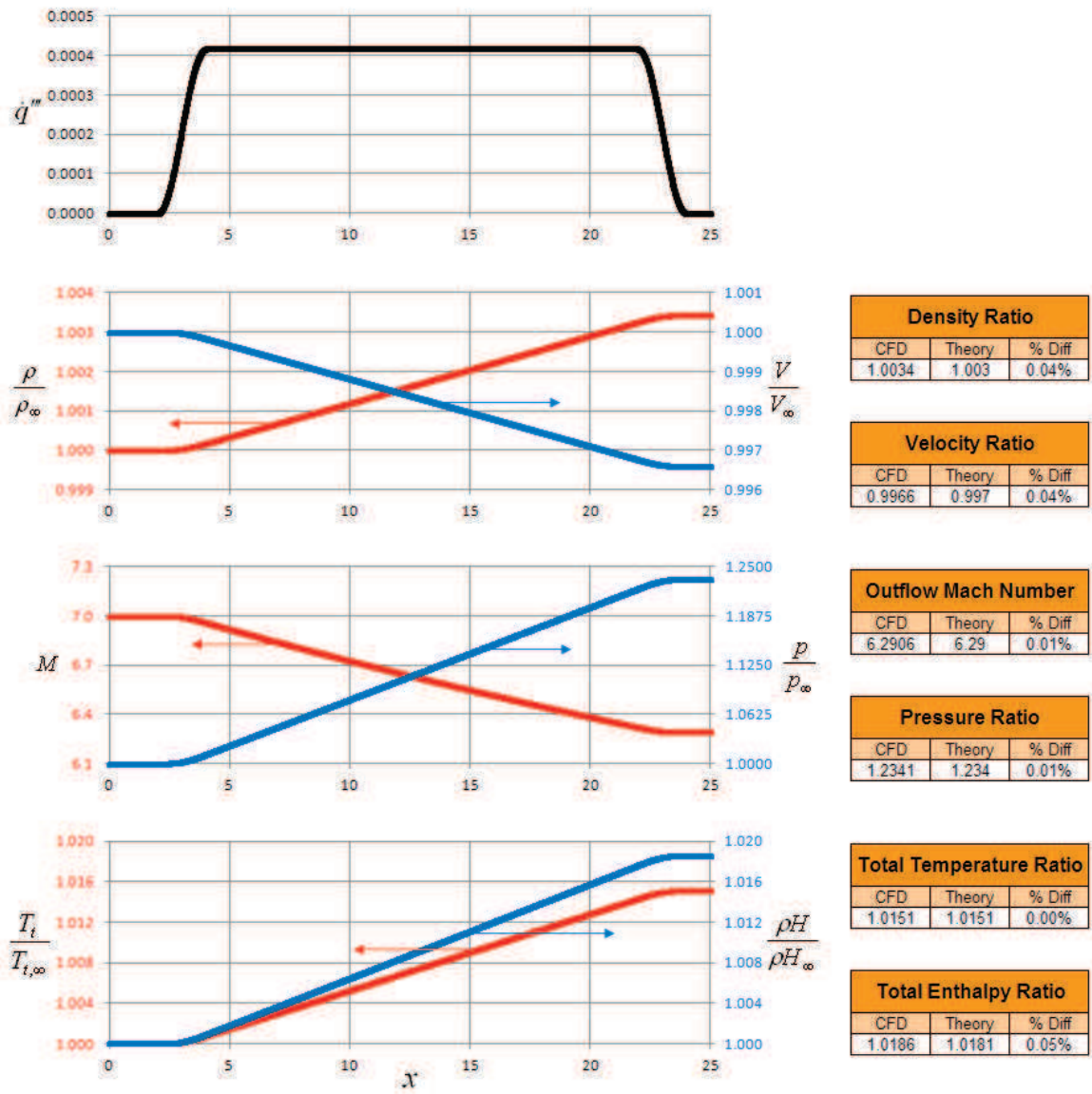


Figure 6.82: Supersonic (Mach 7.0) Cosine Heat Generation.

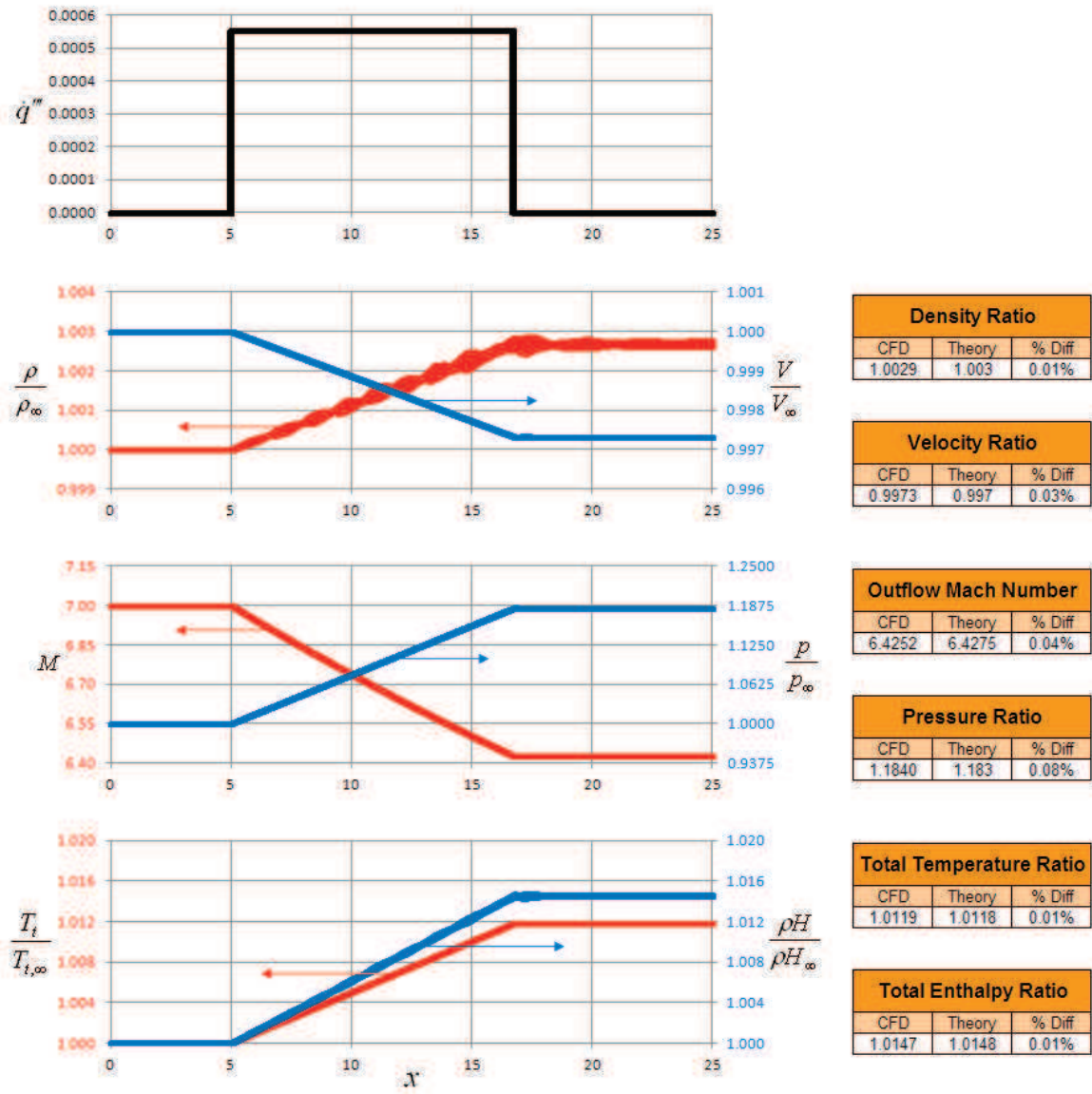


Figure 6.83: Supersonic (Mach 7.0) Constant Heat Generation.

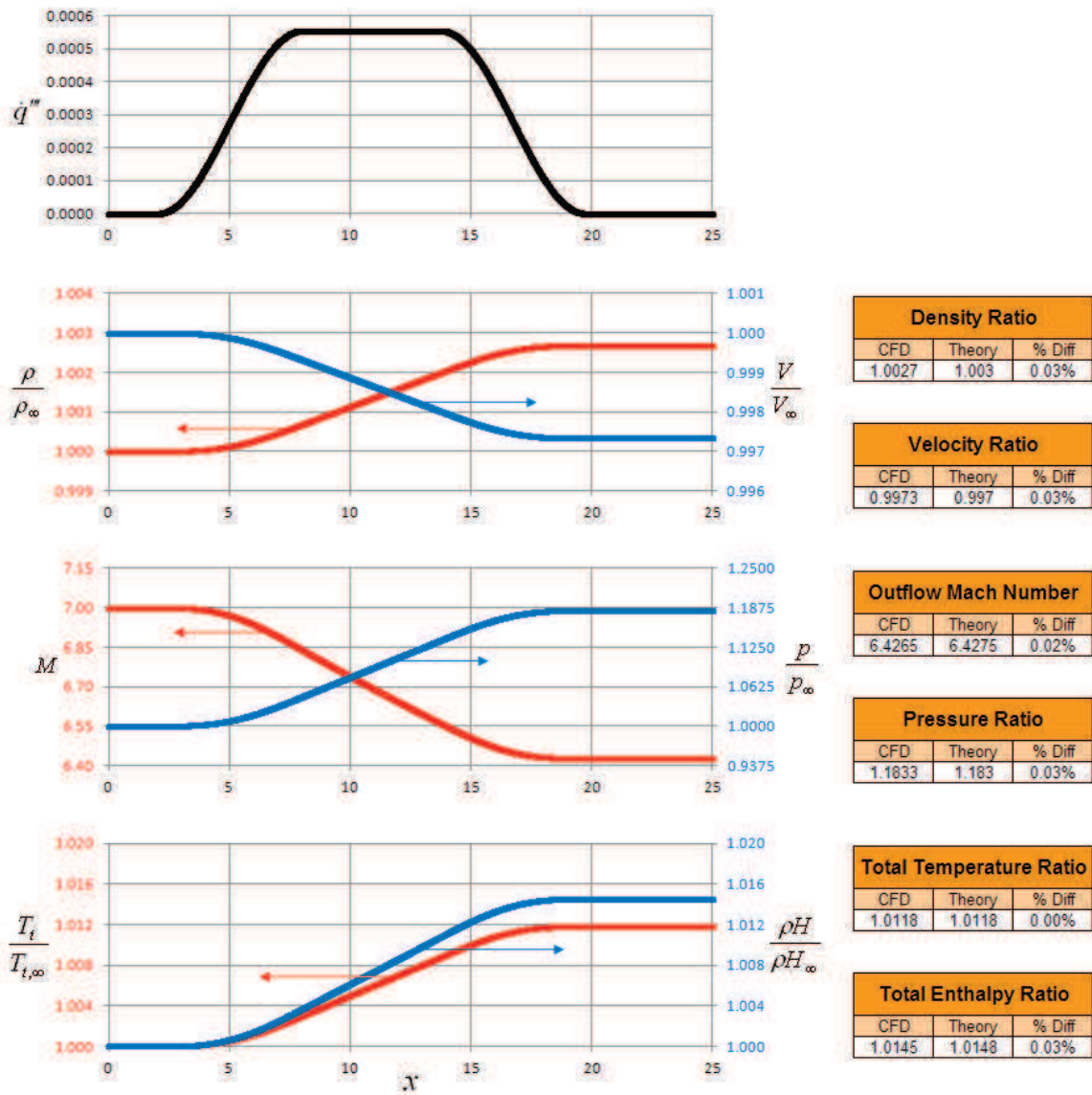


Figure 6.84: Supersonic (Mach 7.0) Cosine Heat Generation

6.2.1.4 GHV Cross-Section (Mach 11)

The capabilities of the combustion model are demonstrated on a generic hypersonic vehicle. The cross-section in Figure 6.85 represents the centerline of the hypersonic vehicle designed to operate at Mach 11. The flow used for propulsion passes through two oblique shocks, through the combustor, and then expands over the aft underbody of the vehicle. The vehicle is designed to produce a positive lift at design speeds.

Figure 6.85 shows the solution for both the combustor on and off. The amount of heat generation was increased in the on-condition until steady level operating conditions were reached (i.e. thrust equals drag). When the engine is on, the pressure in the combustion chamber increases significantly and the Mach number decreases, as expected. An initial estimate of the total heat generation along with temperatures and pressures along the surface of the vehicle have now been generated. If this model was being used to refine the early stages of a design, several concepts could be quickly tested at on- and off-design conditions, the stability of the system can be checked, and the combustion model can be refined as the combustor and its effects on the flow become more accurately accessed.

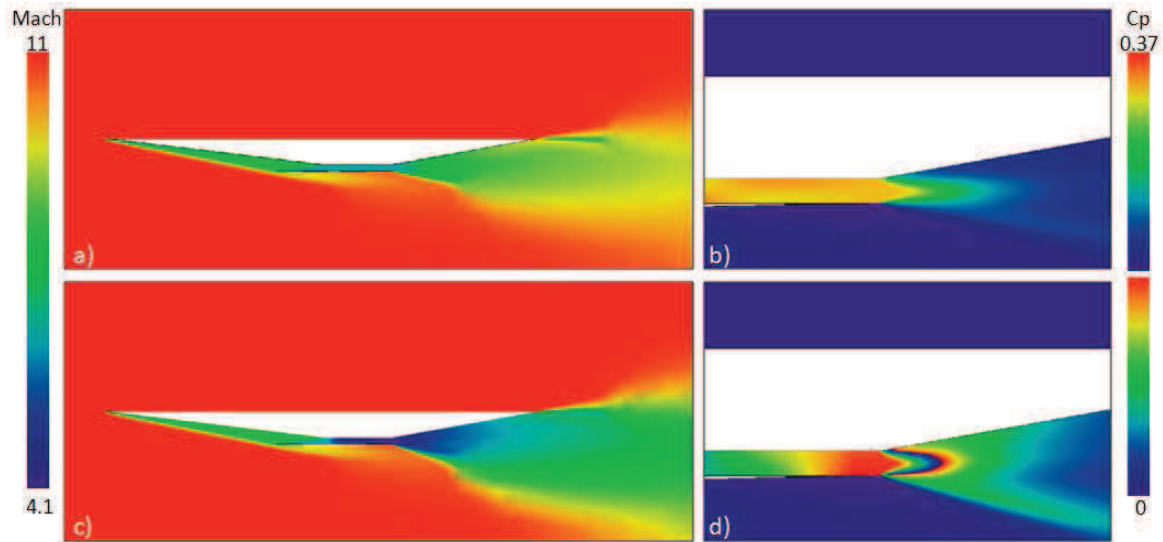


Figure 6.85: Flow around Generic Hypersonic Vehicle and Exit of Combustor.
a) Local Mach Number, Engine Off; b) Chamber Pressure, Engine Off;
c) Local Mach Number, Engine On; and d) Chamber Pressure, Engine On.

Table 6.7 compares the solution from Euler3D to shock-expansion theory (John, 1984). Zone 1 corresponds to the LE underbody, and zone 2 represents the properties in the combustion chamber. The solution compares very well with theory, within 10% on the pressure, Mach number, and angles.

Table 6.7: Comparison of GHV Solution to SE Theory.

Coeff. of Pressure			Local Mach #			Shock Angle		
Zone	C_p	% diff	Zone	M	% diff	---	(deg)	% diff
1	0.0296	7%	1	9.68	8%	LE, Top	10	10%
2	0.0460	-9%	2	6.70	4%	LE, Bot'm	5	2%

Walters (2009) used the new GHV to further demonstrate the capabilities of the combustion model. Walters added fuel and “heat” through perfect combustion distributed evenly through the chamber. The hydrogen was used to estimate the heat to mass ratio of the fuel. Fuel was added to the system to find the theoretical “net zero”, where the thrust balances the drag in steady flight. Walters was able to plot the net force in the x -direction versus the fuel flow

rate and determine an appropriate estimate to generate “net zero”. Walters also tested the engine-on GHV for Mach number and angle of attack stability. The GHV was rotated through angles from -2 to 2 degrees AOA and speeds from Mach 9.5 to 11.5. The drag coefficient increases with Mach number, showing that the configuration has speed convergence. The lift coefficient decreases with approximately the inverse of Mach number, which matches theory for very high speeds. The lift is nearly linear with the angle of attack.

Stability and Performance Estimates. The GHV was adapted to include two new traits: Extended nozzle and variable nozzle lip length. The capabilities of the quasi-combustion model were demonstrated on a generic hypersonic vehicle (GHV). This geometry is shown in Figure 6.86. The flow used for propulsion passes through two oblique shocks, through the combustor, and then expands over the aft underbody of the vehicle. Figure 6.87 shows the local Mach number around the vehicle at Mach 11 and zero angle of attack. Two trade studies were performed on the geometry: The length of the nozzle varied from 100 to 150% of the inlet length, and the length of the lower lip surface of the nozzle was varied from 25 to 75% of the length of the upper surface. Figure 6.88 and Figure 6.89 show the forces and moments created by the GHV under variations of nozzle length, lower lip length, and engine throttle (shown as heat generation q_c). The required amount of energy q_c is shown for each geometry tested along with the lift and moment created by the section at that throttle setting. Elongating the nozzle increases the height and drag of the vehicle so that a shorter nozzle is more desirable. A longer lower surface increases the thrust in the nozzle, but also increases the down force and nose-up moment. The thrust, drag, lift, and moment coefficients are:

$$C_T - C_D = \frac{T - D}{\frac{1}{2} \rho V^2 l_{body}} \quad C_L = \frac{L}{\frac{1}{2} \rho V^2 l_{body}} \quad C_M = \frac{M}{\frac{1}{2} \rho V^2 l_{body}^2}$$

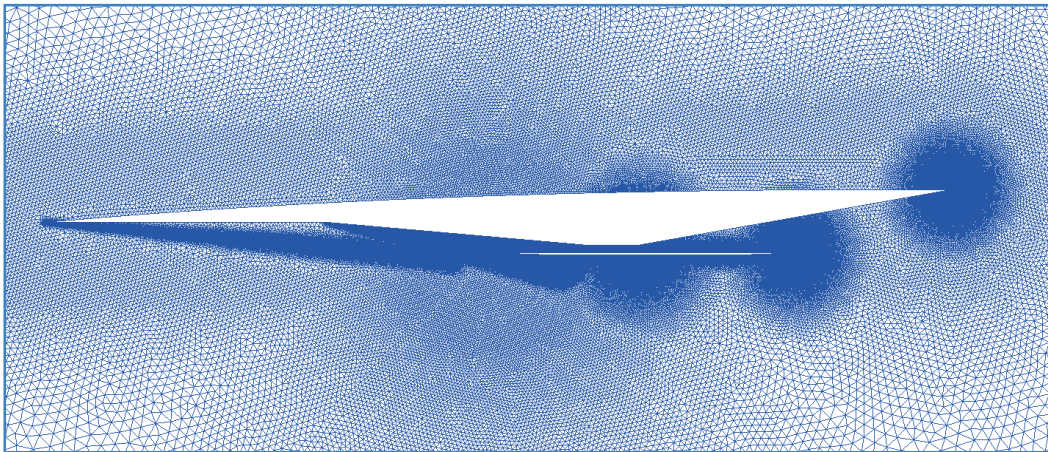


Figure 6.86: GHV Geometry and Mesh.

Babcock (2004) used CFD to estimate stability derivatives by isolating each parameter. This method has been used here to estimate the effects of flight speed, angle of attack, and pitch rate to produce their stability derivatives. Stability derivatives were estimated for a vehicle with 125% nozzle length and 50% lower surface. The data, shown in Figures 6.90, 6.91, and 6.92 shows the variation about the nominal flight conditions. The data was curve fit to second order polynomials and then analytically differentiated to calculate the derivatives shown in the figures.

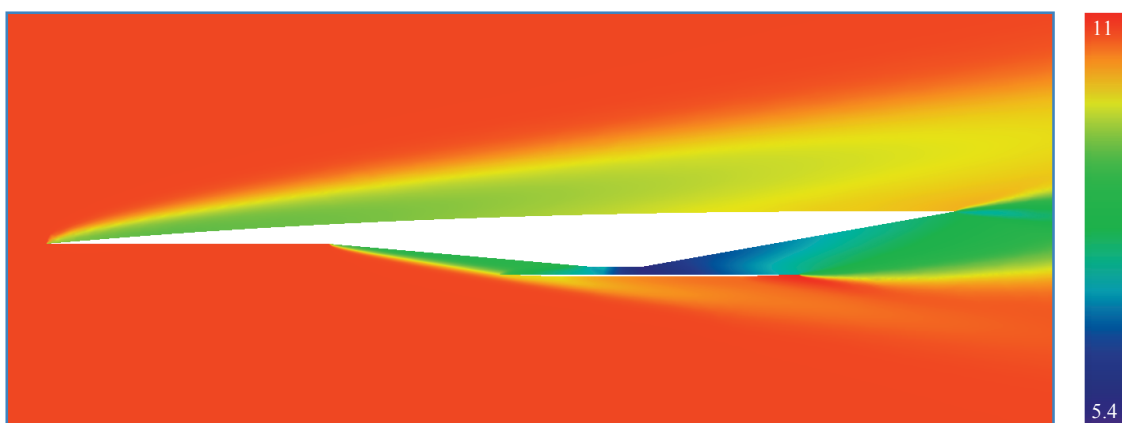


Figure 6.87: Local Mach Number around GHV at Mach 11 and 0° AOA.

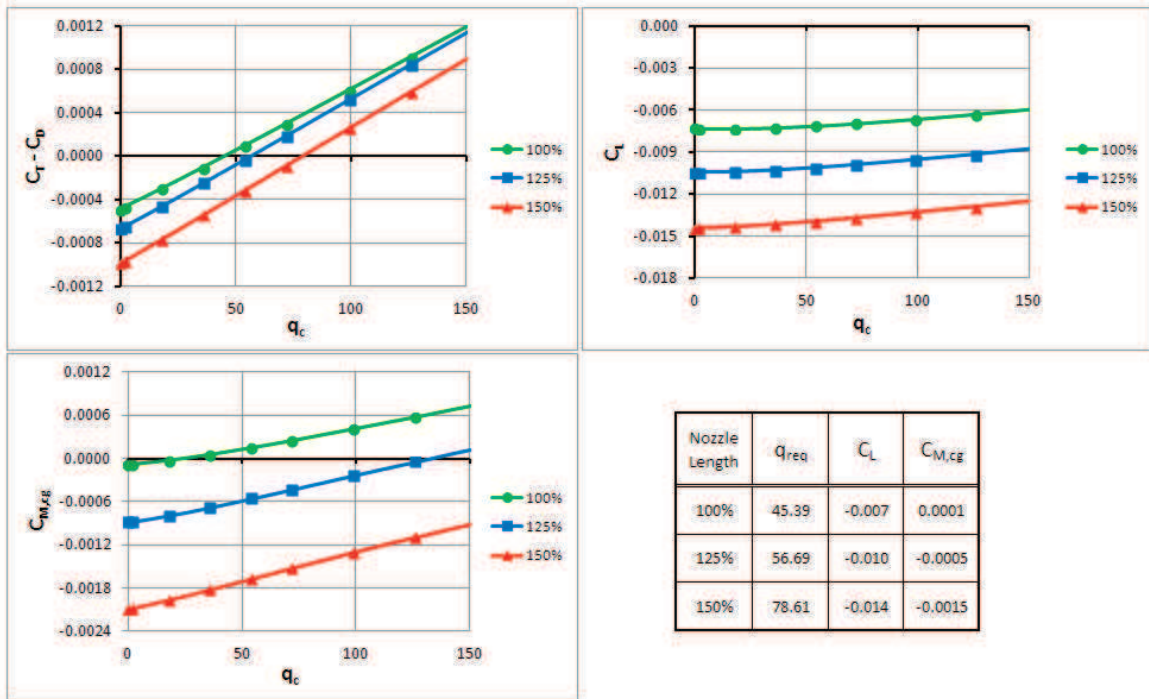


Figure 6.88: Variation of Force and Moment Coefficients with Nozzle Length (as Percent of Inlet Length).

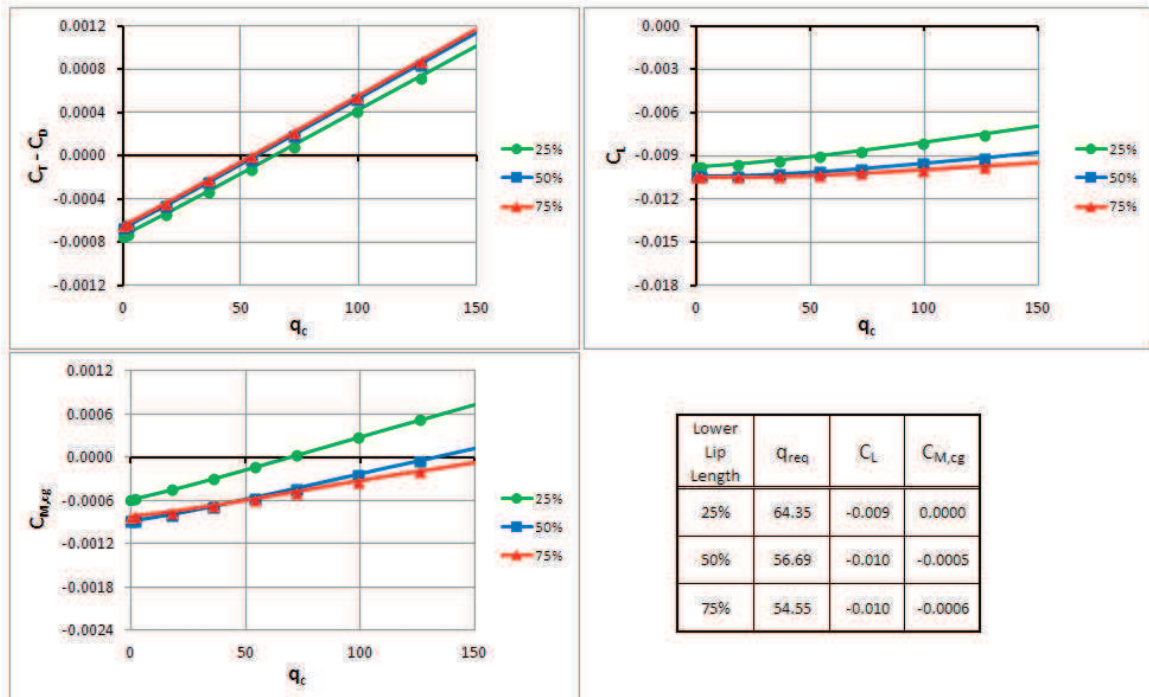


Figure 6.89: Variation of Force and Moment Coefficients with Lower Lip Length (as Percent of Upper Surface).

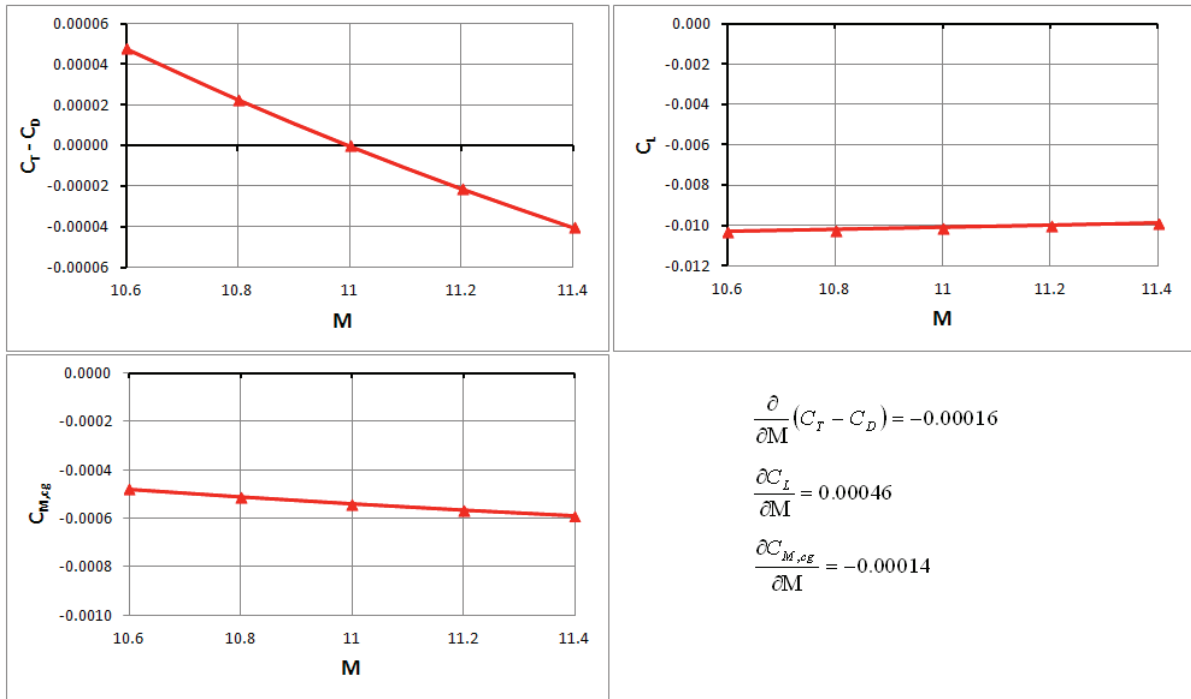


Figure 6.90: Variation of Force and Moment Coefficients with Mach Number with their Stability Derivatives.

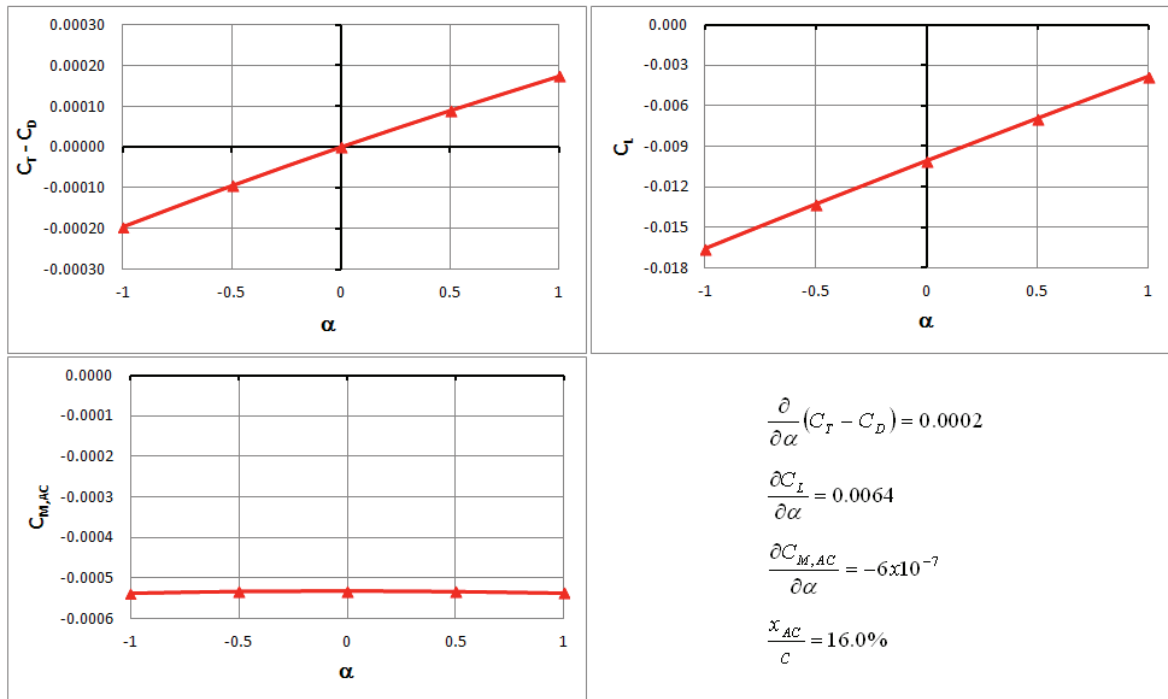


Figure 6.91: Variation of Force and Moment Coefficients with Angle of Attack with their Stability Derivatives.

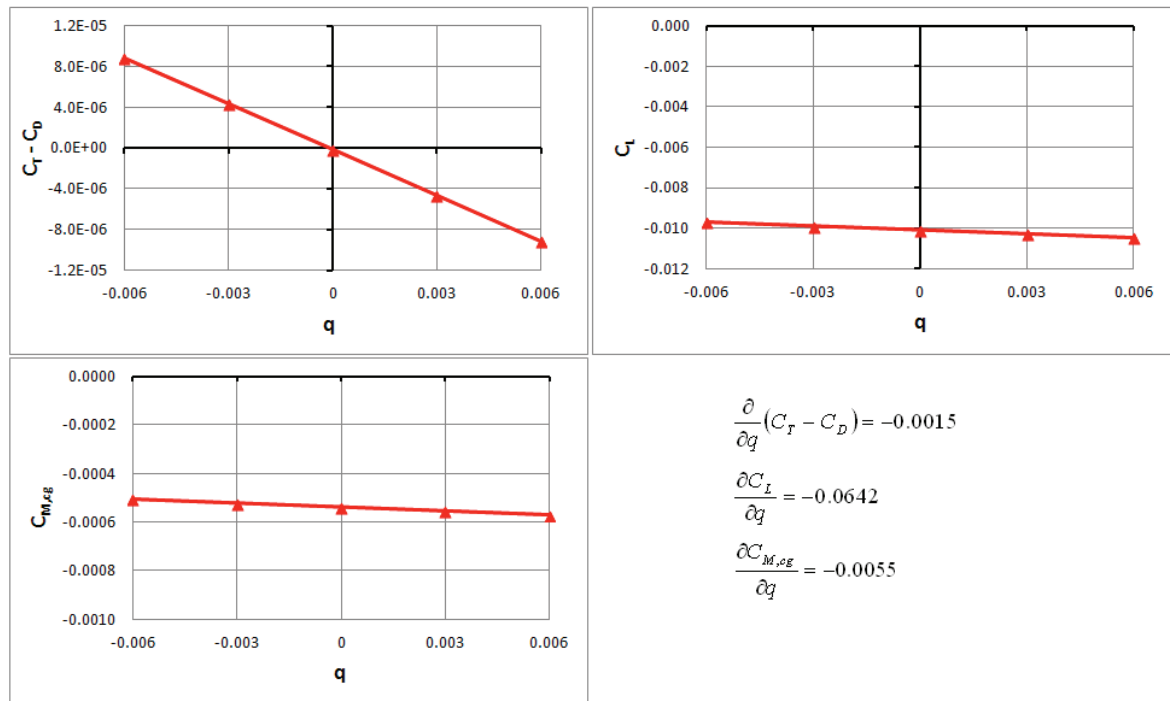


Figure 6.92: Variation Force and Moment Coefficients with Pitching Rate with their Stability Derivatives.

6.2.2 Rocket BC

A simple nozzle was designed using the method of characteristics (Emmons, 1958) to have a design Mach number of 2.2. The nozzle geometry can be seen in Figure 6.94. A sharp corner at the throat forms an expansion fan to help maintain isentropic flow at the design condition. The nozzle was tested at three back pressures: 84, 90, and 100% of the design pressure. Below the design condition, a normal shock appears in the diverging section. The position of the normal shock in Figure 6.93 is upstream of that predicted by theory. Figure 6.95 shows that the shocks take on a curved shape, where the shock is perpendicular to the wall and the centerline. The curvature of the shock enlarges the area where the shock occurs. The CFD data in Figure 6.93 shocks at the same Mach number as the theory line because the two-dimensional flow curves the effective cross-section, increasing their area. As the shock

moves through the nozzle, the effective cross-sectional area approaches the actual cross-sectional area.

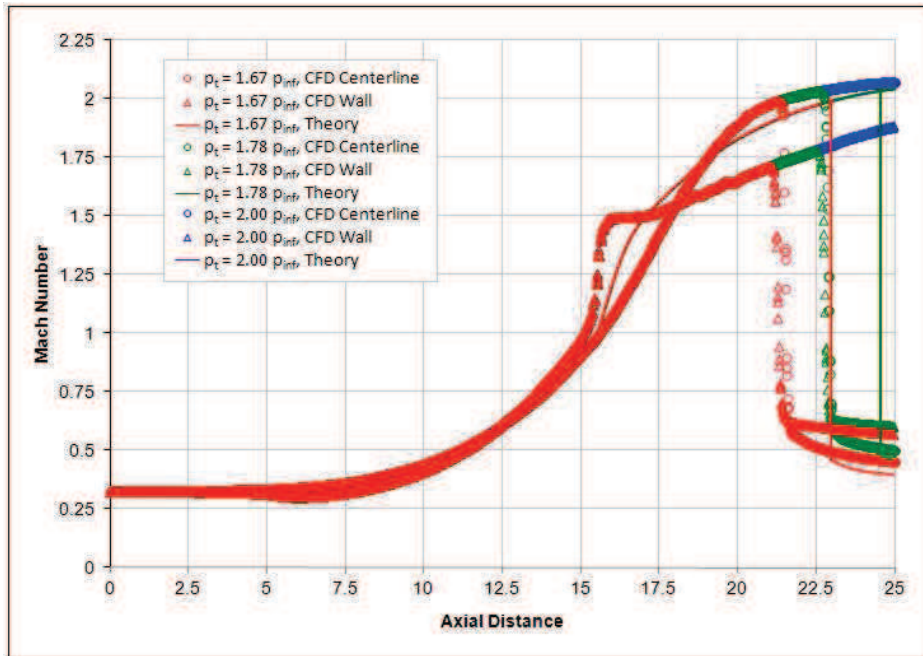


Figure 6.93: Local Mach along Centerline and Wall Surface of Rocket Nozzle.

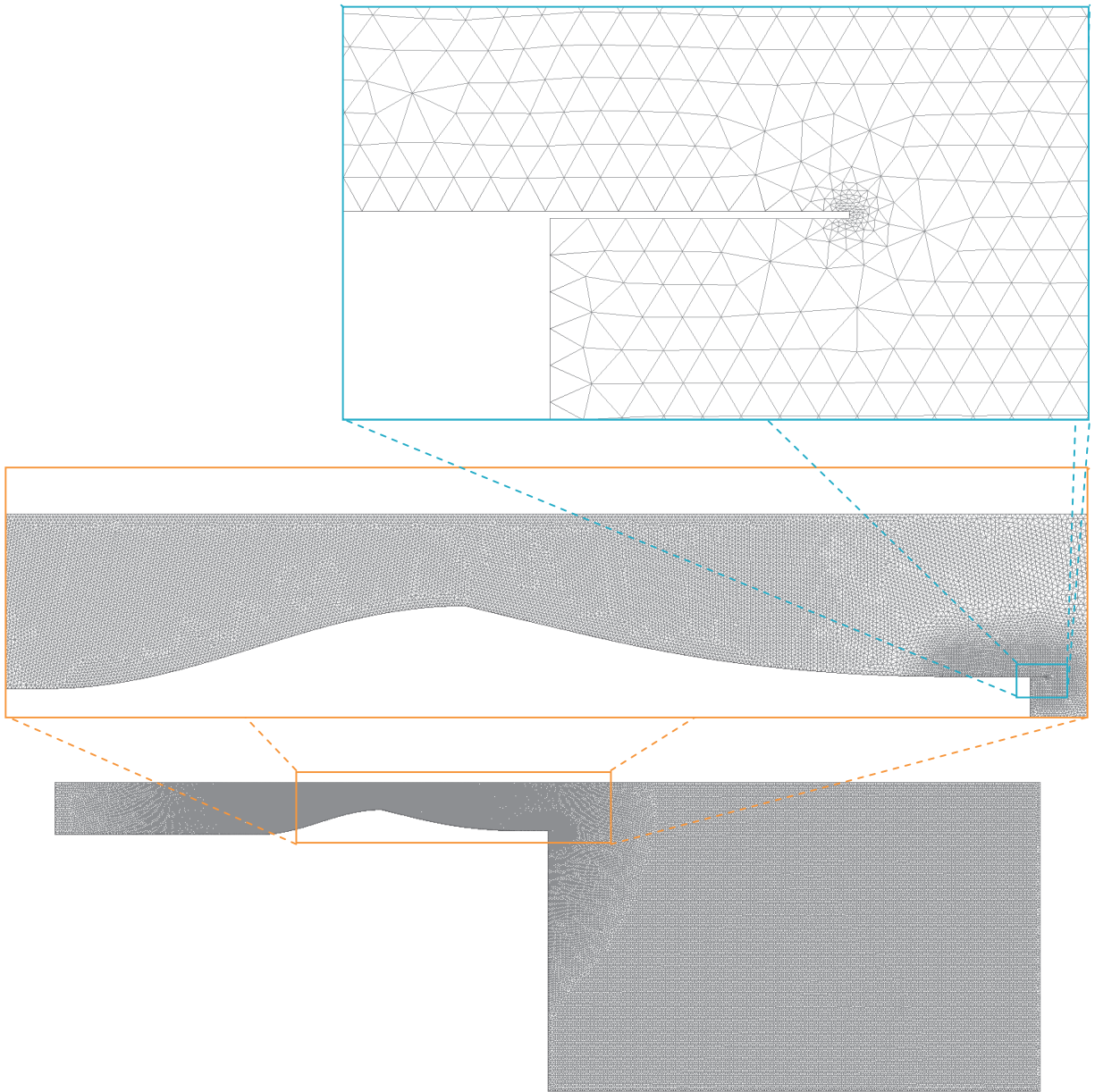


Figure 6.94: Rocket Nozzle Mesh.

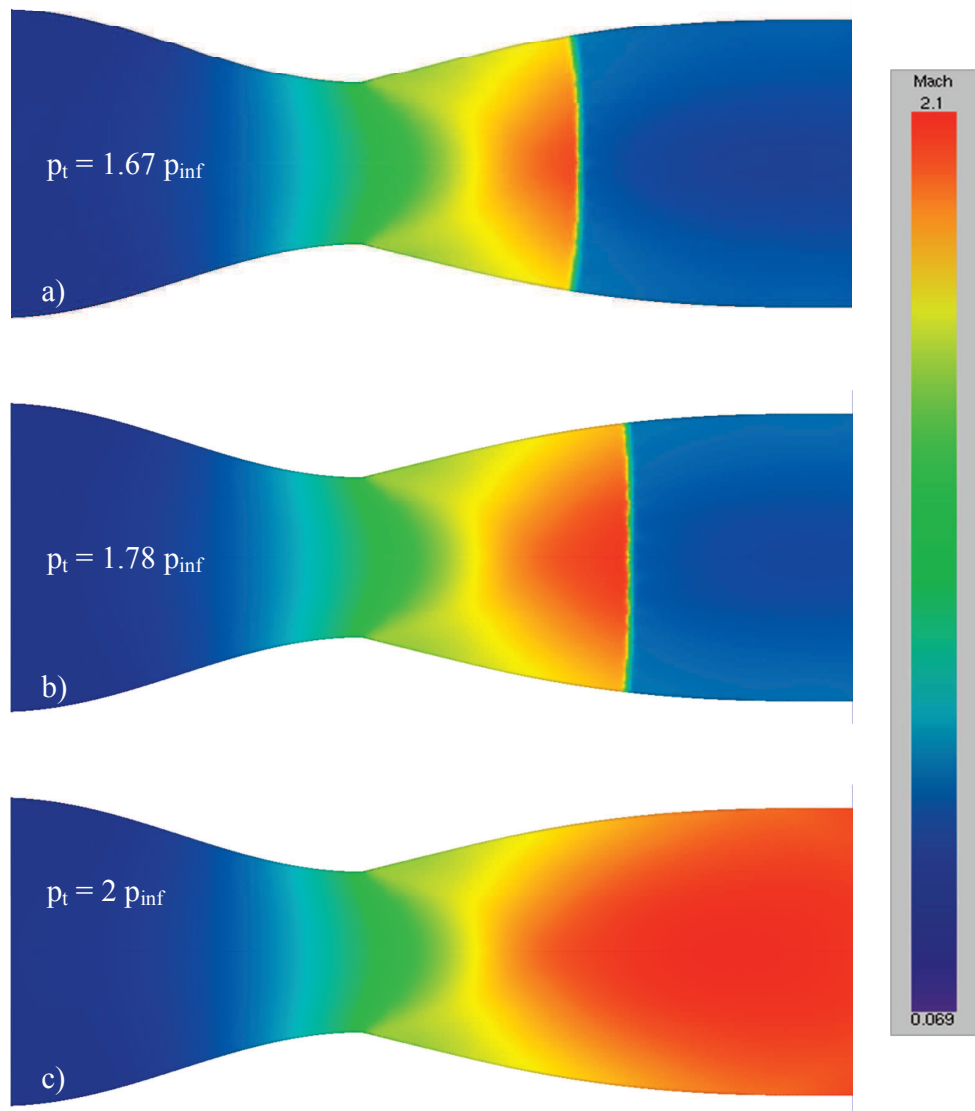


Figure 6.95: Local Mach Number within Rocket Nozzle:
a) $p_t = 1.67 p_{inf}$, b) $p_t = 1.78 p_{inf}$, and c) $p_t = 2 p_{inf}$.

6.2.3 Engine BC

Three engine inlets are used to demonstrate how the engine inflow boundary conditions operate. A subsonic inlet is designed with smooth leading edges and operated at on- and off-design conditions. The off-design conditions show changes in the incoming stream tube due to freestream velocity and movement of the stagnation points with angle of incidence. Two supersonic inlets are demonstrated: The first inlet uses a normal shock to slow the flow to subsonic speeds before entering the engine. The other inlet creates an oblique shock and normal shock in combination to slow the flow. Spillover was tested for both supersonic inlets and is illustrated for the oblique shock inlet. A coupled engine is assembled from the subsonic inlet and rocket nozzle. The coupled engine was used to demonstrate proper coupling. Two engines in the same simulation were used to show that the coupling connections were not crossed for multiple engines.

6.2.3.1 Subsonic Engine Inflow

A two-dimensional engine inlet was designed with a smooth leading edge to encourage subsonic operations. The inlet was designed to allow a stream tube the size of the inlet leading edges at a freestream Mach number of 0.6. The mass flow rate generated by this stream tube is designated as the design mass flow. The engine was demonstrated at on- and off-design conditions, including at an angle of attack. Figure 6.96 shows the mesh used to model all cases. The pressure, Mach number, and internal energy (temperature) distributions are also shown in Figure 6.96. From the local Mach number distribution, the flow stagnates on the leading edge of the inlet and then accelerates over the inner lip. The area past the lip

increases so the flow slows down inside of the inlet. A second stagnation point occurs on the tip of the spinner.

The mass flow rate was held at the design mass flow, and the freestream Mach number (and hence velocity) was varied. Figure 6.97 shows the local Mach number distributions around the inlet at freestream Mach 0.5, 0.6, and 0.7. An approximate stream tube is also drawn on each picture. The stream tubes narrow at higher speeds and widen at lower speeds as expected to maintain the mass flow rate. The stagnation point also shifts to accommodate the approach angle of the stream tube.

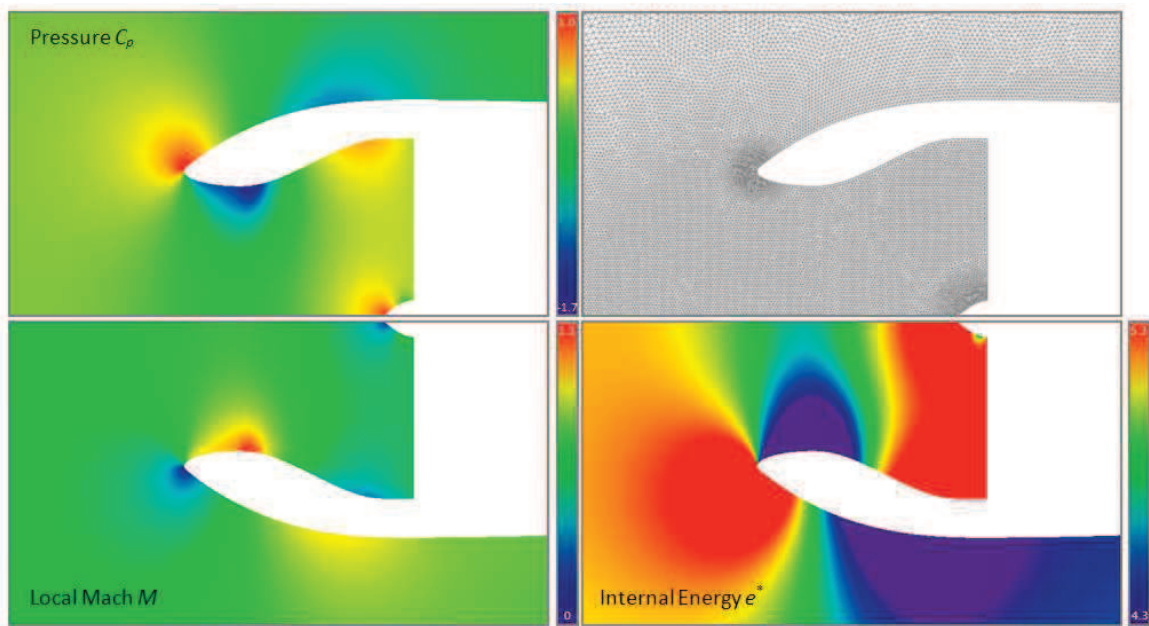


Figure 6.96: Pressure (C_p), Mach, and Internal Energy for Subsonic Inlet at Mach 0.6.

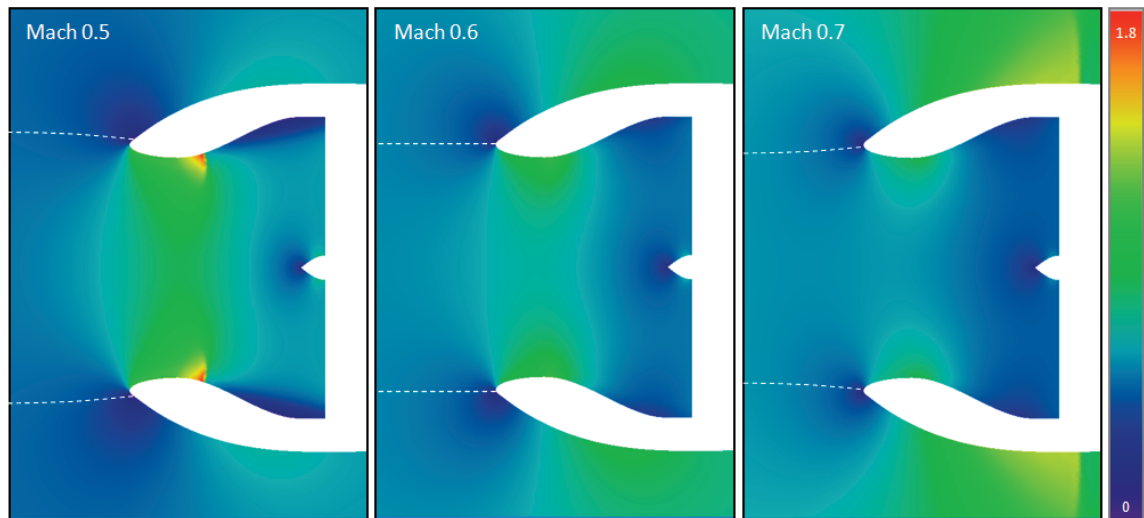


Figure 6.97: Local Mach Number at Various Freestream Velocities.

Figure 6.98 shows the subsonic inlet, pulling in the design mass flow, at 0 and 4-degrees of incidence. The stagnation points both shift downward because of the rotation of the flow. The velocity along the lower lip accelerates well into the transonic region exhibiting a shock; a transonic shock also appears on the top surface of the cowl. The flow normalizes near the engine inflow plane.

6.2.3.2 Supersonic Engine Inflow (Normal Shock)

Another two-dimensional inlet was designed with a sharp leading edge to encourage a normal shock at supersonic operations. The inlet was designed to allow a stream tube the size of the inlet leading edges at a freestream Mach number of 1.5; the mass flow rate generated by this stream tube is designated as the design mass flow. The engine was demonstrated at on- (Mach 1.5) and off-design (Mach 0.7) conditions, shown in Figure 6.99. Spillover at supersonic speeds occurs by pushing the normal shock in front of the inlet lips, allowing some of the mass flow to move around the inlet instead of through it. Spillover was

demonstrated in this research, but pictures of the case are not shown here. Spillover is more clearly seen on the next inlet.

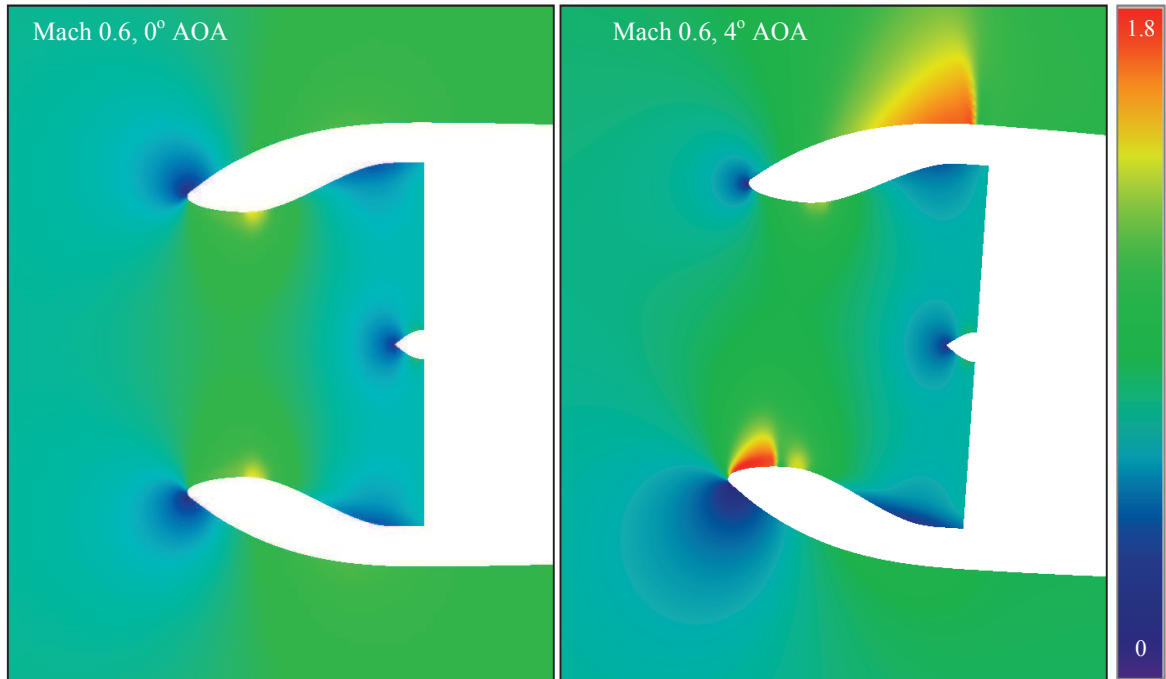


Figure 6.98: Local Mach Number at 0 and 4-Degrees Angle-of-Attack.

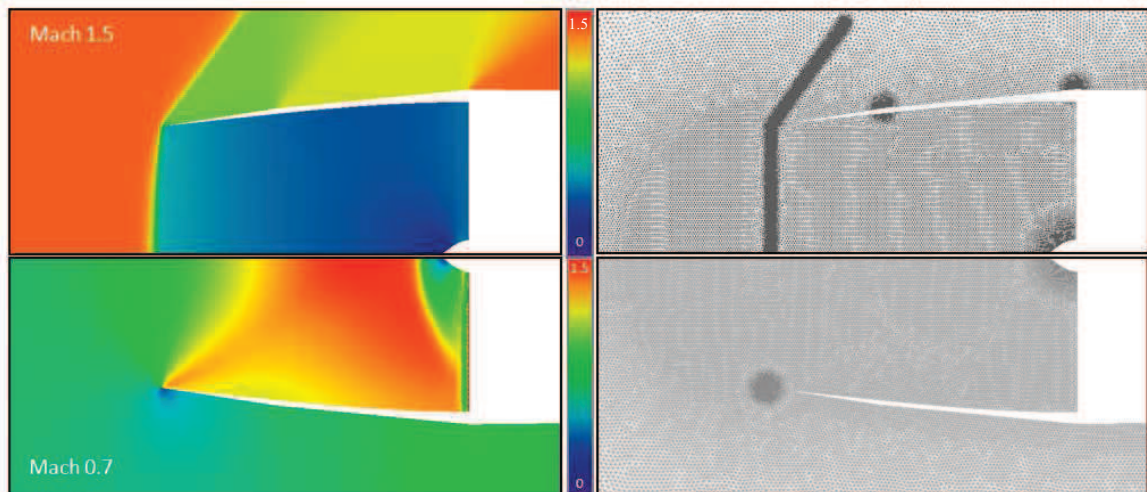


Figure 6.99: Supersonic Normal Shock Inlet at Mach 1.5 (top) and Mach 0.7 (bottom).

6.2.3.3 Supersonic Engine Inflow (Oblique Shock)

A third two-dimensional inlet was designed with a protruding spinner that generates oblique shocks at supersonic speeds. At Mach 2, the inlet is designed so that the oblique shock strikes the inlet lip. A normal shock at the inlet lip slows the flow before entering the inlet. The mass flow rate generated by this stream tube between the inlet lips is designated as the design mass flow. Figure 6.100 shows the mesh used for the oblique shock inlet. Figure 6.101 shows the inlet operating at Mach 2 at both the design mass flow rate and 98% of that mass flow. A dashed box is marked on the pictures, designating where the zoomed view was taken. The flow near the inlet lip shows that the oblique shock and normal shock coalesce at the design conditions. At a lower mass flow rate, spillover occurs by shifting the normal shock forward. The spillover shows that the mass flow rate is maintained by the inflow boundary condition, even at supersonic speeds.

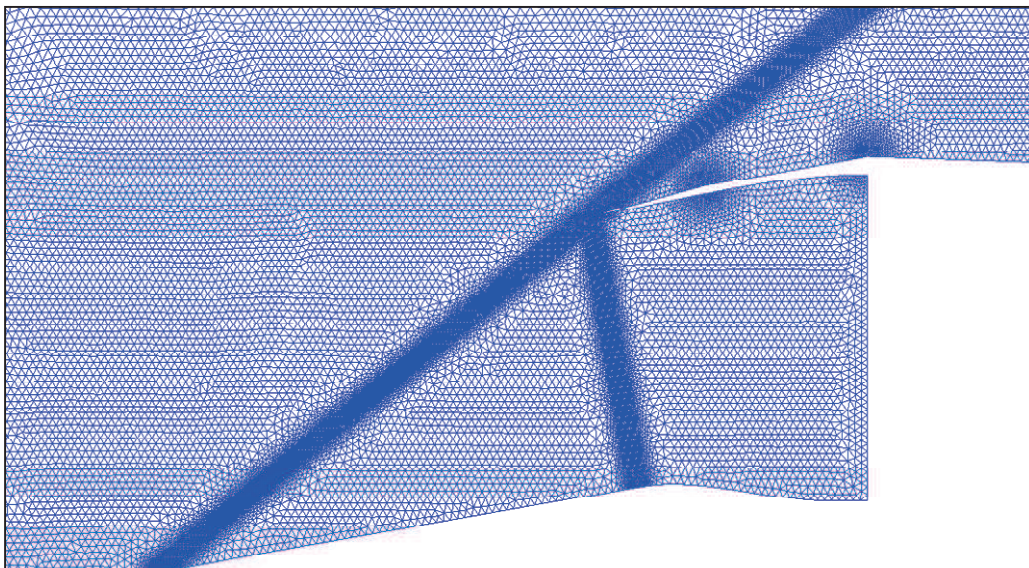


Figure 6.100: Mesh for Supersonic Oblique Shock Inlet.

6.2.3.4 Engine Outflow Cases

The subsonic inlet was coupled together with the rocket nozzle to create the turbojet model shown in Figure 6.102. Figure 6.103 shows the converged pressure and local Mach distributions around the coupled turbojet. The distributions within inlet resemble Figure 6.96 at the same conditions, and the nozzle is similar to the rocket nozzle shown in Figure 6.95. The conditions upstream of the nozzle are not exactly the same as created by the rocket boundary condition, so the exhaust flow contains shocks showing that the flow is under-expanded.

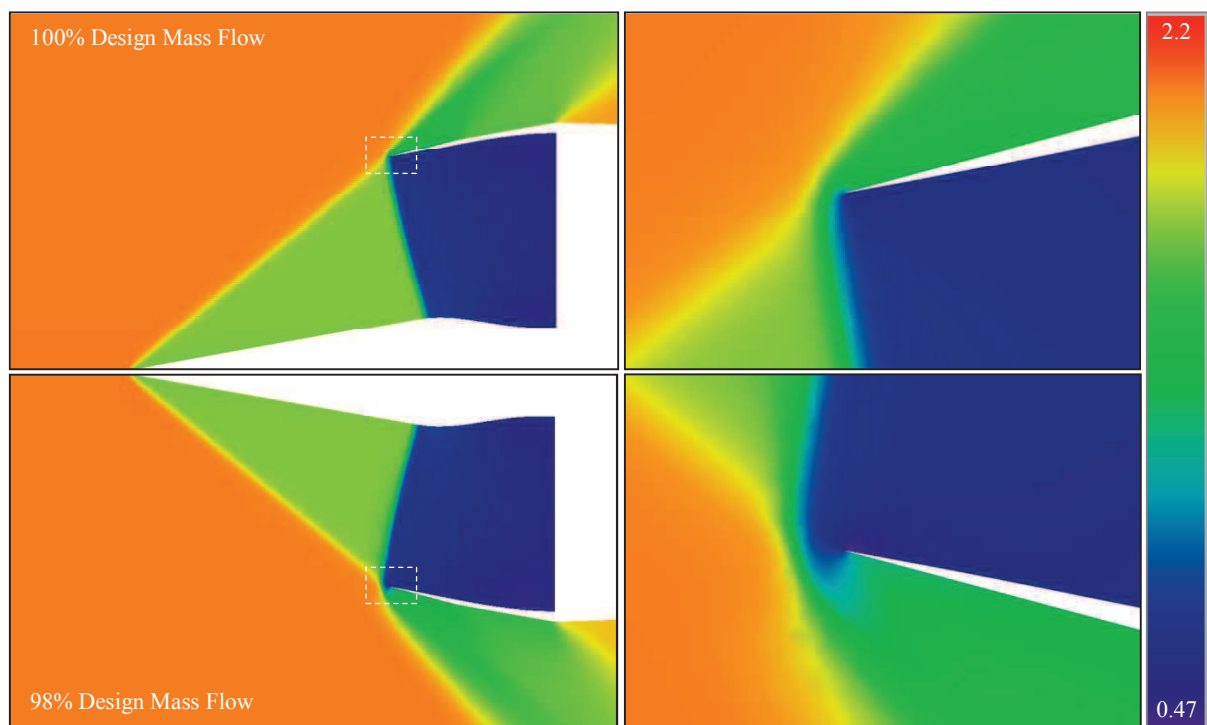


Figure 6.101: Local Mach Number for Oblique Shock Inlet at 100% (top) and 98% (bottom) Design Mass Flow Rates (Mach 2.0).

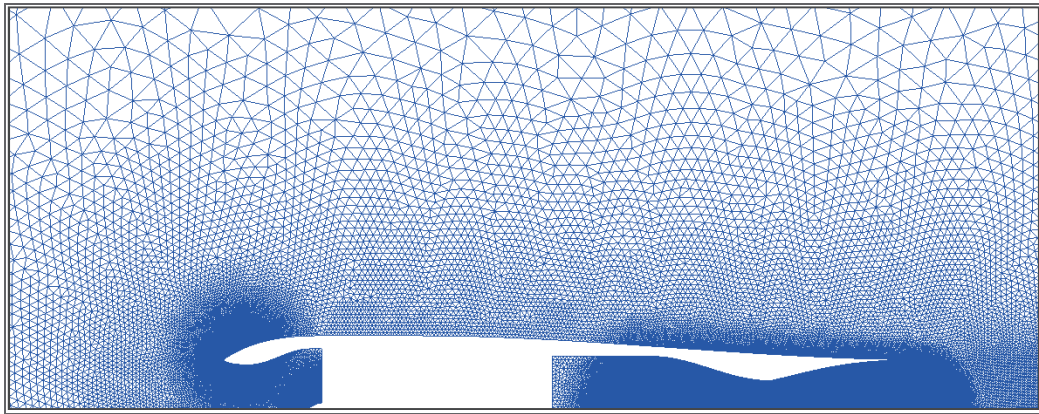


Figure 6.102: Mesh for Coupled Turbojet.

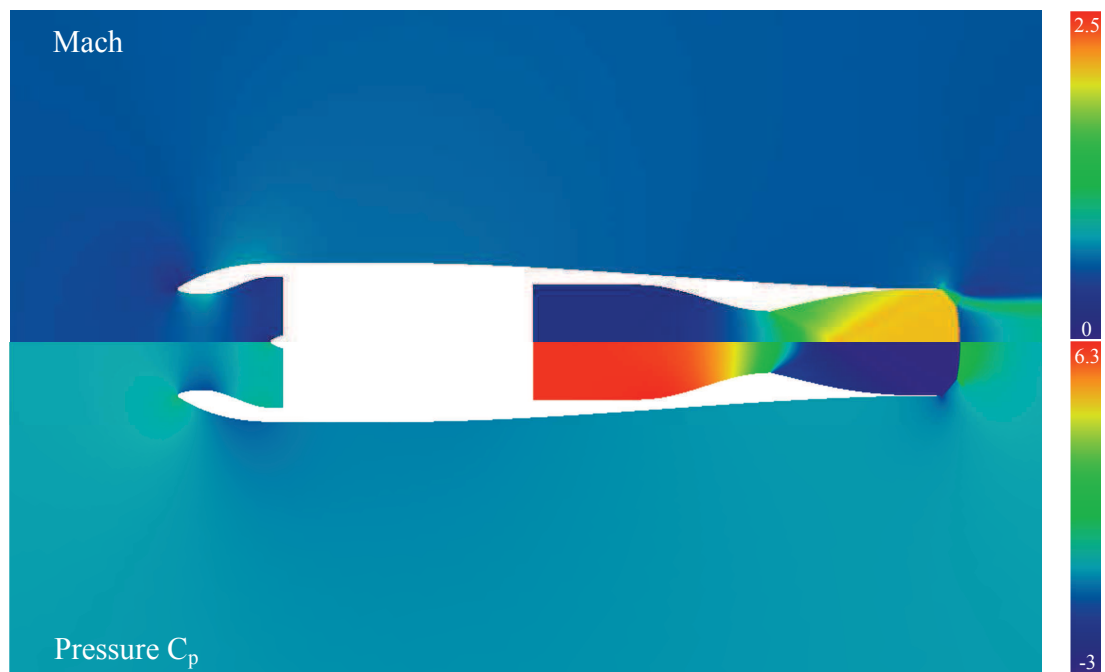


Figure 6.103: Local Mach (top) and Pressure (bottom) around Coupled Turbojet.

6.2.3.5 Multiple Engines

The engine geometry in Figure 6.102 was duplicated, creating two coupled engines in one solution domain. The two engines were given the same mass flow rate and conversation values. The converged solution showed two engines like that shown in Figure 6.103. The

properties on the lower engine were throttled up. Figure 6.104 shows the conditions for both engines, and the converged pressure distribution created by those properties. The lower engine creates more thrust. This creates a nose up moment of 546.94.

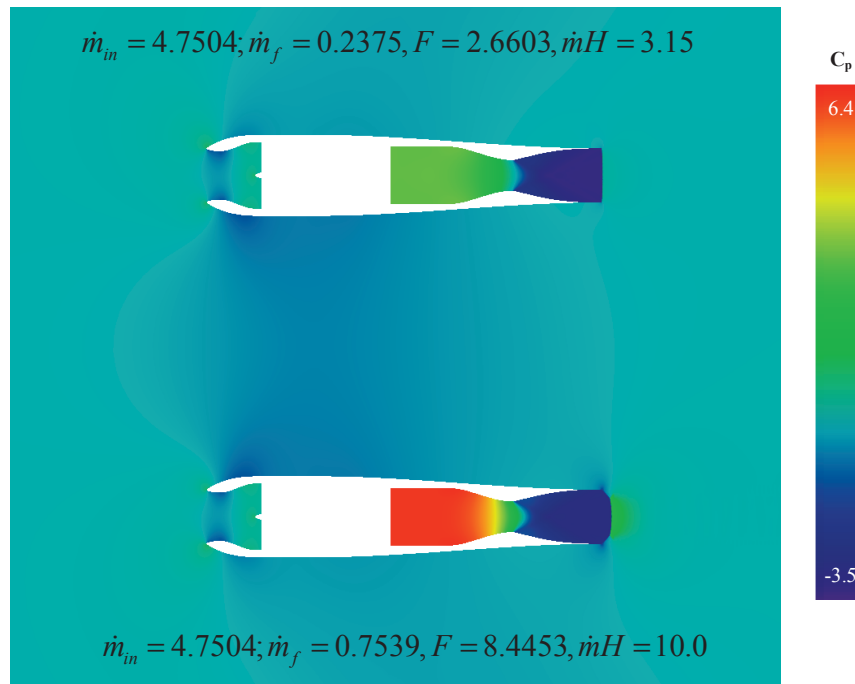


Figure 6.104: Two Coupled Engines in one Domain.

6.3 Viscous Aerodynamics

The viscous test cases are divided into laminar, turbulent, and non-inertial demonstrations. The flat plate boundary layer is used to verify that the viscous terms and Spalart-Allmaras model represent their portions of the physics appropriately. A circular cylinder and ellipse are used to demonstrate simple bodies. Various cavities and fences are presented as the most complicated cases that have been tested using NS2D.

6.3.1 Laminar

The viscous terms are verified with flat plate boundary layer, and then the laminar boundary layer is used to determine an appropriate meshing scheme for laminar flows. Circular cylinder and ellipse flow fields are presented for comparison with CFDsol. Three inclined fences demonstrate the ability of NS2D to model complex flows with application to design. Finally, four cavities represent the most complicated viscous geometries represented with NS2D. Flow fields and acoustic responses are shown for each cavity as a research and design comparison.

6.3.1.1 Laminar (Incomp) Boundary Layer

The flat plate boundary layer was tested to verify the viscous terms in NS2D and NS3D. Six meshes were tested to converge the mesh before accessing the solution. Four of those meshes are shown in Figure 6.105 and Figure 6.106. Blasius' profile is shown on each mesh to give an idea of the relative size of the elements to the boundary layer thickness. The meshes in Figure 6.105 and Figure 6.106 represent spacings of 30%, 23%, 15%, and 11.4% of the boundary layer thickness at the wall. Each mesh was tested with a nominal and reduced amounts of dissipation ($diss = 1.0, 0.1, \text{ and } 0.01$).

NS2D. The velocity profile is also shown for each mesh in Figure 6.105 and Figure 6.106. The velocity profiles are plotted against its similarity variable η . The black line represents Blasius' solution (White, 1991); the solution from NS2D is plotted as red dots. The forward-most and aft-most 10% of the plate is removed from the velocity profile because Blasius' solution is not a fair comparison as the boundary layer begins to form near the leading edge nor where the flow shows influences from the end of the plate, wake, and far field outlet.

The profile matches very well for all four meshes shown here, and the solution tightens up on the Blasius' solution as the spacing is decreased.

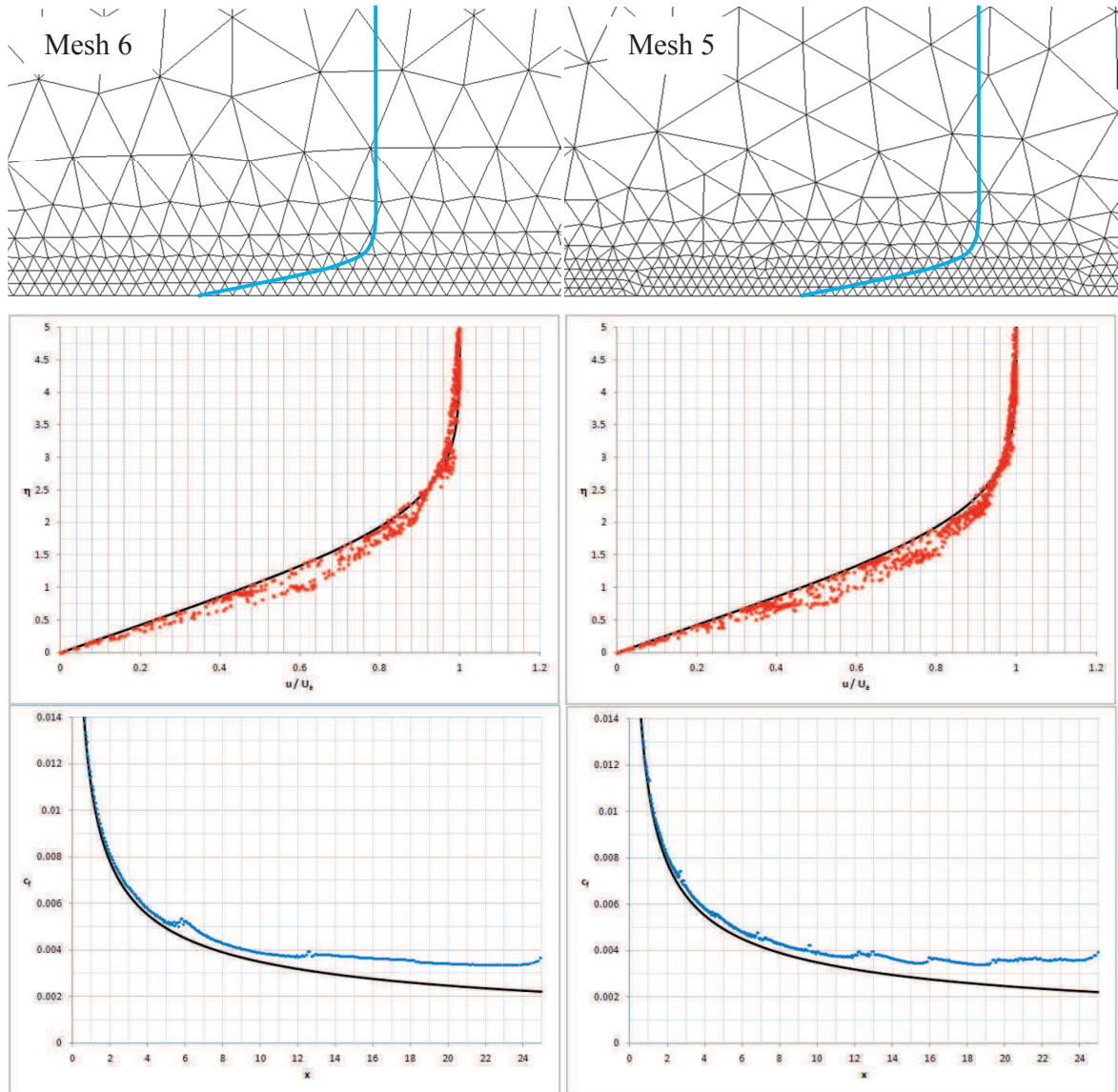


Figure 6.105: Laminar Velocity Profile and Shear Stress for Coarse Meshes (NS2D).

Figure 6.105 and Figure 6.106 also show the wall skin friction distributions for each of the four meshes. Again, the black line represents Blasius' solution. The skin friction from NS2D is shown with blue points and shows artifacts near the discontinuities in mesh spacing.

The mesh spacing needs to smoothly vary along the length of the domain. The skin friction from NS3D is not as sensitive to mesh spacing, but other factors plague the 3D stresses.

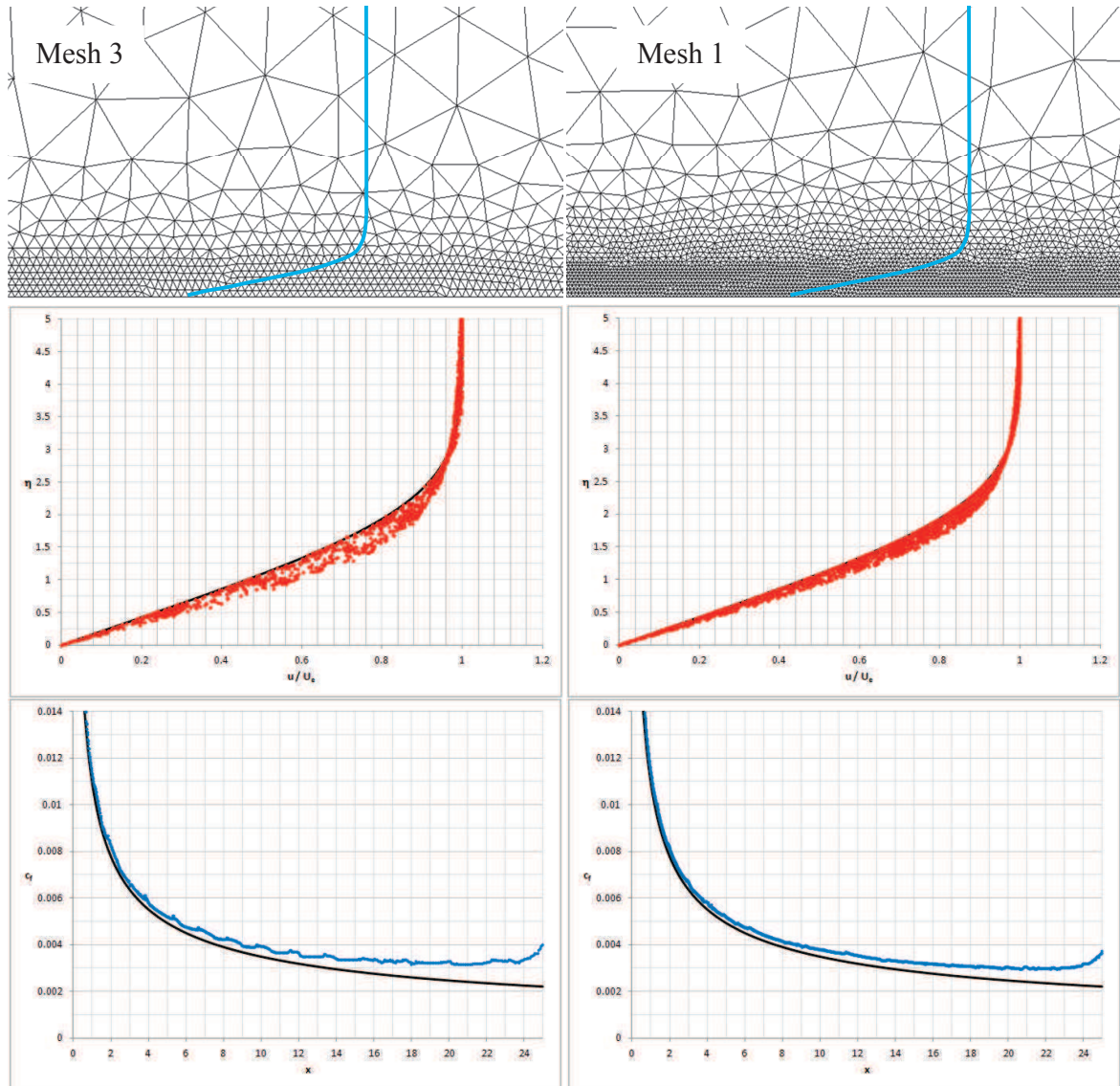


Figure 6.106: Laminar Velocity Profile and Shear Stress for Fine Meshes (NS2D).

Figure 6.107 uses Mesh 3 as a guideline to create an appropriate meshing scheme for laminar flows. At least four elements should be used to span the lower 60% of the boundary layer. The bottom 30% of the boundary layer is nearly linear, so few (but more than one) elements are needed to model this region. The spacing is then increased so that at least two elements

are used in the upper 40% of the boundary layer. Pave2D was used to explore the stream-wise spacing, finding a spacing on the order of half the boundary thickness ($\delta/2$) to be appropriate for the streamwise direction.

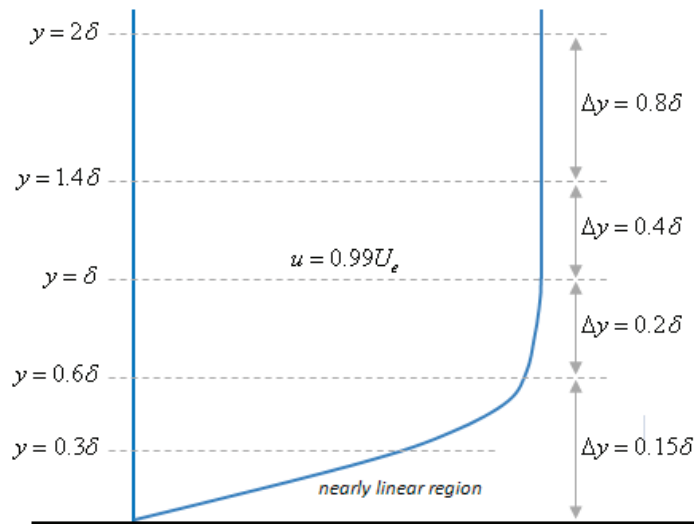


Figure 6.107: Suggested Meshing Scheme for Laminar Flows.

Figure 6.108 shows the boundary layer δ , displacement δ^* , and momentum θ thicknesses along the length of the plate calculated using Mesh 3. The shape factor H is also shown for Mesh 3. All four plots in Figure 6.108 show the effects of the external velocity. The profile properties match the theoretical solution (shown as black lines) very well over the length of the plate.

This process was repeated for plate Reynolds numbers of 800 and 20k. Multiple meshes were created for each Reynolds number. The results for the meshes were similar to that shown in Figure 6.105 and Figure 6.106, and the profile properties were also similar to that seen in Figure 6.108. These results showed that the laminar boundary layer accuracy in NS2D is independent of Reynolds number (below transition, $Re_x < 2 \times 10^5$).

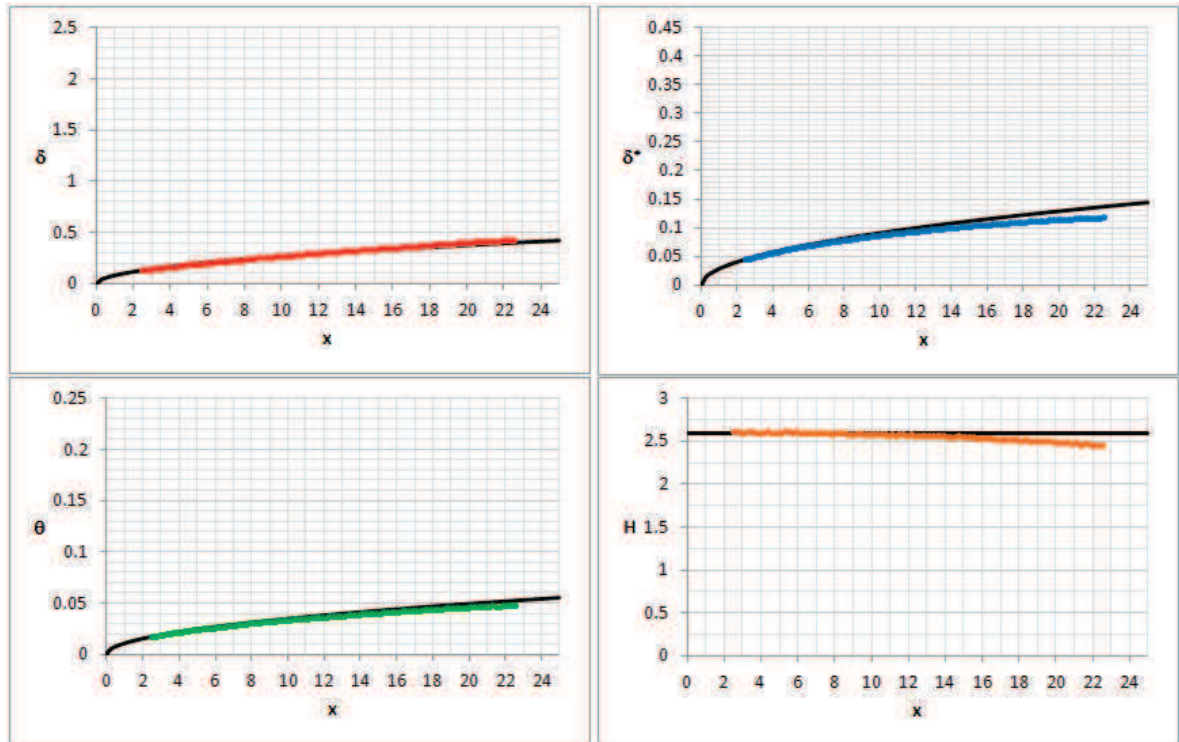


Figure 6.108: Boundary Layer Thickness δ , Displacement Thickness δ^* , Momentum Thickness θ , and Shape Factor H for Mesh 3.

NS3D. These experiments were repeated in NS3D. Figure 6.109 and Figure 6.110 the meshes, velocity profiles, and wall skin friction distributions for each of the four meshes. Again, the black line represents Blasius' solution. The NS3D (red and blue points) solutions are much noisier than their NS2D counterparts, but the NS3D solution does not show artifacts of the mesh. The noise and reduced influence by changes in the mesh seems to come from the third dimension in NS3D.

Mesh 3 shows to be the best mixture of run time and accuracy. The laminar meshing guidelines shown for NS2D can also be used for three-dimensional meshes in NS3D. Figure 6.111 shows the boundary layer profile properties along the length of the plate calculated using

Mesh 3. The profile properties match the theoretical solution (shown as black lines) very well over the length of the plate.

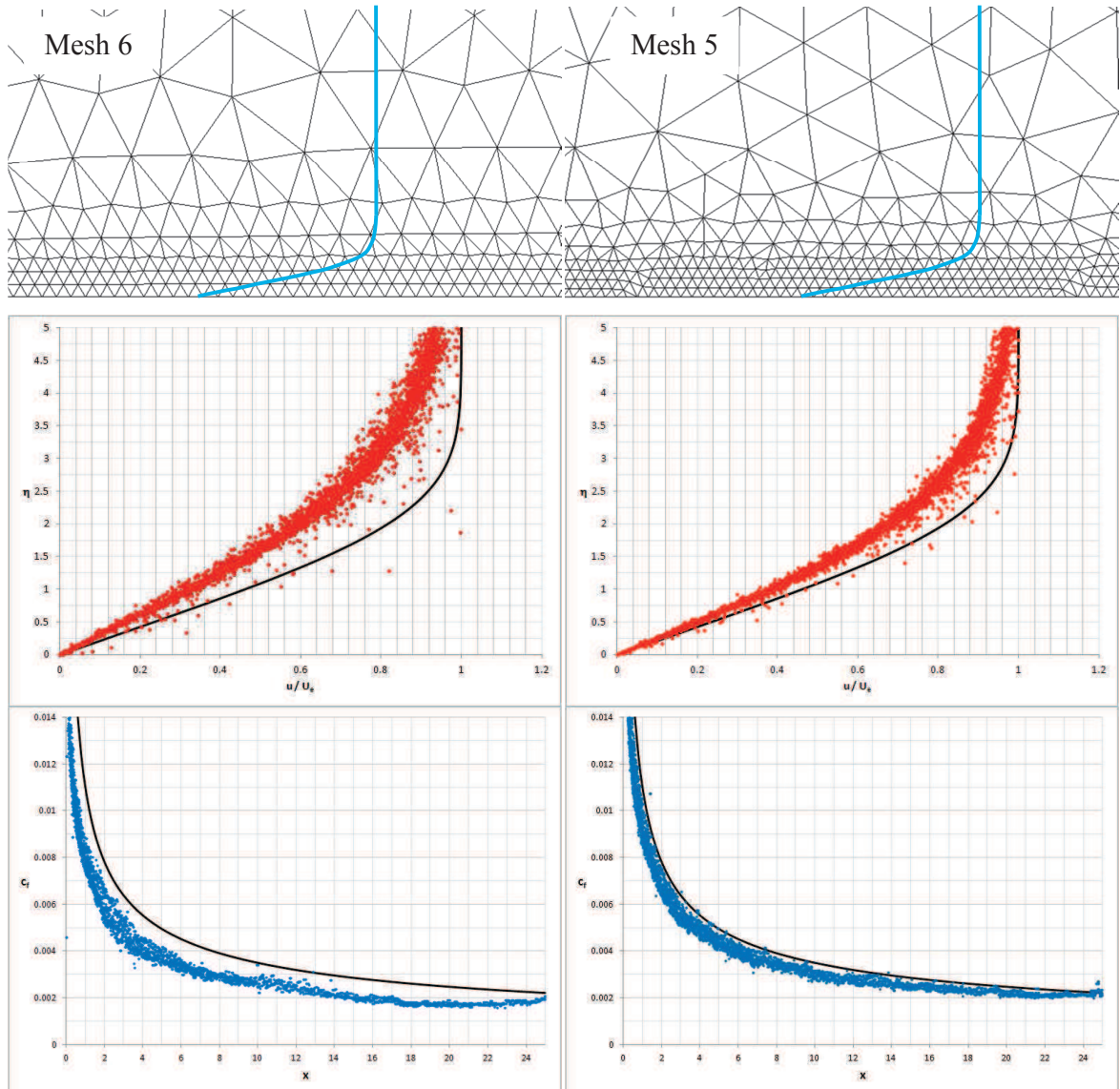


Figure 6.109: Laminar Velocity Profile and Shear Stress for Coarse Meshes (NS3D).

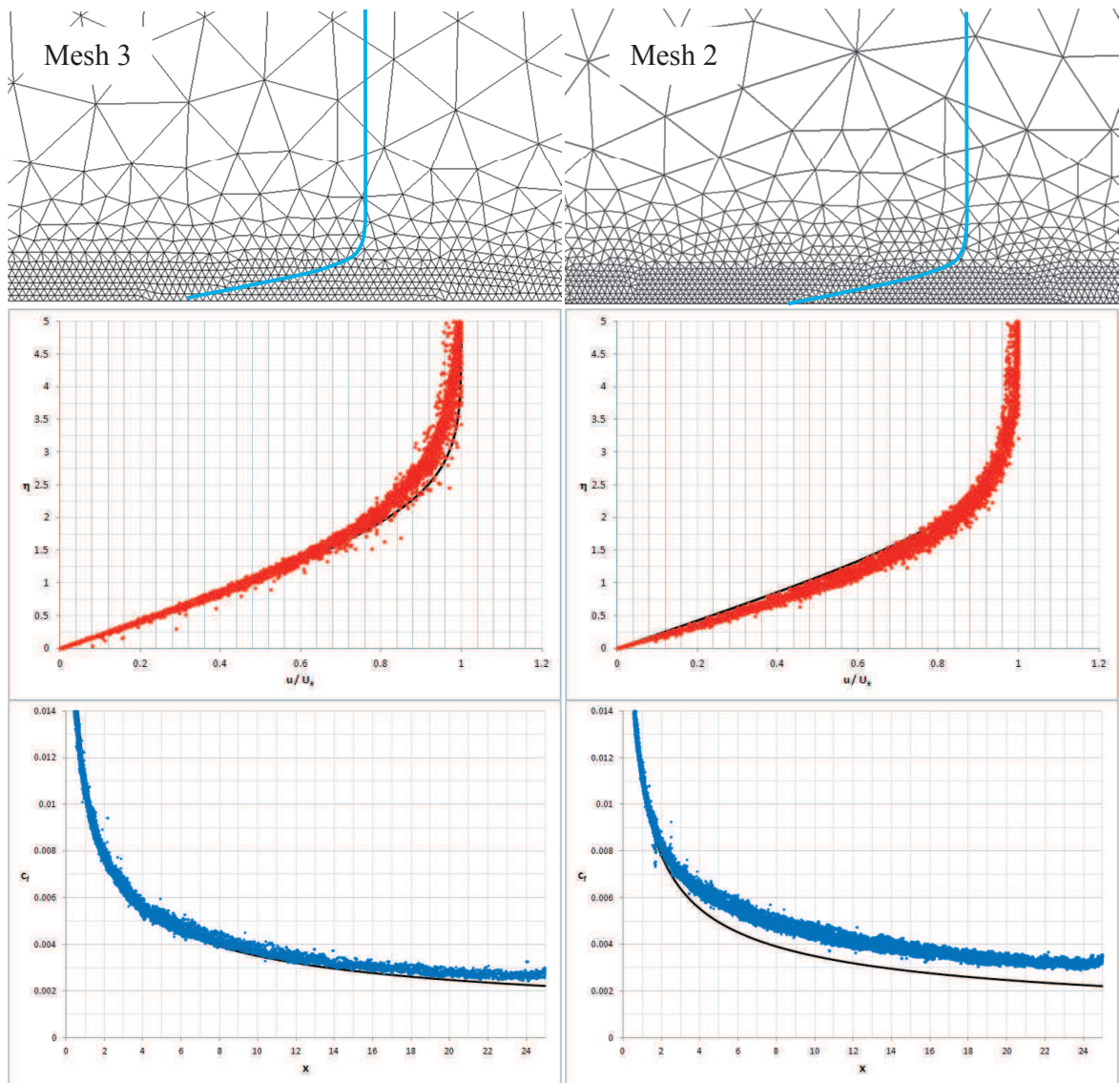


Figure 6.110: Laminar Velocity Profile and Shear Stress for Fine Meshes (NS3D).

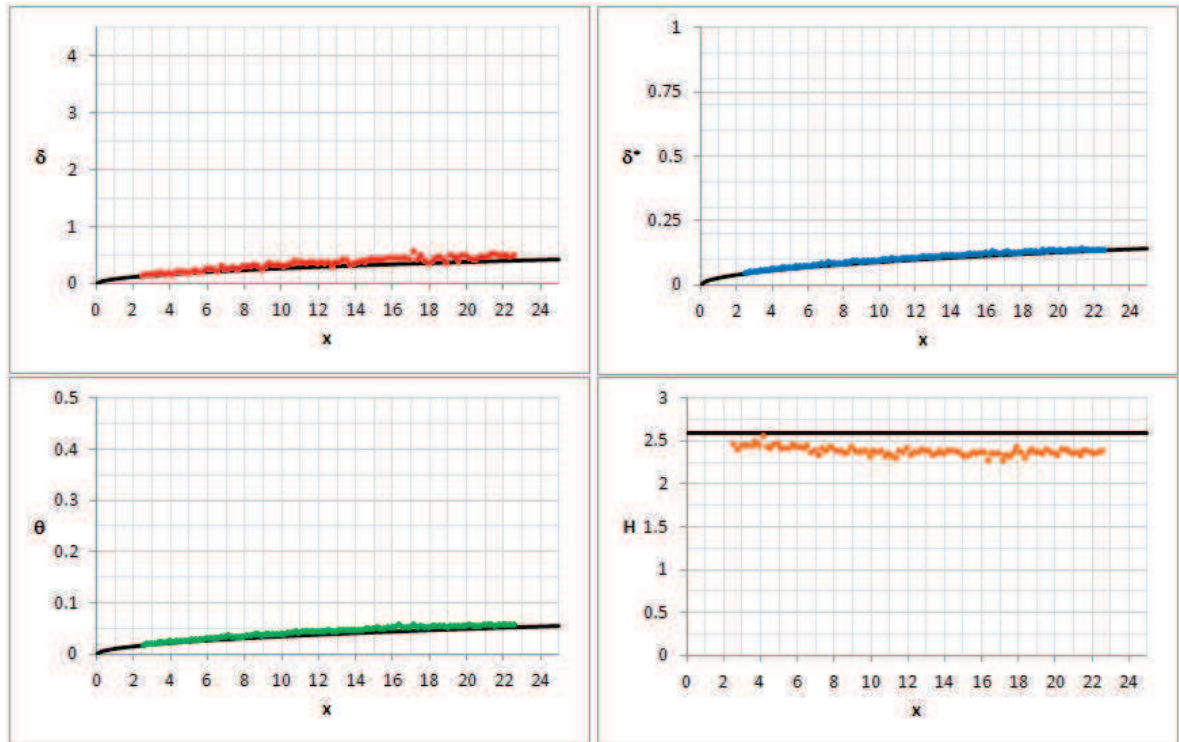


Figure 6.111: Boundary Layer Thickness δ , Displacement Thickness δ^* , Momentum Thickness θ , and Shape Factor H for Mesh 3.

Artificial dissipation is necessary in the external flow for capturing shocks and other flow features. Within the boundary layer, artificial dissipation reduces the skin friction, thickens the boundary layer, and adds error to the near wall solution. To demonstrate that the viscous equations are modeling the boundary layer correctly, aside from the influences of artificial dissipation, an effective Reynolds number Re_{eff} was calculated for the mesh and dissipation $diss$. Blasius' solution is plotted according to the similarity distance η from the wall. This distance can be scaled, creating an effective velocity profile:

$$\eta_{eff} = \eta \sqrt{\frac{Re_{eff}}{Re}}$$

The effective Reynolds number Re_{eff} represents the velocity profile given in the CFD data, chosen to best fit the velocities near the wall. Data from NS3D was compared for three meshes and various dissipation scalars. The results are shown in Figure 6.112. The effective Reynolds number decreases with the amount of artificial dissipation used; and, as the artificial dissipation approaches zero, the effective Reynolds number approaches the actual Reynolds number used in the solution. For this reason, the least amount of artificial dissipation should be used for any viscous solution.

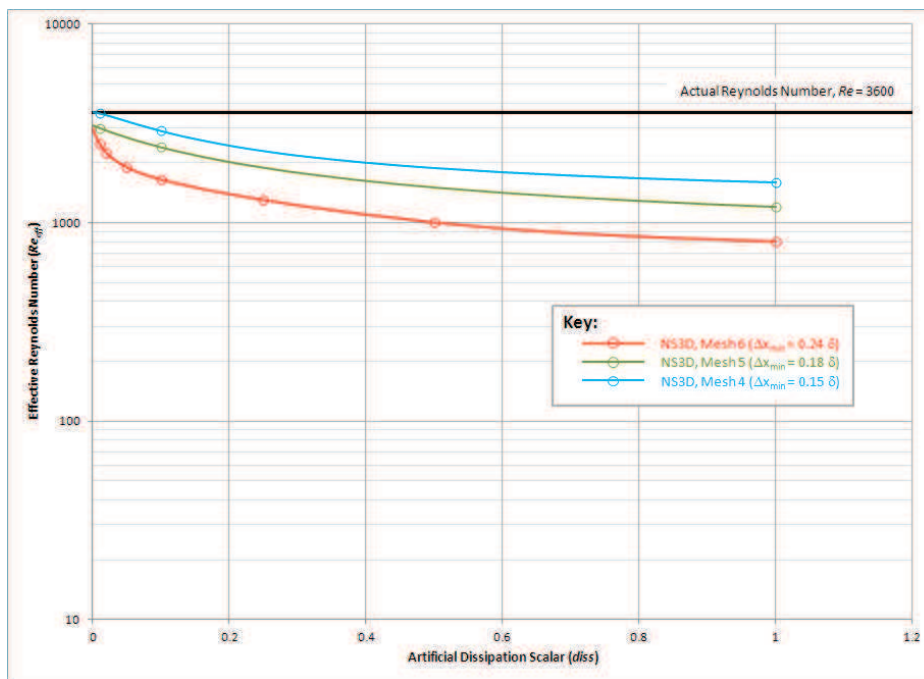


Figure 6.112: Effective Reynolds Number vs. Artificial Dissipation Scalar in NS3D.

The flat plate boundary layer was tested in three different orientations, where the freestream velocity was oriented along with the geometry using α and β :

- (1) The flow aligned with u , stresses in x & y , extruded axis in z
- (2) The flow aligned with v , stresses in y & z , extruded axis in x
- (3) The flow aligned with w , stresses in x & z , extruded axis in y

All three configurations returned the same solution, showing that NS3D has been implemented correctly in all three axis and derivatives.

6.3.1.2 Ellipse

Figure 6.113 shows the flow over an ellipse with 6:1 length-to-thickness ratio. The solution was created using the mesh in Figure 6.114. The freestream flow is defined by Mach 0.3 and a Reynolds number of 4000; higher-order dissipation is used with a scalar *diss* of 0.6. The entropy in the wake shows periodic oscillations forming a vortex street. The dimensionless frequency of the vortices is $f^* = fc / V_{inf} = 1.44$.

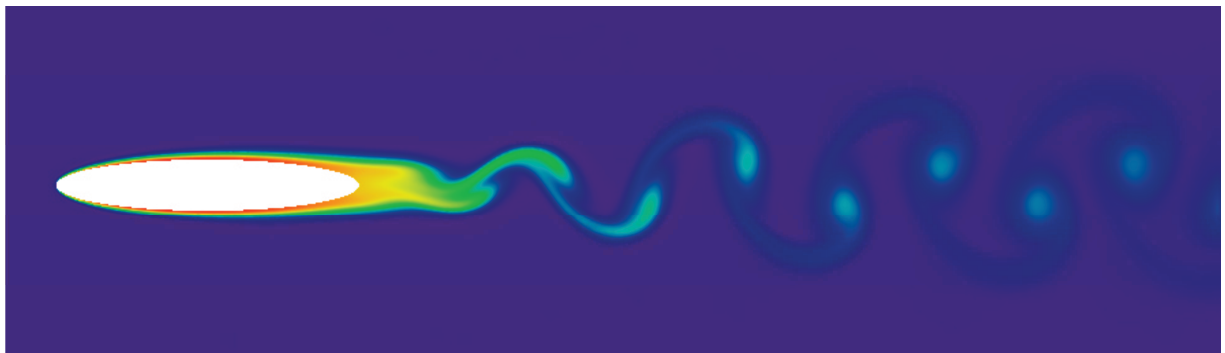


Figure 6.113: Oscillating Wake behind Ellipse (Entropy, $Re_c = 4000$, $diss = 0.6$).

Figure 6.115 shows the inviscid and viscous pressure distributions over the ellipse. The viscous pressure distribution is shown at intervals over the period $T^* = TV_{inf}/c = 0.693$. The viscous pressure matches the inviscid distribution over the forward 40% of the chord. The viscous pressure distribution changes with the release of vortices, but the changes are very small (4% of the overall pressure drop from freestream). Figure 6.116 shows the unsteady lift and drag histories for the ellipse. The lift oscillates at the same frequency as the vortex release. The lift shows an amplitude of 0.17 about a non-zero average of 0.002. The drag

history has a much more diverse frequency spectrum, but the drag shows a definite repetition.

The average drag is 0.05384 with only a 0.07% variation.

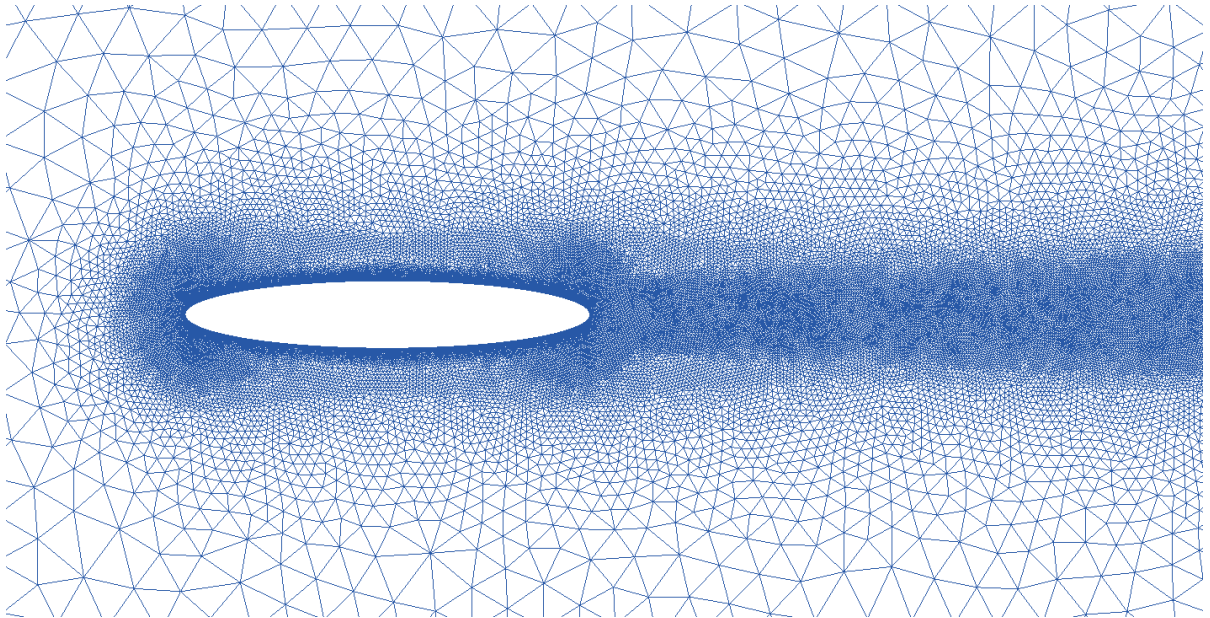


Figure 6.114: Mesh for Ellipse ($Re_c = 4000$).

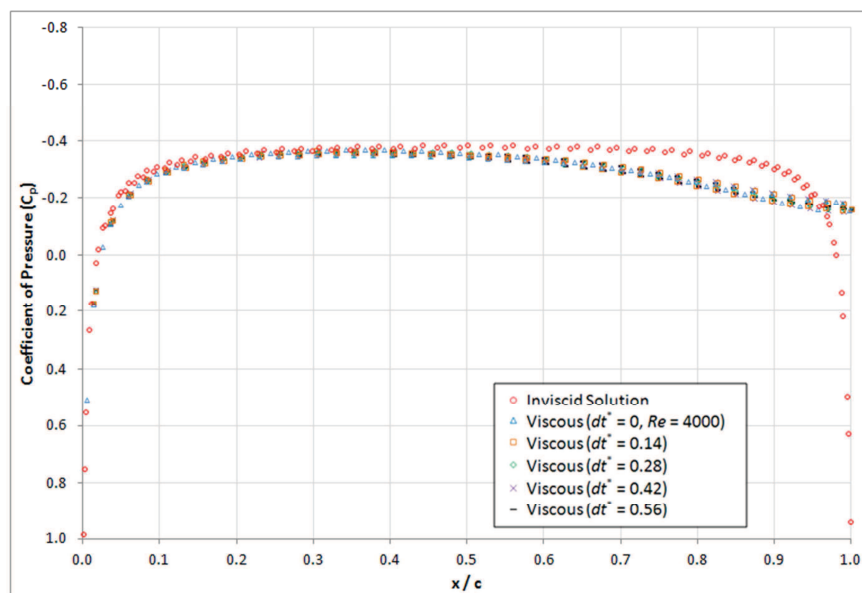


Figure 6.115: Surface Pressure at Various Times over Ellipse ($Re_c = 4000$, $diss = 0.6$).

6.3.1.3 Cylinder

The next six figures show the velocity and entropy distribution around a circular cylinder at Reynolds numbers of 1.54, 9.6, 26, 41, 105, and 200. The lowest three Reynolds numbers create a steady (non-oscillating) flow field. Laminar cylinder flows above a Reynolds number of 40 showing oscillations in the separation points and wake. The two highest Reynolds numbers (105 and 200) produce a definite vortex street.

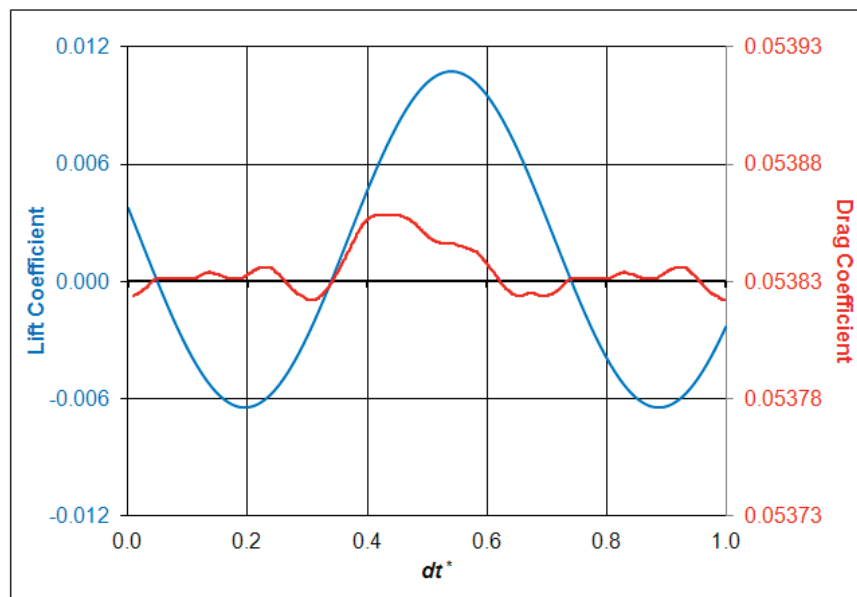


Figure 6.116: Lift and Drag Histories for Ellipse ($Re_c = 4000$, $diss = 0.6$).

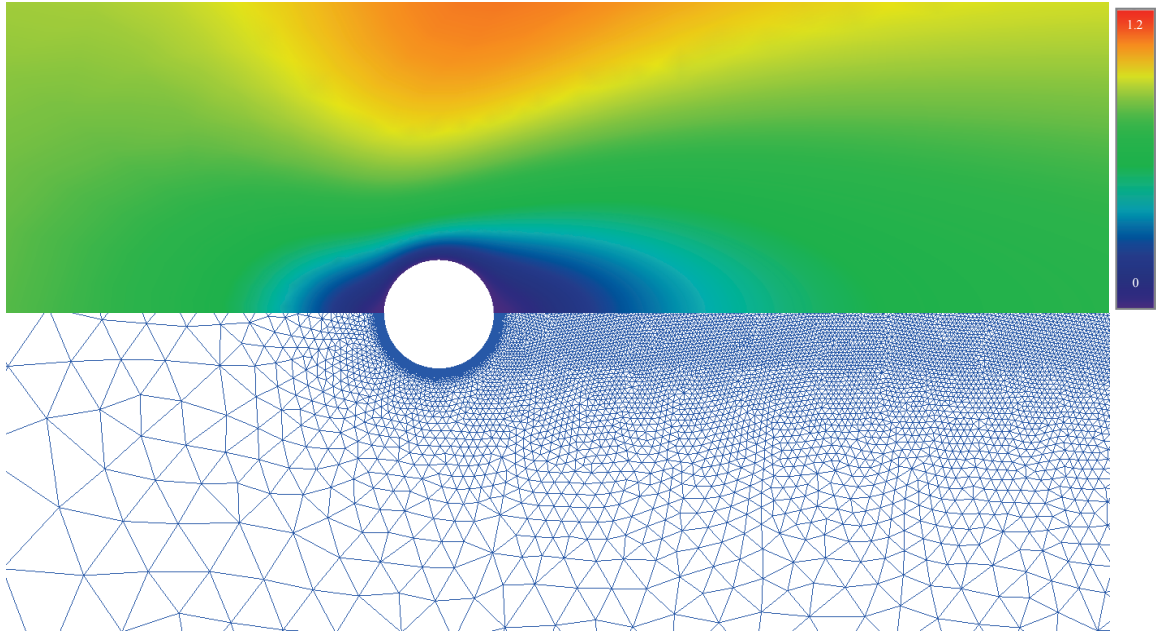


Figure 6.117: Velocity Distribution near Circular Cylinder at $Re = 1.54$ (NS2D).

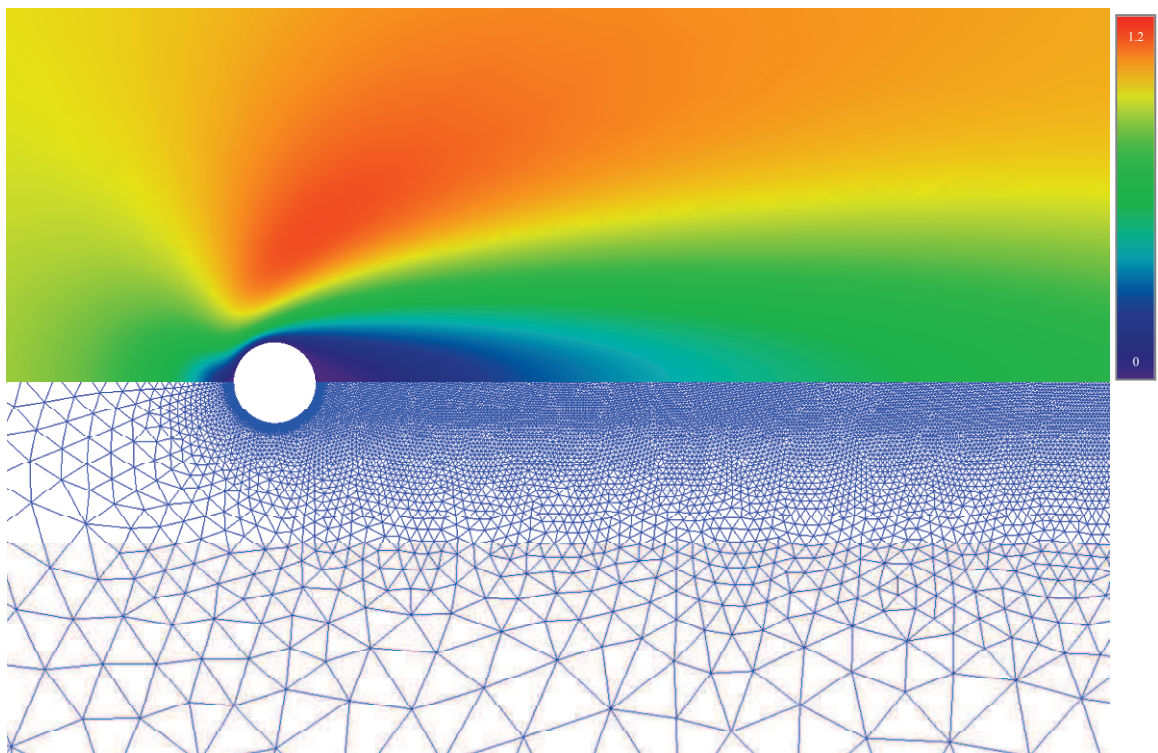


Figure 6.118: Velocity Distribution near Circular Cylinder at $Re = 9.6$ (NS2D).

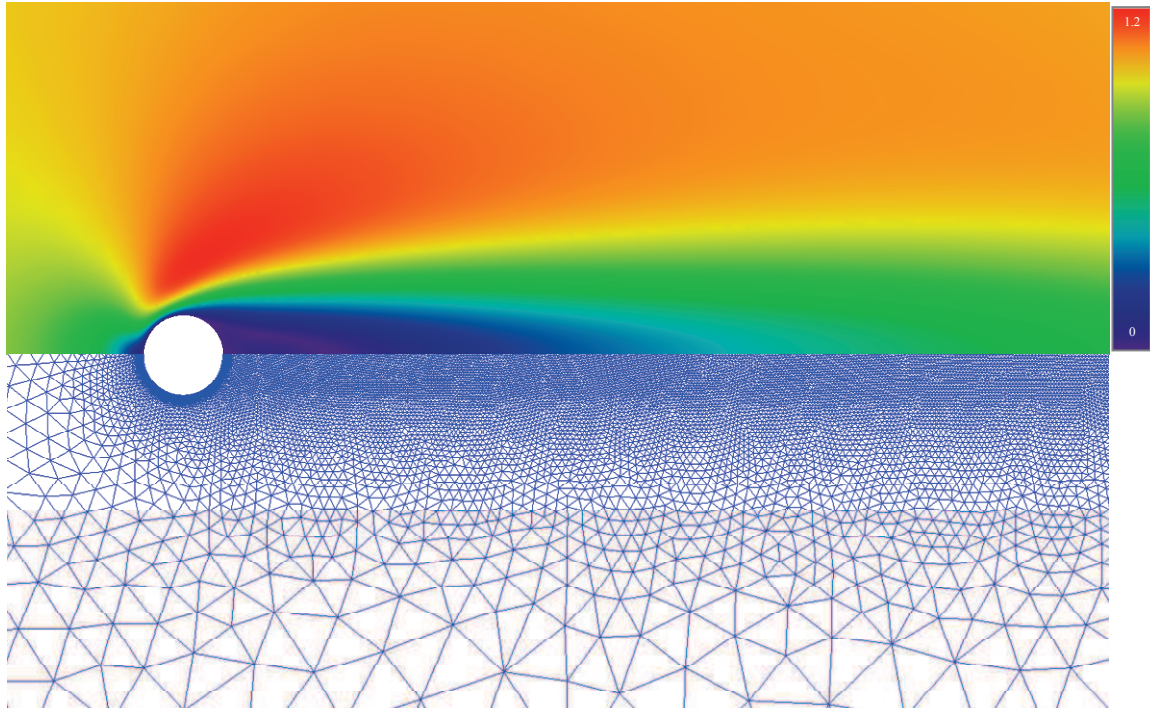


Figure 6.119: Velocity Distribution near Circular Cylinder at $Re = 26$ (NS2D).

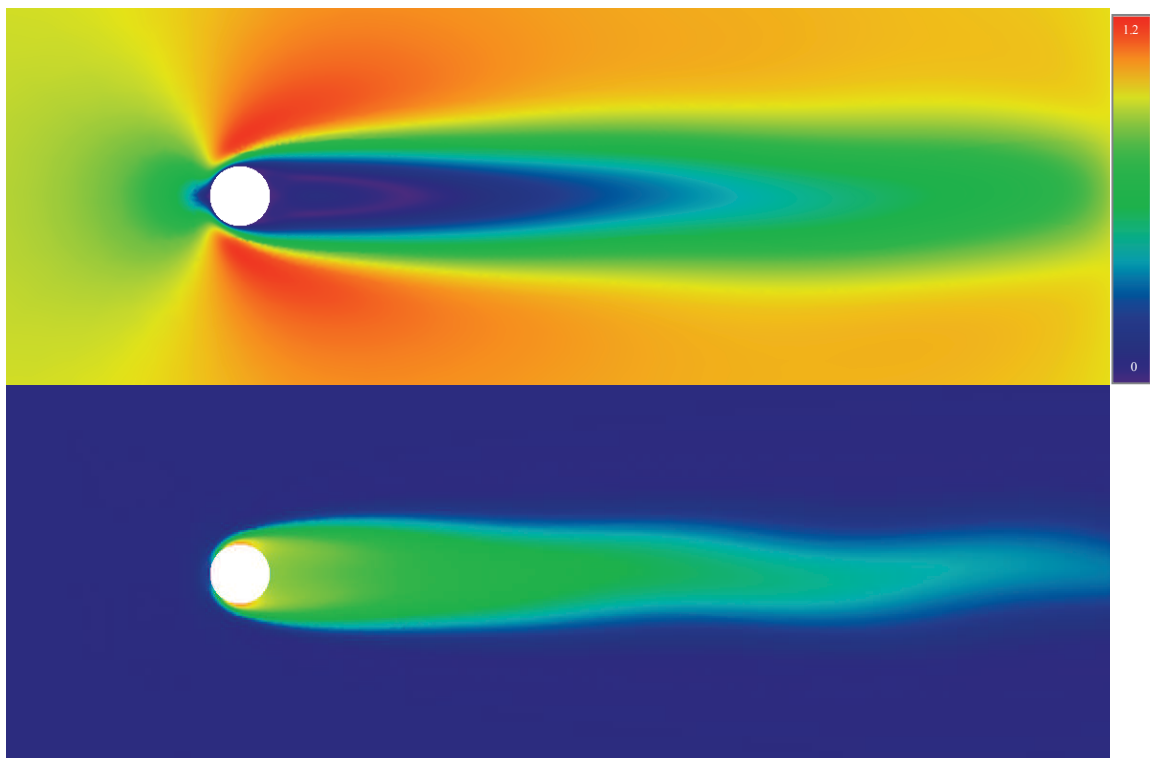


Figure 6.120: Cylinder at $Re = 41$: Velocity (top) and Entropy (bottom) (NS2D).

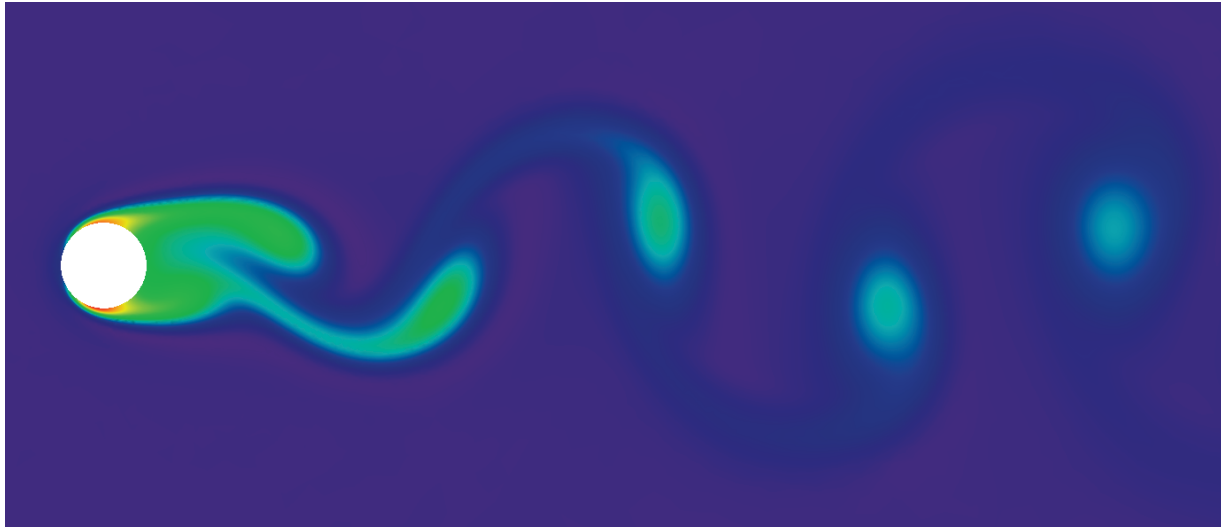


Figure 6.121: Oscillating Wake Behind Circular Cylinder (Entropy, $Re = 105$) (NS2D).

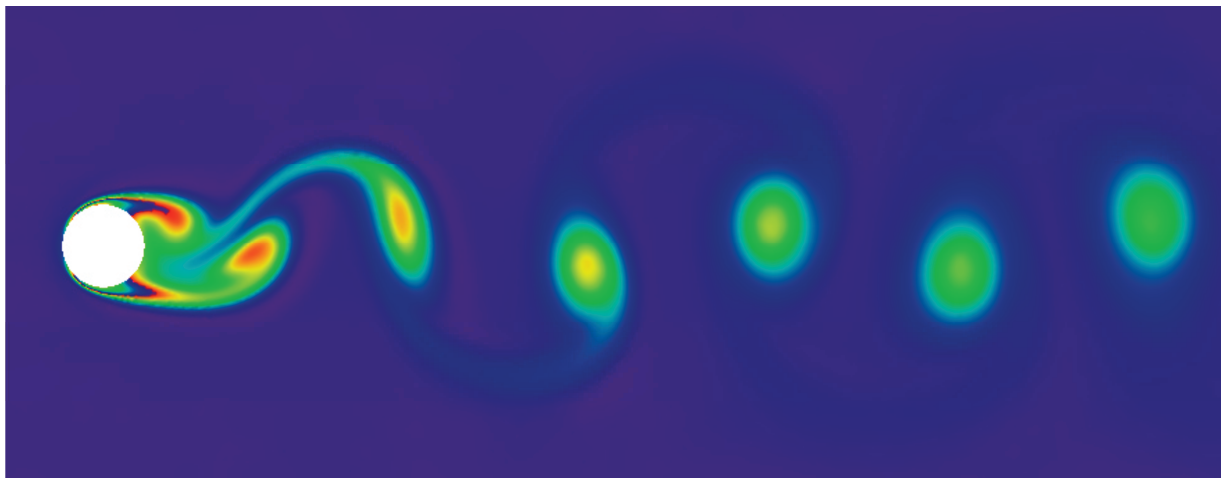


Figure 6.122: Oscillating Wake Behind Circular Cylinder (Entropy, $Re = 200$) (NS2D).

The Reynolds number 41 and 105 cases were tested in NS2D in the non-inertial frame using translational velocities equivalent to the freestream applied to the cylinder in the inertial frame. The $Re = 41$ case did not oscillate in the inertial frame but showed the clear oscillations seen in Figure 6.120 when tested in the non-inertial frame. The $Re = 105$ case was

tested with the translational velocity in v_x . The domain was rotated to align with the y -axis and the flow was induced with v_y . The same solution was obtained in both orientations.

The Reynolds number 105 case was tested in NS3D in three different orientations:

- (1) The flow aligned with v_x , stresses in x & y , cylinder axis in z
- (2) The flow aligned with v_y , stresses in y & z , cylinder axis in x
- (3) The flow aligned with v_z , stresses in x & z , cylinder axis in y

All three configurations returned the same solution. This was another check that NS3D has been implemented correctly in all three axis, derivatives, and non-inertial motion.

6.3.1.4 Inclined Fence

Three inclined fences are demonstrated here. The fences were investigated in cooperation with another research project and are included here because of their complex flow fields.

The first fence (shown in Figure 6.123 through Figure 6.125) is solid inclined at a 30-degree angle, and the height of the fence is approximately 6 times the thickness of the boundary layer at its location. The solid fence produces a strong startup vortex and then releases vortices as bursts into the wake. The flow under the fence stagnates and adds to the vortex.

The percentage value to the right shows an approximate swing in density in the wake, with an estimated uncertainty value. The uncertainty was estimated using the error in the limits of the scale and a worst case error in the “eye-balled” range. The wake of the solid fence starts off by ripping two very solid vortices (shown below). After the vortices drift downstream, the wake converges on the solution shown on this and the previous page. The shear layer and acoustics suffer from the stagnation-vortex dominated flow.

6.3.1.5 Inclined Fence with Holes

The solid fence was improved by cutting holes to allow flow to pass through the fence, eliminating the stagnation under the fence. The holes are evenly spaced and equal sized. Figure 6.126 through Figure 6.128 show the flow downstream of the porous fence. The holes in the fence produce small jets of higher speed flow, bounded by highly vertical flow. The downstream region resembles a turbulent flow field even though no turbulence model was used to produce this solution. The percentage swing in density in the wake is again shown with an estimated uncertainty value. The startup for the porous fence is very interesting: Each slot develops oscillating vortices (shown in Figure 6.128), when the vortices interact they wash each other out into a thick well-mixed wake (seen in Figure 6.126). The porous fence decreases the downstream noise and eliminates the stagnation just under the fence, spreading out the shear layer between the external flow and wall.

Incremented Porous Fence. A second fence was tested with a different hole pattern. The size of the holes was incremented so that more mass passed through the upper portion of the fence and less through the lower portion. The flow field behind both fences can be seen in Figure 6.129 and Figure 6.130. White dashed lines are shown on the figures, representing locations where instantaneous velocity profiles were measured.

The figures below show the velocity deficit behind the two fences compared to a laminar velocity profile, representing the natural shear thickness. The equally sized holes produce a large velocity defect at the top of the fence, created by the solid section at the top of the fence. The incremented holes produce a much more linear velocity distribution with a profile at the wall that resembles a separating boundary layer.

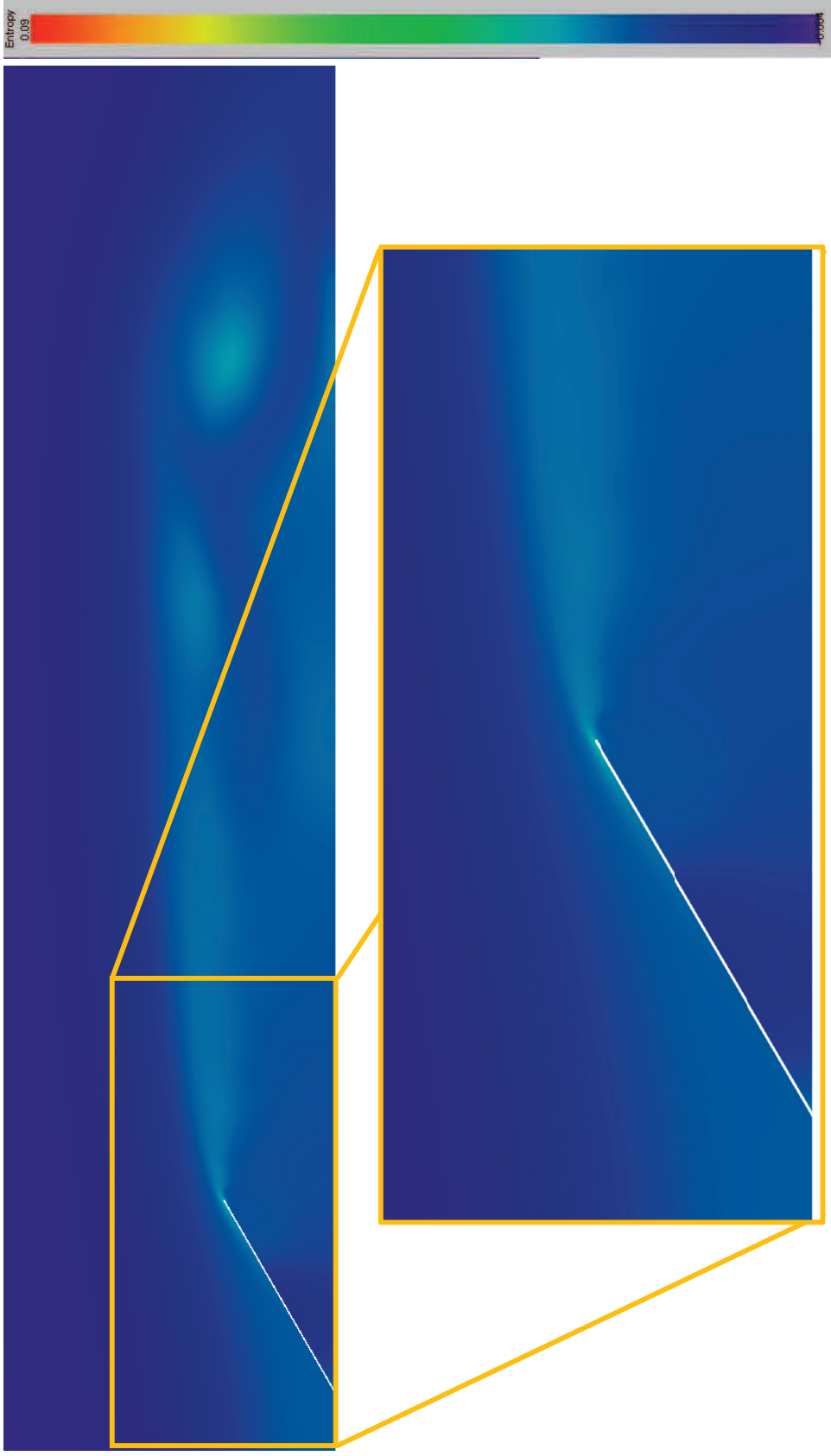


Figure 6.123: Entropy Distribution Downstream of Solid Fence.

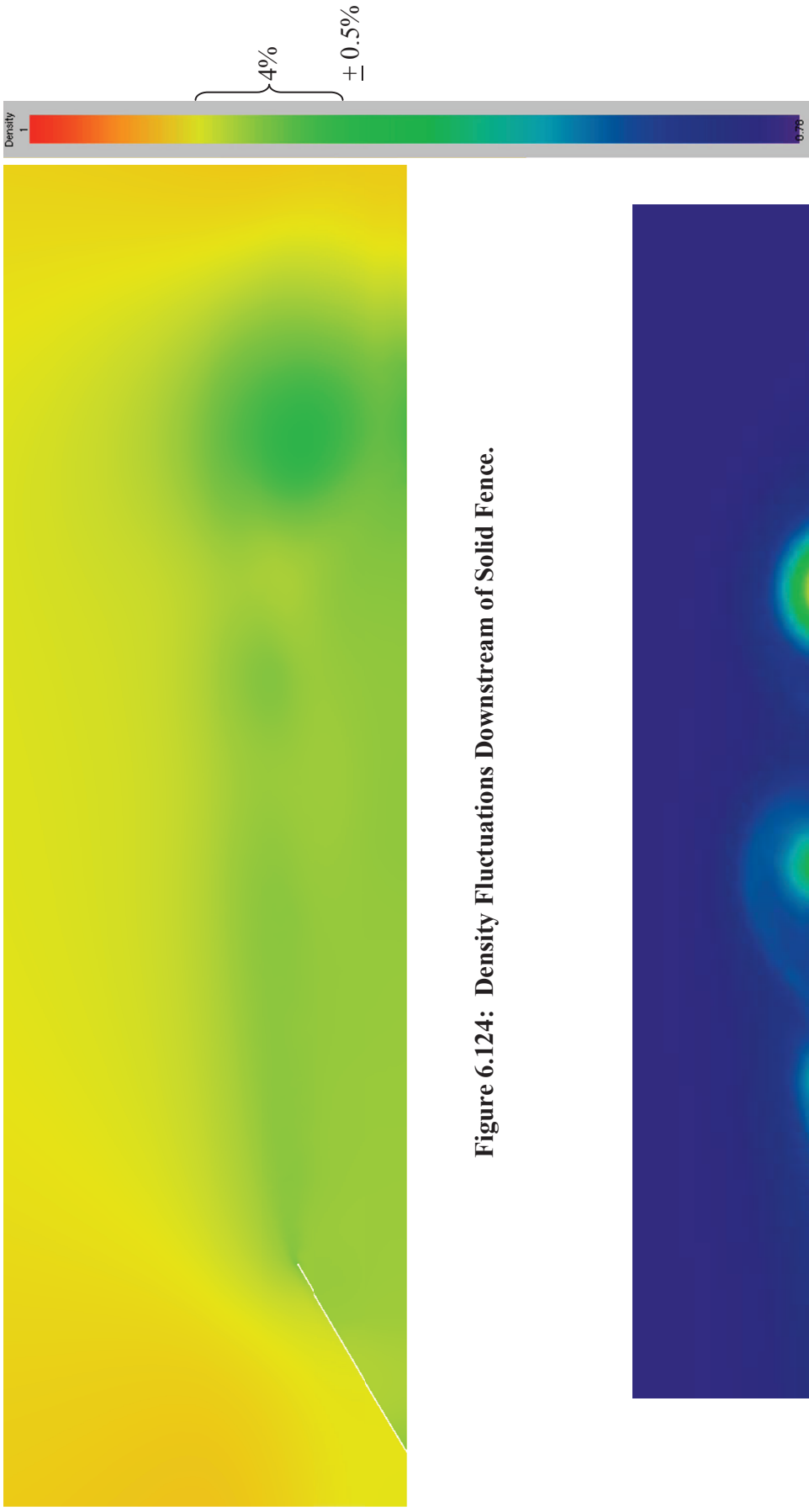


Figure 6.124: Density Fluctuations Downstream of Solid Fence.

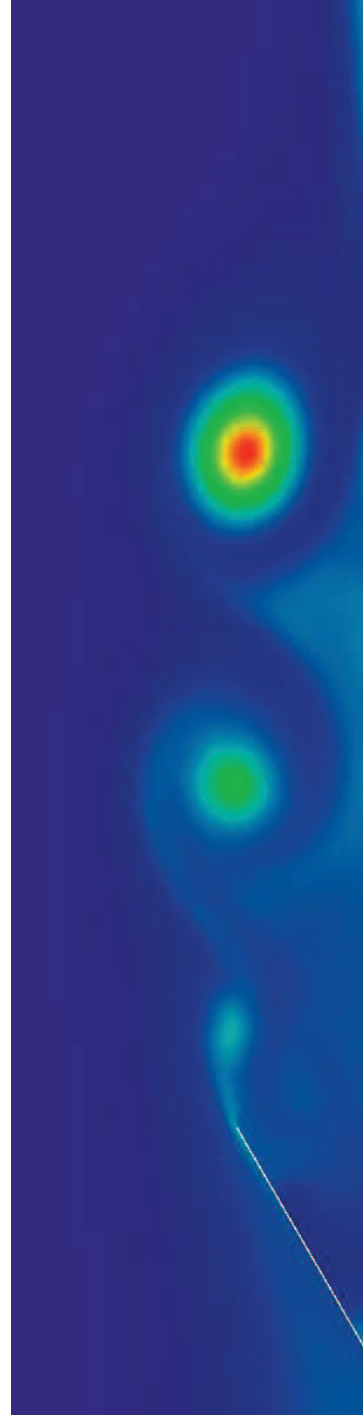


Figure 6.125: Entropy Showing Vortices Downstream of Solid Fence.

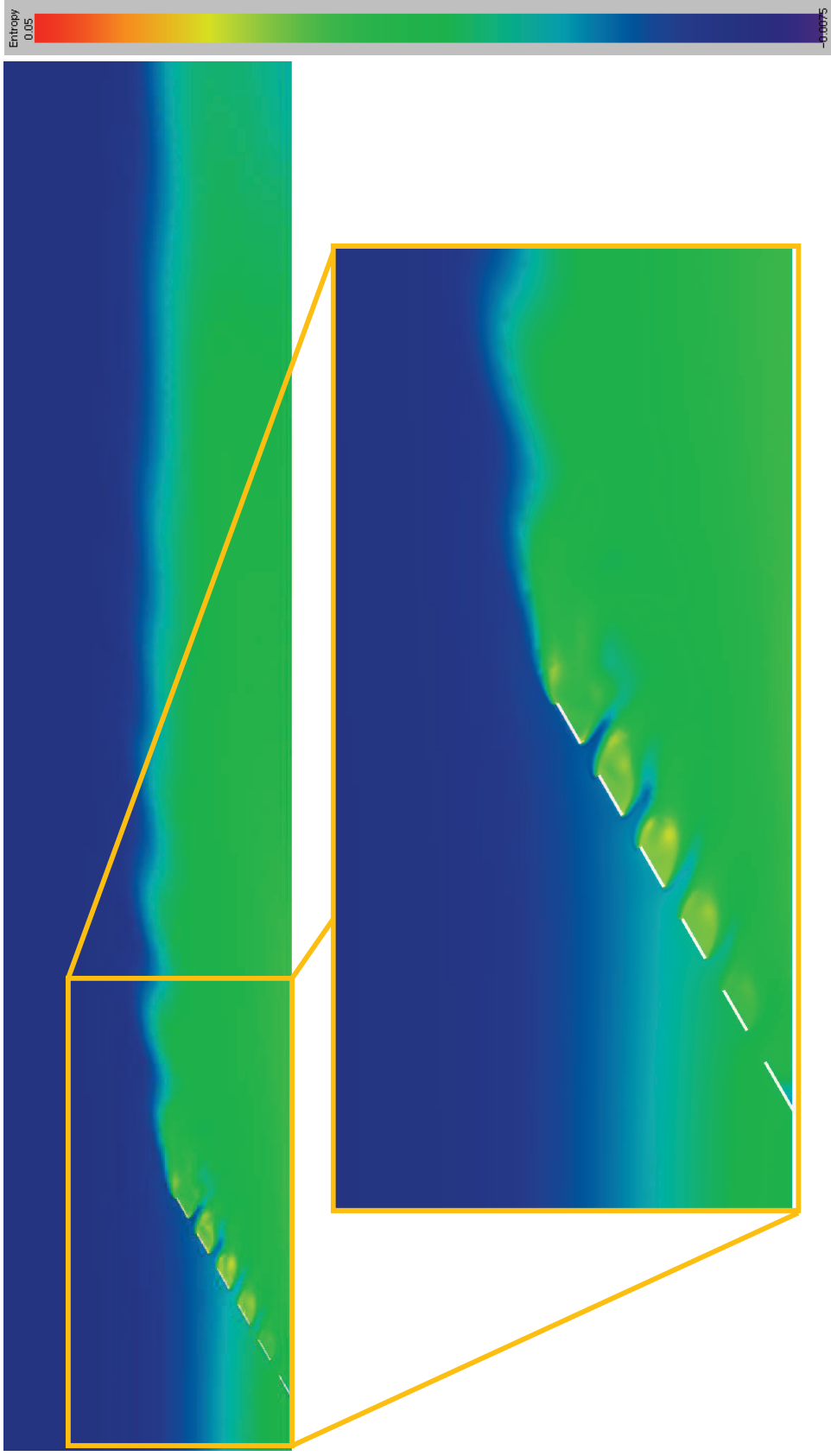


Figure 6.126: Entropy Distribution Downstream of Porous Fence.

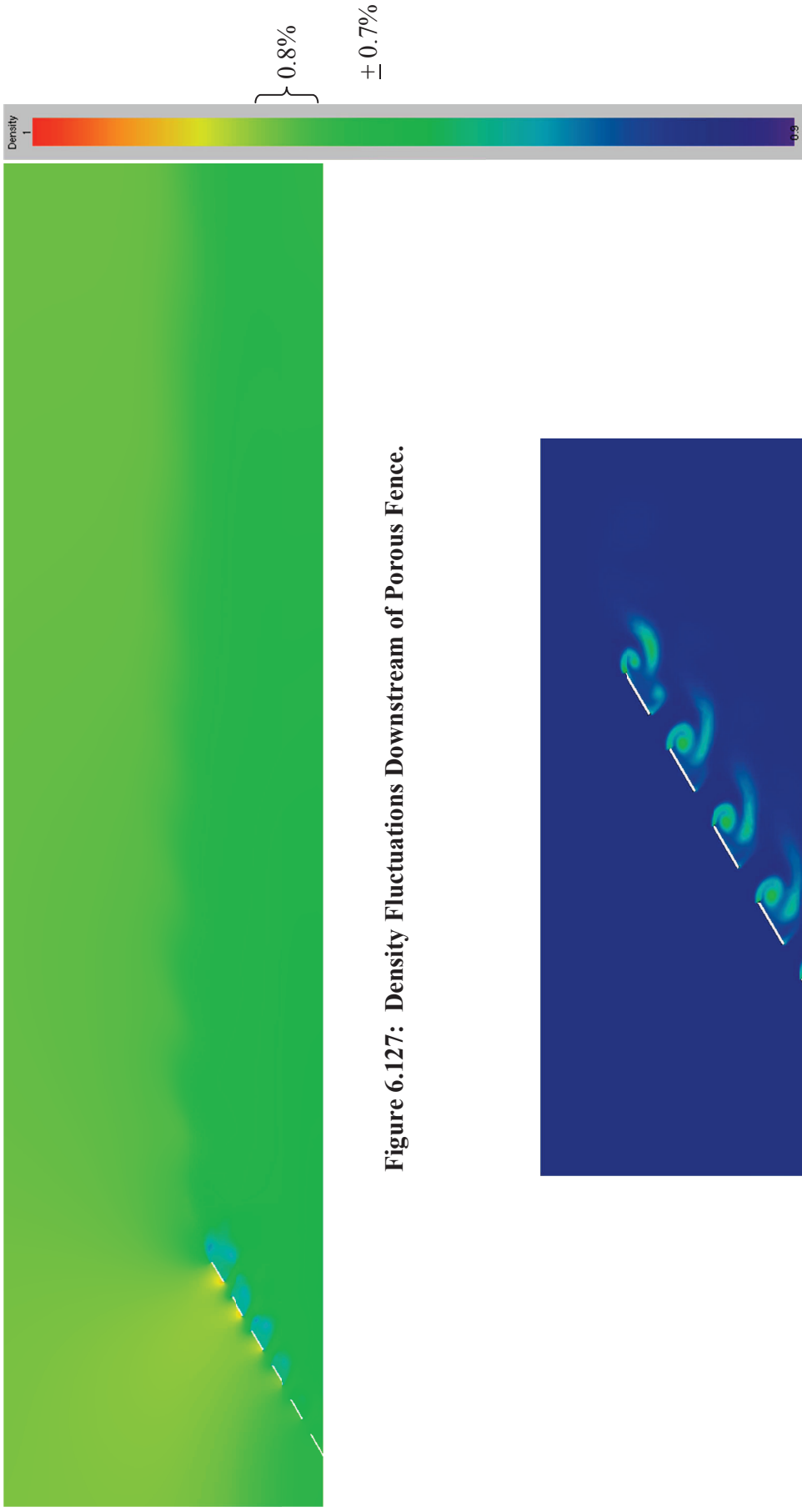


Figure 6.127: Density Fluctuations Downstream of Porous Fence.

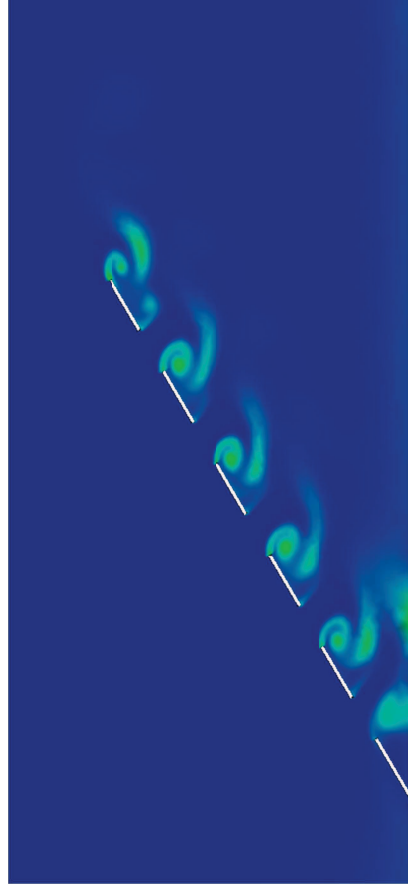


Figure 6.128: Entropy Showing Vortices Produced by Holes in Porous Fence.

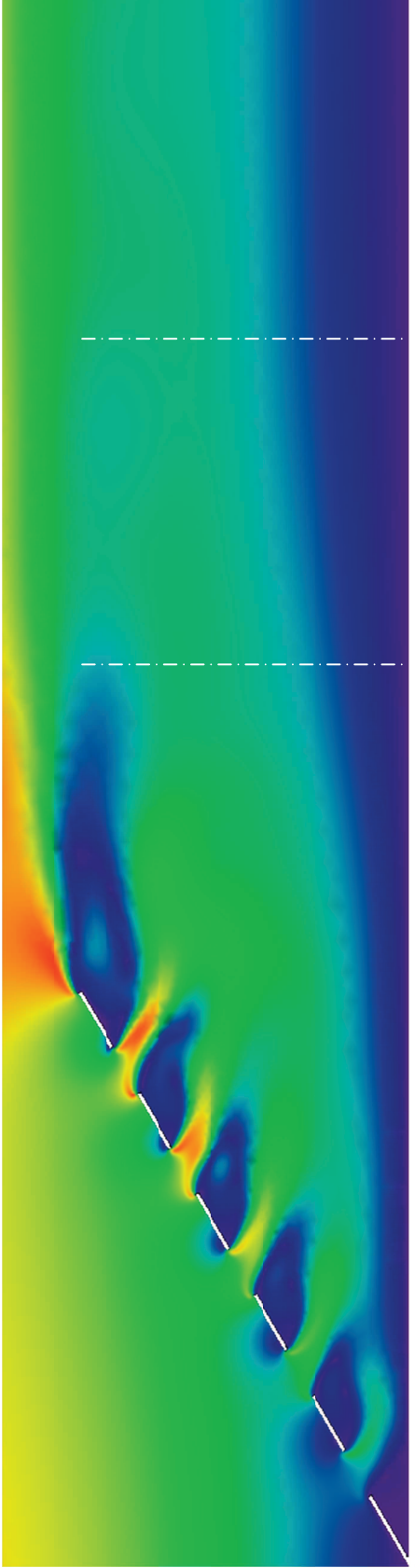


Figure 6.129: Velocity Distribution around Equally Sized Holes

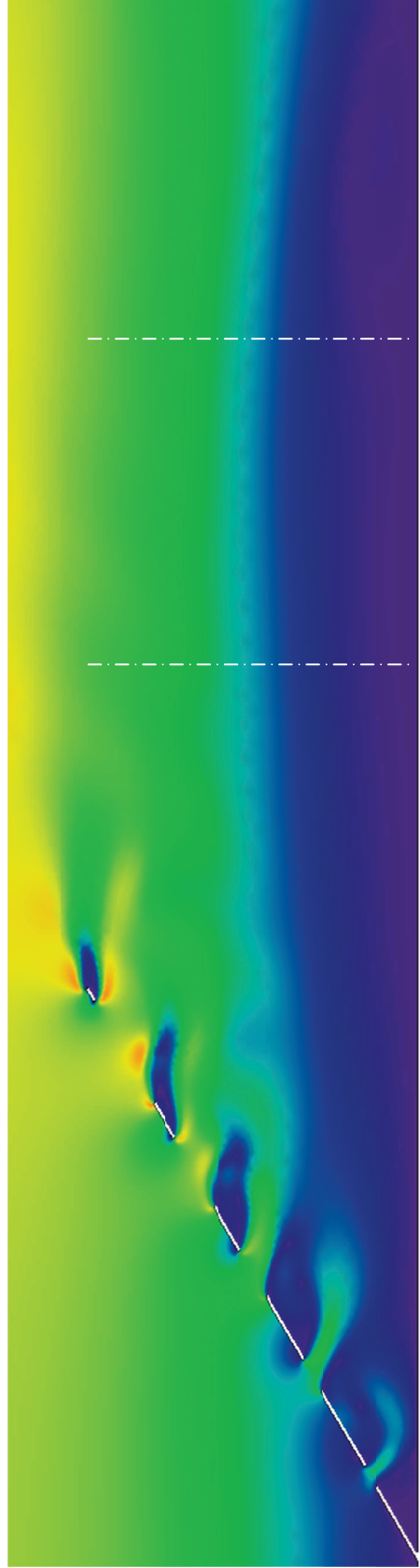


Figure 6.130: Velocity Distribution around Incremented Holes

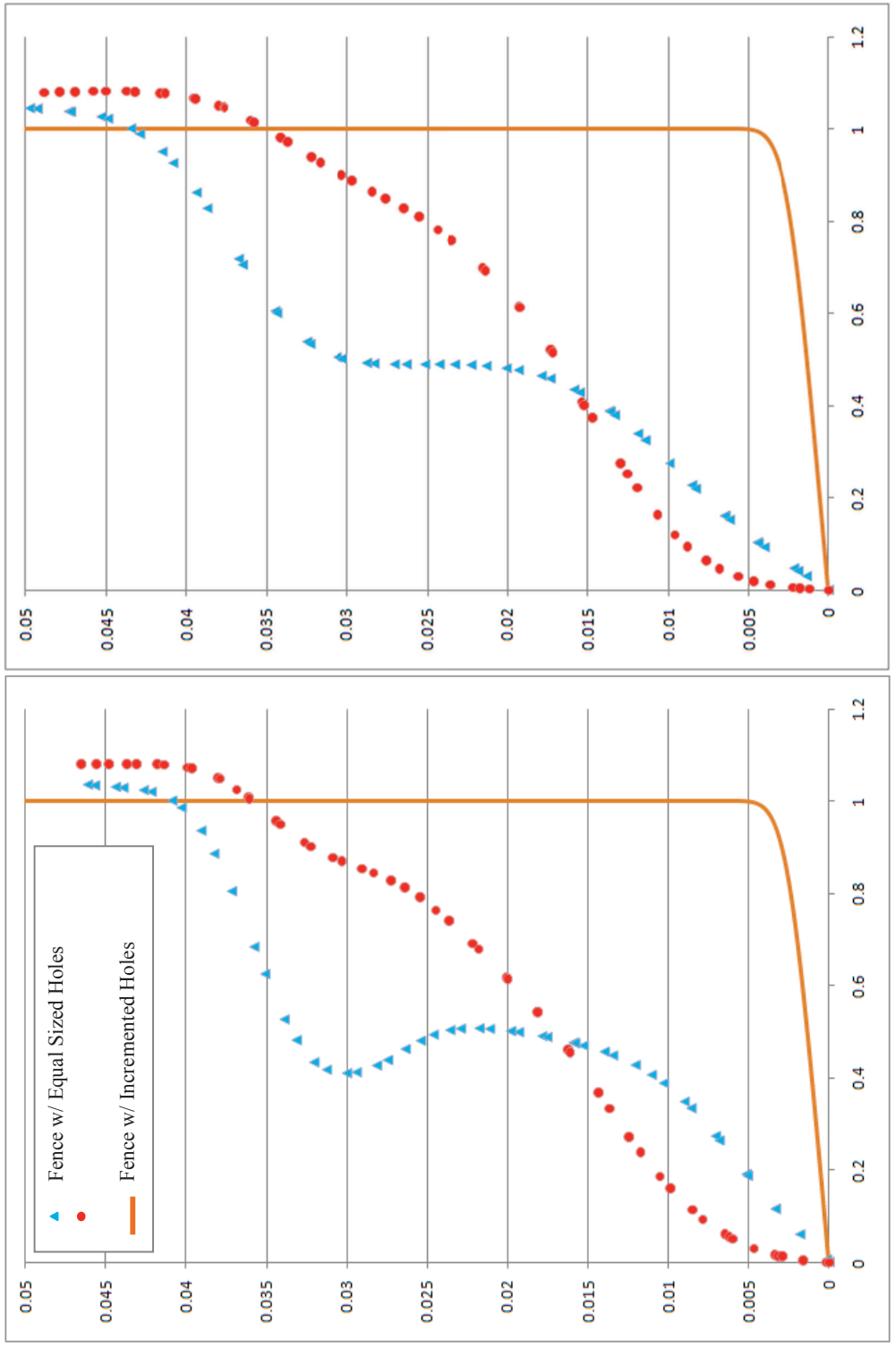


Figure 6.131: Instantaneous Velocity Profile One (left) and Two (right) Fence Heights Downstream.

6.3.1.6 Square Cavity

NS2D was used to model the viscous aerodynamics of a rigid walled cavity at transonic conditions. A square cavity was simulated at Mach 0.9 and an altitude of 30,000 ft for a Reynolds number of 3.13×10^6 . The no-slip condition was applied to the wall one length upstream and downstream of the cavity to build and maintain a boundary layer outside of the cavity. The far field was created cavity lengths upstream, downstream, and above the cavity. The four pictures in Figure 6.132 show the physics of cavity flows. The vortices ripping off the upwind corner strike the downwind corner (left pictures), which create acoustic waves in the form of pressure and density variations (right). The acoustic waves travel upstream, striking the upwind corner. The shear layer generates the vortices in the flow, and the acoustic waves striking the corner, increasing the strength of the subsequent vortex. The flow inside of the cavity circulates, driven by the external flow through the shear layer. The flow exhibits a “chugging” behavior, in and out of the cavity. The vortices and acoustic waves drive each other and, in turn, drive the chugging behavior.

Three cavities of different depths were tested for their acoustic response at Mach 0.281 and a Reynolds number of 6.15×10^6 . The first cavity is a square cavity ($L = D$); the second cavity is one-quarter that depth ($L = 4 D$); and, the final cavity is the twice the depth ($L = D/2$). The mesh at the top of each cavity was the same (shown in Figure 6.133). The mesh is refined in the shear layer to increase the accuracy of the acoustic waves and vortices traveling between the corners. The mesh size throughout the cavity itself is constant (shown at the bottom of Figure 6.133). The shallow cavity was modeled with 129k elements; the square cavity needed 198k elements; and, the deep cavity required 221k elements.

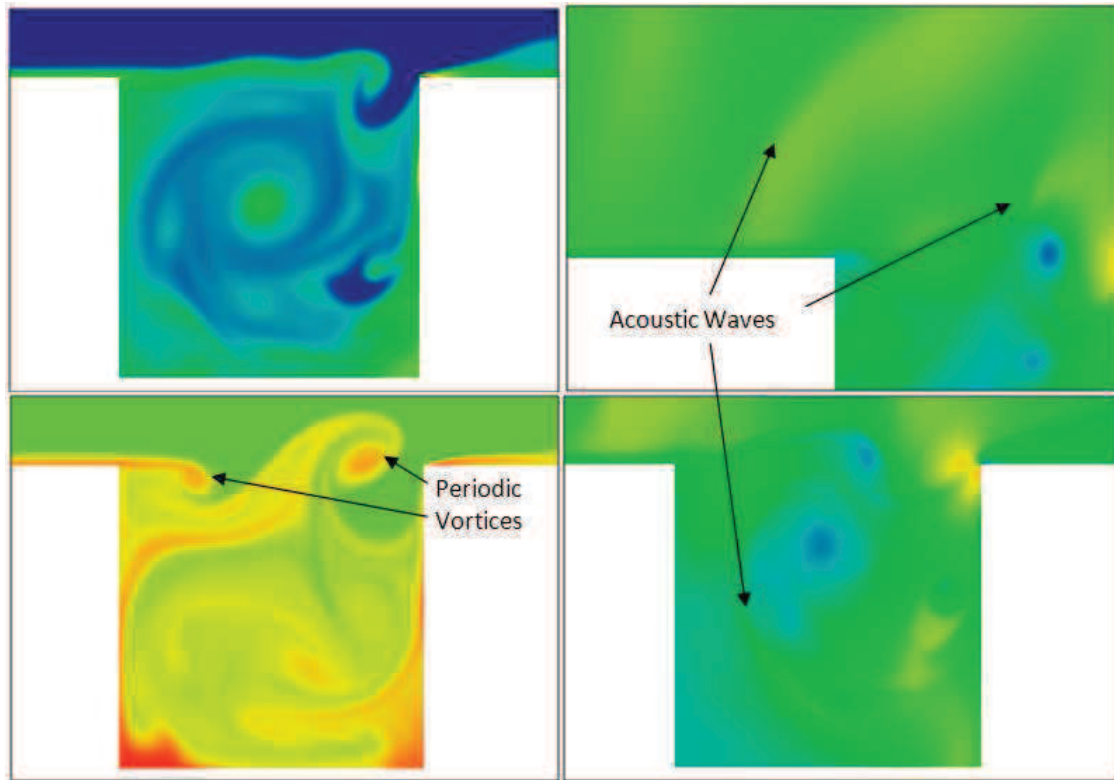


Figure 6.132: Square Cavity Simulation (NS2D).

Each cavity was initialized with freestream conditions above the cavity and zero velocity within the cavity. The external flow was required to spin-up the cavity. The shallow, square, and deep cavities required 1.5M, 1.3M, and 4.2M iterations, respectively, to produce the solutions shown below. A time step of 1.67×10^{-6} s was allow for frequencies up to 75k.

Flow in the square cavity shows a one-to-one circulation with a slight chugging effect (shown in Figure 6.134). The shallow cavity circulates in the downstream 1/3 of the cavity, while the forward 2/3 of the cavity acts like a backward facing step (shown in Figure 6.135). The deep cavity should spin-up counter-rotating circulations (shown in Figure 6.136), but the solution was abandoned before the lower circulation could begin to spin-up. The lower half of the deep cavity has only spun 1/3 of a rotation and would require an estimated 2 to 4 times longer to converge the lower section. Figure 6.137 shows the frequency spectrum at 17

locations around and above the square cavity. Figure 6.138 shows a similar plot for the shallow cavity. The deep cavity was not converged, so spectra are not given for this cavity.

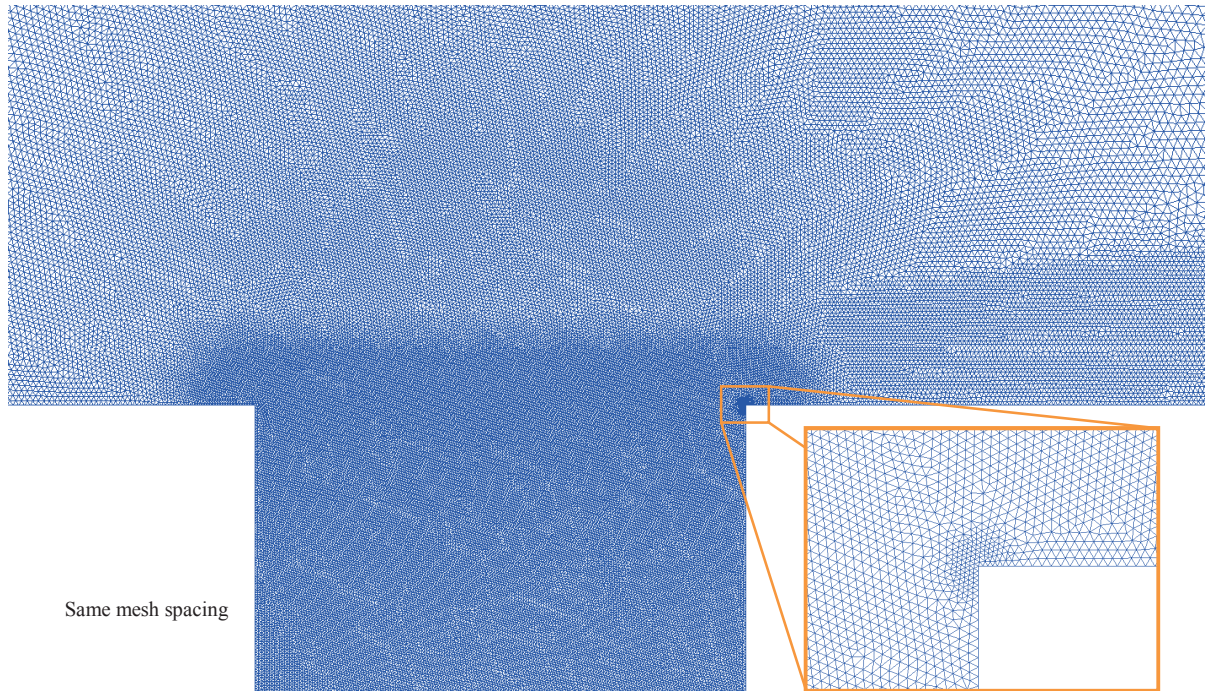


Figure 6.133: Mesh for Top of Cavities ($L:D = 1:1, 1:2,$ and $4:1$).



Figure 6.134: Entropy (left) and Velocity (right) in Square Cavity ($L:D = 1:1$).

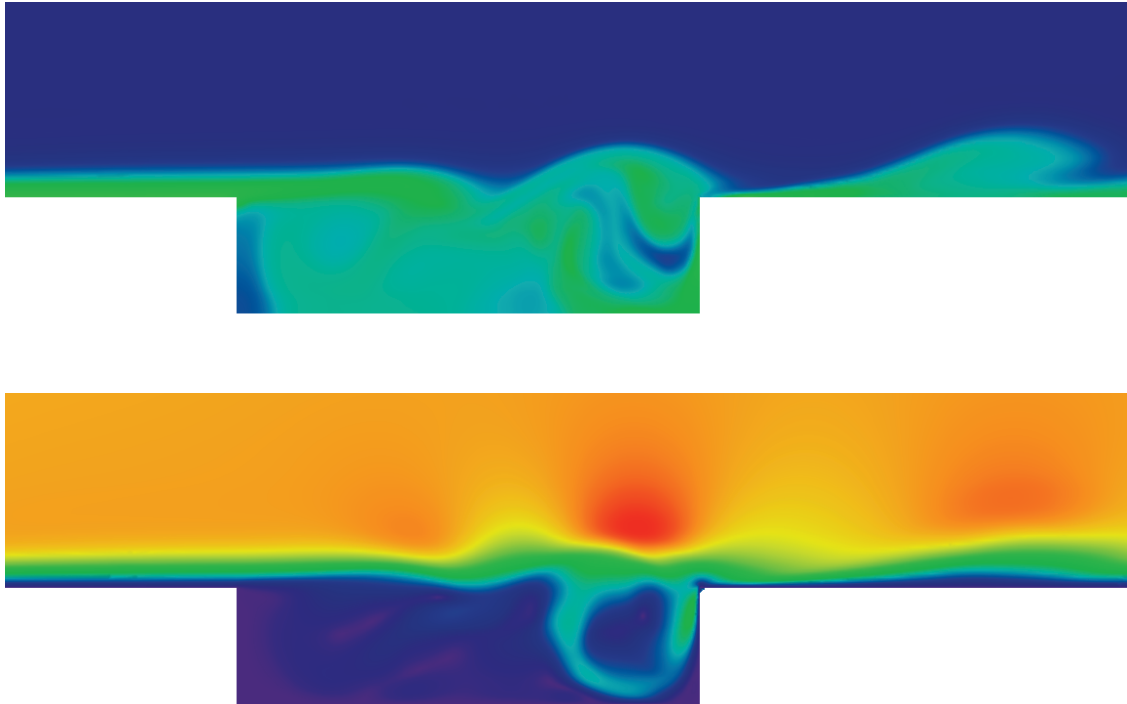


Figure 6.135: Entropy (top) and Velocity (bottom) in Shallow Cavity ($L:D = 4:1$).

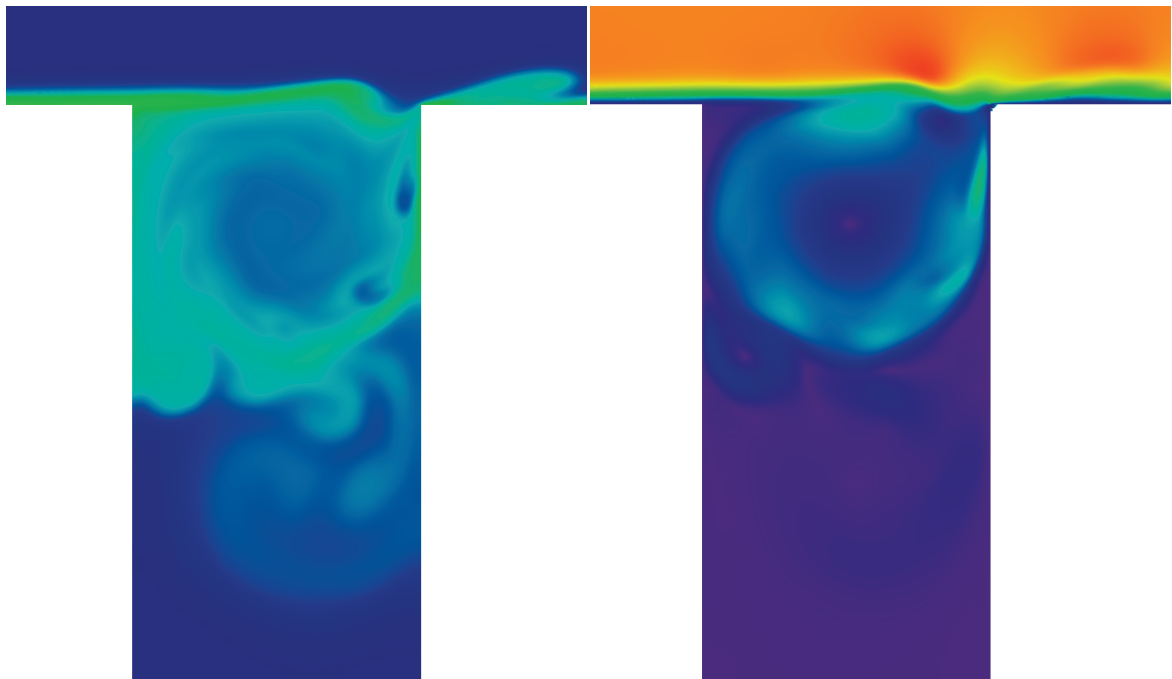


Figure 6.136: Entropy (left) and Velocity (right) in Deep Cavity ($L:D = 1:2$).

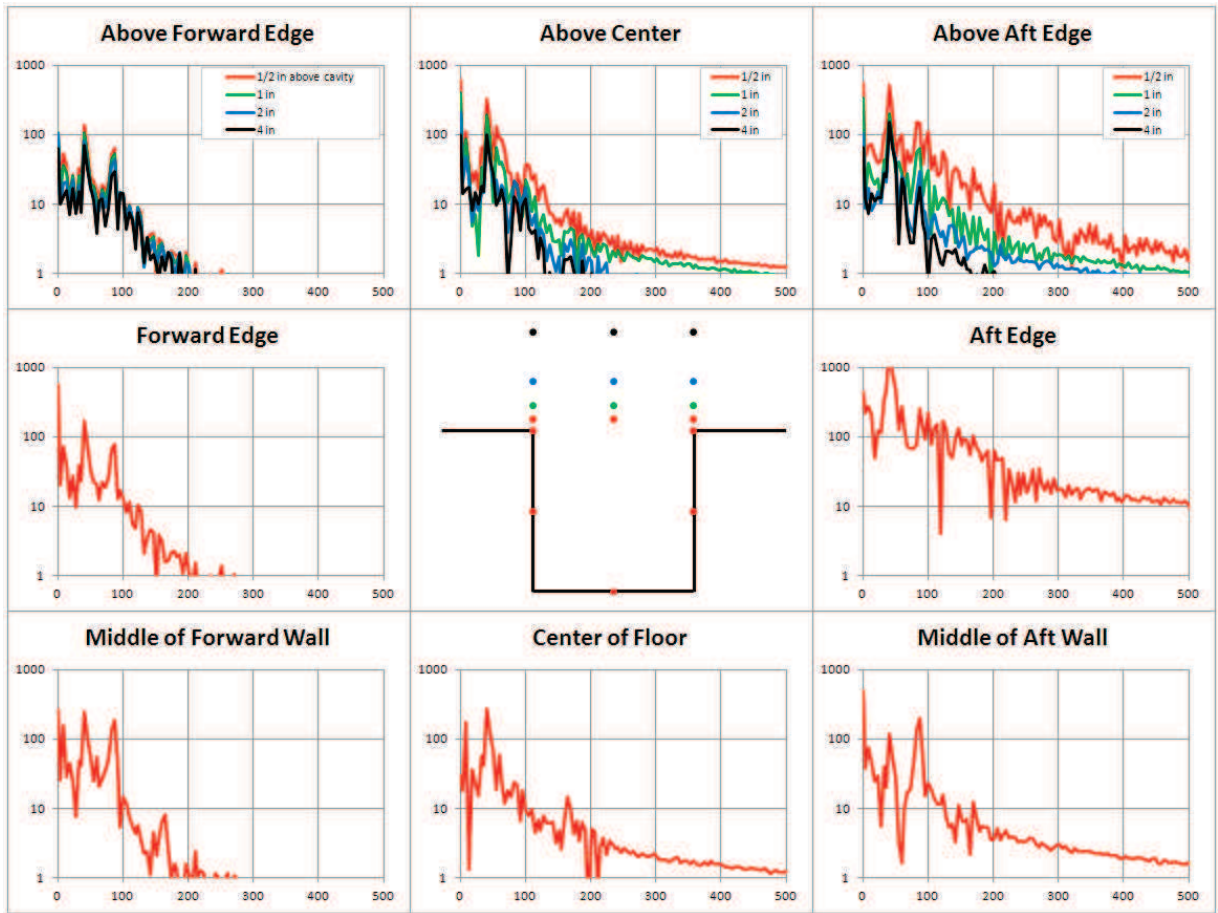


Figure 6.137: Pressure Frequency Spectra for 17 Locations around Square Cavity.

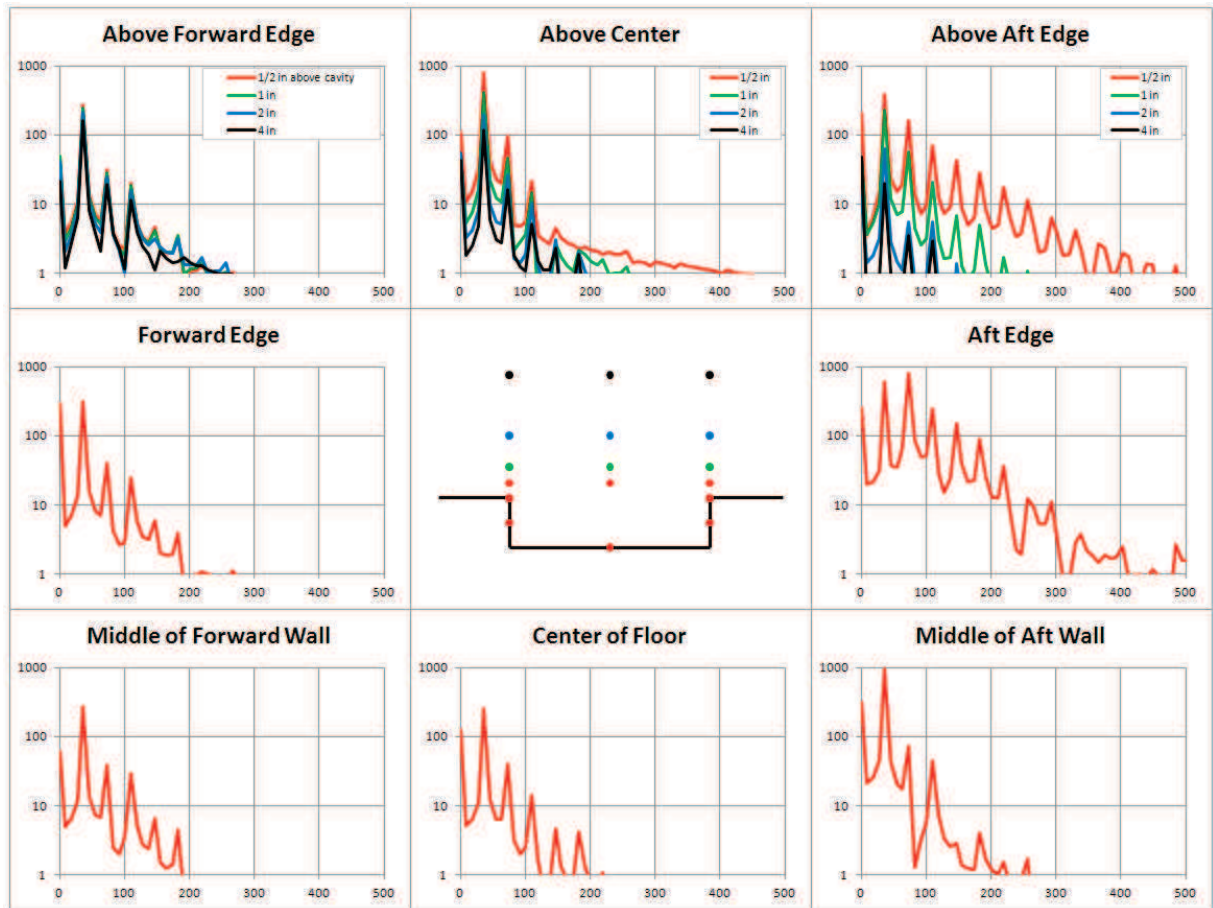


Figure 6.138: Pressure Frequency Spectra for 17 Locations around Shallow Cavity.

6.3.1.7 Lipped Cavity

Cavities are often fitted with passive flow control. These devices are used to offset the shear layer and contain the main vortex in the cavity. Sharp and round lips have been used to accomplish both of these tasks. A cavity with sharp lips was tested in NS2D at Mach 0.75. The cavity is 10.6-ft long (with the flow) and 13.6-ft deep, giving the cavity a 1.28 aspect ratio. Based on the stream wise length, the Reynolds number is 1.225×10^7 . The cavity has sharp edges that protrude 6" to 9" into the cavity along the shear flow. The sharp edges were

designed to eliminate some of the problems with the shear layer and cavity acoustics. The entropy distribution shown in Figure 6.139 shows the primary vortex within the cavity and shear layer. At the instant shown in Fig, the shear layer contains three vortices, which form on the upstream lip and break on the downstream lip. Distinct acoustic waves are released when the vortex breaks over the downstream lip. Figure 6.140 shows the acoustic waves leaving through the mouth of the cavity. The acoustic waves are emanating from the downstream corner into the freestream. The acoustic waves do travel upstream, as seen by the steep incline of the wave over the mouth of the cavity.

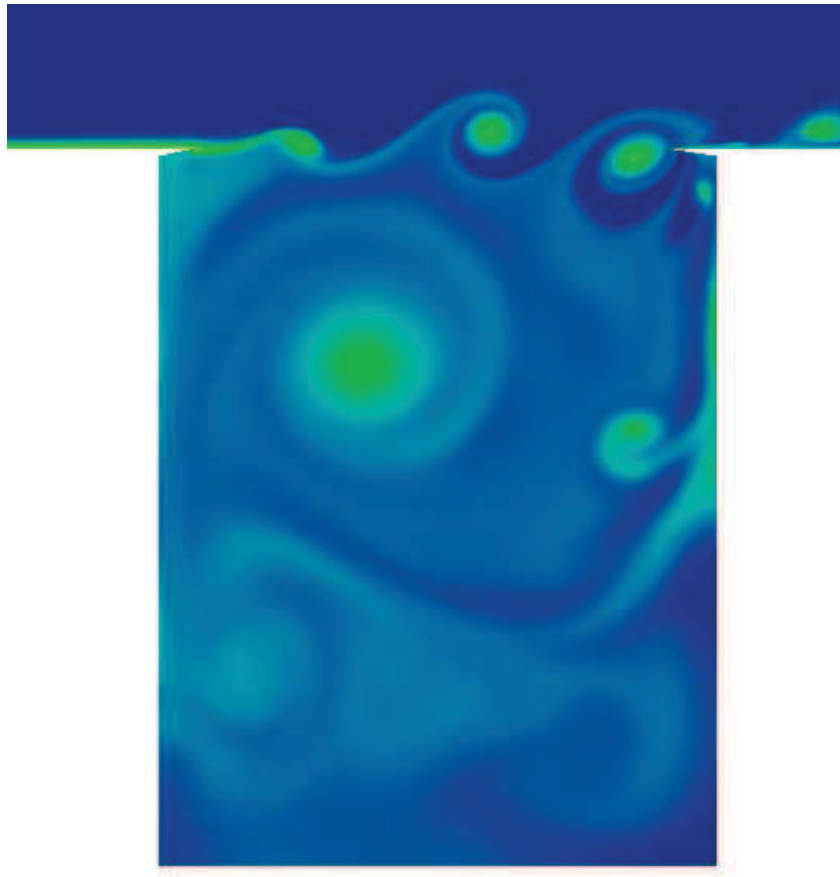


Figure 6.139: Lipped Cavity (Entropy, NS2D).



Figure 6.140: Acoustic Waves above Lipped Cavity (Density, NS2D).

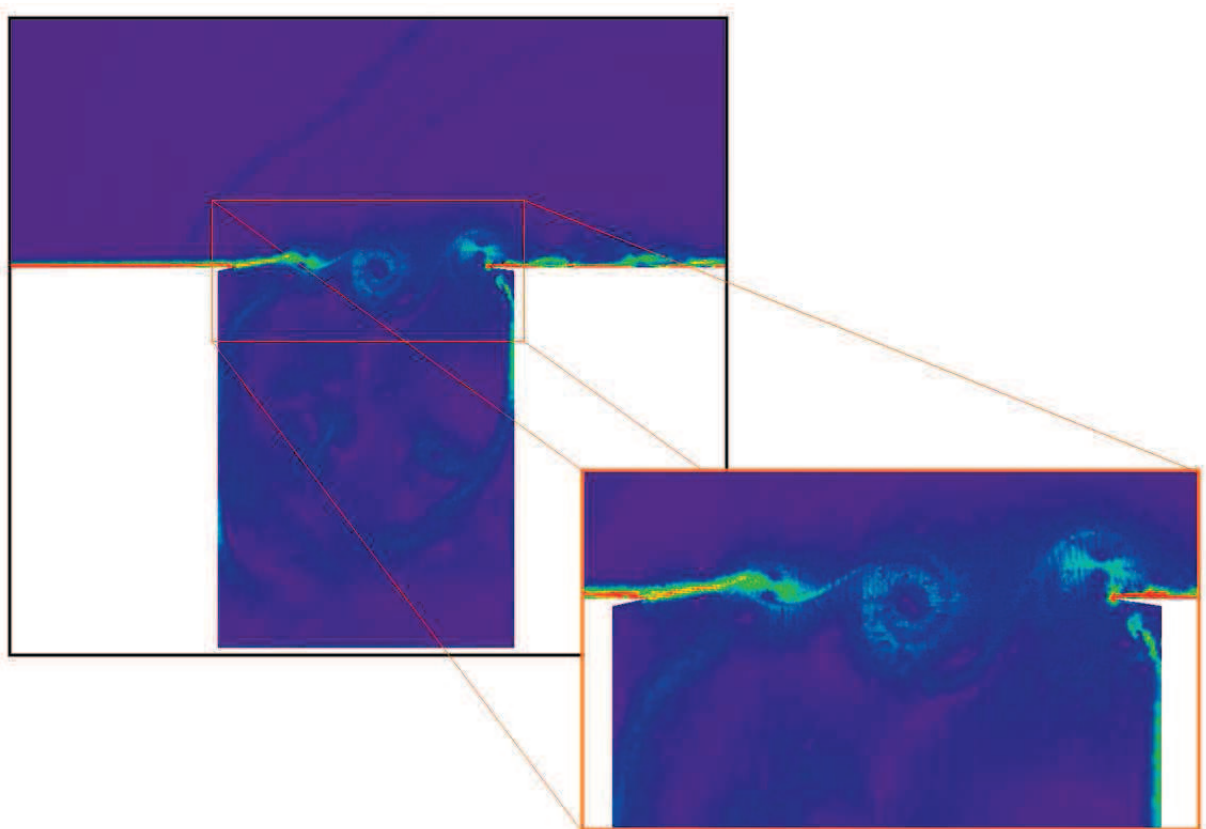


Figure 6.141: Vorticity in Lipped Cavity Flow.

6.3.2 Turbulent

The SA turbulence model has been tested on two geometries. A flat plate was used to develop guidelines for modeling a turbulent boundary layer using the SA model in NS2D. Rumsey (2012) presents CFD data for the flat plate and a similar bump-plate, which were used to demonstrate the capabilities and problems with the SA model.

6.3.2.1 SA Mesh Convergence

A guideline has already been created for laminar profiles. The guideline specifies four element across the lower 60% of the boundary layer and larger elements at the top and above the boundary layer. We would like to refine this guideline for turbulent boundary layers modeled using the SA model. Comparing the theoretical velocity profile, the highest gradients and curvatures occur in three regions nearest the wall: The viscous sublayer ($y^+ < 5$), buffer region ($5 < y^+ < 30$), and log-law regions ($30 < y^+ < 0.1\delta^+$). Theory can also be used to predict the eddy viscosity profile. The eddy viscosity is most active in the log-law region and upper boundary layer, which will be modeled using the laminar spacing guideline. The three near-wall regions will be modeled by a set number of elements across each region (N_{vis} , N_{buf} , N_{log}). Fourteen combinations were tested, as shown in Table 6.8. The spacing outside of the log-law region begins at 0.1δ and decreases to keep the spacing less than three times the spacing in the log-law region. Ideally, the spacing was kept at twice the spacing of the region nearer the wall. The mesh convergence was tested on a flat plate with a freestream Mach number of 0.3 and Reynolds number Re of 5×10^5 for reference dimension $refdim$ of 1. The SA variable $\hat{\nu}_\infty$ was initialized to a freestream value of 3. Artificial dissipation was minimized in the boundary layer using a zero dissipation length of $dislen$ of 0.1.

Figure 6.142 shows the velocity and eddy viscosity profiles within the turbulent boundary layer. The velocity profile outside of the buffer region ($y^+ > 30$) for a spacing at the wall of $y^+ = 2.5$ ($N_{vis} = 2$, $N_{buf} = 4$, and $N_{log} = 12$). The finest mesh converges to the theoretical velocity profile across the viscous sublayer and log-law region, shown in Figure 6.142 as dashed lines. The eddy viscosity profiles are more sensitive to the number of elements in the log-law region and converge for eight elements across the log-law region. The theoretical eddy viscosity profile is also shown in Figure 6.142. The mesh converged distribution matches the theoretical distribution well. The magnitude of the two distributions differs, but the theoretical profile is only a loose estimate of the eddy viscosity.

Table 6.8: Profiles Tested for SA Grid Convergence.

	$y^+ < 5$	$5 < y^+ < 20$	$20 < y^+ < 0.1\delta^+$	$0.1 < y/\delta < 0.5$	$0.5 < y/\delta < 0.9$	$0.9 < y/\delta < 1.3$	$0 < y < 1$	$1 < y < 2$	$2 < y < 4$
Plot #	N_{vis}	N_{buf}	N_{log}	dy_1/δ	dy_2/δ	dy_3/δ	dy_1	dy_2	dy_3
0	1	1	1	0.100	0.20	0.40	0.05	0.1	0.2
---	1	1	2	0.100	0.20	0.40	0.05	0.1	0.2
1	1	2	5	0.050	0.15	0.40	0.05	0.1	0.2
2	1	3	8	0.040	0.15	0.40	0.05	0.1	0.2
---	1	4	12	0.030	0.13	0.40	0.05	0.1	0.2
3	2	4	12	0.030	0.13	0.40	0.05	0.1	0.2
4	2	6	18	0.030	0.13	0.40	0.05	0.1	0.2
5	2	6	24	0.015	0.05	0.20	0.03	0.1	0.2
6	3	6	24	0.015	0.05	0.20	0.03	0.1	0.2
7	3	8	32	0.010	0.04	0.16	0.03	0.1	0.2
8	4	10	40	0.008	0.03	0.12	0.03	0.1	0.2
9	5	12	48	0.007	0.03	0.12	0.03	0.1	0.2
---	6	14	56	0.006	0.03	0.12	0.03	0.1	0.2
---	7	17	68	0.005	0.03	0.12	0.03	0.1	0.2

Figure 6.143 shows the convergence of skin friction at four locations across the plate ($x = 3$, 4, 6, and 7, corresponding to $Re_x = 1.5M$, 2M, 3M, and 3.5M). The skin friction at all four locations shows a convergence to 5% of the finest mesh for a spacing near the wall of $y^+ = 5$ ($un\# = 2$, $N_{vis} = 1$, $N_{buf} = 3$, $N_{log} = 8$). The skin friction along the plate is also shown in Figure 6.143 compared to the theoretical laminar profile and empirical turbulent skin friction from

Bertin and Smith (1998). (The transitional skin friction is a spline created to match the values and slopes at 2×10^5 and 3×10^6 .) The skin friction shows the freestream value \hat{v}_∞ of 3 has created a wholly “turbulent” boundary layer. The skin friction matches the empirical trend beyond $x = 5$ ($Re_x = 2.5M$).

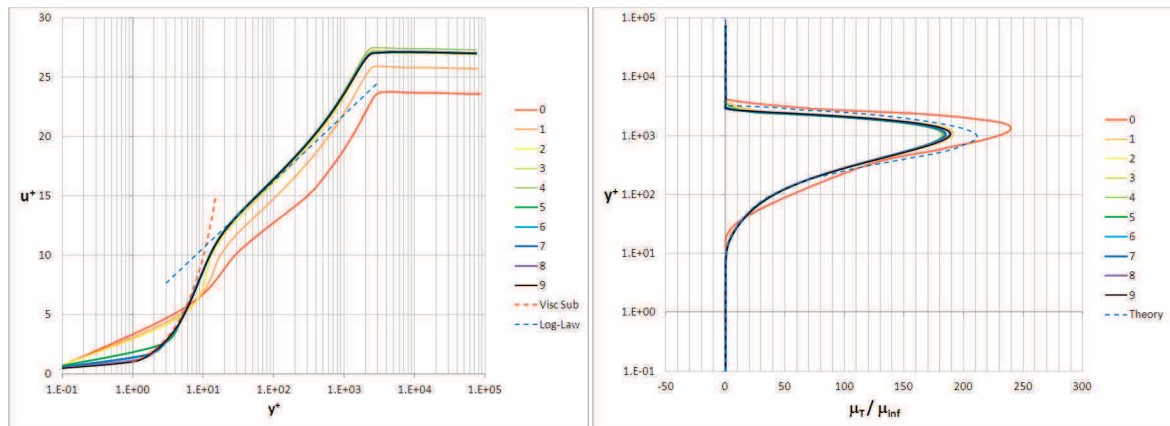


Figure 6.142: Grid Convergence of SA Velocity and Eddy Viscosity Profiles at $x = 7$.

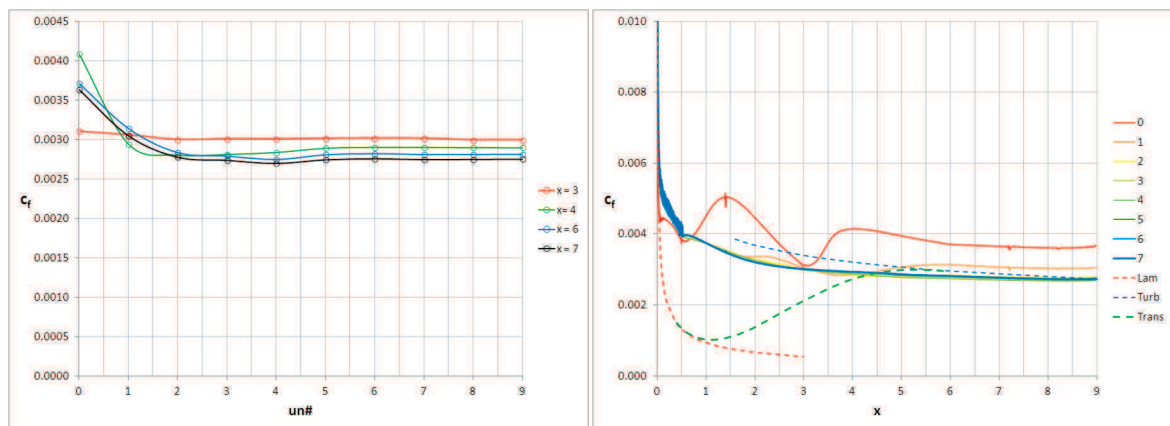


Figure 6.143: Convergence of SA Skin Friction with Near-Wall Spacing.

Finally, the profile properties δ^+ , θ , H , and $\mu_{T,max}$ are shown in Figure 6.144. The shape factor H is the least sensitive to near-wall spacing. The displacement thickness δ^+ and momentum thickness θ converge for a wall spacing of $y^+ = 5$ ($N_{vis} = 1$). The maximum eddy

viscosity $\mu_{T,max}$ is sensitive to the number of elements across the log-law region. The maximum eddy viscosity converged for 5 elements across the log-law region ($N_{log} = 5$).

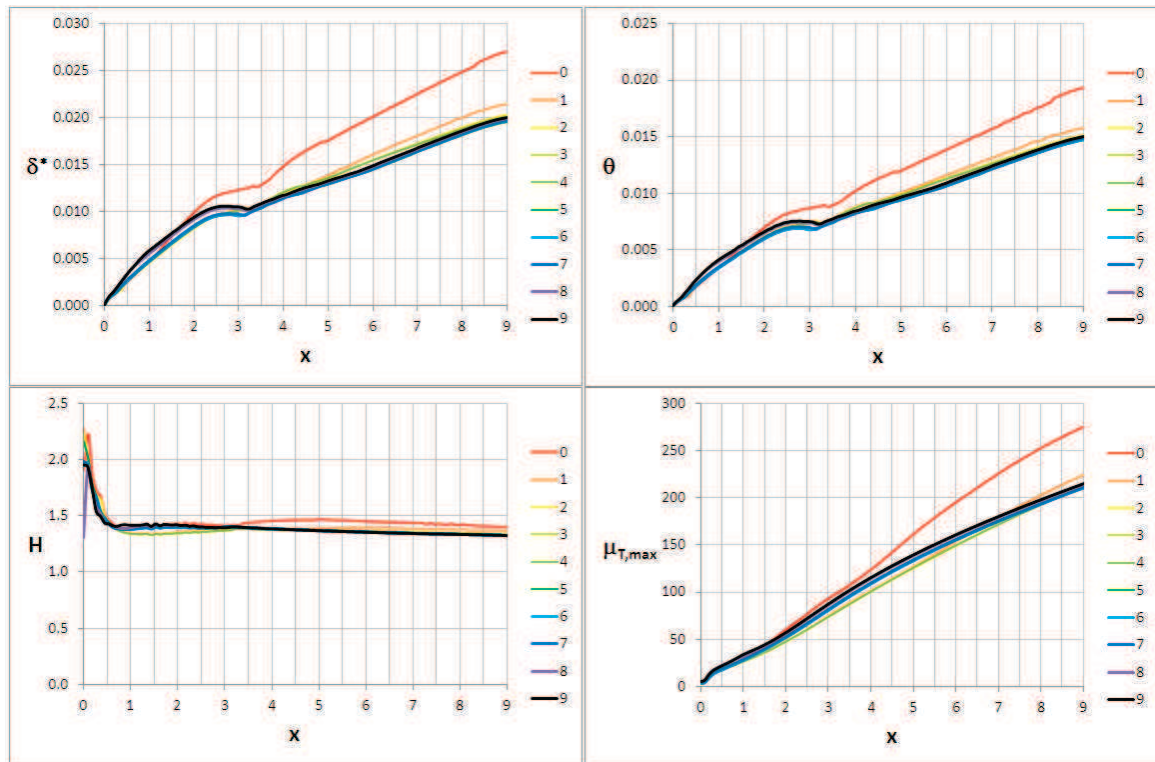


Figure 6.144: Grid Convergence of Profile Properties with SA Turbulence Model.

Taking all of these comparisons into account, a guidelines is created to use the laminar spacing outside the log-law region ($y > 0.1 \delta$). At minimum one element should span the viscous sublayer (N_{vis}), at least three elements for the buffer region (N_{buf}), and at least five elements (preferred eight elements) across the log-law region (N_{log}). For efficient meshing, a y^+ of 5 should be used at the wall ($N_{vis} = 1$, $N_{buf} = 3$, $N_{log} = 8$). For greater accuracy in the velocity profile, a y^+ of 2.5 should be used ($N_{vis} = 2$, $N_{buf} = 4$, $N_{log} = 12$).

6.3.2.2 Rumsey Flat Plate (SA)

The NASA Langley Turbulence Modeling Resource site (Rumsey, 2012) has collected variations of several turbulence models and data from several cases. These cases can be used to verify the accuracy of a given turbulence model implementation. Rumsey presents data for a zero pressure gradient flat plate from two compressible finite volume codes FUN3D and CFL3D with the SA model. Rumsey has made a family of five structures grids available. These meshes are used to demonstrate mesh convergence and then compare solutions one-to-one on the same grid. These grids are named here according to their relative spacing: Mesh 0, Mesh 1, ..., Mesh4, with Mesh 4 being the coarsest grid. Rumsey's grids have been converted to unstructured triangular meshes (Mesh#.g2d) using Convert_Plate.

(Convert_Plate subdivides each quad using either rising, falling or alternating diagonals. For alternating diagonals, NS2D was unstable. The alternating diagonal creates two types of nodes: Those connected to 4 segments, and those connected to 8 segments. The stability of these nodes are very different. Rising and falling diagonals are stable in NS2D and give similar solutions. These diagonal produce similar nodes, which connect to 6 segments.)

Figure 6.145 shows the converted Mesh 4. A black trapezoid is used to illustrate the existence of the plate along the bottom boundary. The plate measures 2 units in length with a Reynolds number Re_L of 10^7 . The coarsest mesh (Mesh 4) has a minimum spacing at the wall of approximately $y^+ = 1.7$, which is finer than the spacing suggestions in the previous section. The subsequent meshes are nested by cutting each element edge in half, so that the elements in Mesh 0 are one-eighth the dimension of the elements in Mesh 4. The minimum

spacing at the wall is approximately $y^+ = 0.1$. Rumsey specifies the following freestream conditions: $mach = 0.2$, $Re = 5 \times 10^6$, $refdim = 1$, and $\hat{v}_\infty = 3$.

A solution was converged on Mesh 4 (800k iterations, $ncyc = 4$) and Mesh 3 (1.6M iterations, $ncyc = 4$). The Mesh 3 solution was used as an initial starting point for the other three meshes (Mesh 2 – 2.4M iterations; Mesh 1 – 1.6M iterations; Mesh 0 – 1.2M iterations). The solution for Mesh 0 was the only solution that did not present convergence over 400k iterations or more. Mesh 0 was converging but needed 400 to 800k more iterations. The convergence of Mesh 0 only proved to be a problem for the downstream profiles shown in Figure 6.148.

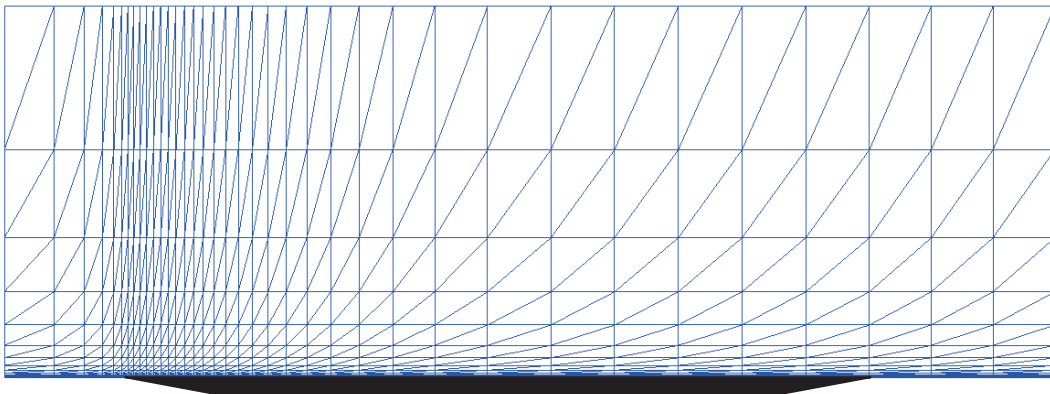


Figure 6.145: Geometry for Rumsey’s Flat Plate Grids (Mesh 4).

Figure 6.146 shows the mesh convergence of skin friction along the plate. The results from NS2D are compared with results from FUN3D on a similar triangular mesh. The skin friction is erratic for Mesh 4 and Mesh3, which have not been refined enough in the stream-wise direction. Mesh 2 begins to show convergence. Mesh 1 and Mesh 0 begin to resemble the FUN3D solution.

Two cuts are taken at $x = 0.97$ and 1.83 . The velocity and eddy viscosity profiles at these cuts are shown in Figure 6.147 and Figure 6.148, respectively, compared to FUN3D and CFL3D. The profiles at $x = 0.97$ match very well with the other CFD results. At $x = 1.83$, the NS2D profiles converge for Mesh 1. (Mesh 0 showing a lacking of convergence.) The NS2D results compare well with the profile from FUN3D in the buffer and external regions of the plot. u^+ and y^+ will only match in the external flow when the same wall skin friction is used for both profiles. Strangely, the wall gradients are similar, but the log-law regions of the two profiles are very different. FUN3D predicts a higher velocity in the log-law region than NS2D. The downstream profiles are much more sensitive to mesh spacing in NS2D.

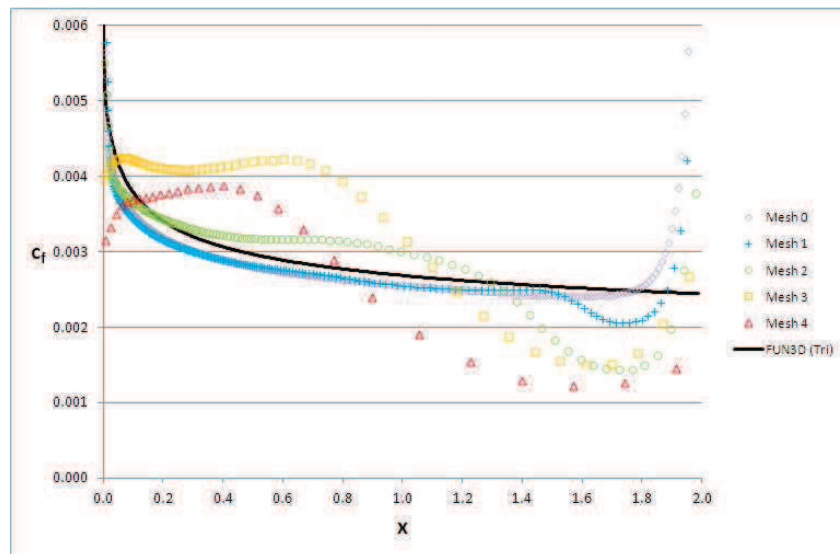


Figure 6.146: Skin Friction along Rumsey Plate for 5 Meshes in Family.

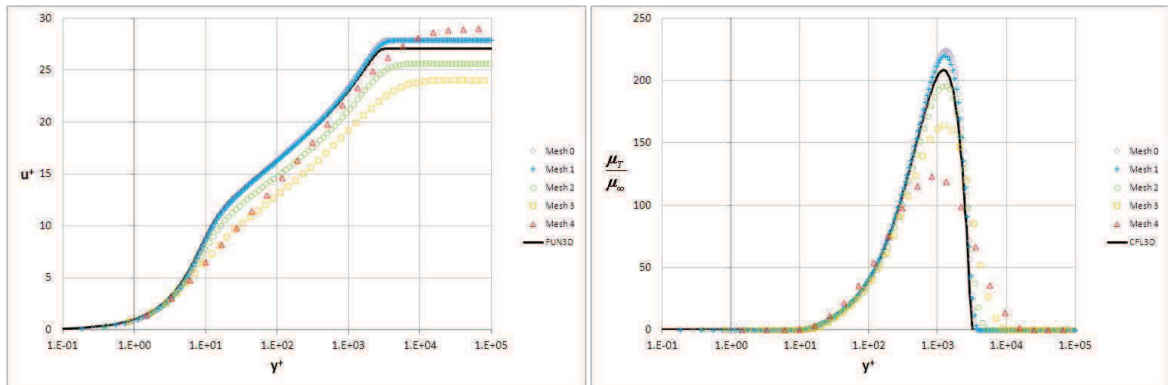


Figure 6.147: Convergence of Velocity and Eddy Viscosity Profiles at $x = 0.97$.

Figure 6.149 compares the maximum eddy viscosity predicted by NS2D for each of the five meshes to that predicted by FUN3D on the finest grid. The NS2D distribution begins to resemble that from FUN3D when the streamwise spacing has converged beyond Mesh 2. The comparison with Rumsey's data verifies the implementation of the SA model in NS2D.

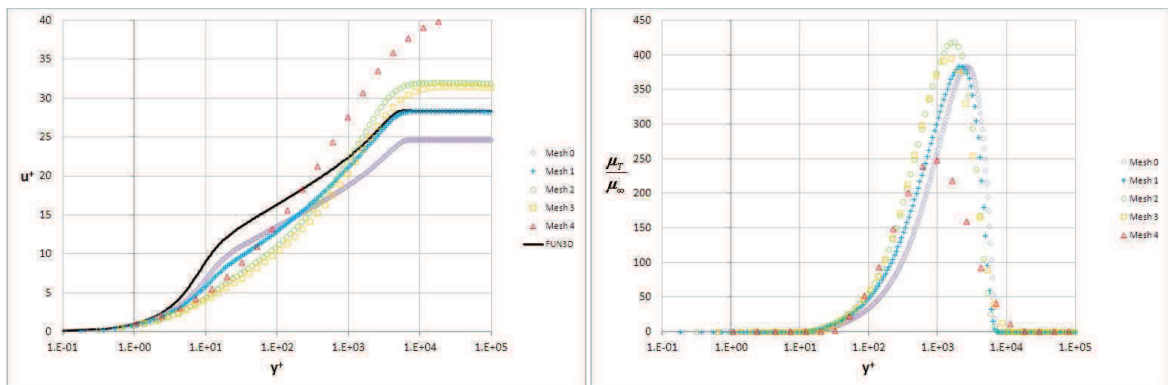


Figure 6.148: Convergence of Velocity and Eddy Viscosity Profiles at $x = 1.83$.

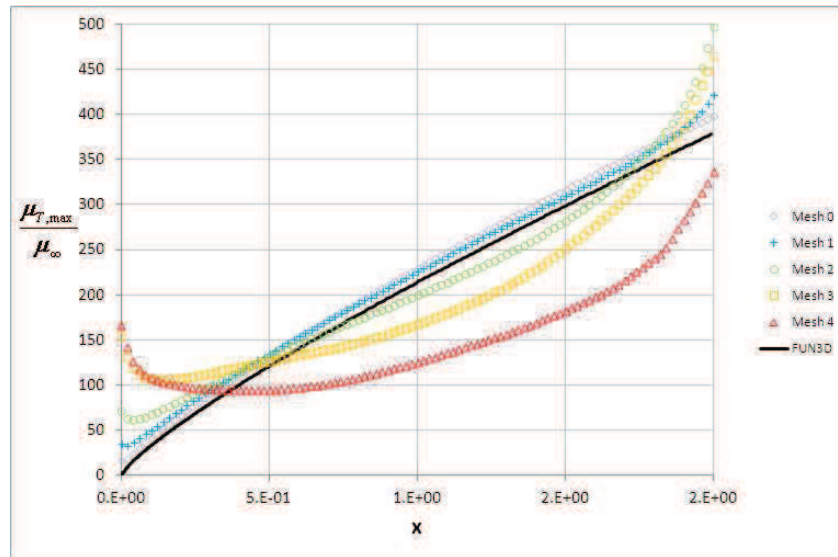


Figure 6.149: Maximum Eddy Viscosity as a Function of Distance along Plate.

6.3.2.3 Rumsey Bump Plate (SA)

Rumsey (2012) also presents data for a bump on a second plate. The bump is small and smooth but presents a problem to NS2D. The bump geometry is shown in Figure 6.150 on the coarsest mesh converted from Rumsey's grids using Convert_Bump. The bump meshes show the same nesting as the previous case. The bump was tested with freestream conditions: $mach = 0.2$, $Re = 3 \times 10^6$, $refdim = 1$, and $\hat{v}_{\infty} = 3$. The plate has a length of 1.5 units with the bump (0.05 units high, 0.9 units long). This gives the plate a Reynolds number Re_L of 4.5×10^6 . Figure 6.151 shows the convergence of pressure for the bump case. Only the coarsest mesh cannot produce the desired pressure distribution.

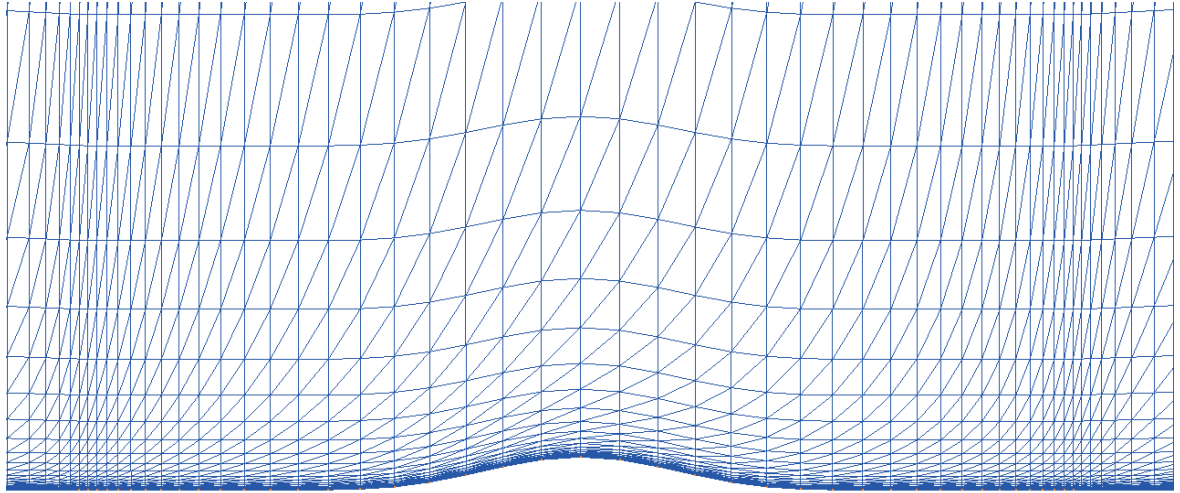


Figure 6.150: Rumsey's Coarsest Grid for Bump Plate Case.

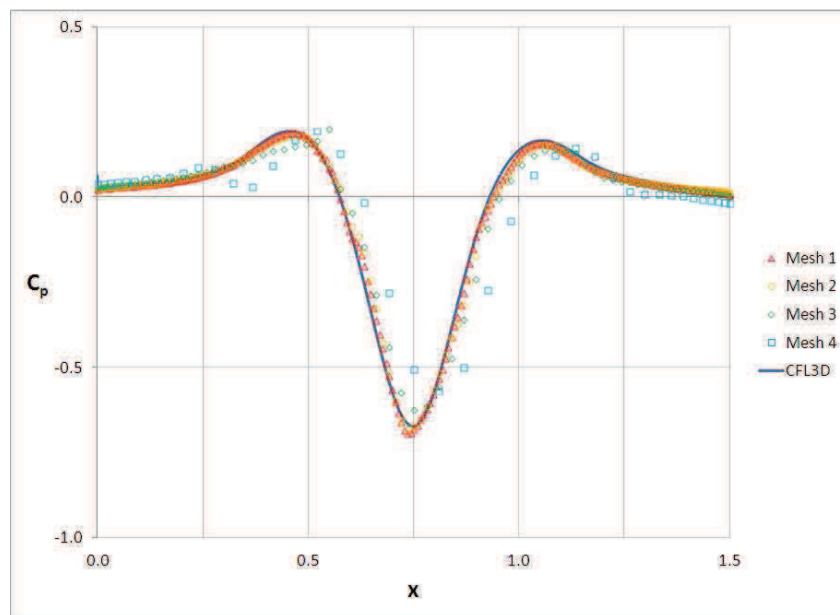


Figure 6.151: Pressure Distribution over Bump Plate for Four NS2D Solutions.

NS2D struggles to produce a reasonable skin friction over Rumsey's bump (seen in Figure 6.152). Figure 6.153 shows the pressure, velocity, and eddy viscosity distributions around the bump and plate. The solutions are smooth and do not help explain the discontinuities in the skin friction distribution. Figure 6.154 shows the velocity vectors around the bump. The

velocity vectors show the boundary layers and downstream separation. The velocities vectors show some perturbations normal to the bump but the solution still looks relatively smooth and reasonable. Rumsey presents a velocity and eddy viscosity profile at the top center of the bump. These profiles are compared to the solutions calculated with NS2D on the four meshes, shown in Figure 6.155. On the finest mesh (Mesh 1), NS2D predicts a velocity profile that resembles Rumsey's profile. NS2D under-predicts the velocity through the log-law region. (The velocity distributions can be collapsed into a single profile if the skin friction convergence in Figure 6.152 are taken into account.) The eddy viscosity predicted using NS2D shows a similar distribution to Rumsey's profile, but NS2D predicts magnitudes twice that seen in Rumsey's profile. Figure 6.156 shows the maximum eddy viscosity along the length of the plate. NS2D matches Rumsey's profile over the flat region upstream of the bump. NS2D does not predict the decrease in eddy viscosity over the bump, like that seen in Rumsey's profile. NS2D either over predicts turbulent production or under predicts the destruction of turbulence.

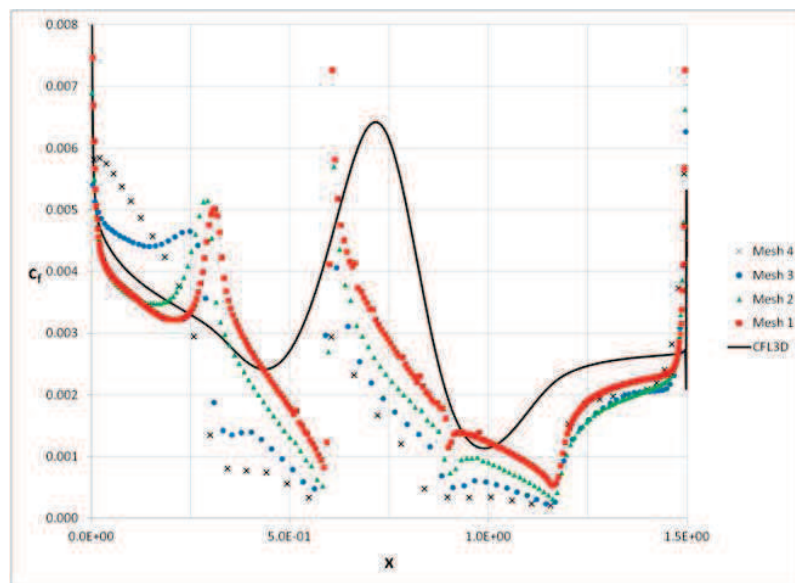


Figure 6.152: Skin Friction along Bump Plate for Four NS2D Solutions.

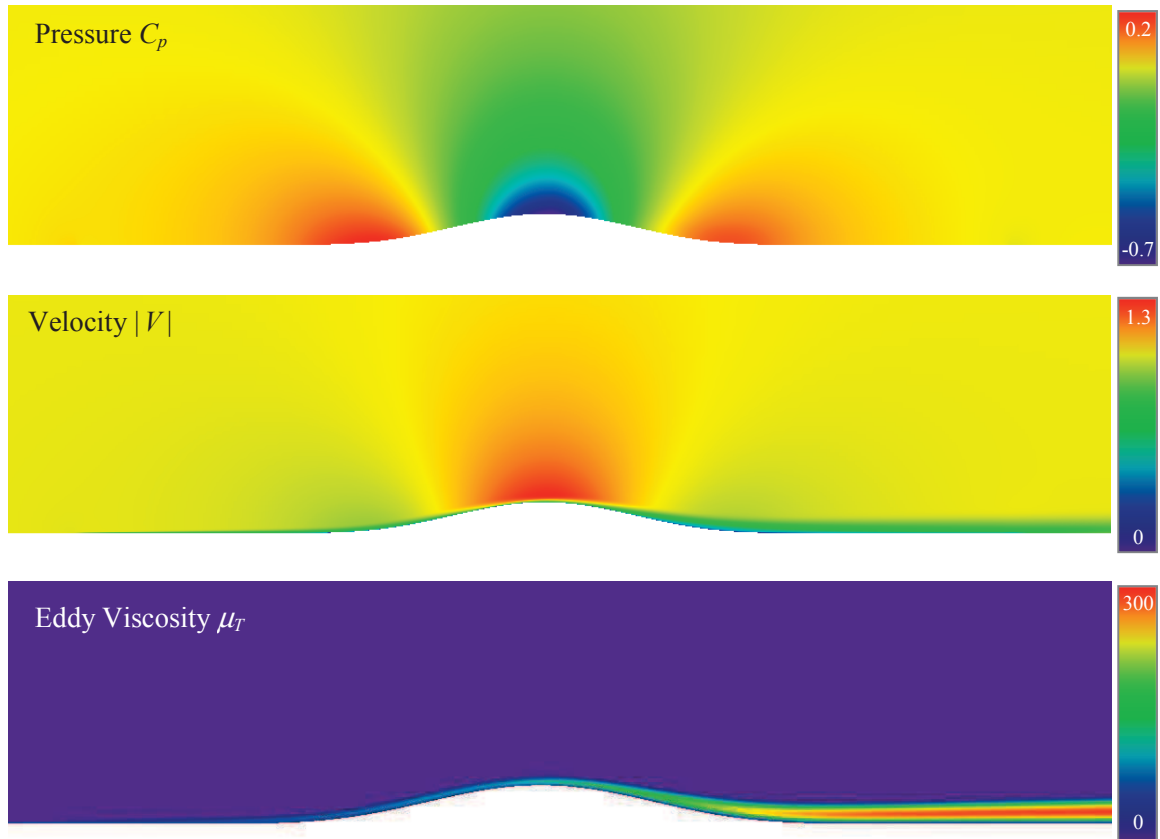


Figure 6.153: Pressure, Velocity, and Eddy Viscosity Distribution around Bump Plate.

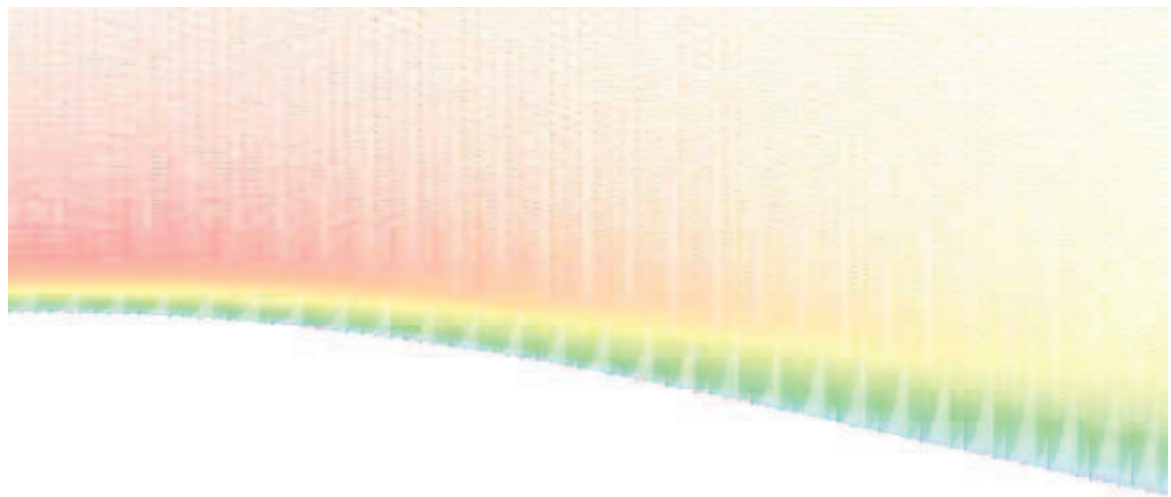


Figure 6.154: Velocity Vectors around Bump Plate.

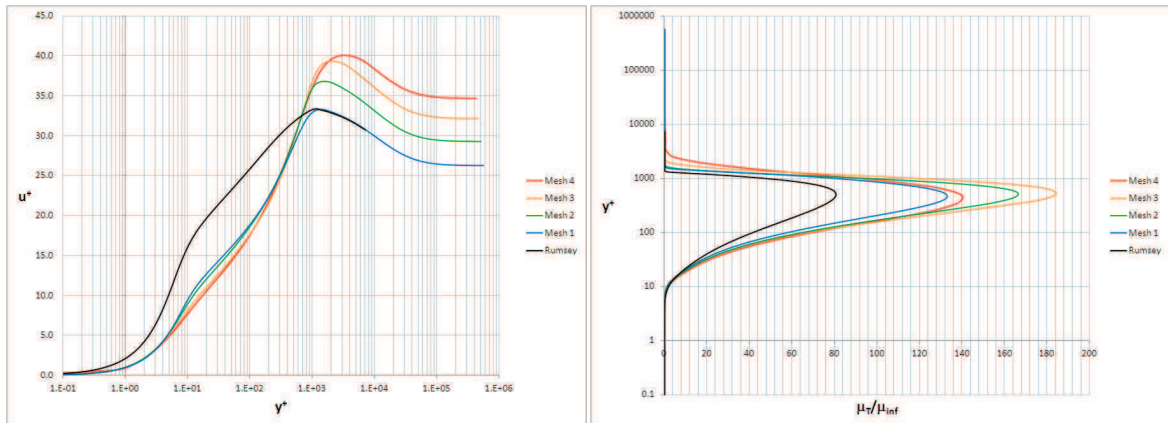


Figure 6.155: Velocity and Eddy Viscosity Profiles at Top of Bump ($x = 0.75$).

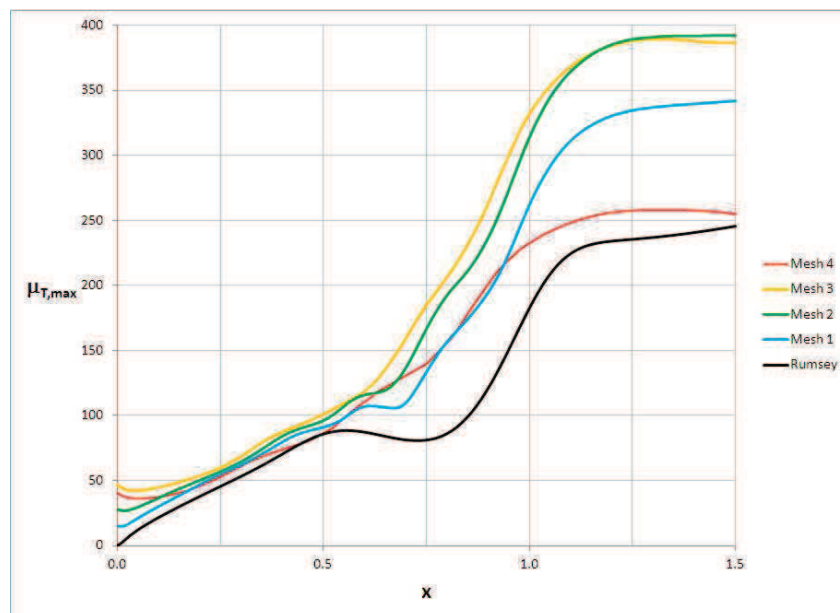


Figure 6.156: Maximum Eddy Viscosity along Bump Plate for Four NS2D Solutions.

The destruction of turbulence is calculated using a series of scalar functions. The production term is calculated using the vorticity, which is derived from the velocity derivatives. What if the vorticity is artificially increased over the bump because of an error in the finite element discretization, mesh, or combination? Orthogonal elements show a decoupling of the gradient, which could artificially increase the vorticity over a curved surface. Take the elements shown in Figure 6.157. The two elements share two nodes x_2 and x_3 :

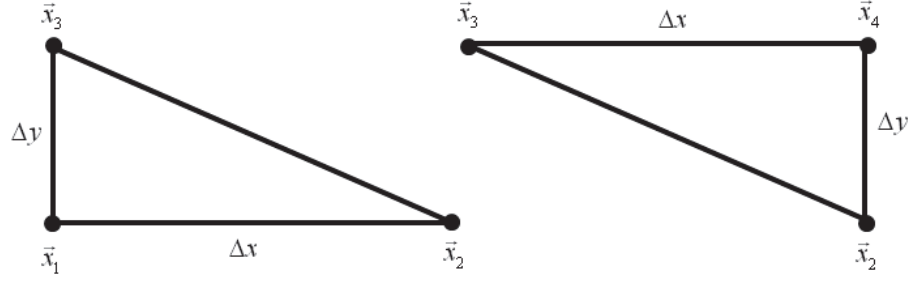


Figure 6.157: Orthogonal Elements.

From Eq. 4.16, the gradient of a property p on the lower triangular element is calculated:

$$x_{13} = 0 \quad y_{13} = -\Delta y \quad x_{12} = -\Delta x \quad y_{12} = 0 \quad (6.1)$$

$$x_{23} = x_{13} - x_{12} = \Delta x \quad y_{23} = y_{13} - y_{12} = -\Delta y \quad (6.2)$$

$$\left(\frac{\partial p}{\partial \vec{x}} \right)_{123} = \frac{1}{\Delta x \Delta y} \begin{bmatrix} -\Delta y & \Delta y & 0 \\ -\Delta x & 0 & \Delta x \end{bmatrix} \begin{Bmatrix} p_1 \\ p_2 \\ p_3 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{\Delta x} (p_2 - p_1) \\ \frac{1}{\Delta y} (p_3 - p_1) \end{Bmatrix} \quad (6.3)$$

The gradient of p on the upper triangular element is calculated:

$$x_{24} = 0 \quad y_{24} = -\Delta y \quad x_{34} = \Delta x \quad y_{34} = 0 \quad (6.4)$$

$$x_{23} = x_{24} - x_{34} = -\Delta x \quad y_{23} = y_{24} - y_{34} = -\Delta y \quad (6.5)$$

$$\left(\frac{\partial p}{\partial \vec{x}} \right)_{234} = \frac{1}{\Delta x \Delta y} \begin{bmatrix} 0 & \Delta y & -\Delta y \\ -\Delta x & 0 & \Delta x \end{bmatrix} \begin{Bmatrix} p_2 \\ p_3 \\ p_4 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{\Delta x} (p_3 - p_4) \\ \frac{1}{\Delta y} (p_4 - p_2) \end{Bmatrix} \quad (6.6)$$

If the properties are components of velocity, then the shear stress on each element becomes:

$$(\tau_{xy})_{123} = \frac{\mu}{\text{Re}} \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right)_{123} = \frac{\mu}{\text{Re}} \left(\frac{u_3 - u_1}{\Delta y} + \frac{v_2 - v_1}{\Delta x} \right) \quad (6.7)$$

$$(\tau_{xy})_{234} = \frac{\mu}{\text{Re}} \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right)_{234} = \frac{\mu}{\text{Re}} \left(\frac{u_4 - u_2}{\Delta y} + \frac{v_3 - v_4}{\Delta x} \right) \quad (6.8)$$

Similarly the vorticity on each element is calculated:

$$\Omega_{123} = \left| \frac{\partial u}{\partial y} - \frac{\partial v}{\partial x} \right|_{123} = \left| \frac{u_3 - u_1}{\Delta y} - \frac{v_2 - v_1}{\Delta x} \right| \quad (6.9)$$

$$\Omega_{234} = \left| \frac{\partial u}{\partial y} - \frac{\partial v}{\partial x} \right|_{234} = \left| \frac{u_4 - u_2}{\Delta y} - \frac{v_3 - v_4}{\Delta x} \right| \quad (6.10)$$

If nodes 1 and 2 exist on a no-slip wall, then $u_1 = u_2 = v_1 = v_2 = 0$; and, if the velocity off of the wall is perturbed in the wall normal direction, then $u_3 = u_4 = U$ and $v_3 = -v_4 = \Delta v$. The shear stresses and vorticity on the neighboring elements become:

$$(\tau_{xy})_{123} = \frac{\mu}{\text{Re}} \left(\frac{U - 0}{\Delta y} + \frac{0 - 0}{\Delta x} \right) = \frac{\mu}{\text{Re}} \frac{U}{\Delta y} \quad (6.11)$$

$$(\tau_{xy})_{234} = \frac{\mu}{\text{Re}} \left(\frac{U - 0}{\Delta y} + \frac{\Delta v - (-\Delta v)}{\Delta x} \right) = \frac{\mu}{\text{Re}} \left(\frac{U}{\Delta y} + \frac{2\Delta v}{\Delta x} \right) \quad (6.12)$$

$$\Omega_{123} = \left| \frac{U}{\Delta y} \right| \quad \Omega_{234} = \left| \frac{U}{\Delta y} - \frac{2\Delta v}{\Delta x} \right| \quad (6.13)$$

The vorticity should be the same on the two elements to predict a consistent turbulent source term. The inconsistencies seen in Eq. 6.13 could reflect badly through Oliver's (2008) adaptation to \hat{S} (Eq. 3.154). This becomes coupled with Eqs. 6.11 and 6.12 when the eddy viscosity is used to calculate the Reynolds stress applied in the momentum equation. The Reynolds stresses are also different based on the upper or lower element in the pair. The highly stretched elements in Rumsey's bump grids would create orthogonal elements. The explanation is incomplete; otherwise a solution would be found, and the code corrected. This problem needs further investigation.

CHAPTER VII

COMPARISION WITH NASA-CFDSOL

CFDs_{sol} was demonstrated for several cases representing its final capability. The cases encompassed steady and unsteady subsonic, transonic, and supersonic fields. Various geometries were used to demonstrate the usefulness of the code for inviscid and viscous solutions. The new additions were also demonstrated: Quasi-combustion, rocket BC, transpiration, and non-inertial frame. CFD_{sol} was verified with analytical and numerical solutions. As part of the NASA contract, the four in-house OSU codes were also compared to CFD_{sol} as an evaluation of the accuracy and efficiency of CFD_{sol}.

7.1 Inviscid Aerodynamics

The inviscid test cases are divided into groups according to flow regime: Subsonic, transonic, and supersonic. Several subsonic airfoils were used to demonstrate the low speed capabilities. Two airfoils were tested at low subsonic speeds and used to demonstrate time-accuracy. A cylinder, ellipse, and sphere were generated for comparison with their viscous counterparts. Five airfoils were tested at transonic speeds. Simple compression and

expansion corners were verified at supersonic speeds, followed by a supersonic double-wedge airfoil.

7.1.1 Subsonic

Six subsonic cases were used to demonstrate the capabilities of CFDsol under inviscid conditions at Mach numbers from 0.3 to 0.6. A circular cylinder, ellipse, and sphere were demonstrated at inviscid conditions in preparation for later viscous solutions. An NACA 0012 airfoil was demonstrated in comparison to theory and experimental data, and an RAE 2822 airfoil was compared to additional experimental data. Finally, the discussion is closed with a time-accurate solution of the Wagner airfoil.

7.1.1.1 NACA 0012 Airfoil (Mach 0.3, 5 deg)

An NACA 0012 airfoil was tested under two steady conditions: A low subsonic case was compared to a theoretical solution, and a high subsonic case was compared to experimental data. The airfoil was tested at a low subsonic speed and an angle of attack to create two distinct pressure distributions in comparison to an analytical solution. The airfoil was tested at Mach 0.3 at 2-degrees angle of attack using the mesh shown in Figure 7.1 and compared with a Smith-Hess panel method (Arena, unpublished; Katz, 2001), corrected for compressibility using Prandtl-Glauert (Anderson, 2001). The pressure matches very well over the entire airfoil.

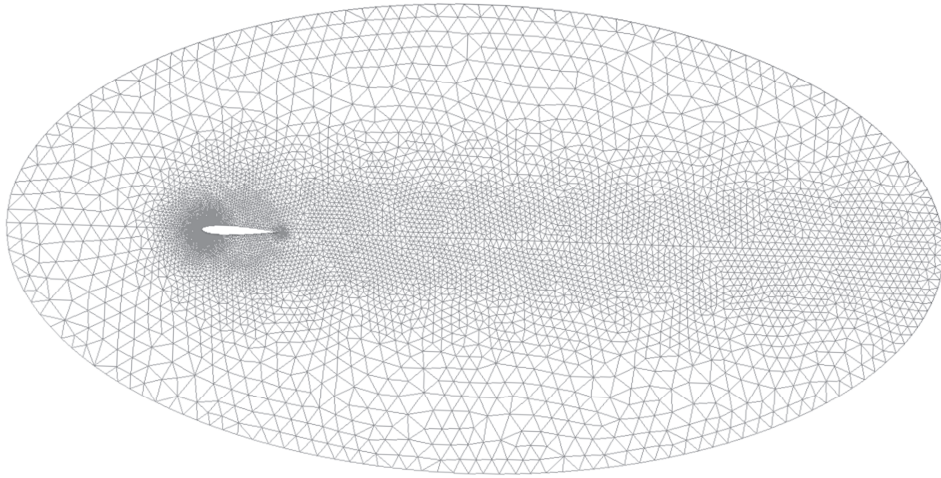


Figure 7.1: NACA 0012 Airfoil Mesh (Mach 0.3, 2-degrees AOA).

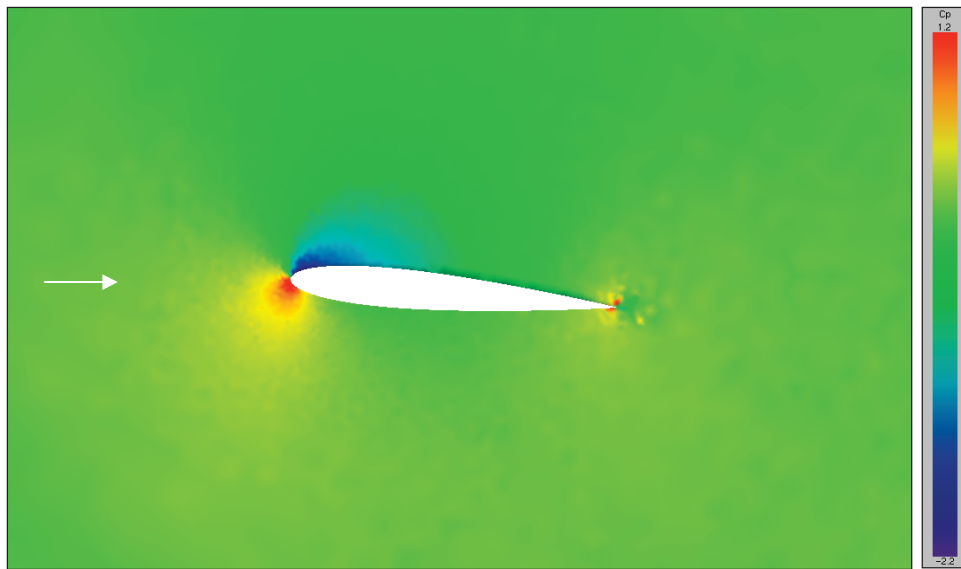


Figure 7.2: Pressure Distribution around NACA 0012 Airfoil at Mach 0.3 and 2° AOA.

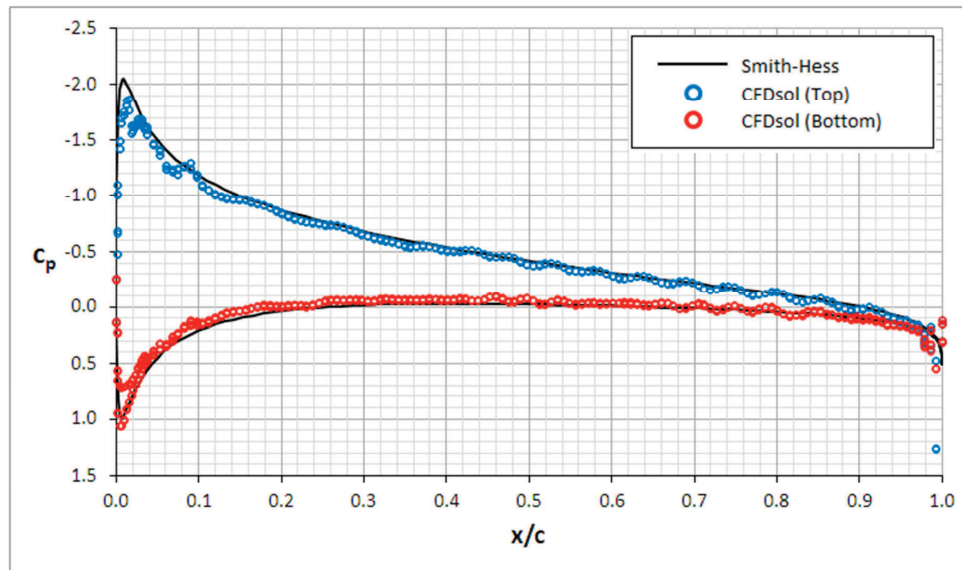


Figure 7.3: Pressure Distribution around NACA 0012 Airfoil at Mach 0.3 and 2° AOA.

7.1.1.2 NACA 0012 Airfoil (Mach 0.502, 1.77 deg)

The NACA 0012 was also demonstrated at a high subsonic speed and compared to experimental data: Mach 0.502 at 1.77 degrees angle-of-attack. The mesh is shown in Figure 7.4. Results are shown in Figure 7.5 and Figure 7.6. Figure 7.5 shows the pressure and Mach distributions predicted by CFDsol, and Figure 7.6 compares the surface pressure from CFDsol and Euler2D to experimental data from Barche (1979). The pressure on the lower surfaces matches very well, but the pressure on the top surface differs near the leading edge. CFDsol and Euler3D predict the same distribution, so the loss in suction must be due to a lack of refinement near the leading edge.

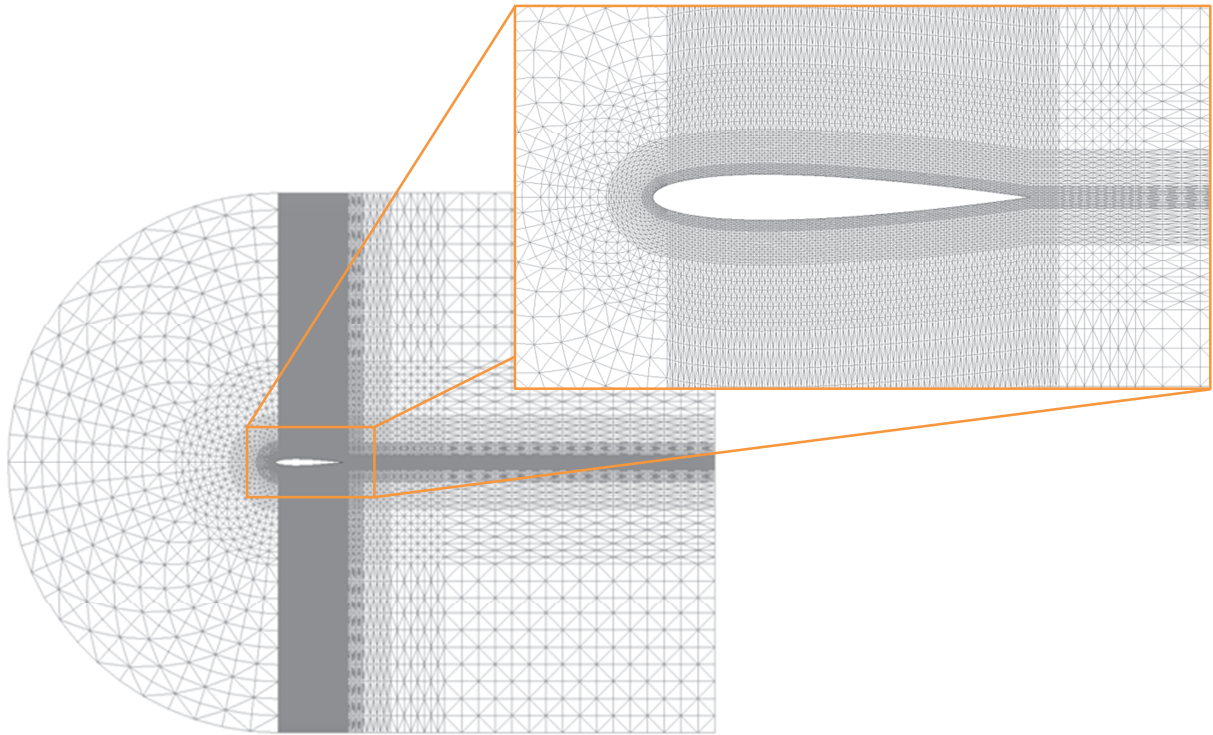


Figure 7.4: Mesh for NACA 0012 Airfoil (Mach 0.502, 1.77° AOA).

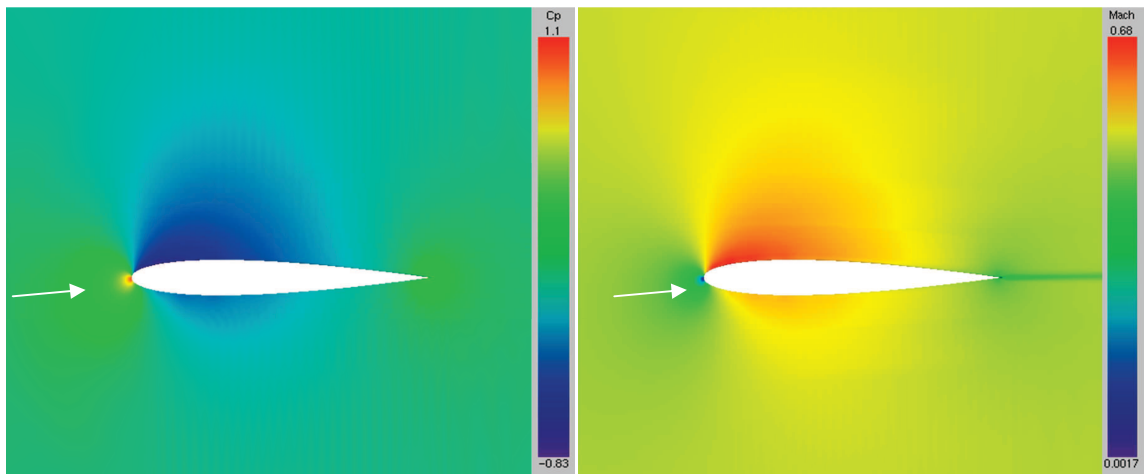


Figure 7.5: Pressure and Mach Distributions around NACA 0012 Airfoil (Mach 0.502).

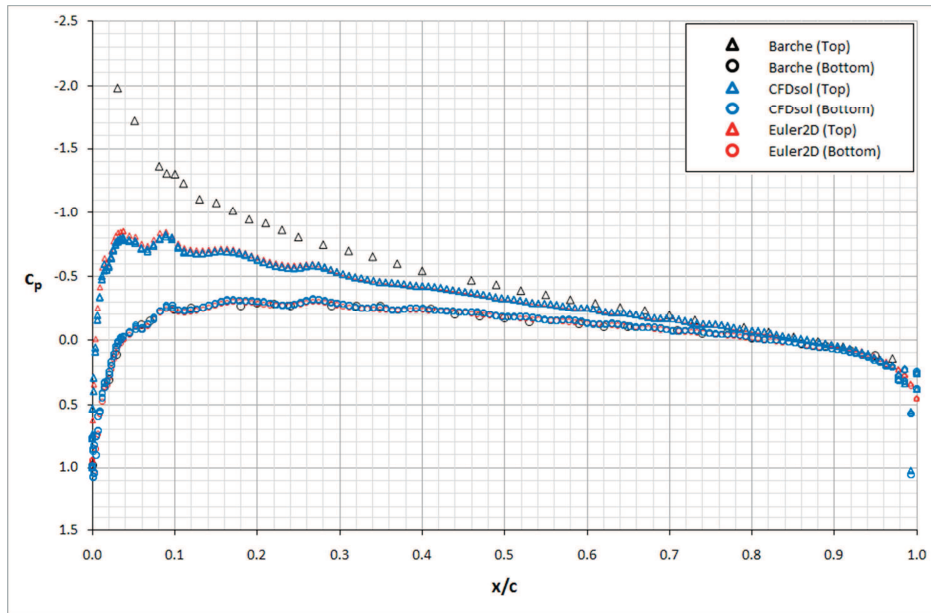


Figure 7.6: Pressure Distribution over NACA 0012 Airfoil at Mach 0.502, 1.77° AOA.

7.1.1.3 RAE 2822 Airfoil (Mach 0.6, 2.57 deg)

An RAE 2822 airfoil was also demonstrated at a high subsonic condition (Mach 0.6, 2.57° AOA) and compared to experimental data. The mesh for this condition is shown in Figure 7.7. The pressure and Mach distributions predicted by CFDsol are shown in Figure 7.8 and Figure 7.9. The surface pressure predicted by CFDsol is compared to that from Euler2D and experimental data from Barche (1979) in Figure 7.10. The largest differences are seen near the trailing edge on the lower surface. The pressure from CFDsol matches that from Euler2D, even near the trailing edge.

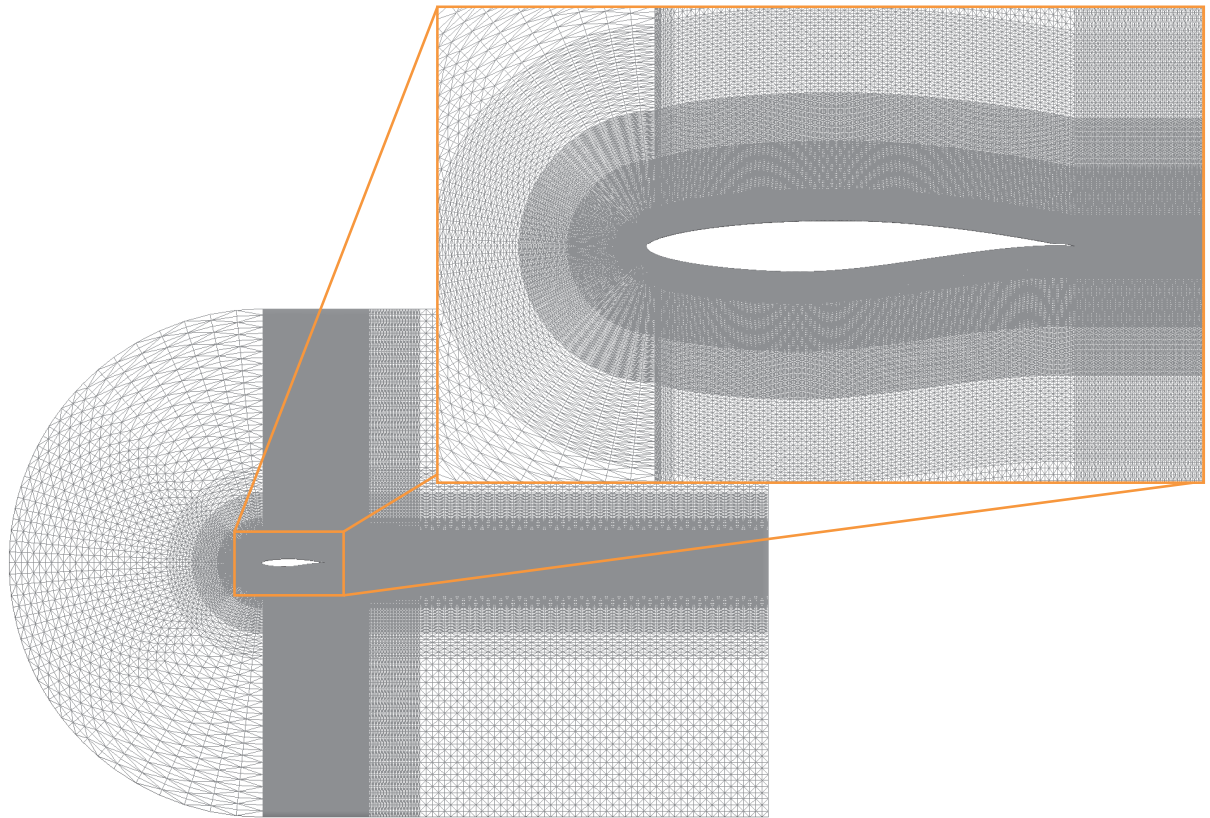


Figure 7.7: Mesh for RAE 2822 Airfoil (Mach 0.6, 2.57° AOA).

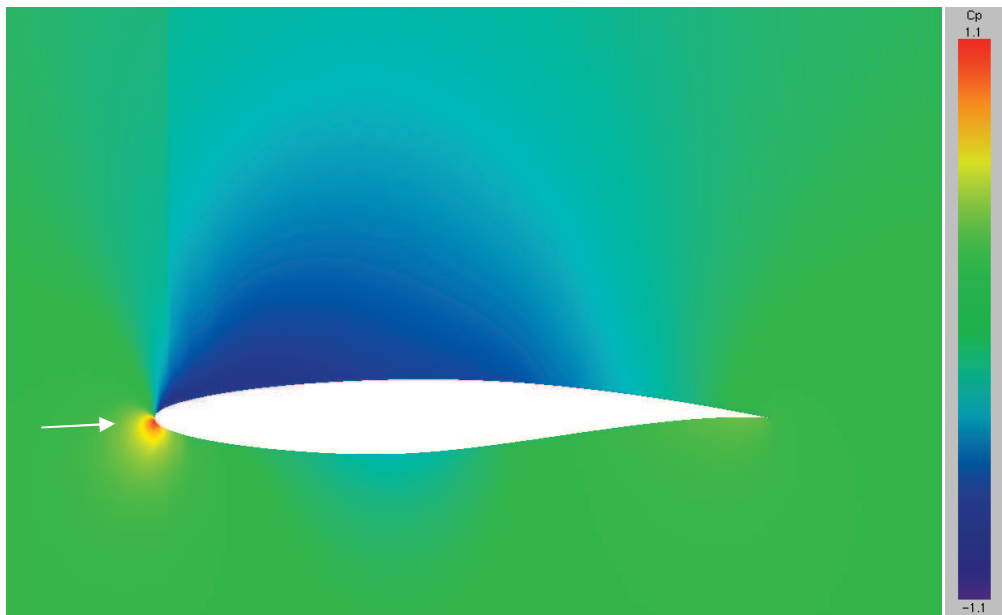


Figure 7.8: Pressure Distribution around RAE 2822 Airfoil (Mach 0.6).



Figure 7.9: Mach Distribution around RAE 2822 Airfoil (Mach 0.6).

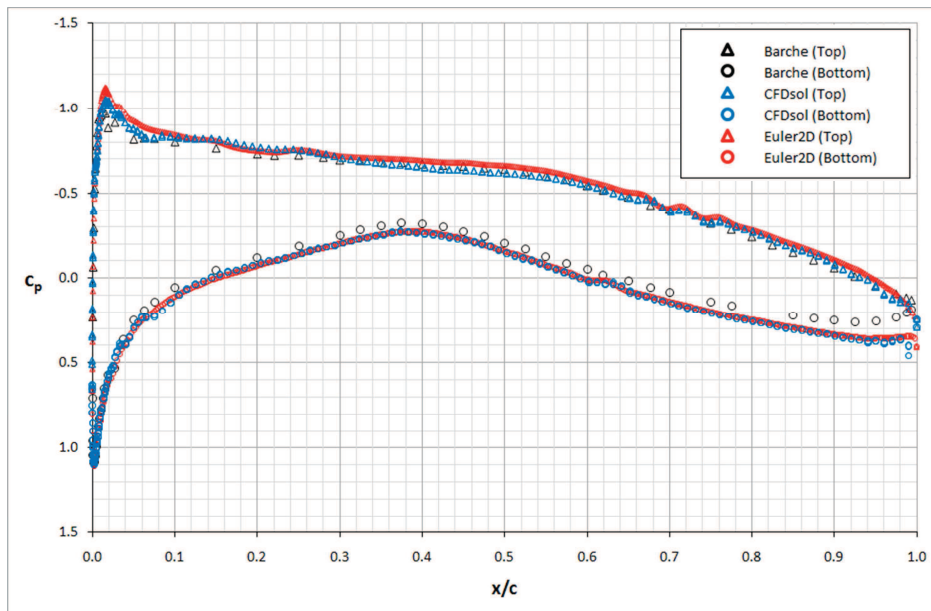


Figure 7.10: Pressure Distribution over RAE 2822 Airfoil (Mach 0.6, 2.57° AOA).

7.1.1.4 Ellipse

An ellipse with 6:1 ratio was tested under inviscid conditions, in preparation for a viscous boundary layer. The inviscid pressure and velocity distributions are shown in Figure 7.11 for the mesh shown in Figure 7.12. The mesh was generated using three mesh sources so that the mesh was finer at the leading and trailing edges. The spacing was chosen to resemble the radius of curvature of the ellipse surface. (A viscous mesh was added to the near-wall region in preparation for viscous testing.) The inviscid surface pressure distribution for the ellipse is shown in Figure 7.13.

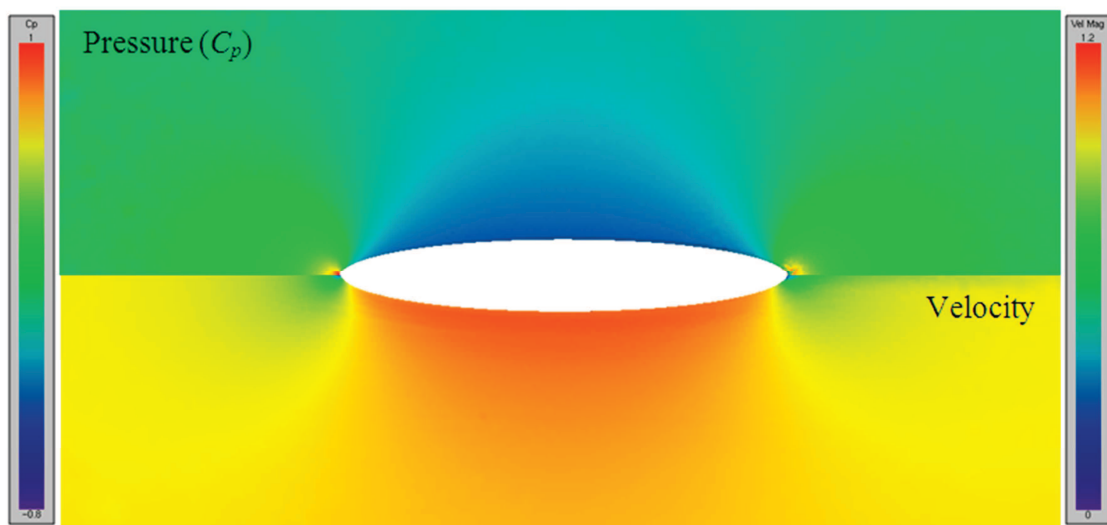


Figure 7.11: Pressure (top) and Velocity (bottom) around Inviscid Ellipse (Mach 0.3).

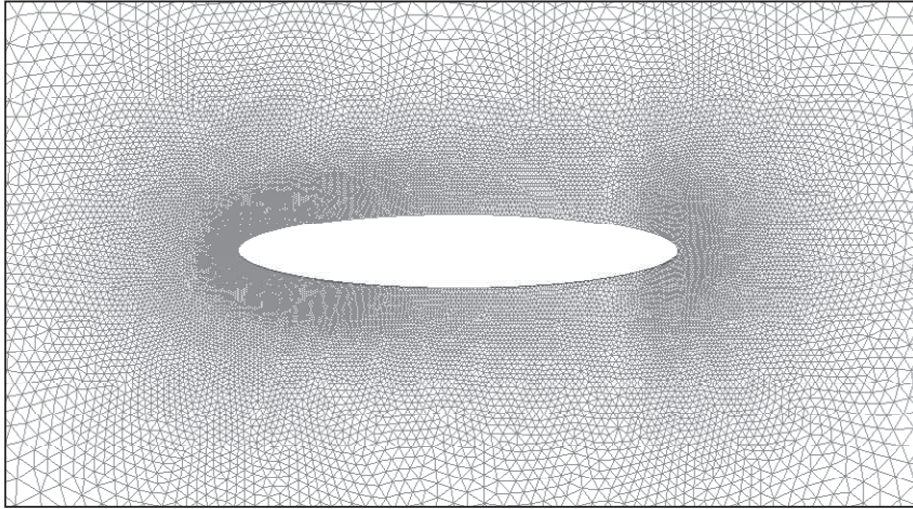


Figure 7.12: Ellipse Mesh.

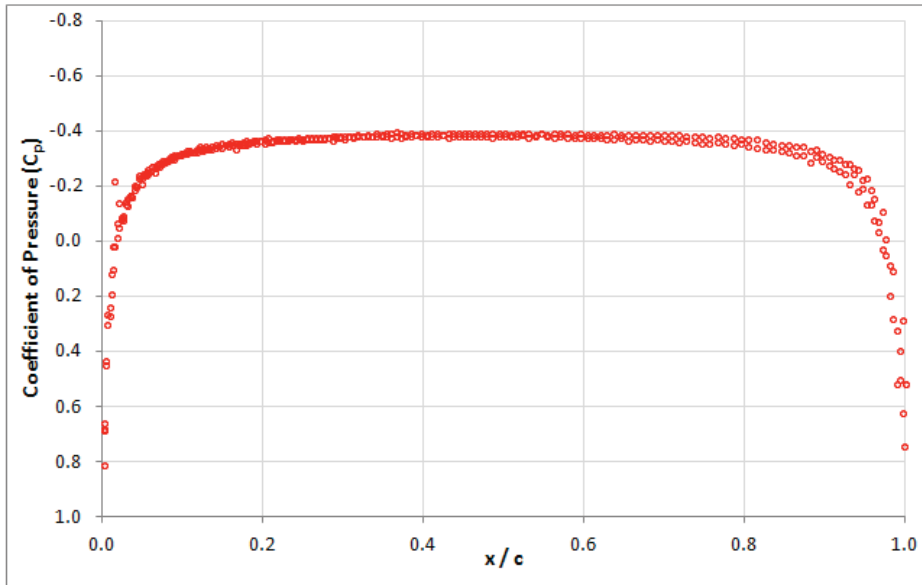


Figure 7.13: Pressure over Inviscid Ellipses (Mach 0.3).

7.1.1.5 Cylinder

A circular cylinder was tested under inviscid conditions to demonstrate a simple, symmetric two-dimensional body. Only half of the cylinder was modeled to minimize the run time.

The mesh was generated so that the spacing at the cylinder surface is 6% of the radius of the cylinder, doubling at 3.75 radii from the center of cylinder. The pressure and velocity distributions around the cylinder are shown in Figure 7.15 for the mesh shown in Figure 7.14. The surface pressure distribution is shown in Figure 7.16 compared with potential theory corrected for compressibility. The pressure distribution is asymmetric across the centerline because artificial dissipation triggers the flow to separate near the aft stagnation point. Otherwise the distribution matches theory very well.

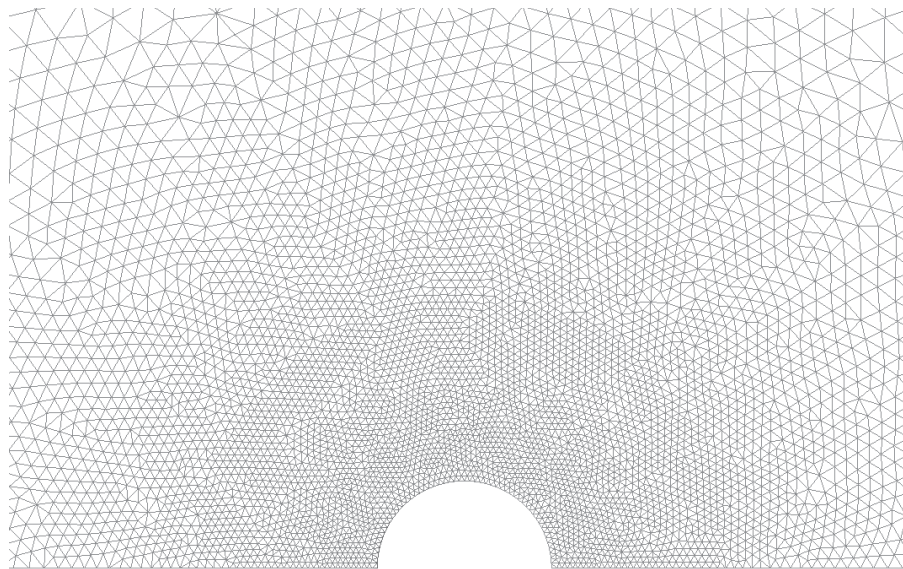


Figure 7.14: Inviscid Cylinder Mesh.

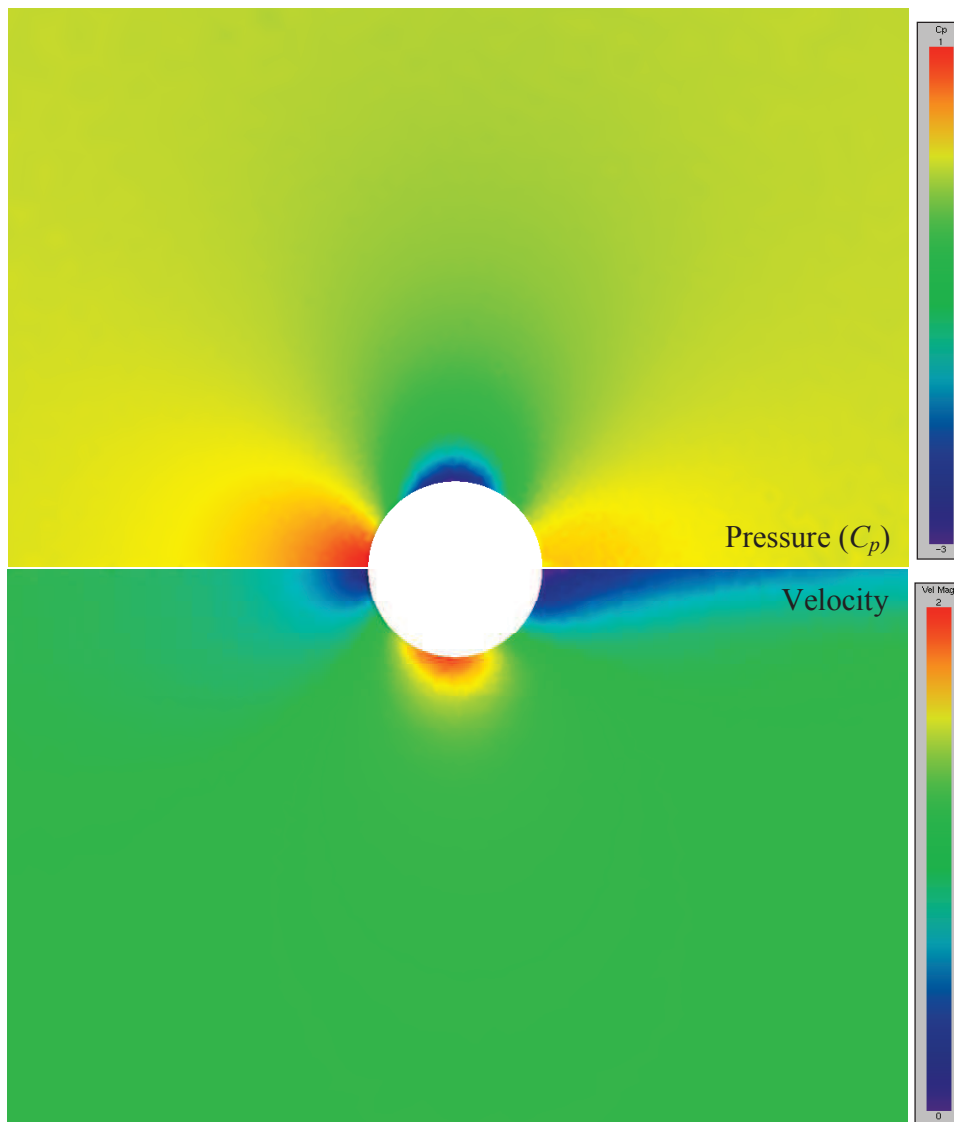


Figure 7.15: Pressure (top) and Velocity (bottom) Distrib. around Inviscid Cylinder.

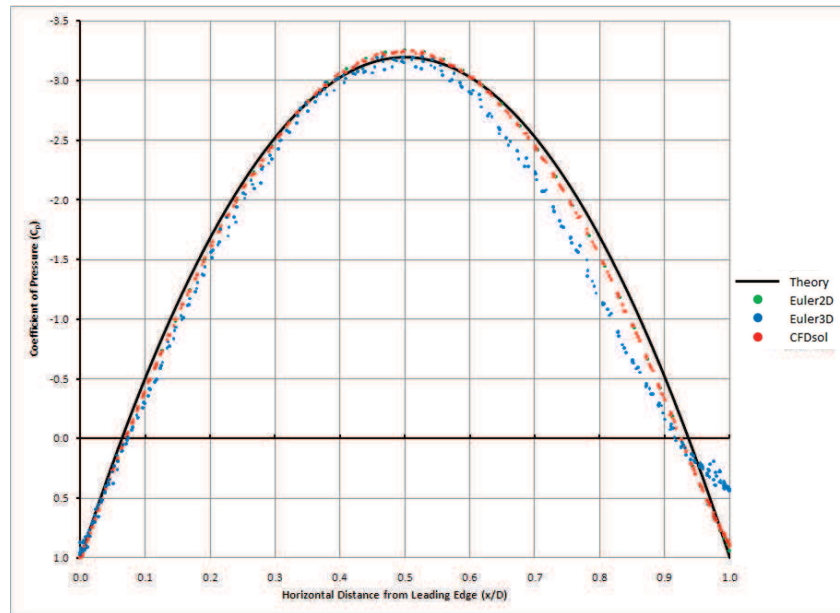


Figure 7.16: Pressure Distribution over Cylinder.

7.1.1.6 Sphere

A sphere was tested under inviscid conditions to demonstrate a simple, axisymmetric (three-dimensional) body. One quarter of the sphere was modeled for inviscid testing. The mesh was generated using a similar method to the inviscid cylinder. The surface mesh is 8.4% of the radius, and the spacing doubles at five times the radius. The pressure and velocity distributions along a symmetry plane are shown in Figure 7.17 for the mesh shown in Figure 7.18. The surface pressure distribution is shown in Figure 7.19 compared with potential theory. The pressure distribution is asymmetric across the centerline because artificial dissipation triggers the flow to separate near the aft stagnation point. Otherwise the distribution matches theory reasonably well.

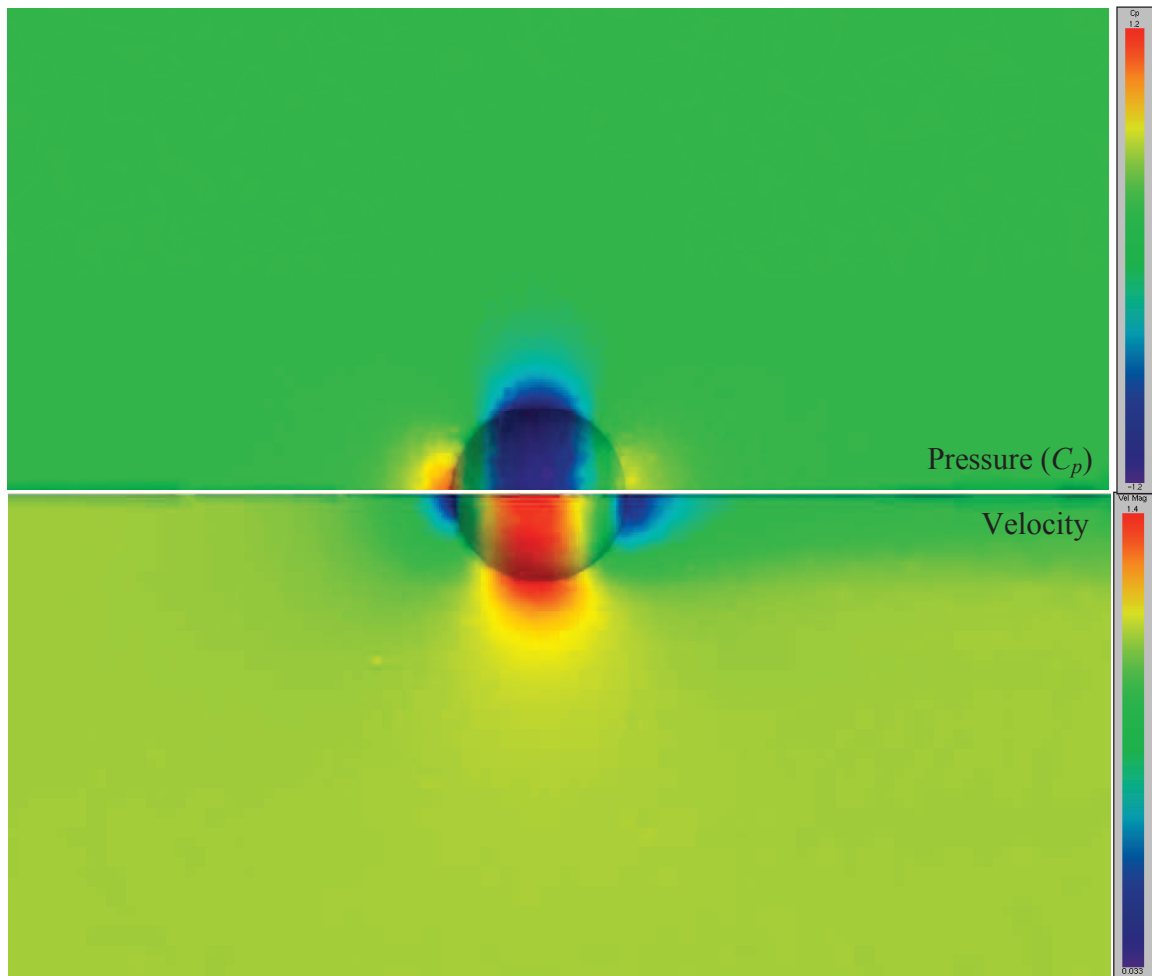


Figure 7.17: Pressure (top) and Velocity (bottom) Distributions along Symmetry Plane around Inviscid Sphere (Mach 0.3).

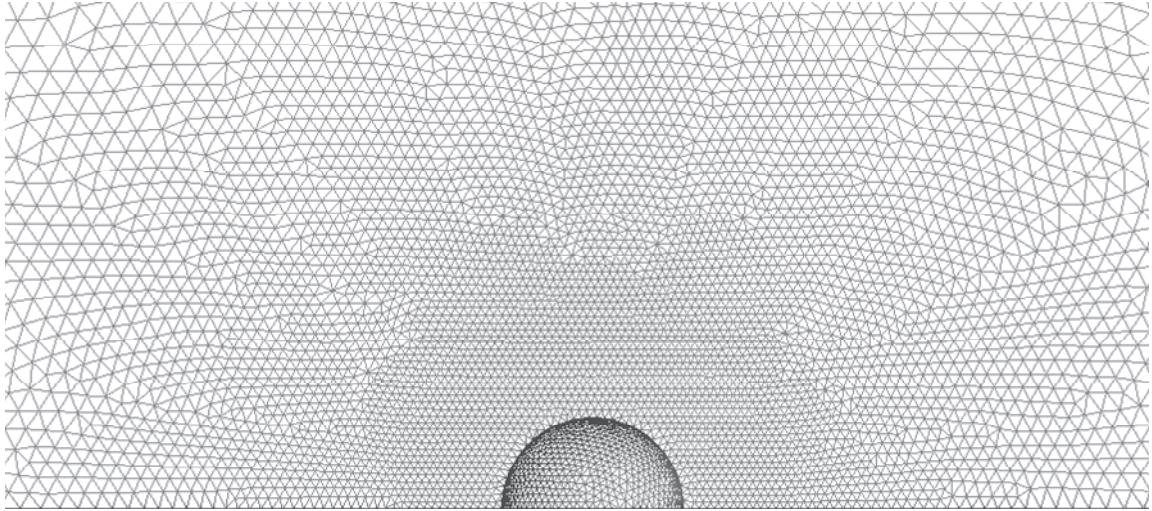


Figure 7.18: Inviscid Sphere Mesh.

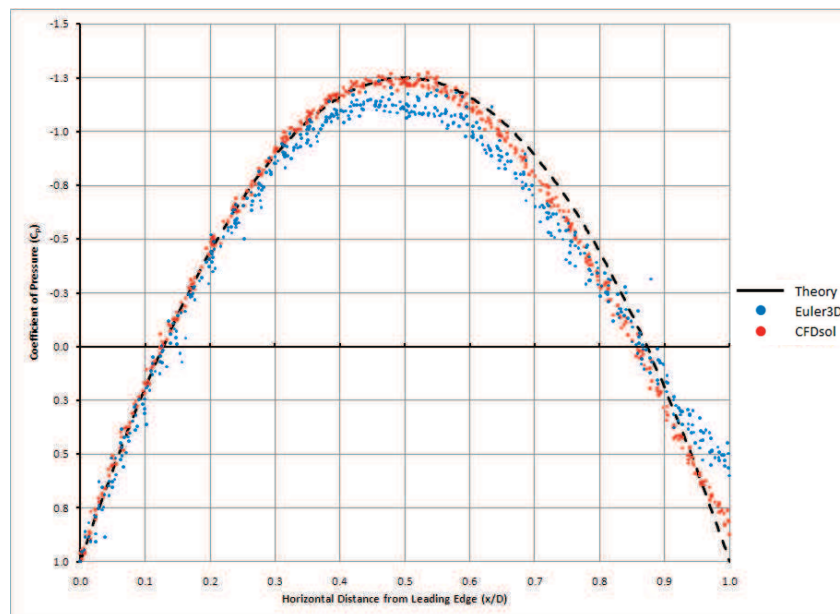


Figure 7.19: Pressure Distribution over Sphere.

7.1.2 Transonic

The complexity of flow solutions is greatly increased by demonstrating transonic flow fields.

Four air-foils are investigated at different speeds and orientations. The CAST 7 airfoil is

demonstrated at Mach 0.765 and 0.785, which create two distinct transonic conditions. The NACA 0012 airfoil is revisited at a low angle-of-attack and Mach 0.835, generating shocks on both upper and lower surfaces. The NASA 10% thick airfoil is tested at Mach 0.79, and the RAE 2822 airfoil is revisited at Mach 0.73. Each transonic case is compared to an experimental data set. Analytical solutions are not available for these transonic cases, so each case is also compared to Euler2D. When discrepancies arise between the CFD solution and the experimental data, the solutions from Euler2D can be used to show that the discrepancies are due to mesh spacing, model completeness, or accuracy of the geometry.

7.1.2.1 NACA 0012 Airfoil (Mach 835, -0.13 deg)

An additional case was demonstrated for the NACA 0012. Previously, the NACA 0012 was tested at a low subsonic Mach. This case was selected to demonstrate the capabilities of the solver and compare to experimental data: Mach 0.835 at -0.13 degrees angle-of-attack. The resulting flow shows two transonic shocks, differing in location and strength. The mesh shown in Figure 7.20 was used to model the transonic case.

This case is the only test that has a shock on both the top and bottom surfaces at the same time. The airfoil is symmetric, and at low angles of attack, the potential for different shocks exists. The airfoil is held at a slight negative incidence (creating a down force) so we expect the bottom surface to shock later. This shock pattern is shown in the experimental data from Barche (1979). Figure 7.21 shows two very distinct supersonic regions, with staggered shocks. The shock on the lower surface is aft of that on the upper surface, as expected. CFDsol predicts the shock further aft than that shown in the experimental data (Figure 7.22). The locations only differ by about 8% of the chord length, which could easily change with

the addition of a boundary layer. The CFDsol profile is very similar to that generated with Euler2D.

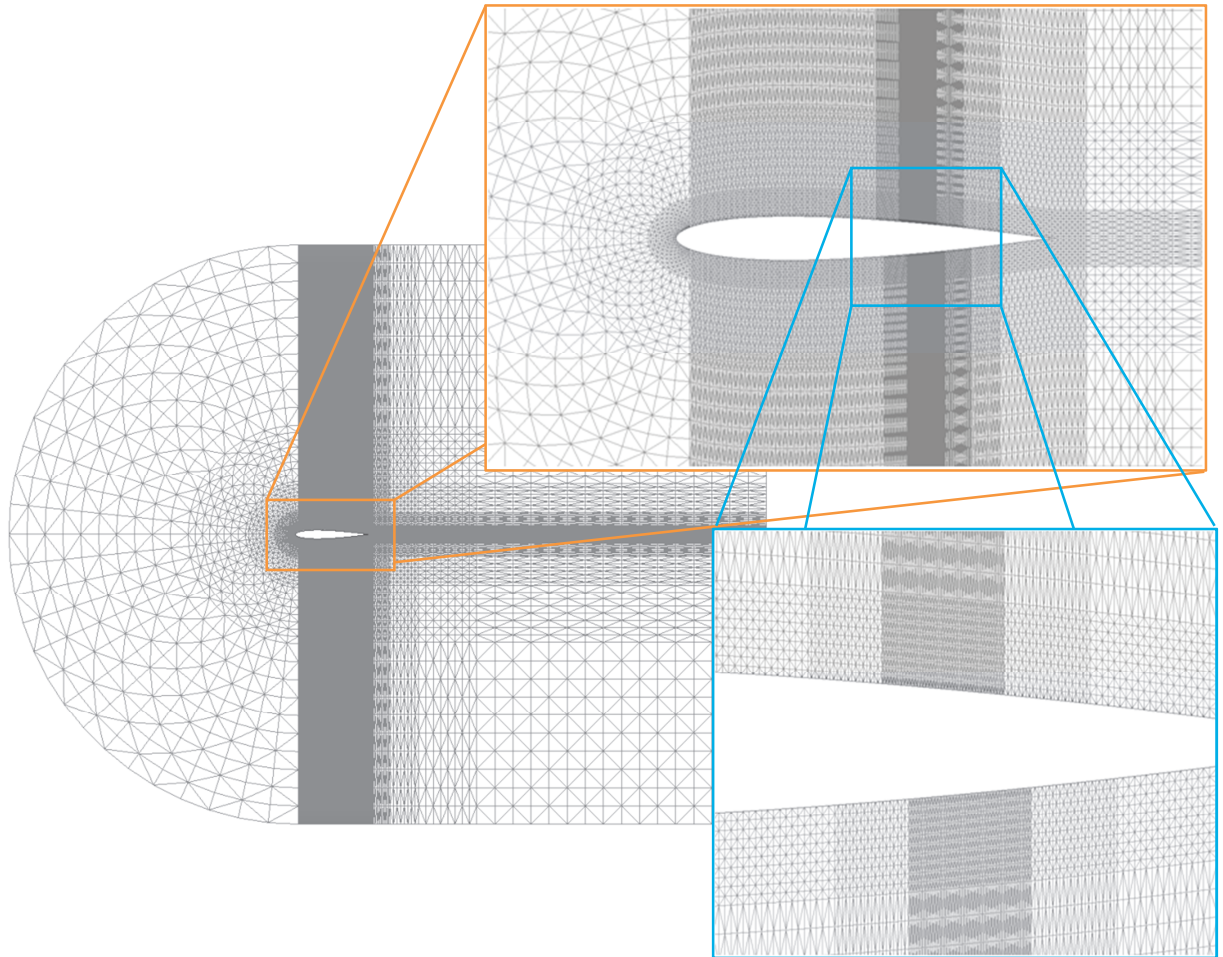


Figure 7.20: Mesh for NACA 0012 Airfoil (Mach 0.835, -0.13° AOA).

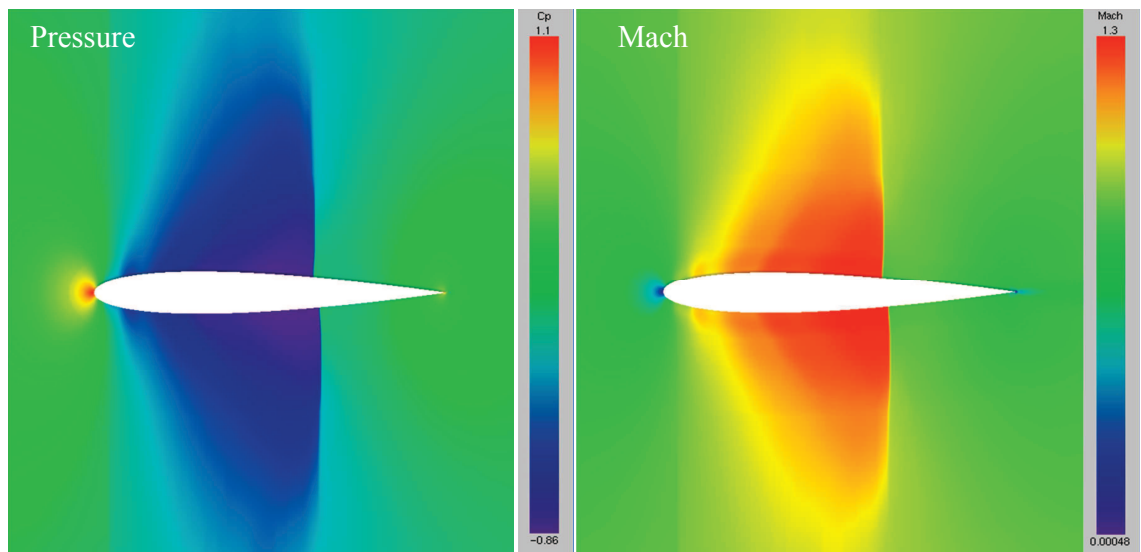


Figure 7.21: Pressure and Mach Distrib. around NACA 0012 Airfoil (Mach 0.835).

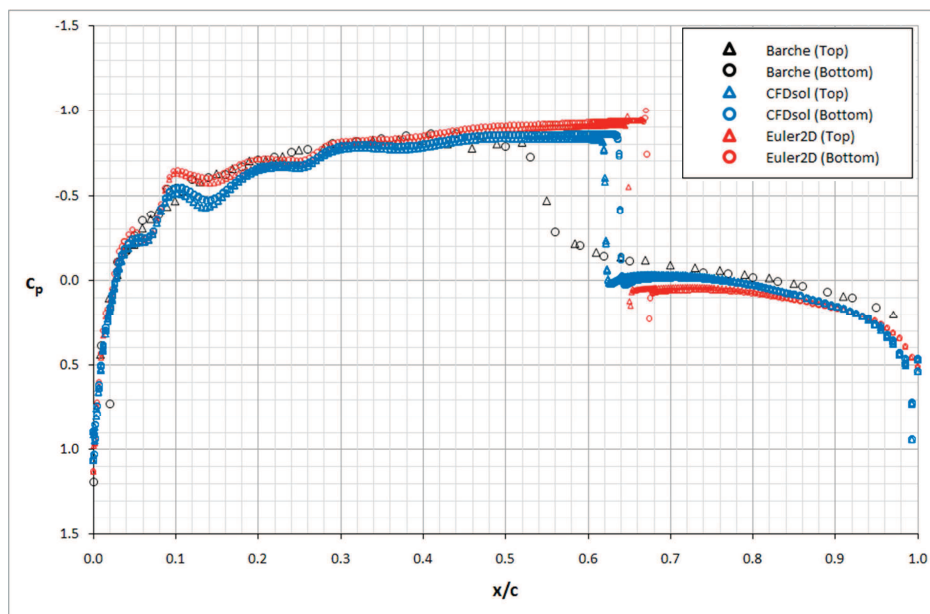


Figure 7.22: Pressure Distribution over NACA 0012 Airfoil (Mach 0.835, -0.13° AOA).

7.1.2.2 RAE 2822 Airfoil (Mach 0.73, 2.8 deg)

The next transonic demonstration case matches its experimental data the best of those shown in this section. The RAE 2822 airfoil was revisited at a transonic condition (Mach 0.73, 2.8° AOA), using the mesh shown in Figure 7.24. The predicted pressure and Mach distributions from CFDsol are shown in Figure 7.23. The surface pressure predicted by CFDsol is compared to that from Euler2D and experimental data from AGARD (1988) in Figure 7.25. The experimental data seems to have an elongated shock region, which can only be explained by viscous interactions. The inviscid analysis of both CFDsol and Euler2D, which track very closely together, predict a very sharp shock region. Both CFD solvers under predict the suction near the leading edge, most likely due to mesh refinement near the stagnation point.

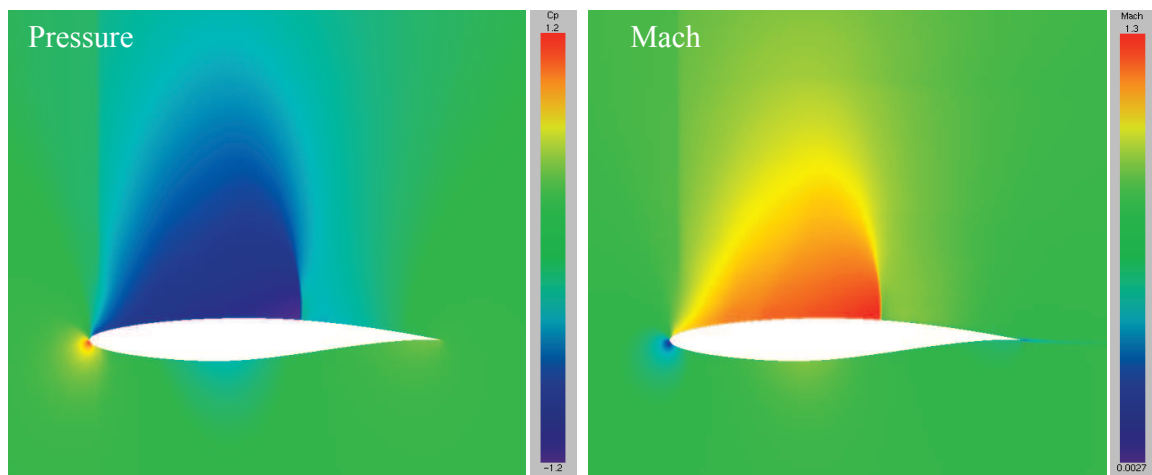


Figure 7.23: Pressure and Mach Distributions around RAE 2822 Airfoil (Mach 0.73).

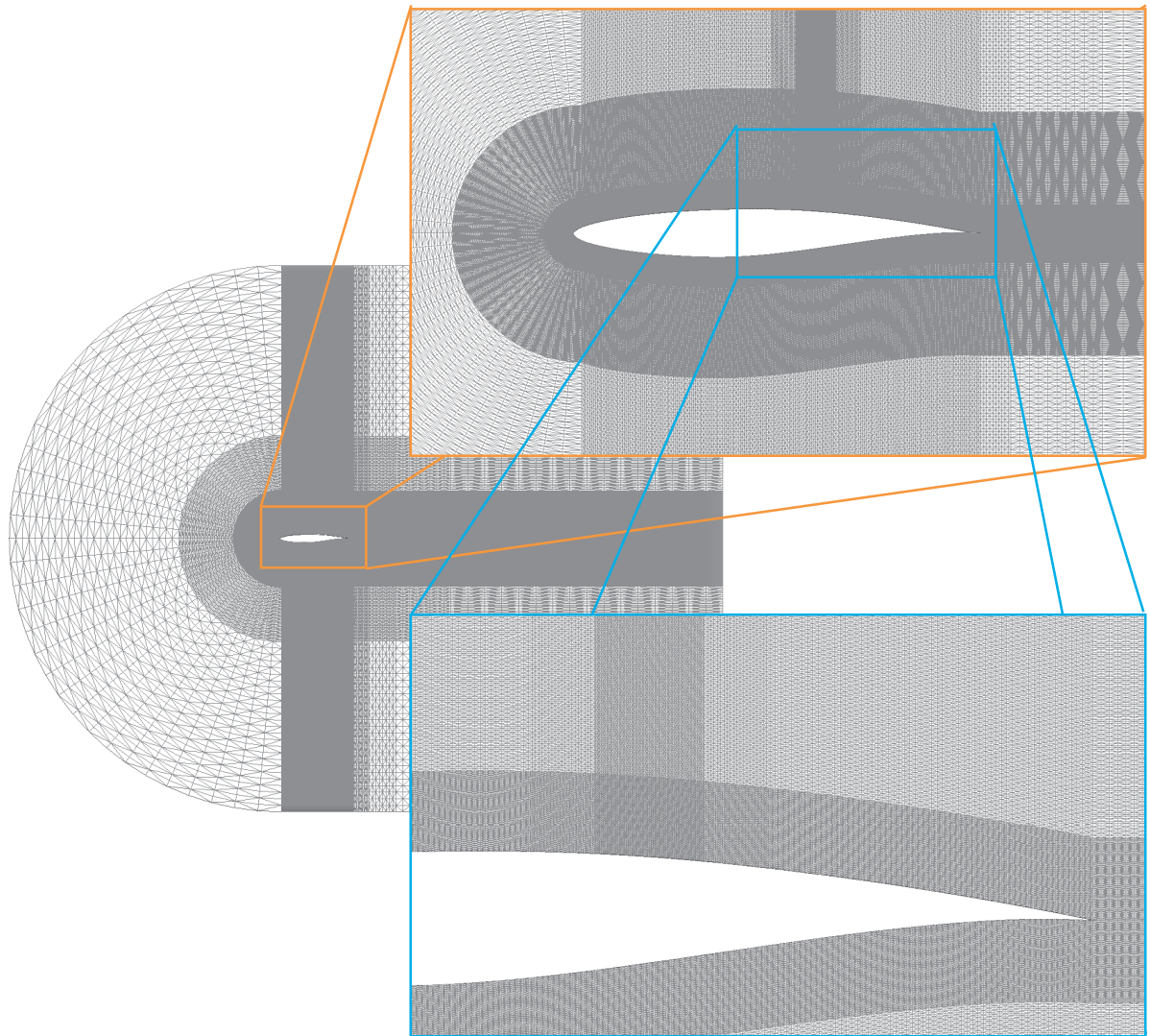


Figure 7.24: Mesh for RAE 2822 Airfoil (Mach 0.73, 2.8° AOA).

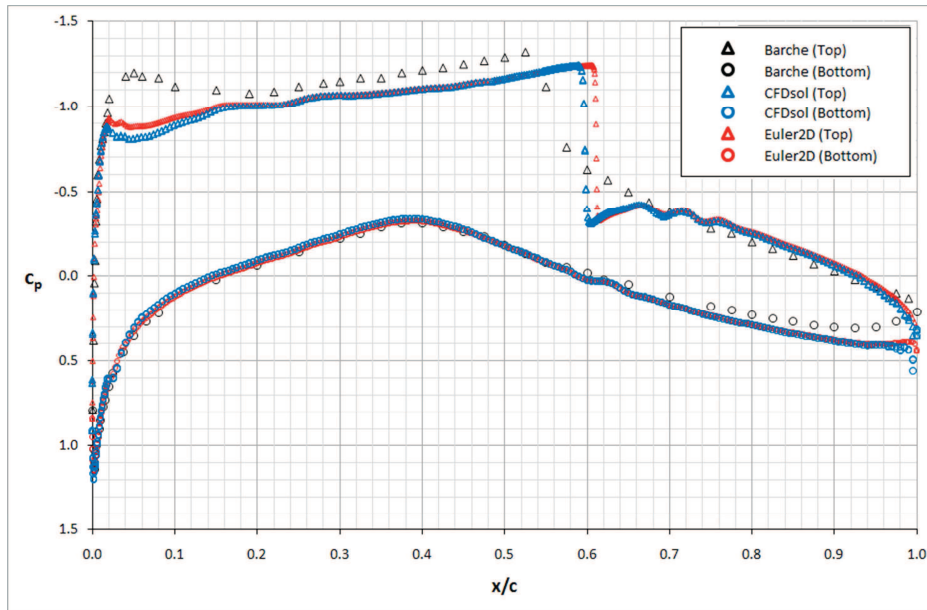


Figure 7.25: Pressure Distribution over RAE 2822 Airfoil (Mach 0.73, 2.8° AOA).

7.1.2.3 CAST 7 Airfoil (Mach 0.765, 2.52 deg)

Another transonic airfoil to be demonstrated is the CAST 7 airfoil. Two different meshes were generated for two specific transonic cases: Mach 0.765 and 0.785, both at 2.52 degrees angle-of-attack. Each case has a transonic shock on the top of the airfoil. Coarse meshes were ran to determine a general location for the shocks (estimated by CFDsol, not using the experimental data). The mesh near the shock was then refined to converge the solution. The final meshes are shown in Figure 7.26 and Figure 7.30. The two meshes contain 1.4 and 1.6 million elements, respectively. Each mesh was mathematically wrapped in a U-shape around the airfoil instead of using the normal Delaunay approach. The wrapping approach allowed the spacing to be specified in each of the three component directions without depending on

the others. The result is a mesh that is exactly two elements wide across the entire domain, limiting any cross-flow influences.

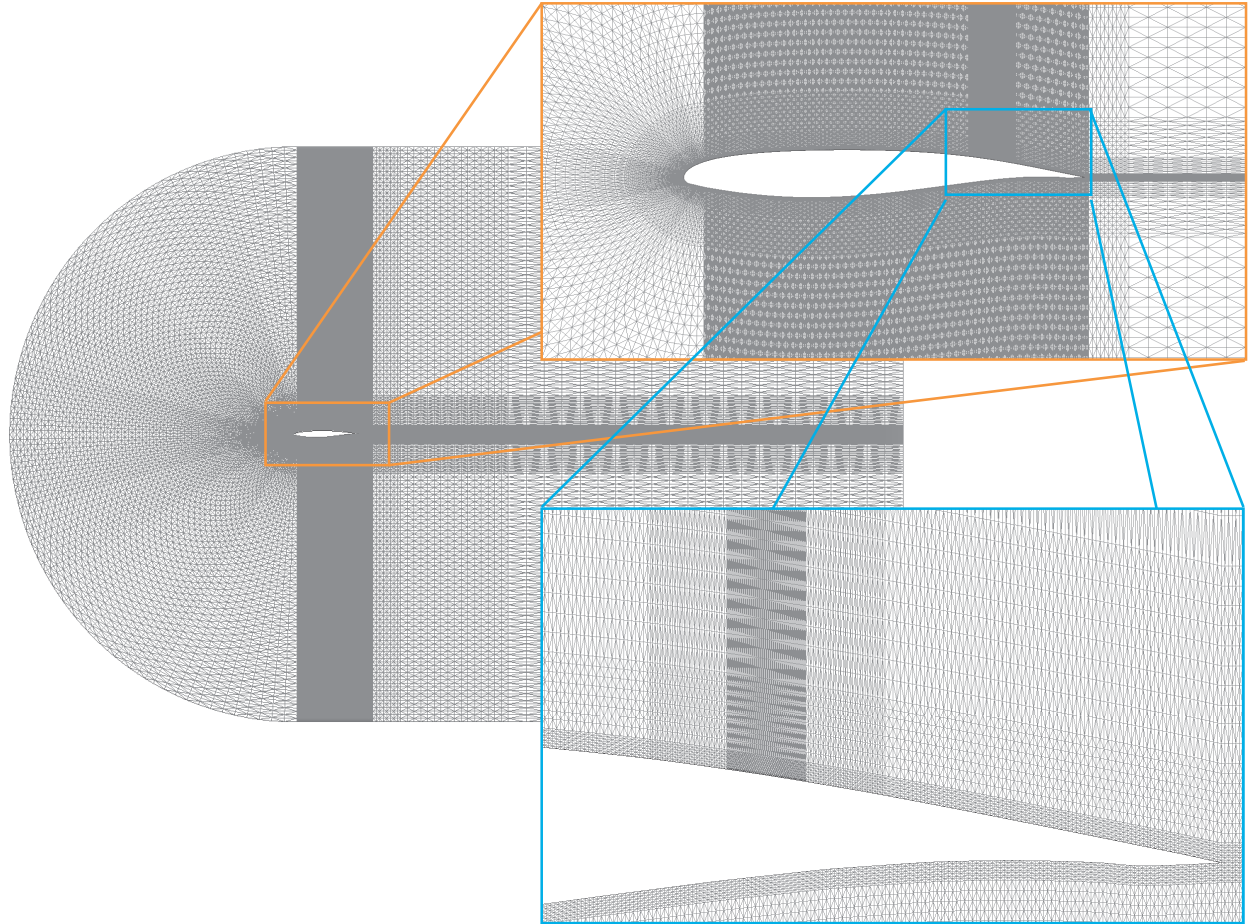


Figure 7.26: Mesh for CAST 7 Airfoil (Mach 0.765, 2.52° AOA).

The results for the lower speed case (Mach 0.765) are shown in Figure 7.27 and Figure 7.28. Figure 7.27 shows the distribution of pressure and local Mach number calculated using CFDsol. Figure 7.28 compares the pressure distribution on the surface of the airfoil calculated using CFDsol with results from Euler2D and experimental data taken from Barche (1979). Both CFD solvers predict a shock further back on the top surface than that seen in

the experimental data, which is most likely due to the inviscid analysis. The addition of a boundary layer would effectively thicken the section, making the flow shock sooner. (Viscous solutions were not demonstrated due to time.) The pressure on the shock-free lower surface follows the experimental data well, as does the pressure near the leading and trailing edges on the top.

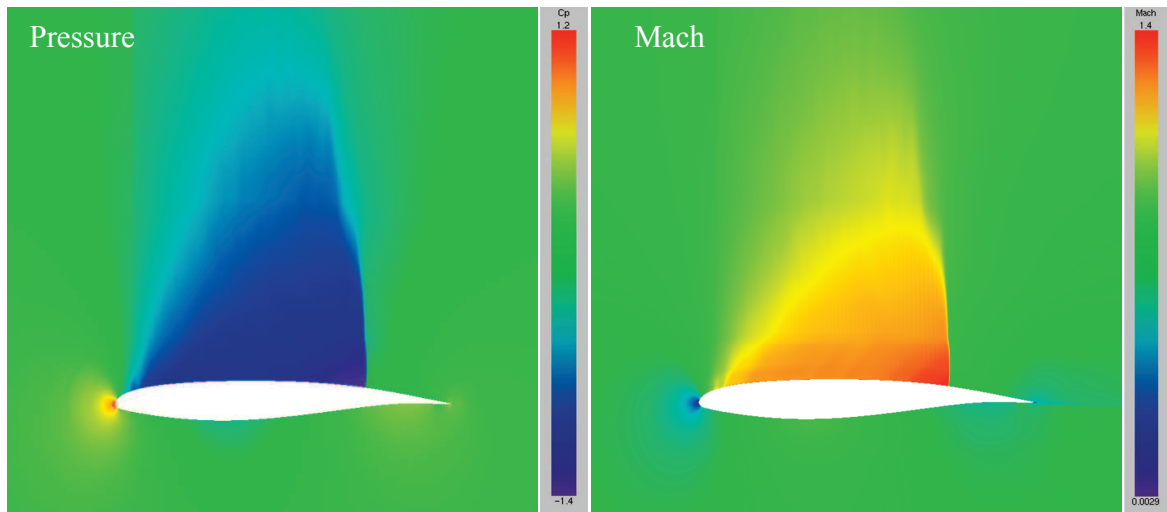


Figure 7.27: Pressure and Mach Distributions around CAST 7 Airfoil (Mach 0.765).

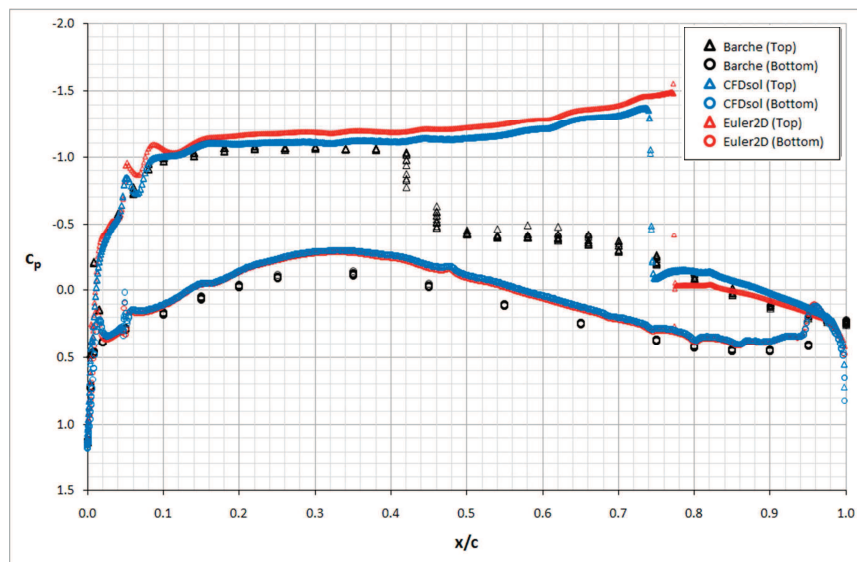


Figure 7.28: Pressure Distribution over CAST 7 Airfoil (Mach 0.765, 2.52° AOA).

7.1.2.4 CAST 7 Airfoil (Mach 0.785, 2.52 deg)

Similar results are shown for the higher speed case (Mach 0.785) in Figure 7.31 and Figure 7.29. Figure 7.31 again shows the pressure and local Mach number distributions predicted by CFDsol, and Figure 7.29 shows a comparison of CFDsol, Euler2D, and data from Barche (1979). The experimental data shows the shock location moves back 10% of the chord, due to the increase freestream Mach number. The two CFD solvers show the shock to move aft 5% of the chord and still predict the transonic shock much later on the top surface. The leading, trailing, and lower surfaces follow the experimental data well.

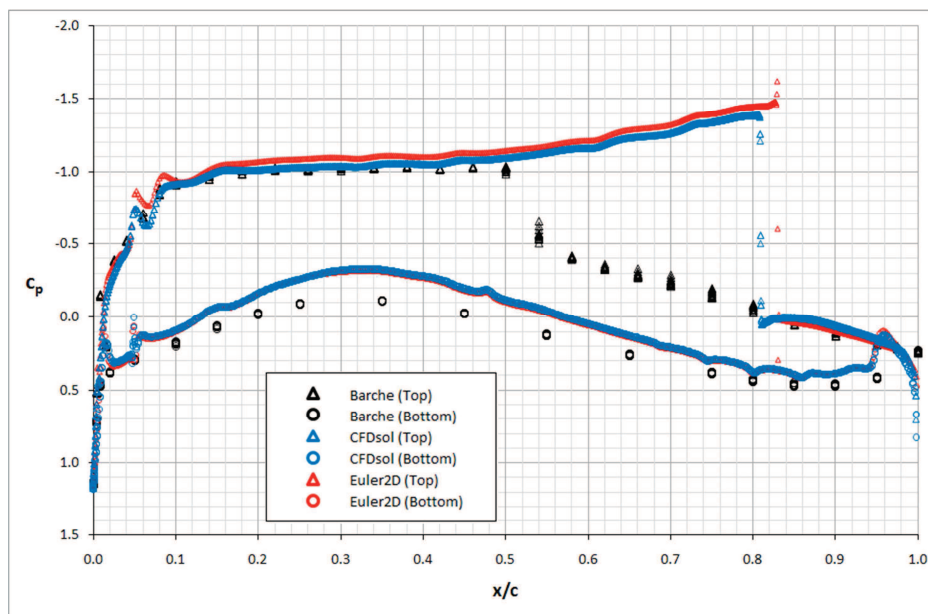


Figure 7.29: Pressure Distribution over CAST 7 Airfoil (Mach 0.785, 2.52° AOA).

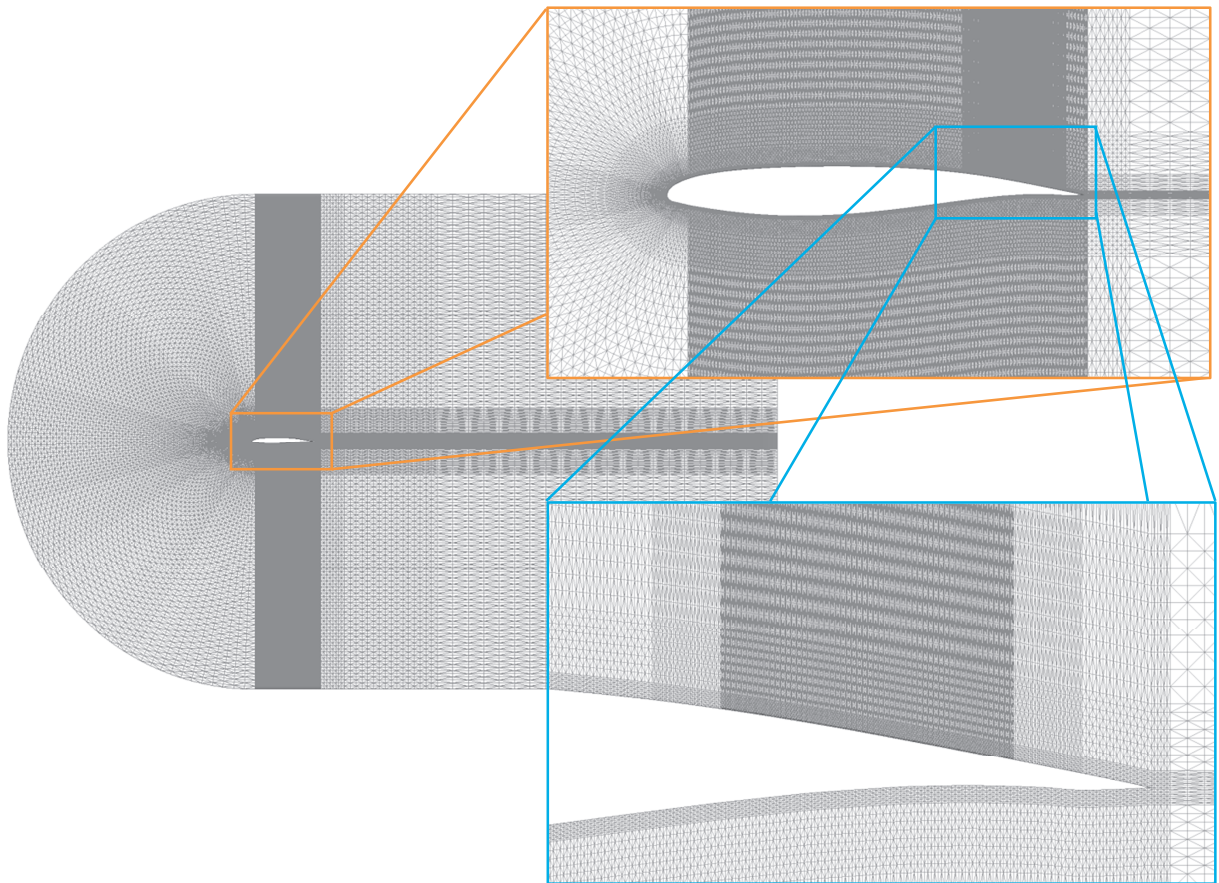


Figure 7.30: Mesh for CAST 7 Airfoil (Mach 0.785, 2.52° AOA).

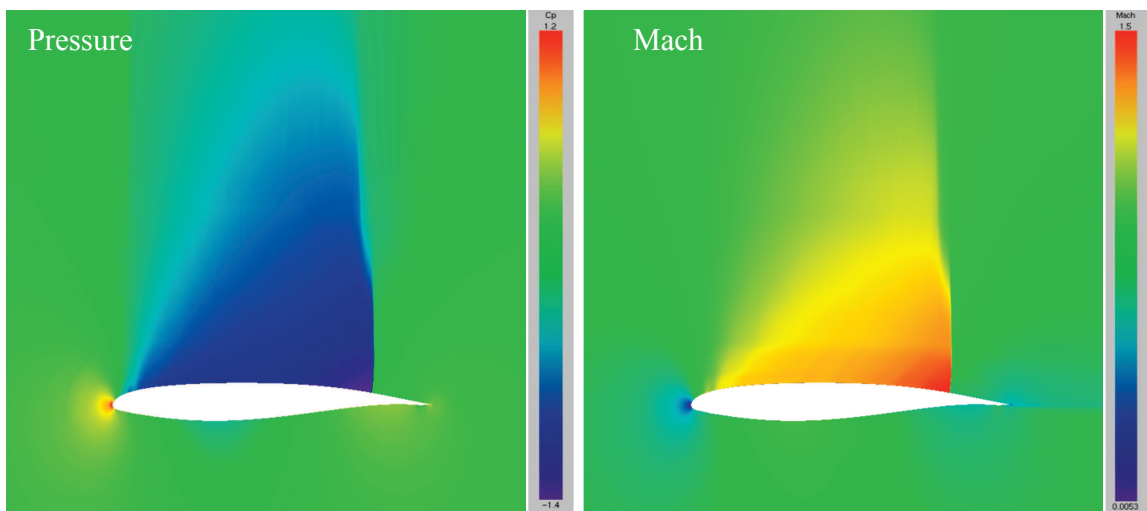


Figure 7.31: Pressure and Mach Distributions around CAST 7 Airfoil (Mach 0.785).

7.1.2.5 NASA 10% Supersonic Airfoil (Mach 0.79, 2 deg)

The next case represents somewhat subtle difference between the two CFD solvers. The mesh used to analyze flow around the NASA 10% supercritical airfoil is shown in Figure 7.32. Both CFDsol and Euler2D predict a strong transonic shock near the trailing edge of the airfoil, so this region is strongly refined. The transonic bubble forms from waves originating from the leading and trailing edges simultaneously.

In Euler2D, once the two waves have passed over the length of the chord, the transonic bubble is formed and nearly converged. In CFDsol, these waves pass over the length of the chord and the bubble is partially formed. Another set of waves then form and travel across the airfoil, refining the bubble. If the solution is allowed to continue, an intermediate shock forms near the mid-chord (shown in Figure 7.33 and Figure 7.34). This shock is very weak, but changes the pressure over the entire top surface. The pressure distributions generated by the two solvers are shown in Figure 7.33 and Figure 7.35 for comparison, and the surface pressure is plotted in Figure 7.37 with experimental data from Barche (1979). Pressure over the lower surface matches very well. Again the CFD results from both CFDsol and Euler2D predict a more aft shock location and a much stronger transonic shock.

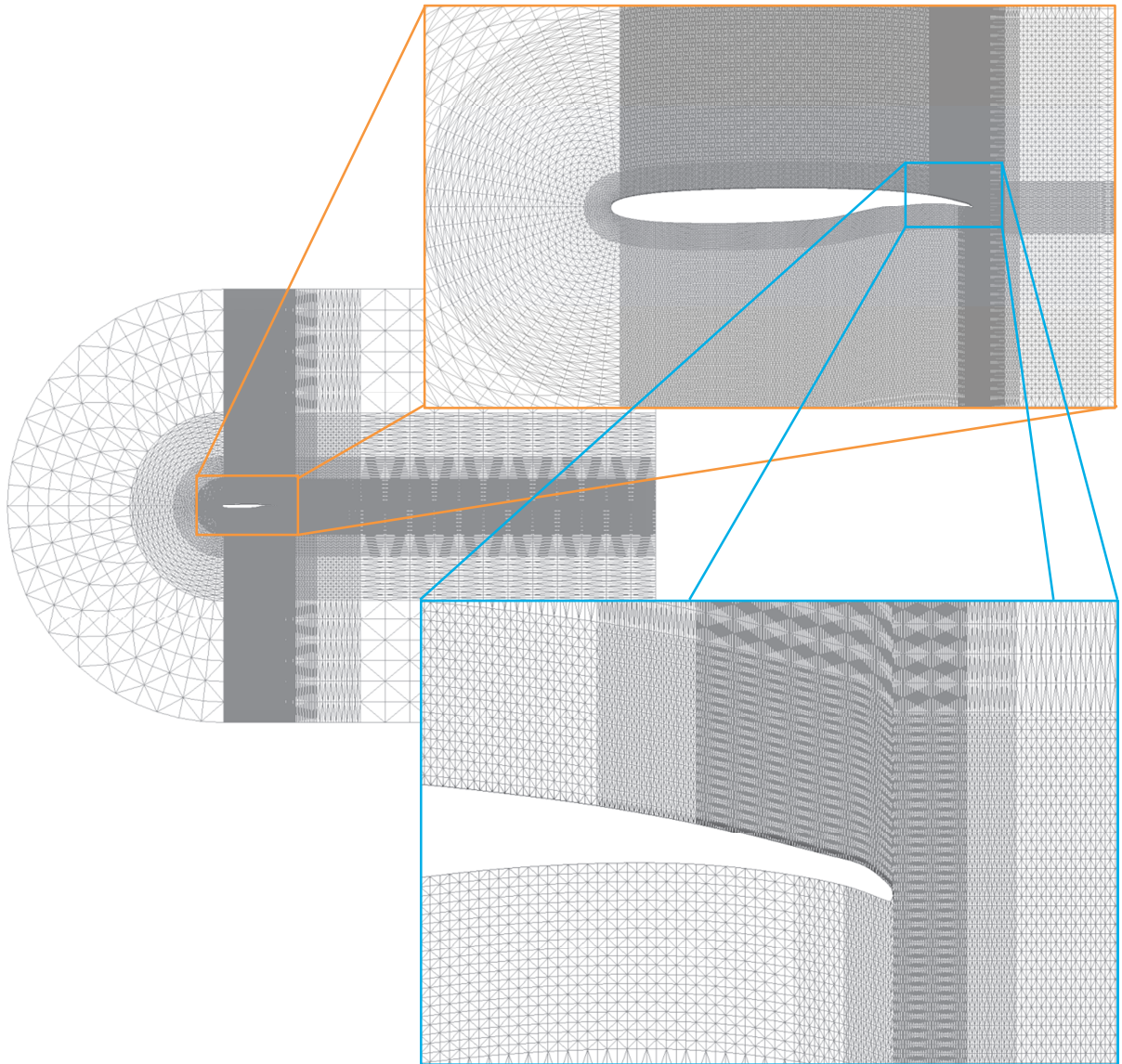


Figure 7.32: Mesh for NASA 10% Thick Supercritical Airfoil (Mach 0.79, 2° AOA).

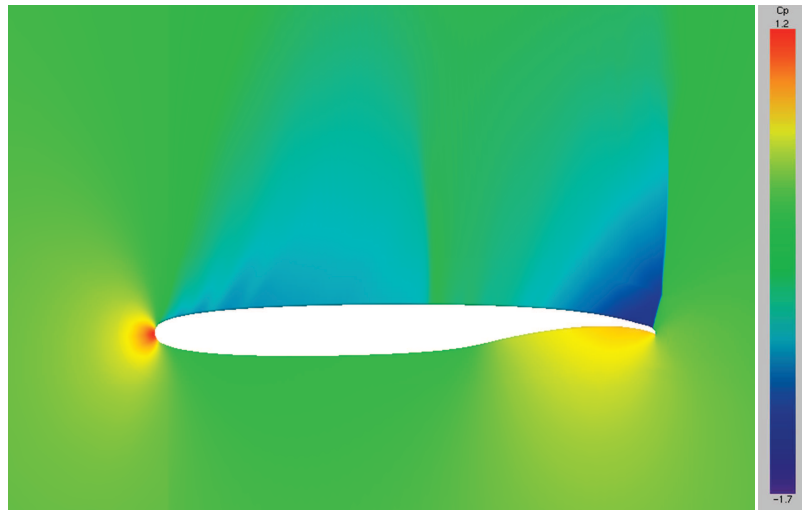


Figure 7.33: Pressure Distribution around NASA 10% Supercritical Airfoil (CFDsol).



Figure 7.34: Mach Distribution around NASA 10% Supercritical Airfoil (CFDsol).

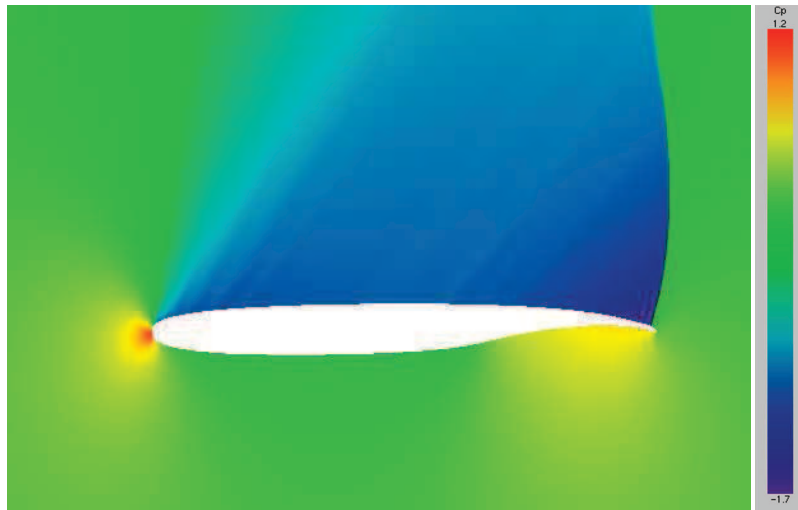


Figure 7.35: Pressure Distribution around NASA 10% Supercritical Airfoil (Euler2D).

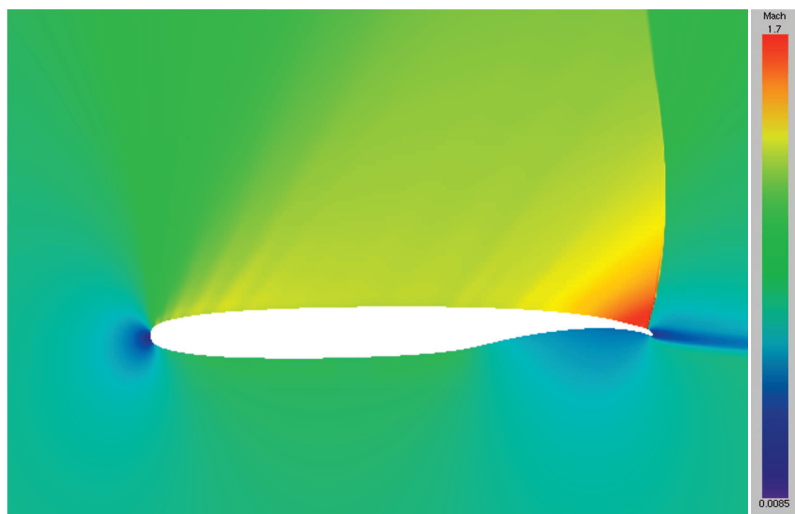


Figure 7.36: Mach Distribution around NASA 10% Supercritical Airfoil (Euler2D).

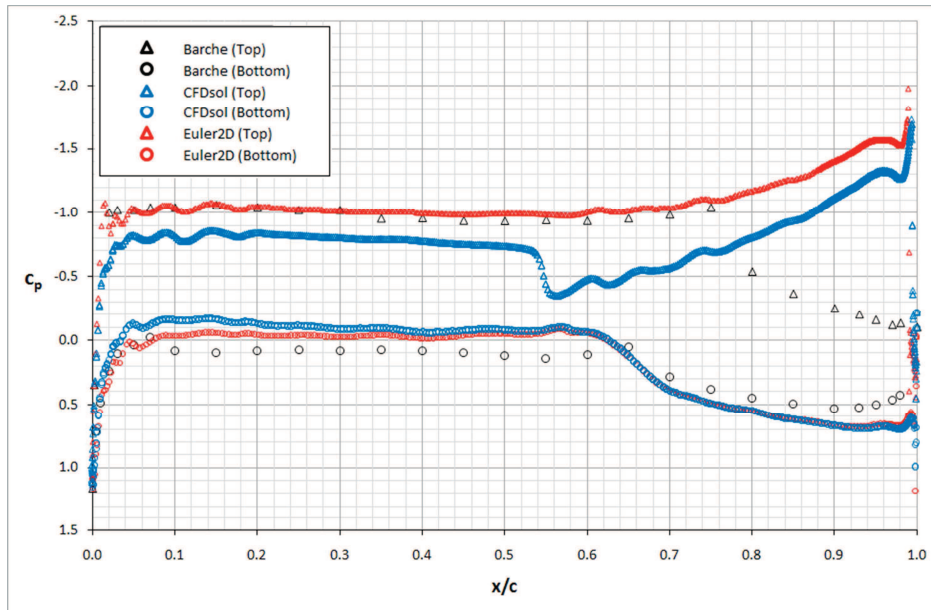


Figure 7.37: Pressure Distribution over NASA 10% Supercritical Airfoil (Mach 0.79).

7.1.3 Supersonic

A double-wedge airfoil was tested to create four distinct shocks and two expansion fans. The airfoil is symmetric both along and across its chord line with a 5-degree half-wedge angle.

At a 2-degree angle-of-attack, the leading edge creates a 3- and 7-degree angle with the free-stream flow, creating different flow fields over the top and bottom surfaces, shown in Figure 7.39. The flow is divided into four regions of concern, shown in Figure 7.41. The solution at Mach 2 was compared with shock-expansion (SE) theory, a reliable analytical method for predicting Mach number and pressure along the airfoil surface and the angle of shock waves (John, 1984). The mesh shown in Figure 7.38 was used to generate a solution in CFDsol shown in Figure 7.40. The CFDsol solution is compared with SE theory in Table 7.1.

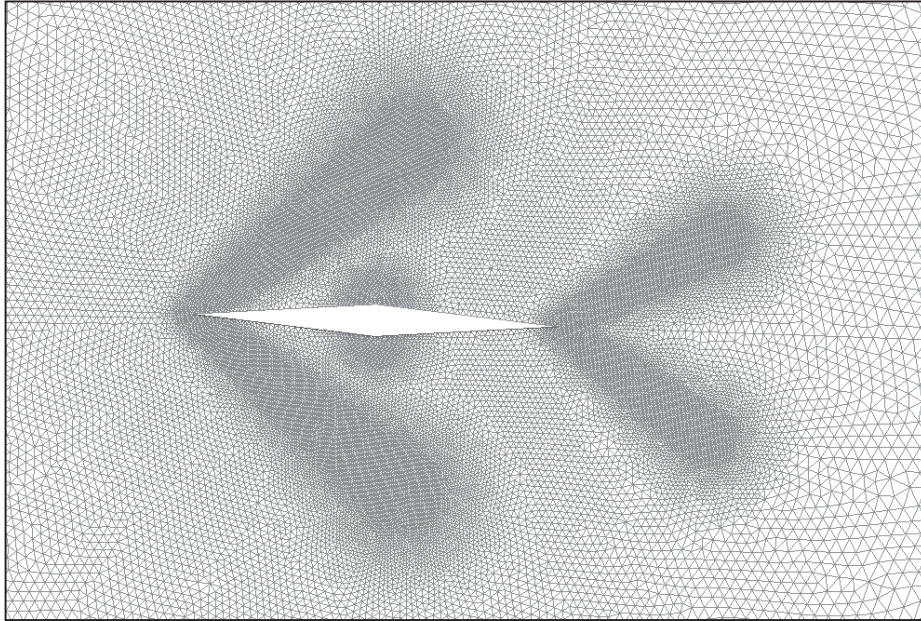


Figure 7.38: Double-Wedge Airfoil Mesh (Mach 2, 2-degrees AOA).

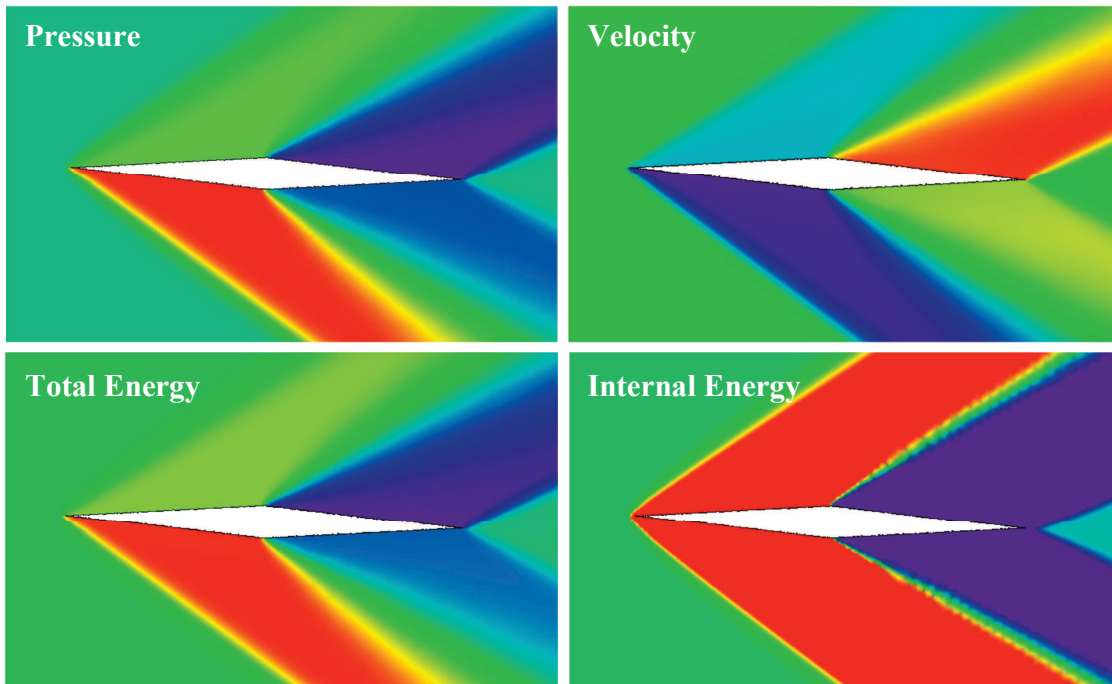


Figure 7.39: Flow around a Supersonic Double-Wedge Airfoil at Mach 2. (Left to Right, Top to Bottom: Pressure, Velocity, Total Energy, Internal Energy.)

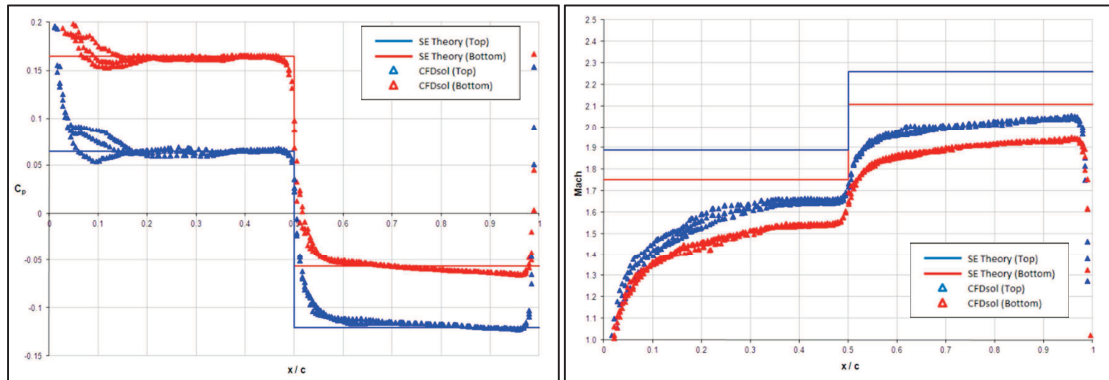


Figure 7.40: Pressure Coefficient (left) and Mach Number (right) around Double Wedge Airfoil at Mach 2 and 2-deg AOA.

Table 7.1: Comparison of Local Mach, Pressure, and Shock Angles to Theory for Double-Wedge Airfoil (Mach 2, 2-deg AOA).

Coeff. of Pressure			Local Mach #			Shock Angle			Aero. Coeff.		
Zone	C_p	% diff	Zone	M	% diff	---	(deg)	% diff	---	---	---
1	0.0639	-1%	1	1.65	-13%	LE, Top	38	10%	c_l	0.0741	-10%
2	0.1639	-1%	2	1.53	-12%	LE, Bot'm	35	2%	$c_{d,w}$	0.0238	15%
3	-0.1184	-2%	3	2.03	-10%	TE, Top	30	-12%	$c_{m,LE}$	0.0319	-13%
4	-0.0614	9%	4	1.93	-9%	TE, Bot'm	31	8%			

The convergence rates in CFDsol were tested at various relaxation factor τ . A τ value of 0.1 represents one tenth of the ideal Courant-Fredrick-Lewis (CFL) condition. The ideal condition represents an element time step equal to the minimum distance across an element divided by the maximum propagation velocity on that element. The supersonic field is very stable in CFDsol, so under-relaxation ($\tau > 1$) was also tested up to twice the ideal CFL condition. The results are shown in Figure 7.42.

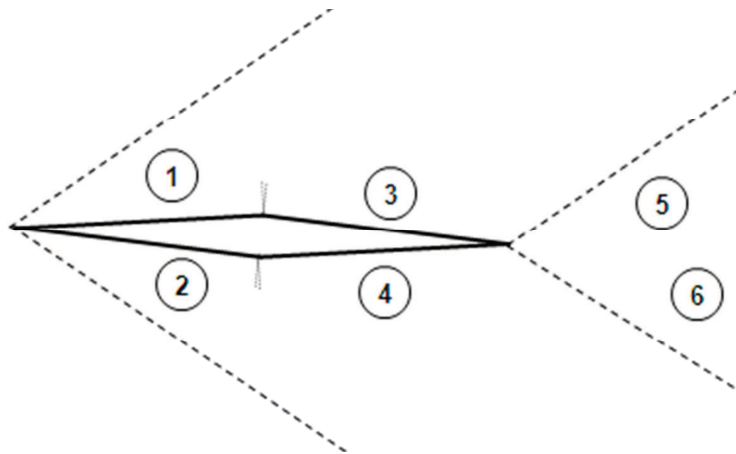


Figure 7.41: Four Flow Regions around Double-Wedge Airfoil.

Three interesting conclusions can be drawn from the comparison: First, smaller relaxation factors τ increase the run time proportional to the ratio of relaxation factor (i.e., from $\tau = 1$ to 0.5, the convergence time doubles). Second, the minimum residual (representing convergence on particular solution) is lower for smaller relaxation factors. Finally, the final convergence of the energy equation by CFDsol is on the order of 10^{-6} RMS change (or residual). After the solution converges, the residual begins to rise due to numerical hunting.

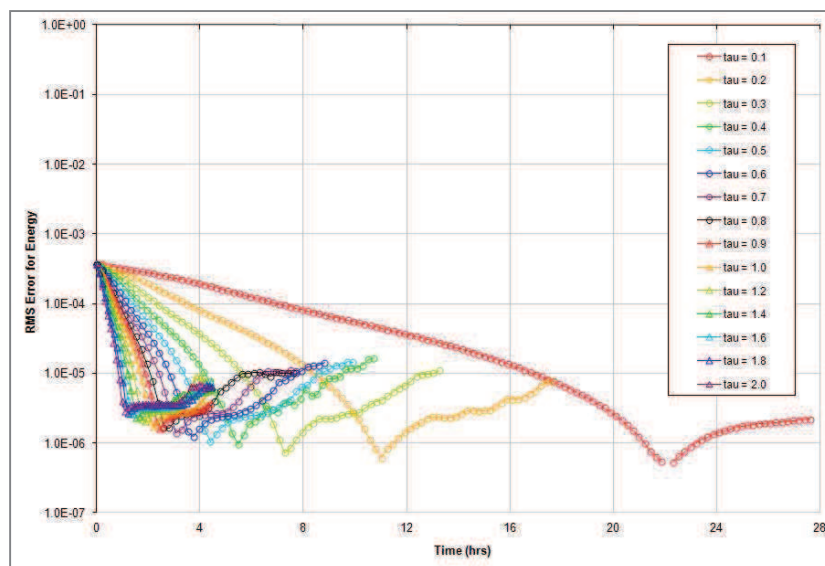


Figure 7.42: Convergence Rate of Energy Residual vs. τ .

7.1.4 Time-Accurate (Acoustic)

The NACA 0012 airfoil was also used to demonstrate unsteady, time-accurate flow conditions: The Wagner problem. The steady, low subsonic mesh (Figure 7.1) was reused here. The airfoil was held at an angle-of-attack of 5 degrees in a Mach 0.3 flow field. The initial conditions for the solution are seeded with the freestream properties, ignoring the presence of the airfoil. Time-accurate cases are often seeded with a steady solution about which the unsteady solution is perturbed. The Wagner problem is seeded with freestream properties so that the field reacts to the presence of the airfoil at the beginning of the solution.

When the Wagner simulation begins, acoustic waves are released from the surface of the airfoil. The acoustic waves radiate outward transmitting the presence of the airfoil to the surrounding airflow. (The acoustic waves can be seen in Figure 7.44.) The airfoil is held at an angle of attack, so the flow over the top and bottom surfaces are different. The velocity difference develops vorticity at the trailing edge. As the solution progresses, the vortex builds to a point that it can no longer be maintained at the trailing edge and releases downstream. The vortex drifts away from the trailing edge at nearly the freestream velocity. The presence of the vortex induces a downwash on the airfoil and decreases its lift. As the vortex moves downstream, downwash is diminished so that the lift approaches its steady-state value.

The lift history predicted by CFDsol is plotted in Figure 7.43 in comparison to Jones' (1940) approximation to the Wagner solution. The initial lift reflects the acoustic response of the freestream properties. The lift quickly drops from its initial value to its minimum at $t^* = 1/2$ because of the release of the acoustic waves. The discrete time over which the lift drops is due to the compressibility of the solution. (An incompressible solution would show a much

faster, if not instant, response.) The lift then begins to increase because the trailing vortex is drifting further downstream. As the vortex passes out of the domain, the solution approaches its steady-state value. The lift history predicted by CFDsol after $t^* = \frac{1}{2}$ matches Jones' approximation very well.

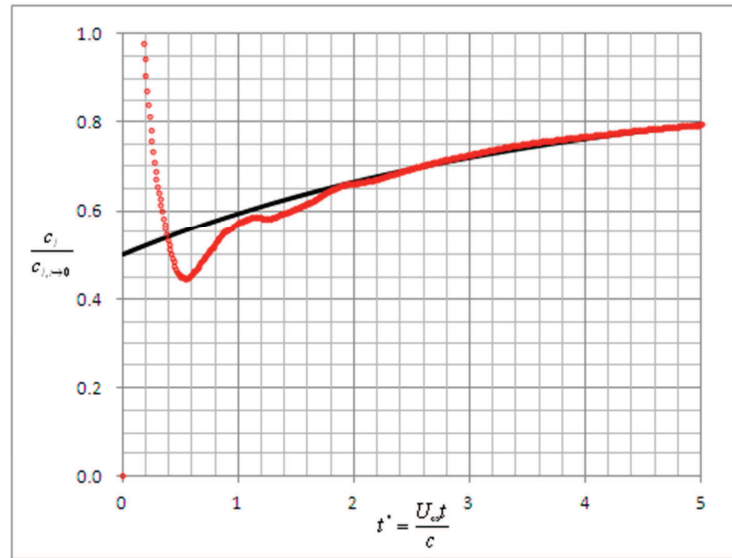


Figure 7.43: Wagner Solution for NACA 0012 Airfoil (Mach 0.3, 2° AOA).

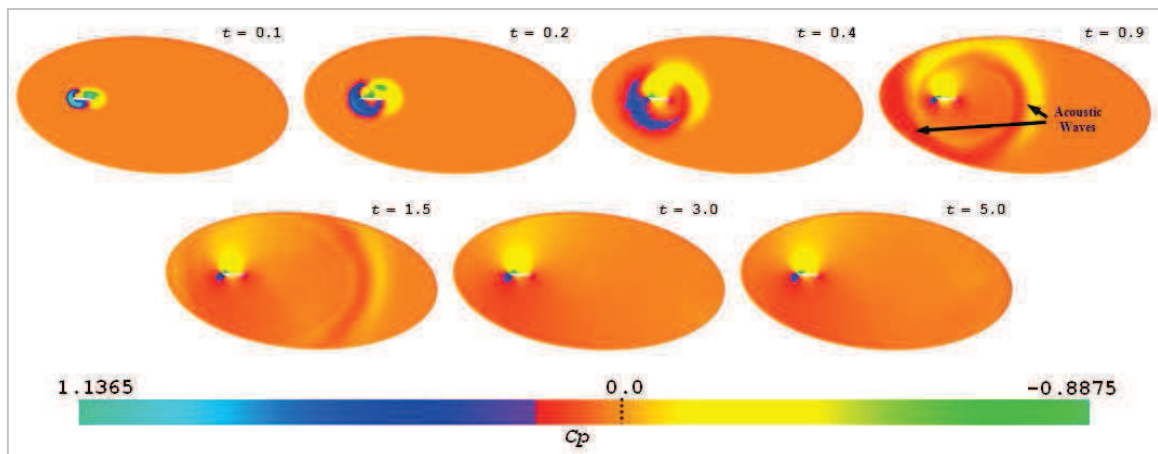


Figure 7.44: Snapshots of Unsteady Pressure Distribution (C_p) for Wagner Solution.

7.1.4.1 Theodorsen Pitching Airfoil (Non-Inertial)

The NACA 0012 airfoil was pitched about its quarter chord in the non-inertial frame. Initial conditions were created by rotating the freestream in the non-inertial frame by 5-degrees and holding the frame in place until a steady solution developed. The airfoil was then released to pivot about the quarter chord at a frequency $\omega = 0.8$ rad/s. The motion was forced using the mass and stiffness of the system. A time step of 10^{-5} was used to obtain a stable solution and produce the velocity and entropy distributions seen in Figure 7.45 and Figure 7.46, respectively. Both distributions show the wake as a dashed white line. Figure 7.47 shows the lift and drag histories, and Figure 7.48 shows the moment history for the pitching airfoil. Several cycles are required to converge the period nature of the flow.

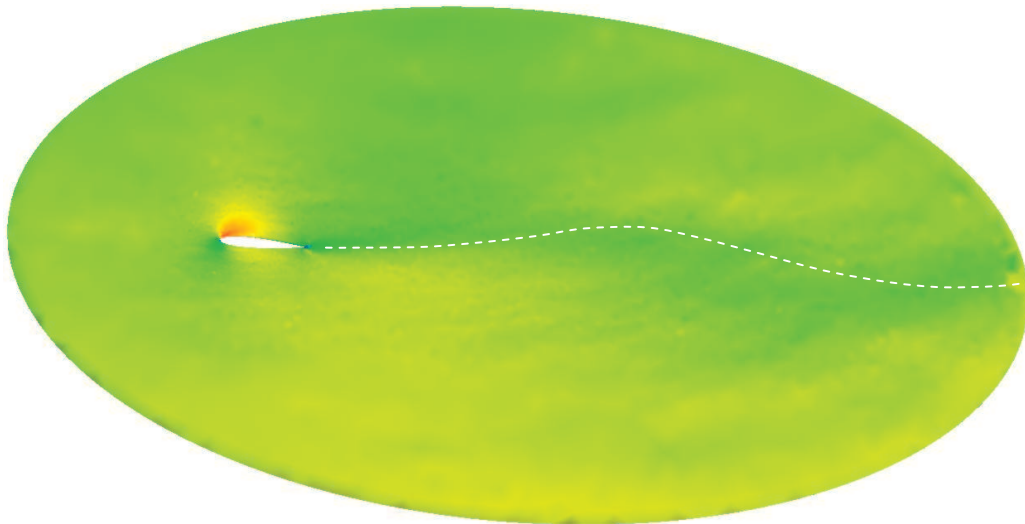


Figure 7.45: Velocity Distribution around Pitching Airfoil (Non-Inertial).

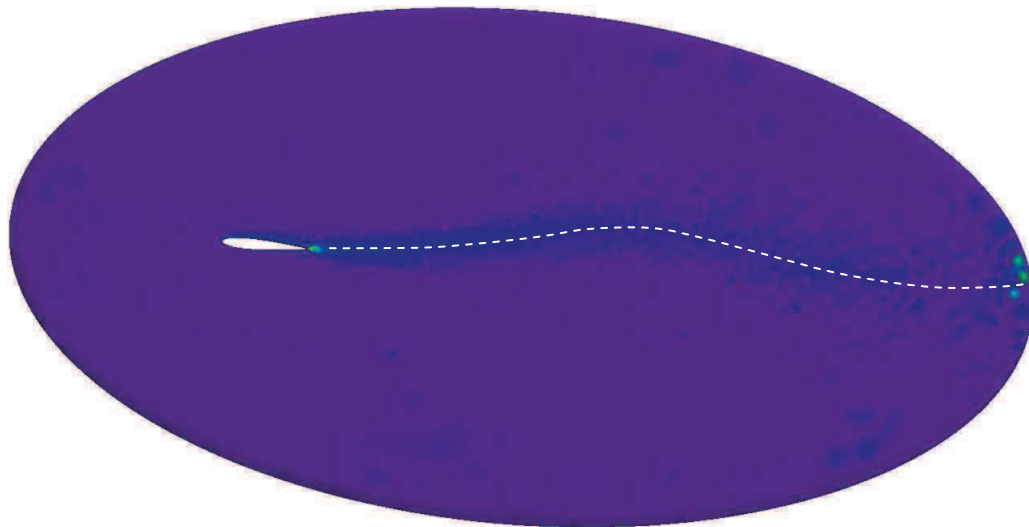


Figure 7.46: Entropy Distribution around Pitching Airfoil (Non-Inertial).

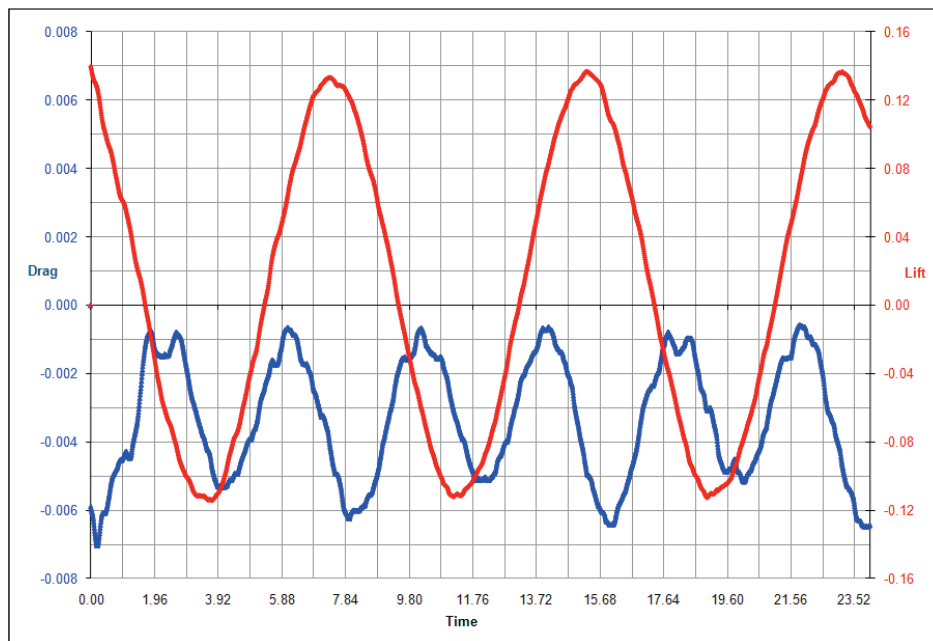


Figure 7.47: Lift and Drag History for Pitching Airfoil (Non-Inertial).

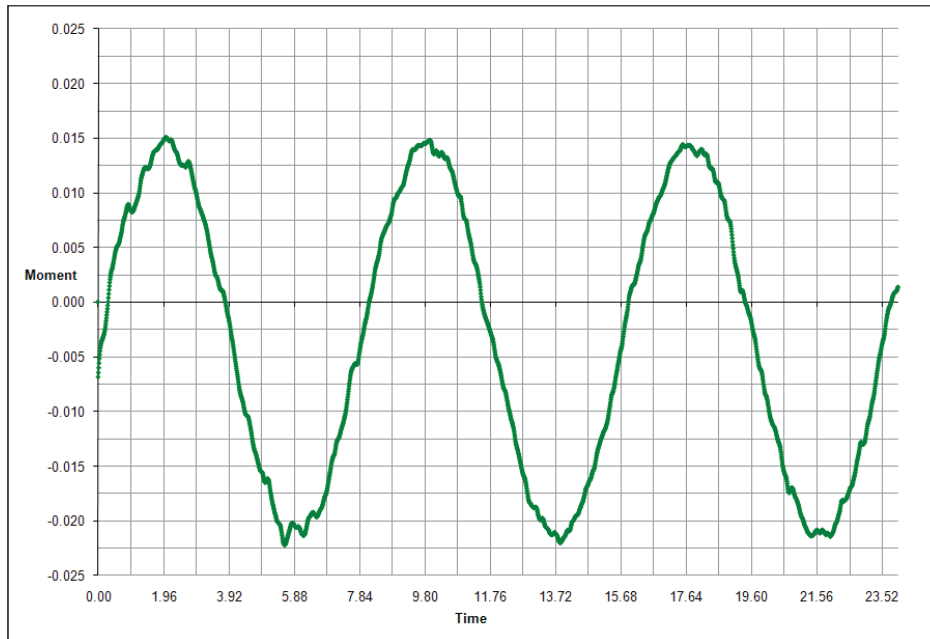


Figure 7.48: Moment History for Pitching Airfoil (Non-Inertial).

7.1.4.2 Theodorsen Plunging Airfoil (Non-Inertial)

The airfoil was then plunged with a maximum displacement of 0.1094 units. The sinusoidal plunging motion creates a maximum velocity of 0.0875 units per second, which is equivalent to rotating the angle of the flow by 5-degrees. The initial conditions were created by plunging the airfoil at a constant 0.0875 units per second. The plunge rate was held until a steady solution developed, and then the airfoil was released. The mass and stiffness of the system was again used to force the airfoil through a frequency $\omega = 0.8$ rad/s. A time step of 10^{-5} was used again to obtain a stable solution. The velocity and entropy distributions are shown in Figure 7.49, where a dashed line signifies the vortex wake. The lift and drag histories are shown in Figure 7.50, and the moment about the quarter chord is shown in Figure 7.51. Several cycles are required to converge the period flow field.

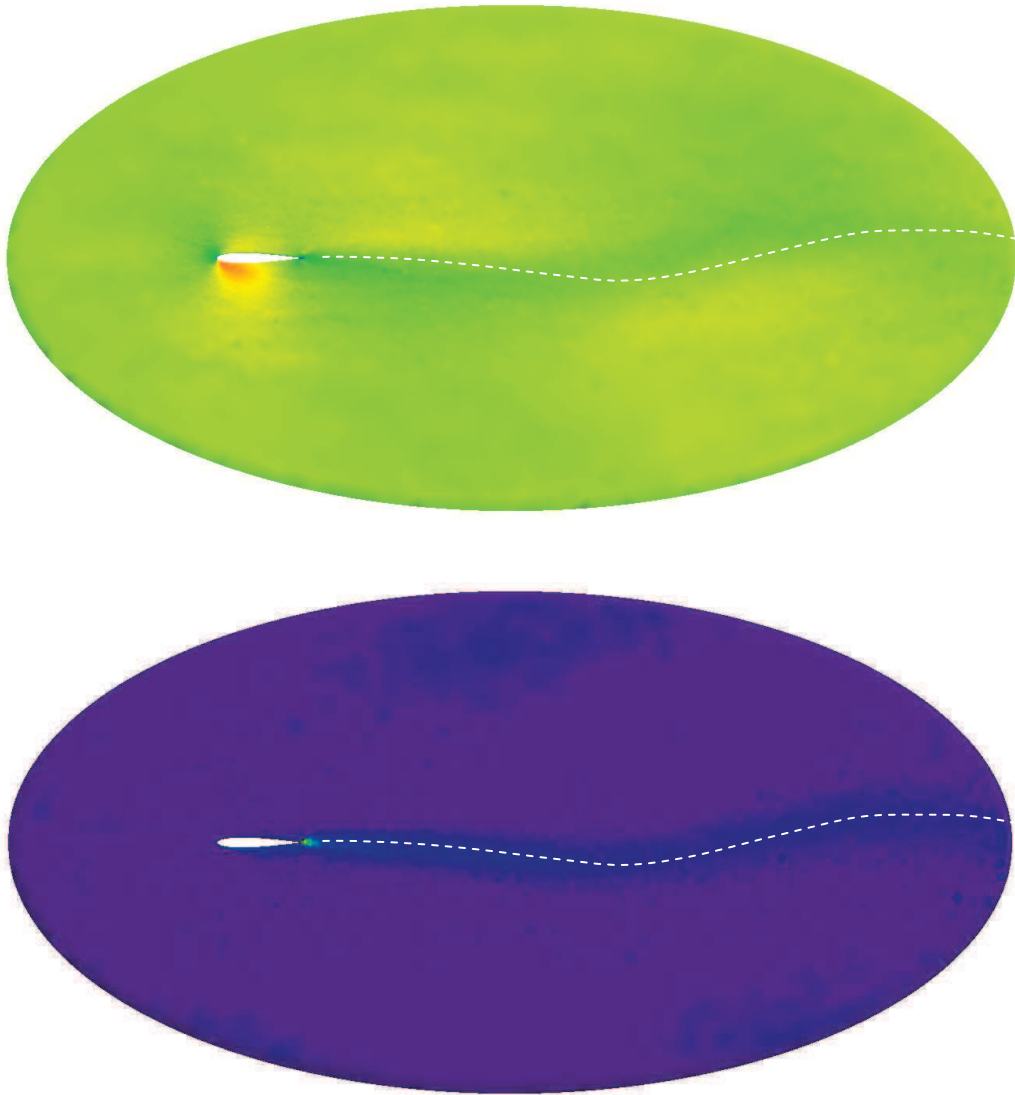


Figure 7.49: Velocity (top) & Entropy (bottom) around Plunging Airfoil (Non-Inertial).

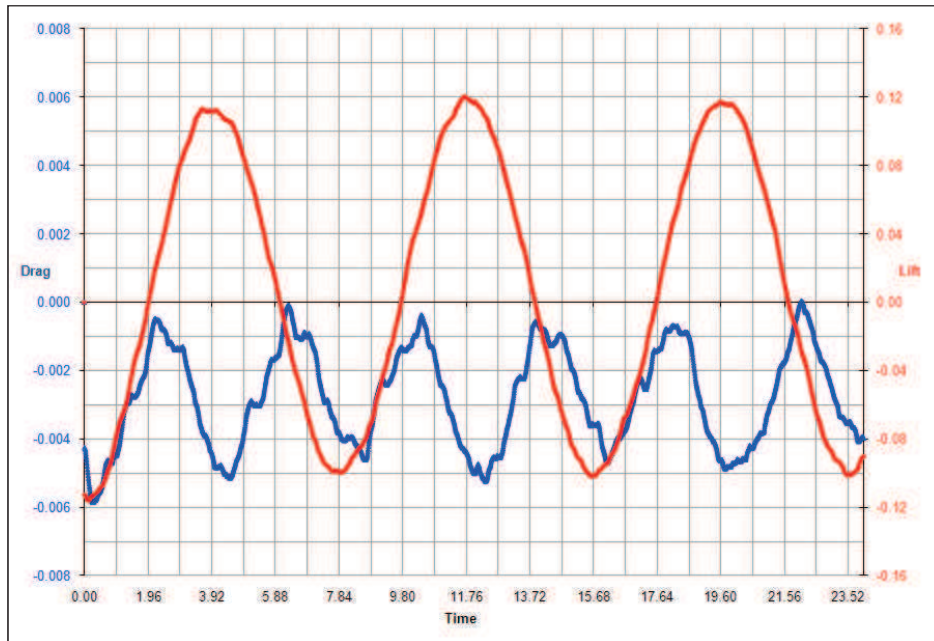


Figure 7.50: Lift and Drag History for Pitching Airfoil (Non-Inertial).

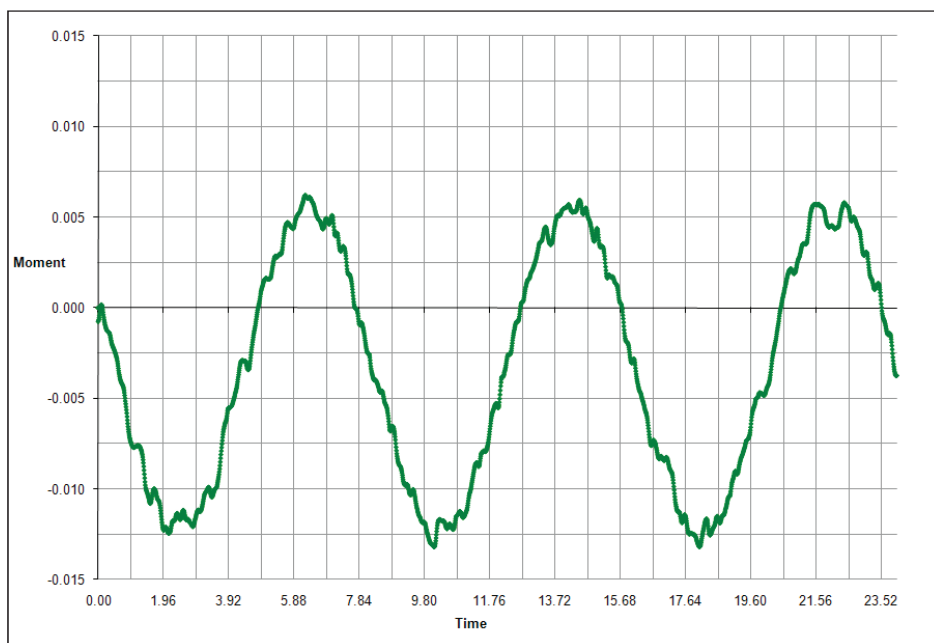


Figure 7.51: Moment History for Pitching Airfoil (Non-Inertial).

7.1.4.3 Theodorsen Pitching Airfoil (Transpiration).

The pitching airfoil was repeated in the inertial frame by transpiring the geometry about the quarter chord to represent pitching motion. Initial conditions were created by transpiring the airfoil to represent a 5-degree rotation. The deflection was held until a steady solution developed. The mode shape was then released to “pivot about the quarter chord”; motion was forced using the mass and stiffness. A time step of 10^{-5} was again used for stability. Velocity and entropy distributions are shown in Figure 7.53, with a dashed white line representing the vortex wake. Figure 7.52 shows the generalized force for the pitching mode. The flow field converges much more quickly with transpiration than for the non-inertial cases.

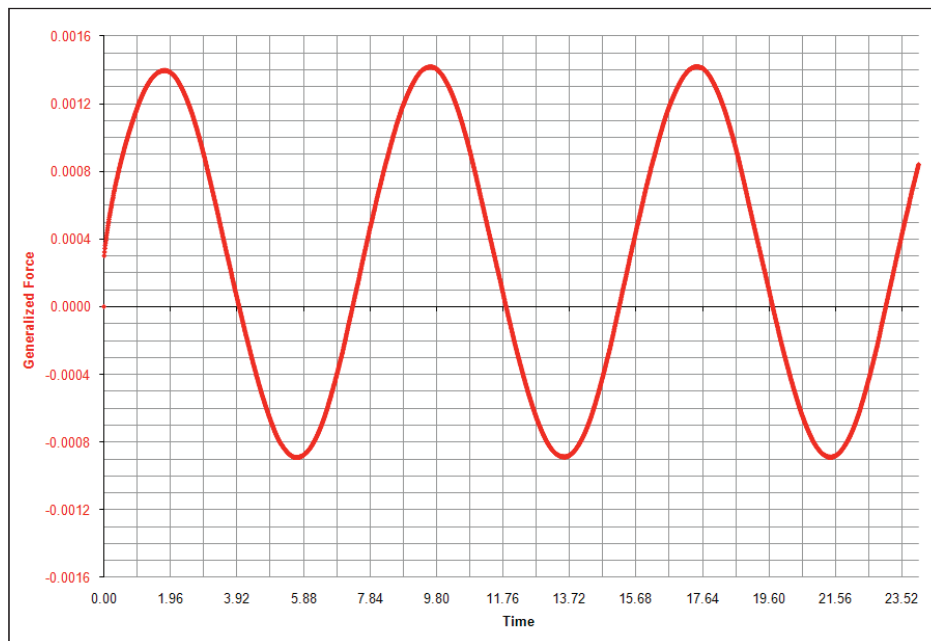


Figure 7.52: Generalized Force for Pitching Mode (Transpiration).

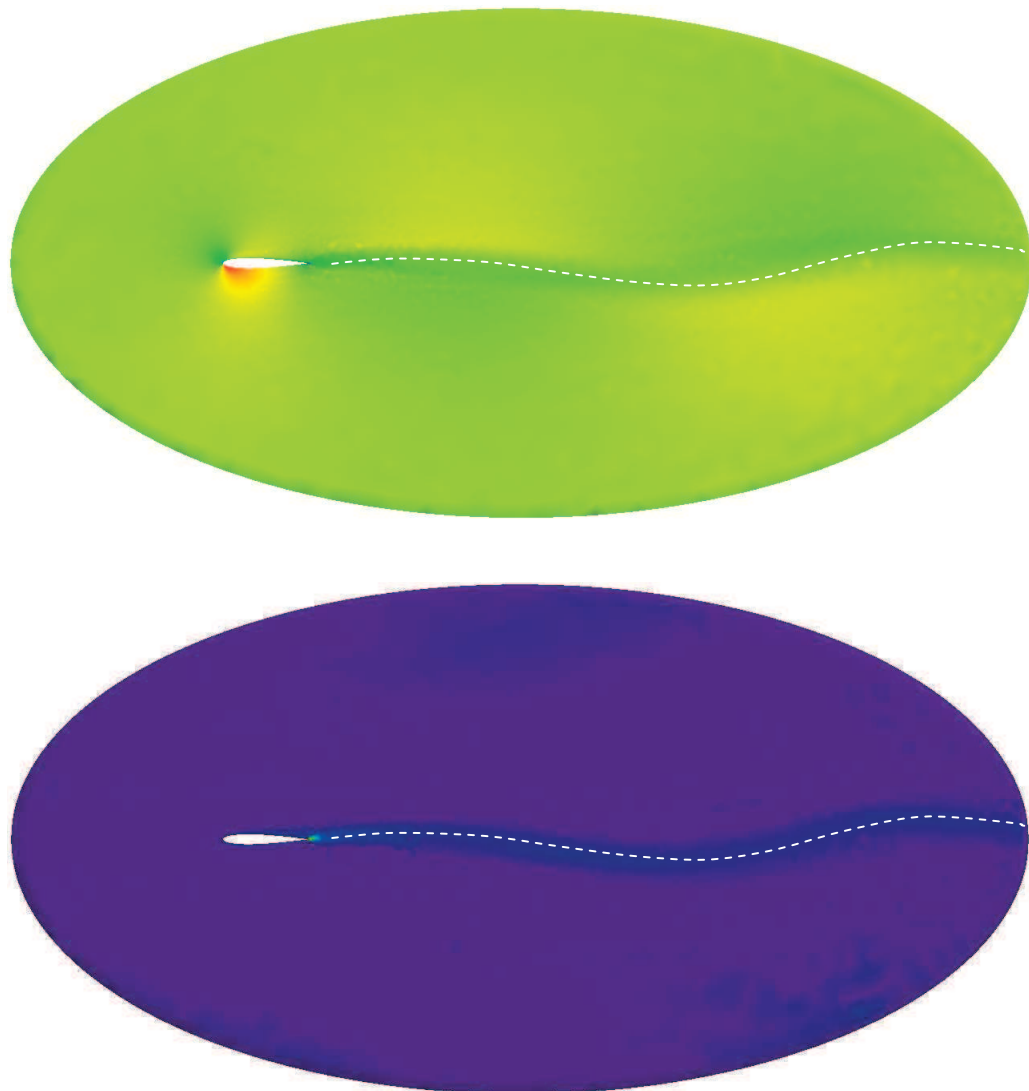


Figure 7.53: Velocity (top) & Entropy (bottom) around Pitching Airfoil (Transp.).

7.1.4.4 Theodorsen Plunging Airfoil (Transpiration).

The plunging airfoil was also repeated in the inertial frame with transpiration. Initial conditions were created by transpiring a velocity across the airfoil at 0.0875 units per second. The deflection was held until a steady solution developed. The mode shape was then released to “plunge”; motion was forced using the mass and stiffness. A time step of 10^{-5} was again used for stability. Velocity and entropy distributions are shown in Figure 7.54 and Figure

7.55, respectively, with a dashed white line representing the vortex wake. Figure 7.56 shows the generalized force for the pitching mode. The flow field converges much more quickly with transpiration than for the non-inertial cases.

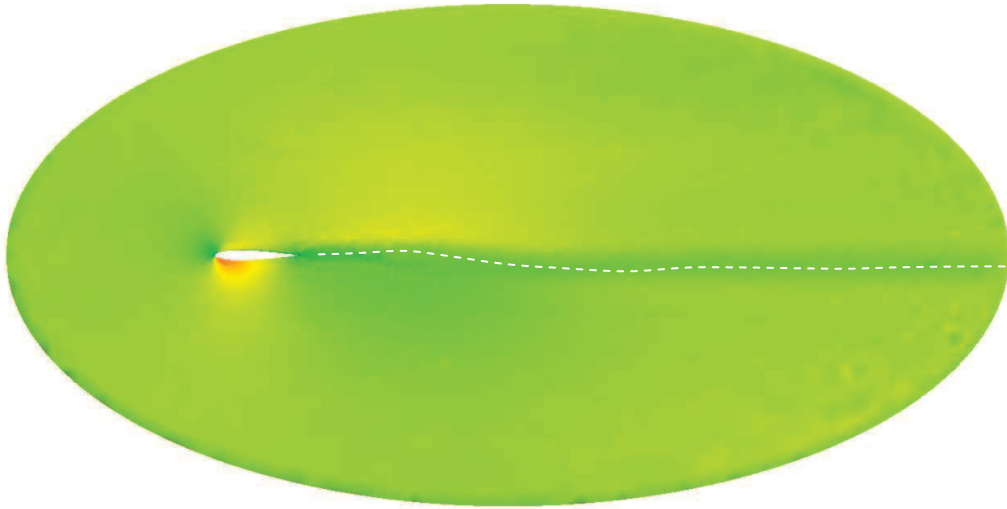


Figure 7.54: Velocity Distribution around Plunging Airfoil (Transpiration).

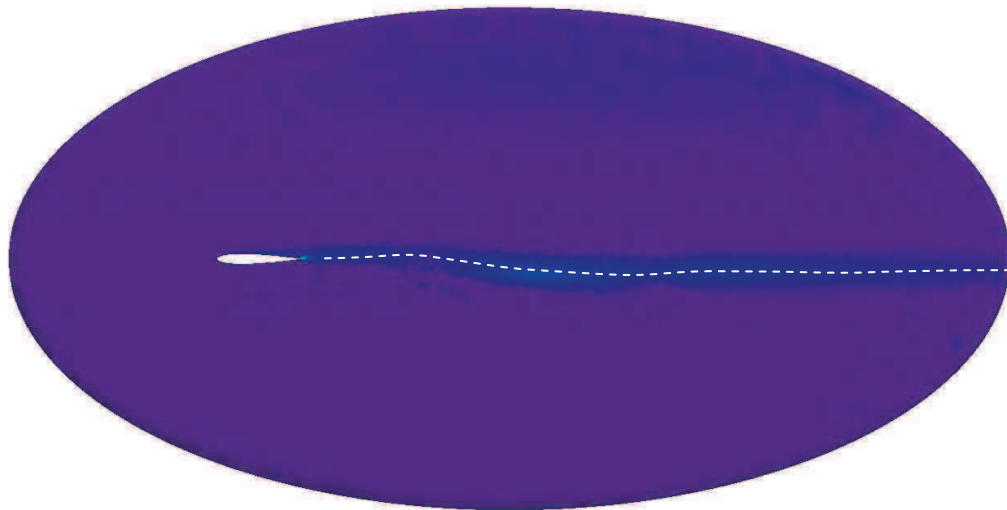


Figure 7.55: Entropy Distribution around Plunging Airfoil (Transpiration).

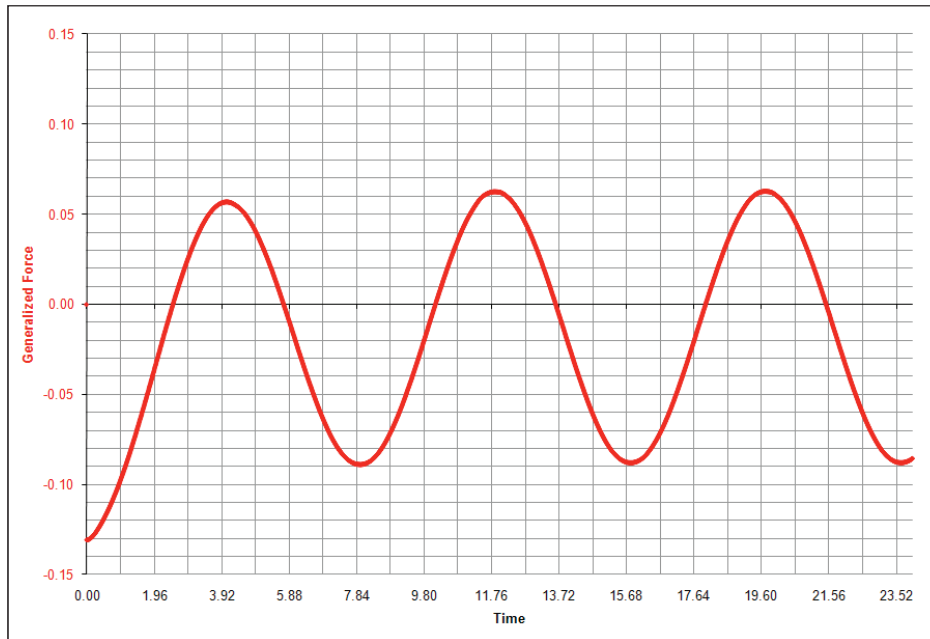


Figure 7.56: Generalized Force for Plunging Mode (Transpiration).

7.2 Propulsion Modeling

The quasi-combustion terms and rocket boundary conditions were tested in CFDsol using a linear combustor and converging-diverging rocket nozzle. The quasi-combustion terms were tested with mass and enthalpy addition at subsonic, supersonic, and hypersonic speeds. The CD nozzle was tested at three total pressures at and below design pressure.

7.2.1 Quasi-Combustion Terms

The quasi-combustion terms have been tested using enthalpy and mass generation. Three different Mach numbers were tested, representing subsonic, supersonic, and hypersonic regimes. The first set of tests adds enthalpy to the flow without mass addition. Two different amounts of energy were added to the supersonic and hypersonic cases. Two different distributions were tested in each situation: A constant hat-function and a cosine-smoothed

function. The solutions from CFDsol were checked versus Rayleigh line flow (John, 1984). Mass was then added to the subsonic and supersonic cases, followed by the combination of mass and enthalpy addition. The properties predicted by adding mass and enthalpy to the flow in CFDsol were compared to theoretical values created by modifying Rayleigh's theory to include mass flow (Bathie, 1996; Mattingly, 1996). These cases show that the quasi-combustion terms work properly and within reasonable accuracy for subsonic, supersonic, and hypersonic speeds.

7.2.1.1 Subsonic Linear Afterburner (Mach 0.4)

Heat was added to the flow in a subsonic combustor to simulate the effects of combustion. Figure 7.57 and Figure 7.58 were produced by modifying the outflow boundary. The boundary integrals along the far field BC were modified with the following code:

```

IF((VNR .LE. 0.00D0) .OR. (AMloc .LT. 1.0d0)) THEN
...
else
  ENER1 = TTINF/GAMMA
  ENER2 = 0.5D0*(UNKNO(2,IP)**2+UNKNO(3,IP)**2
&          +UNKNO(4,IP)**2)/(UNKNO(1,IP)**2)
  rhoE = (ENER1+ENER2)*UNKNO(1,IP)
  pres = PINF * 0.768d0 ! p2 / p1 = 0.768
  rhoH = (rhoE + PINF) * 0.8071549d0 ! rH2 / rH1 = 0.491 * 1.6439
  UNKNO(1,IP) = RHOINF * 0.491d0 ! rho2 / rho1 = 0.491
  UNKNO(2,IP) = RHOZ1 * 2.037d0 ! u2 / u1 = 2.037
  UNKNO(3,IP) = RHOZ2 * 0.0d0
  UNKNO(4,IP) = RHOZ3 * 0.0d0
  UNKNO(5,IP) = rhoH - pres
END IF

```

This modification was required because the Riemann conditions at both the inflow and outflow boundaries allow the solution to “float” slightly while gradients are created to balance the governing equations. The final solution does not match at the inflow or outflow boundaries, but the ratio of their properties is correct. The absolute properties are used to calculate the local Mach number, which is different at both planes. To match the solution,

the outflow condition is modified to resemble the desired properties, trapping the solution where desired.

The results were compared with Rayleigh line theory. The maximum error for the Mach 0.4 cases is 38%. The subsonic results do not match as well as desired. The changes due to the presents of heat addition do not reflect the boundary conditions as much as the initial conditions. Further testing should be completed with the quasi-combustion terms in subsonic flow, but these tests will not occur during the term of this research project.

Mass was added to the flow in a subsonic duct to demonstrate the mass addition terms.

Figure 7.59 and Figure 7.60 were produced by modifying the outflow boundary condition.

The boundary integrals along the far field BC were modified with the following code:

```

IF((VNOR .LE. 0.00D0) .OR. (AMloc .LT. 1.0d0)) THEN
  ...
else
  ENER1 = TTINF/GAMMA
  ENER2 = 0.5D0*(UNKNO(2,IP)**2+UNKNO(3,IP)**2
&          +UNKNO(4,IP)**2)/(UNKNO(1,IP)**2)
  rhoE = (ENER1+ENER2)*UNKNO(1,IP)
  pres = PINF * 0.986d0           ! p2 / p1 = 0.986
  rhoH = (rhoE + PINF) * 0.988176d0 ! rH2 / rH1 = 1.038 * 0.952
  UNKNO(1,IP) = RHOINF * 1.038d0   ! rho2 / rho1 = 1.038
  UNKNO(2,IP) = RHOZ1 * 1.012d0   ! u2 / u1 = 1.012
  UNKNO(3,IP) = RHOZ2 * 0.0d0
  UNKNO(4,IP) = RHOZ3 * 0.0d0
  UNKNO(5,IP) = rhoH - pres
END IF

```

Mass and enthalpy were added to the flow in a subsonic afterburner as a final test of the subsonic capabilities of the quasi-combustion terms in CFDsol. Figure 7.61 and Figure 7.62 were also produced by modifying the outflow boundary condition. The boundary integrals along the far field BC were modified with the following code:

```

IF((VNROR .LE. 0.00D0) .OR. (AMloc .LT. 1.0d0)) THEN
...
else
ENER1 = TTINF/GAMMA
ENER2 = 0.5D0*(UNKNO(2,IP)**2+UNKNO(3,IP)**2
&      +UNKNO(4,IP)**2)/(UNKNO(1,IP)**2)
rhoE = (ENER1+ENER2)*UNKNO(1,IP)
pres = PINF * 0.761d0 ! p2 / p1 = 0.761
rhoH = (rhoE + PINF) * 0.802068d0 ! rH2 / rH1 = 0.534 * 1.502
UNKNO(1,IP) = RHOINF * 0.534d0 ! rho2 / rho1 = 0.534
UNKNO(2,IP) = RHOZ1 * 1.966d0 ! u2 / u1 = 1.966
UNKNO(3,IP) = RHOZ2 * 0.0d0
UNKNO(4,IP) = RHOZ3 * 0.0d0
UNKNO(5,IP) = rhoH - pres
END IF

```

This modifications were required because the Riemann conditions at both the inflow and outflow boundaries allow the solution to “float” slightly while gradients are created to balance the governing equations. The final solution does not match at the inflow or outflow boundaries, but the ratio of their properties is correct. The absolute properties are used to calculate the local Mach number, which is different at both planes. To match the solution, the outflow condition is modified to resemble the desired properties, trapping the solution where desired. The ratio of mass, momentum, and energy flow rates have been calculated for these eight cases below. Each ratio uses the ratio of properties represented graphically:

$$\frac{\dot{m}_2}{\dot{m}_1} = \frac{\rho_2 u_2}{\rho_1 u_1} \quad \frac{P_2}{P_1} = \frac{(\rho_2 u_2^2 + p_2)A}{(\rho_1 u_1^2 + p_1)A} = \frac{p_2}{p_1} \frac{1 + \gamma M_2^2}{1 + \gamma M_1^2} \quad \frac{\dot{m}_2 H_2}{\dot{m}_1 H_1} = \frac{\rho H_2 u_2}{\rho H_1 u_1}$$

The results are grouped by generation and Mach number. The maximum error for the Mach 0.4 cases is 19.6%. The ratio of all three flow rates shows a loss in mass and energy flow rates for all cases where mass or heat are added to the flow. When a conservation equation is not affected by generation, the corresponding mass, momentum, and energy flow rates are properly conserved. When mass is added to CFDsol, the change in mass flow rate was off by as much as 1.2% of the added mass flow; and when heat is added to CFDsol, the change in energy flow rate was up to 1.6% off of the enthalpy flow rate added to the flow.

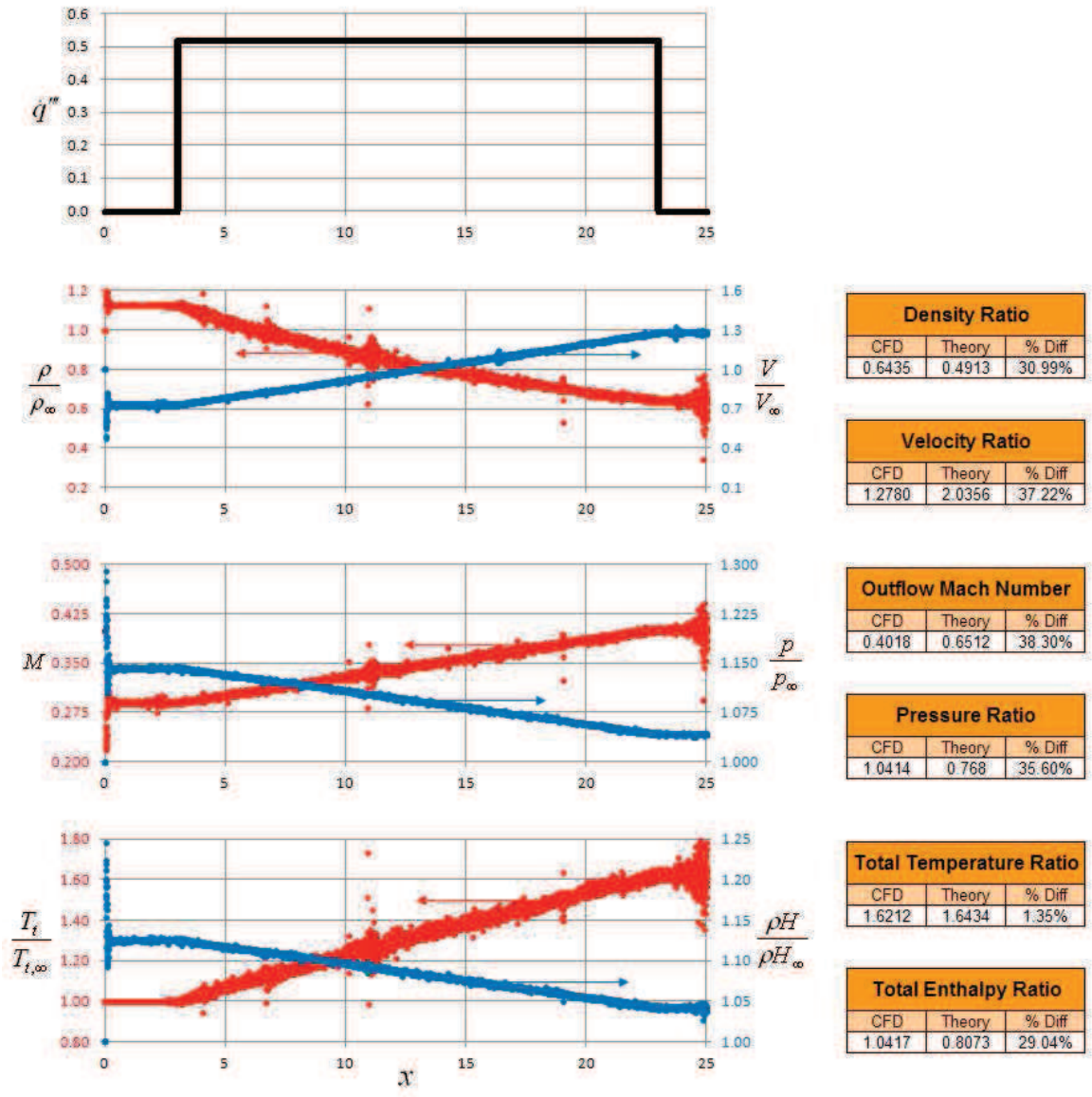


Figure 7.57: Subsonic (Mach 0.4) Constant Heat Generation.

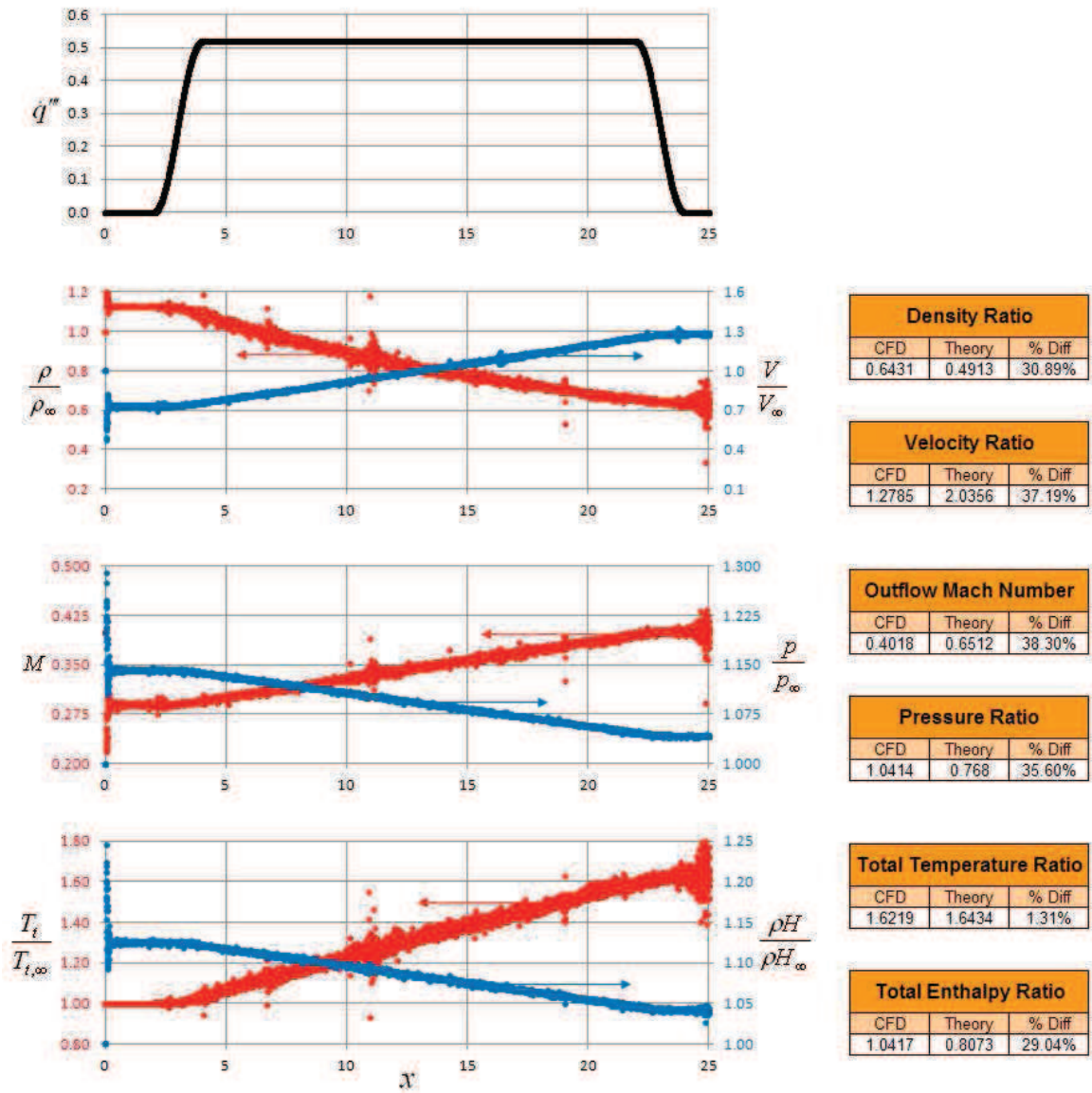


Figure 7.58: Subsonic (Mach 0.4) Cosine Heat Generation.

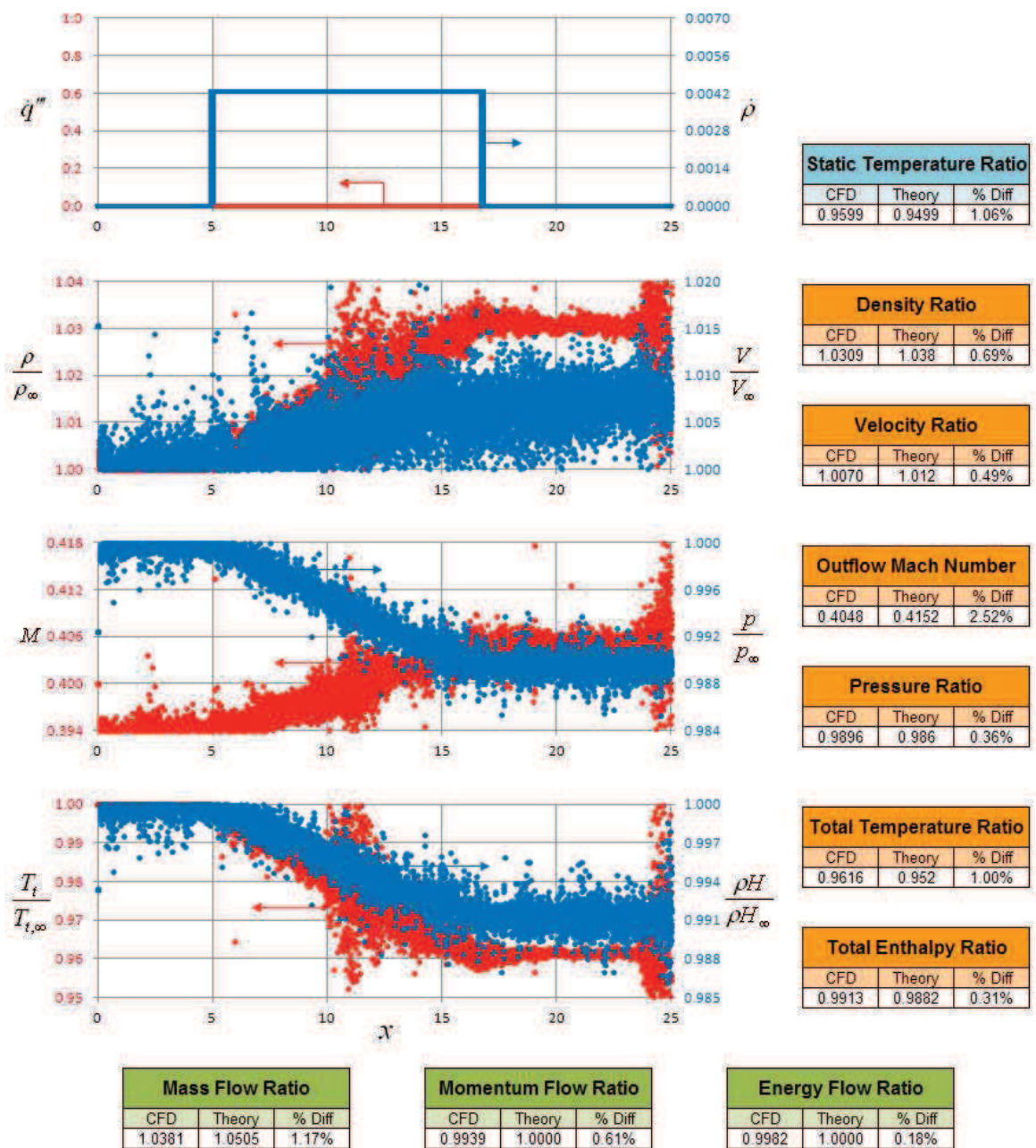


Figure 7.59: Subsonic (Mach 0.4) Constant Mass Generation.

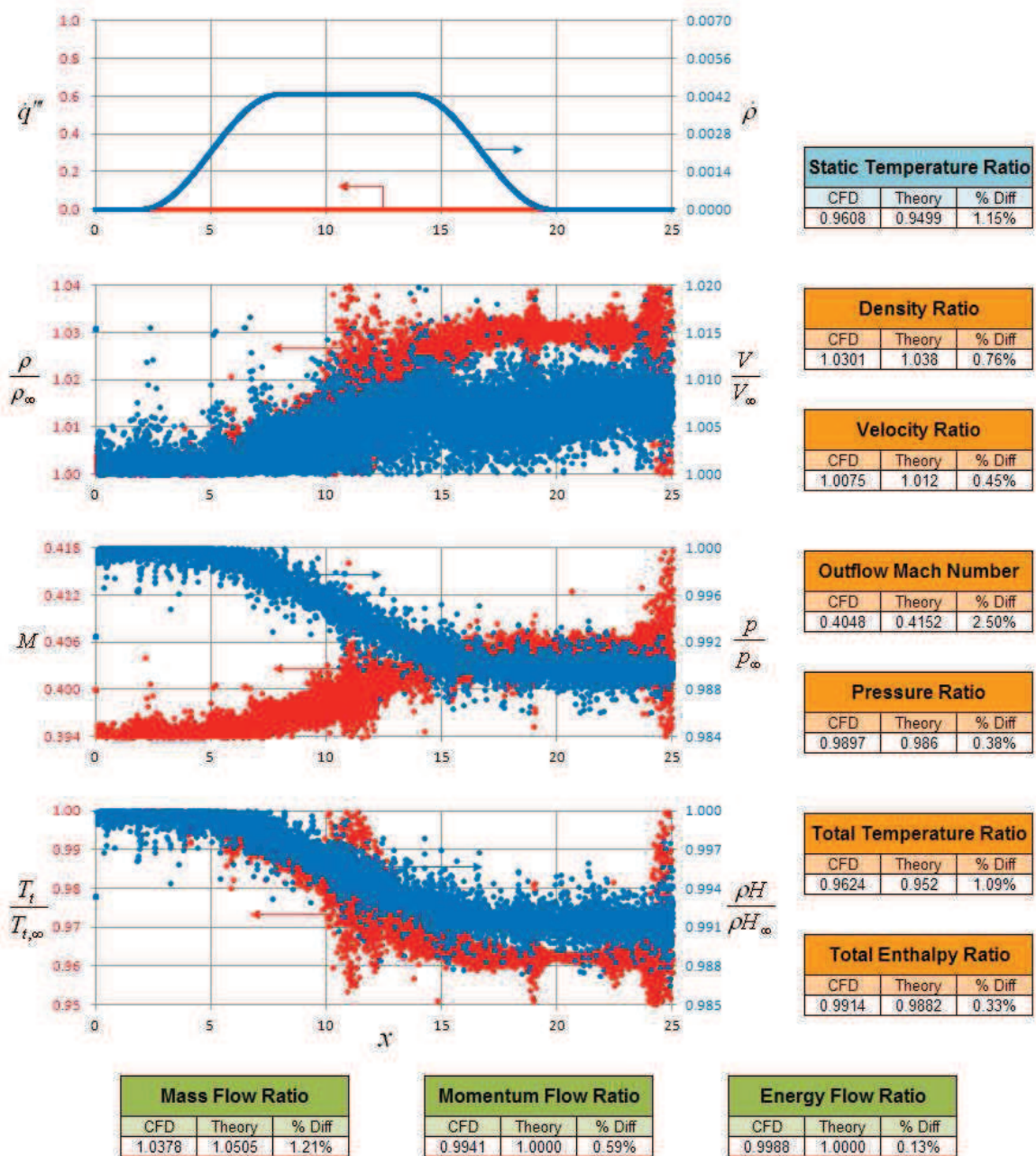


Figure 7.60: Subsonic (Mach 0.4) Cosine Mass Generation.

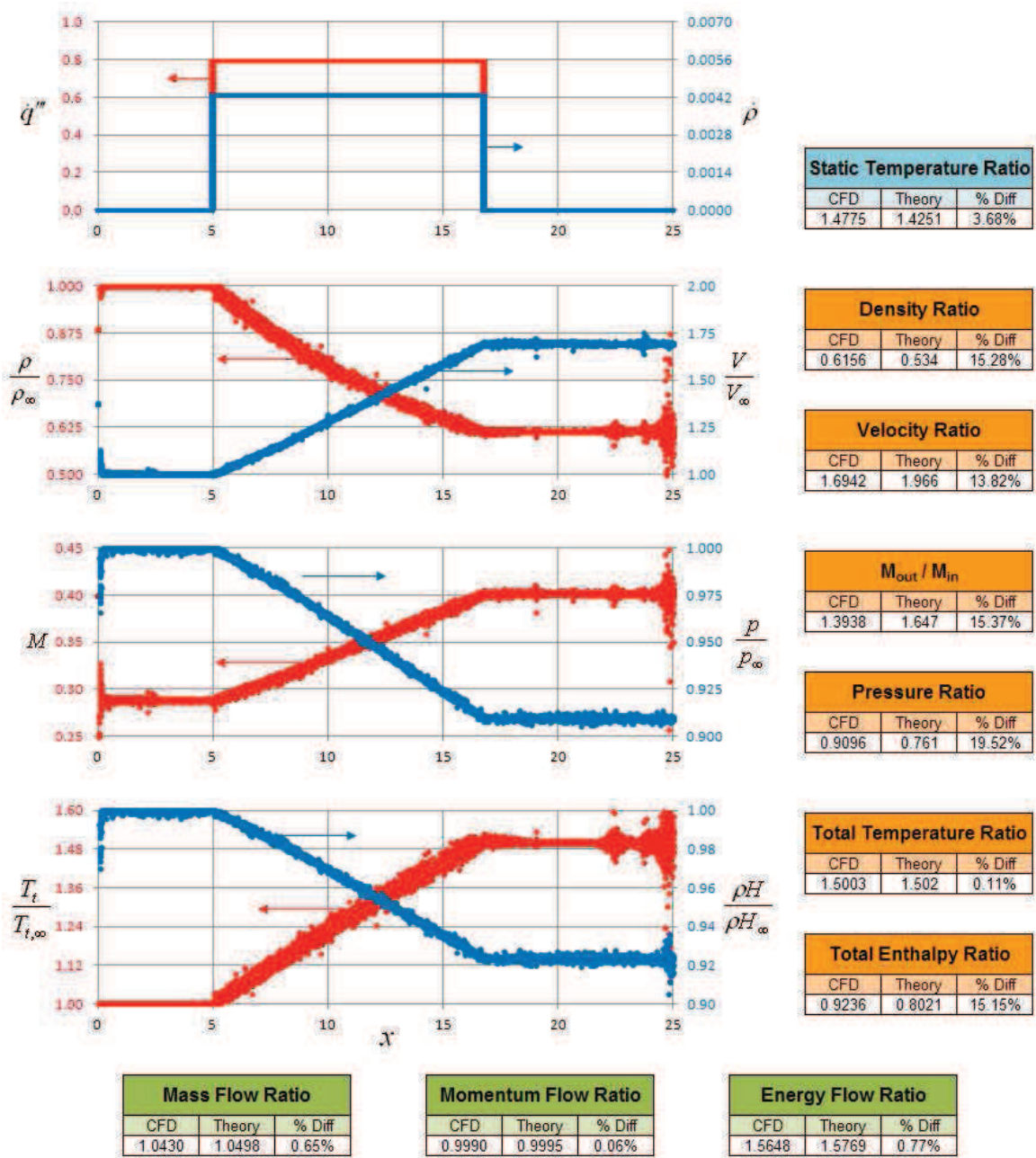


Figure 7.61: Subsonic (Mach 0.4) Constant Mass and Heat Generation.

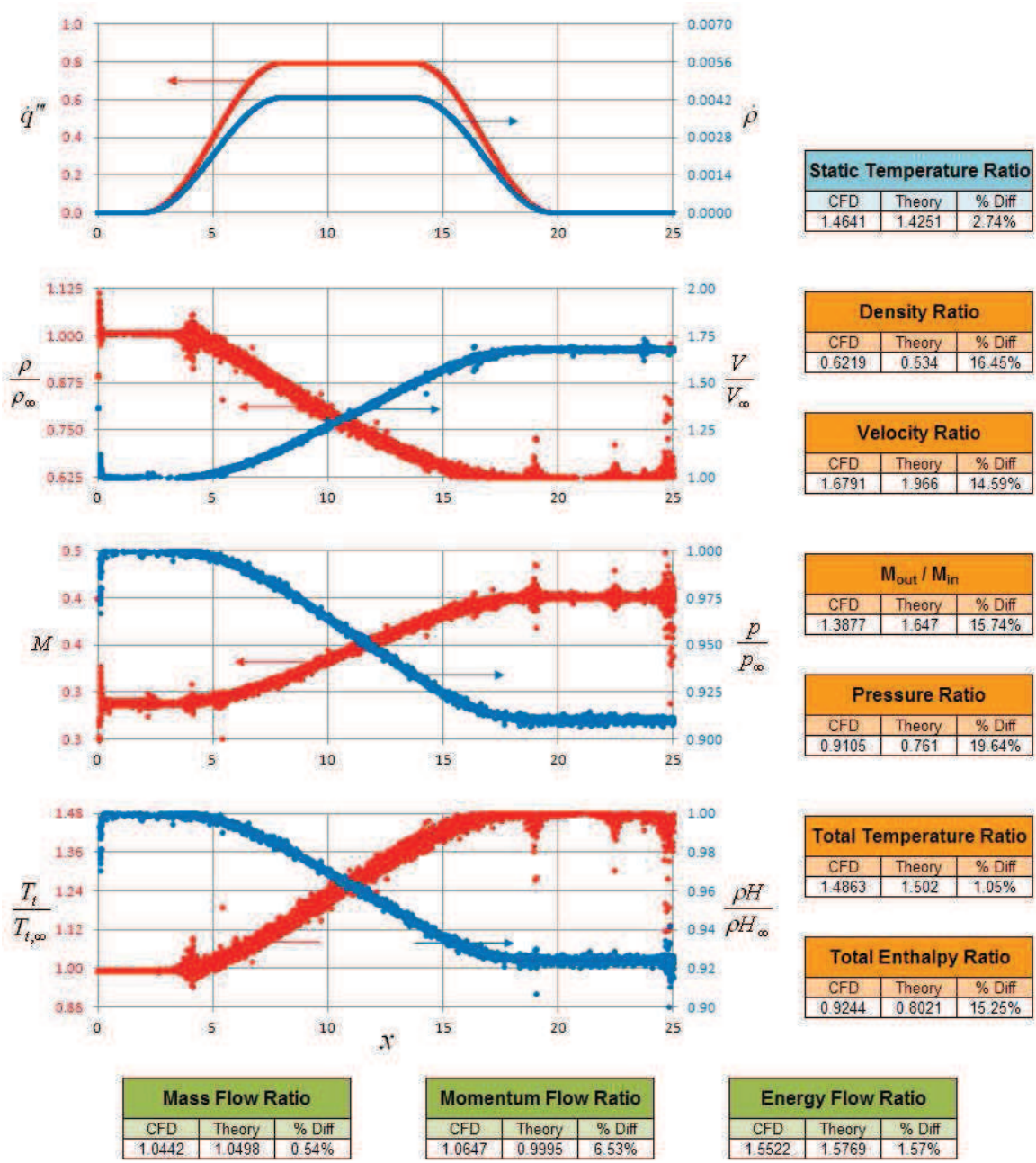


Figure 7.62: Subsonic (Mach 0.4) Cosine Mass and Heat Generation.

7.2.1.2 Supersonic Linear Afterburner (Mach 2)

Heat was added to the flow in a supersonic combustor to simulate the effects of combustion. The outflow boundary conditions do not need to be modified because the characteristics are taken from upstream (domain elements). The maximum error for the Mach 2 cases is 12%.

Mass (and enthalpy) was added to the flow in a supersonic duct to demonstrate the quasi-combustion terms in supersonic flows. The outflow boundary conditions do not need to be modified because the characteristics are taken from upstream (domain elements). The supersonic cases were built using the baseline CFDsol solver. The ratio of mass, momentum, and energy flow rates have been calculated for these eight cases below. Each ratio uses the ratio of properties represented graphically:

$$\frac{\dot{m}_2}{\dot{m}_1} = \frac{\rho_2 u_2}{\rho_1 u_1} \quad \frac{P_2}{P_1} = \frac{(\rho_2 u_2^2 + p_2)A}{(\rho_1 u_1^2 + p_1)A} = \frac{p_2}{p_1} \frac{1 + \gamma M_2^2}{1 + \gamma M_1^2} \quad \frac{\dot{m}_2 H_2}{\dot{m}_1 H_1} = \frac{\rho H_2 u_2}{\rho H_1 u_1}$$

The results are grouped by generation and Mach number. The maximum error for the Mach 2 cases is 11.0%. The ratio of all three flow rates shows a loss in mass and energy flow rates for all cases where mass or heat are added to the flow. When a conservation equation is not affected by generation, the corresponding mass, momentum, and energy flow rates are properly conserved. When mass is added to CFDsol, the change in mass flow rate was off by as much as 0.8% of the added mass flow; and when heat is added to CFDsol, the change in energy flow rate was up to 2.5% off of the enthalpy flow rate added to the flow.

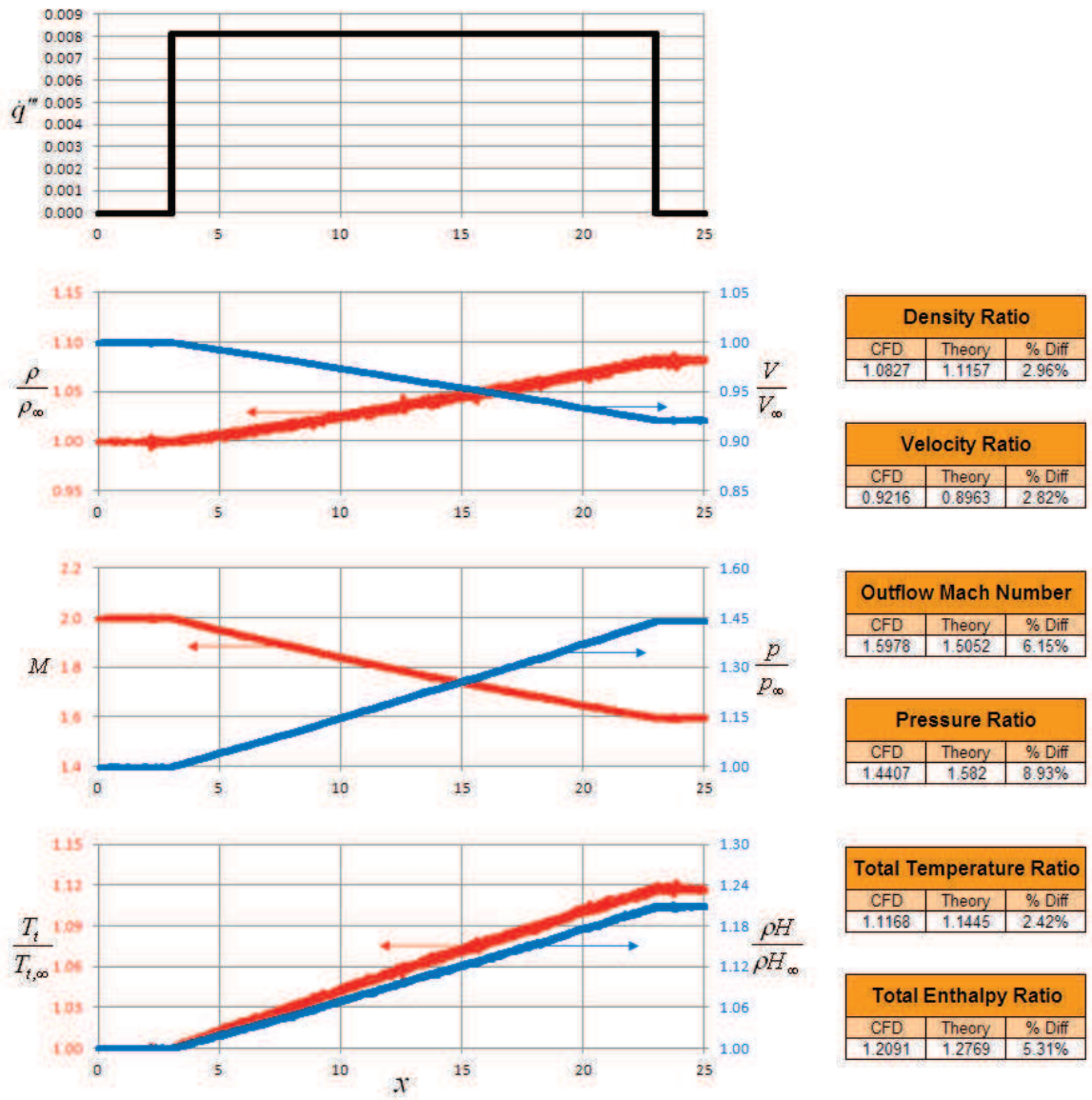


Figure 7.63: Supersonic (Mach 2.0) Constant Heat Generation.

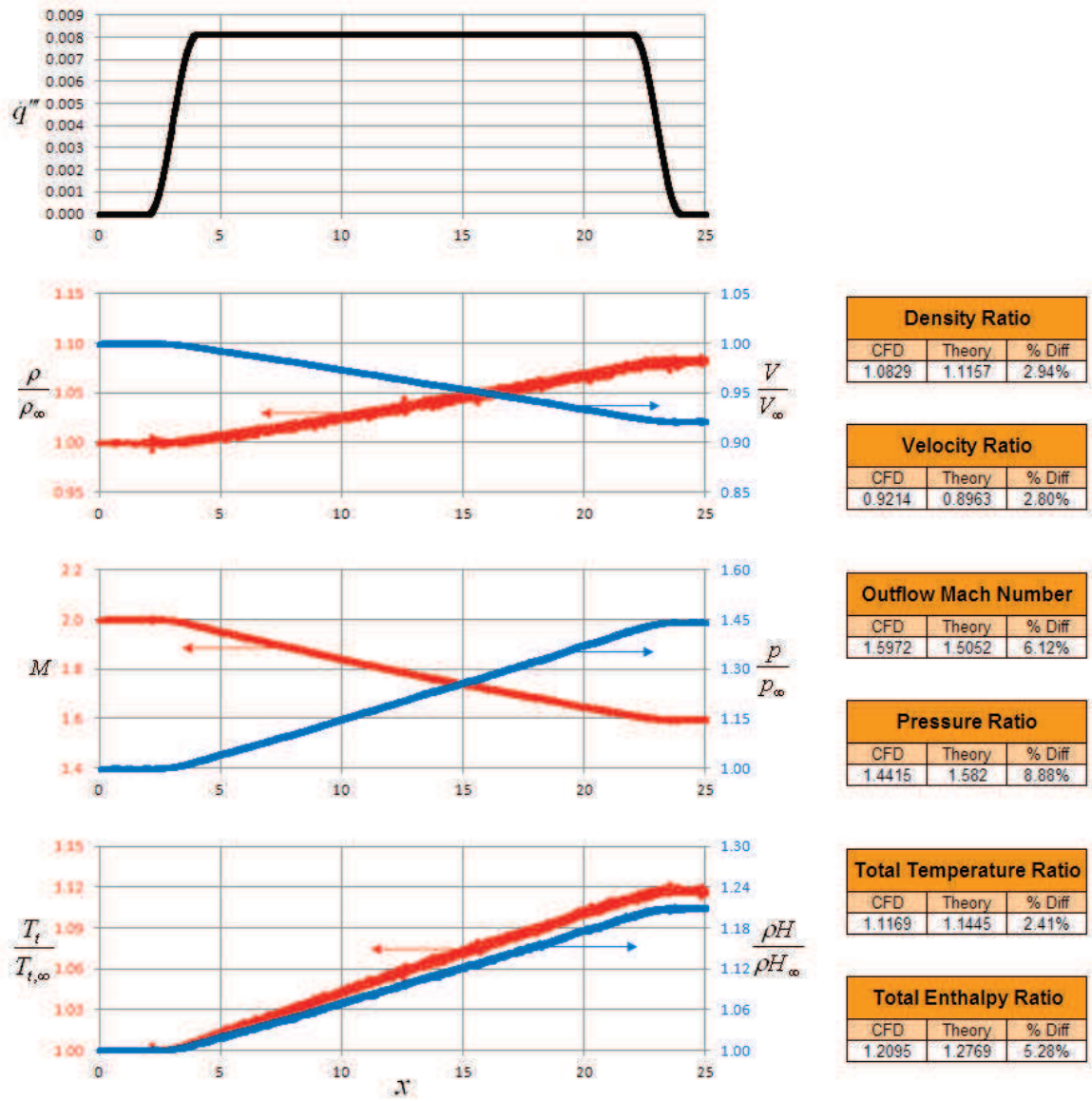


Figure 7.64: Supersonic (Mach 2.0) Cosine Heat Generation.

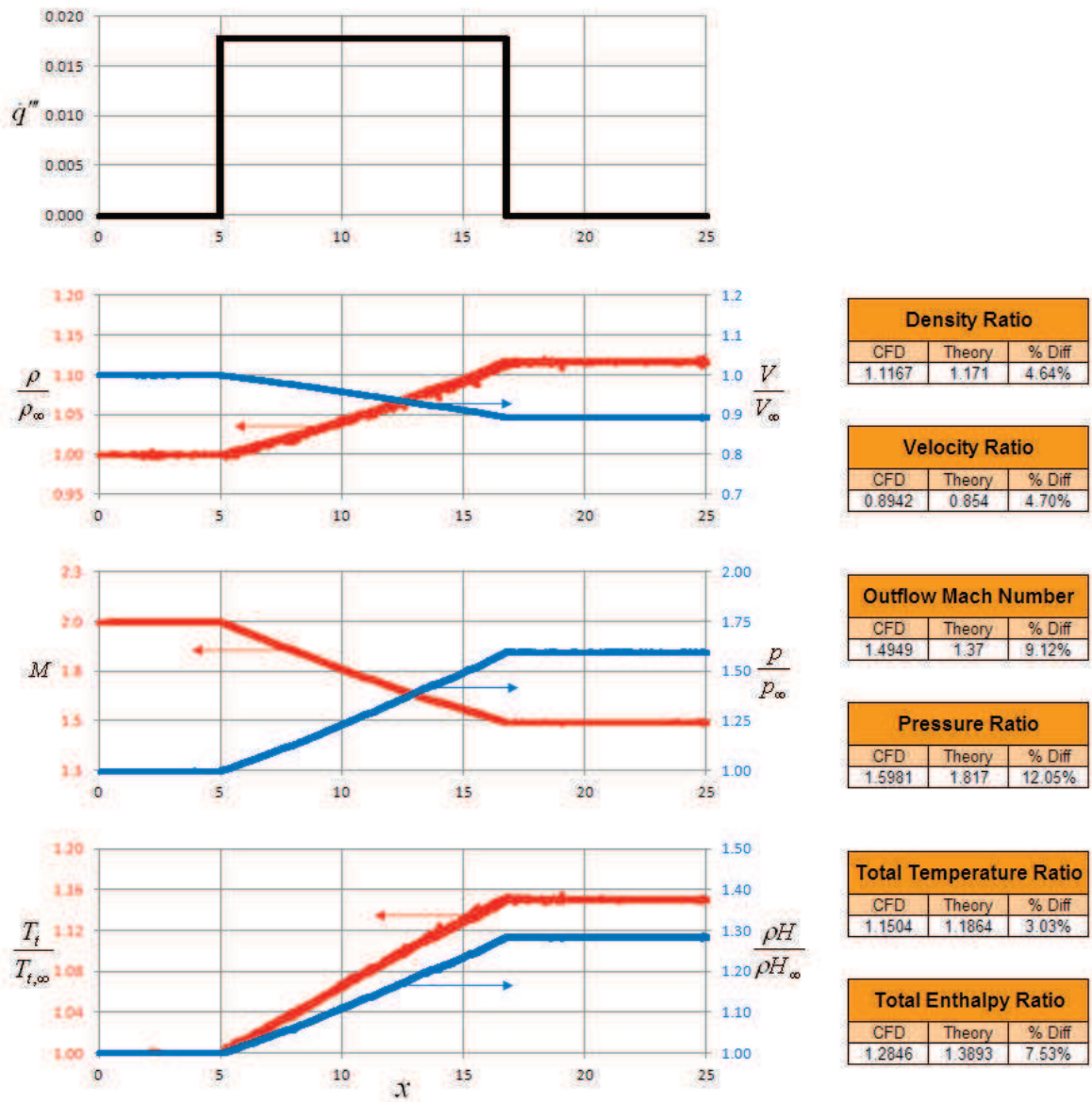


Figure 7.65: Supersonic (Mach 2.0) Constant Heat Generation.

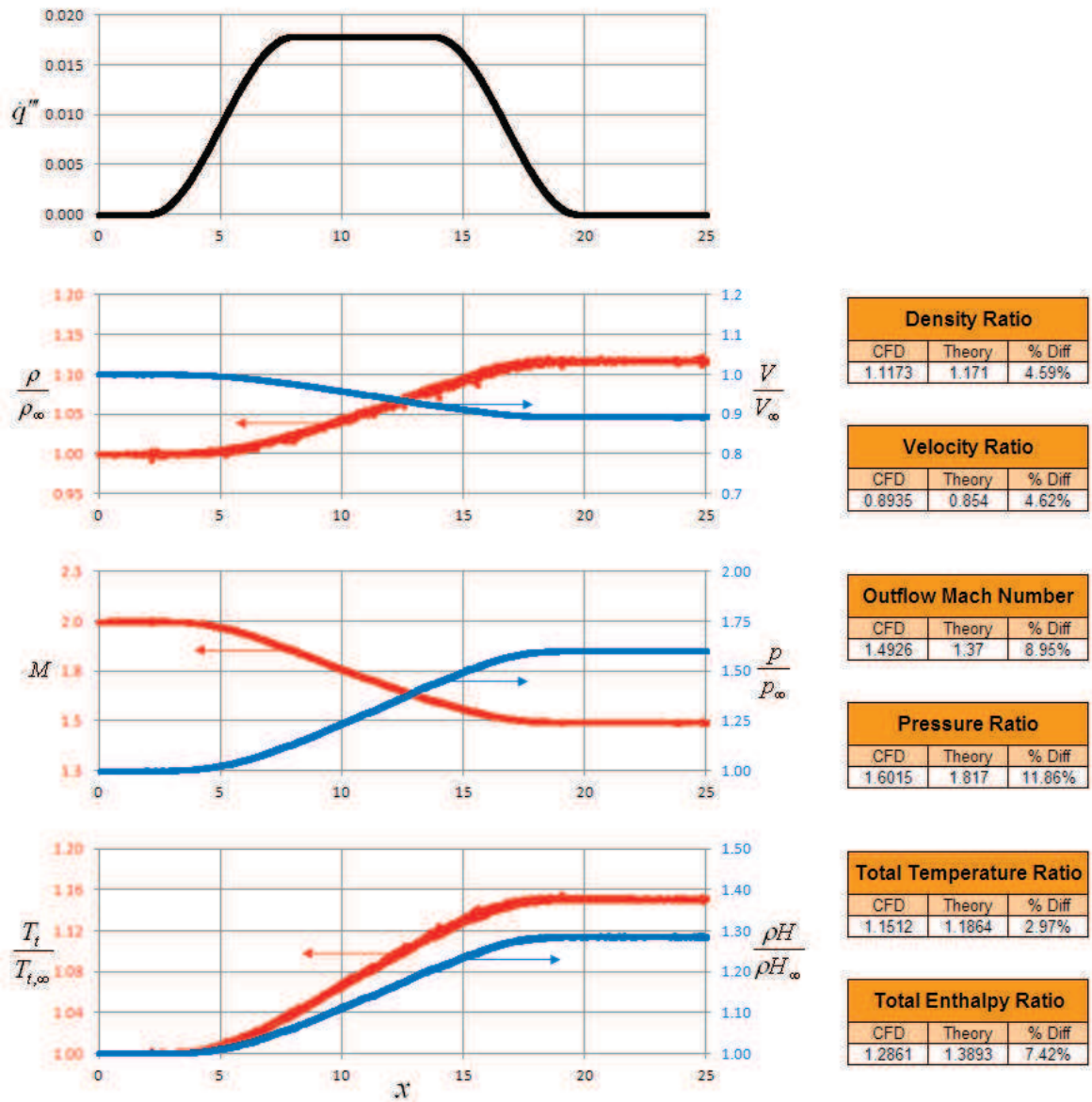


Figure 7.66: Supersonic (Mach 2.0) Cosine Heat Generation.



Figure 7.67: Supersonic (Mach 2.0) Constant Mass Generation.

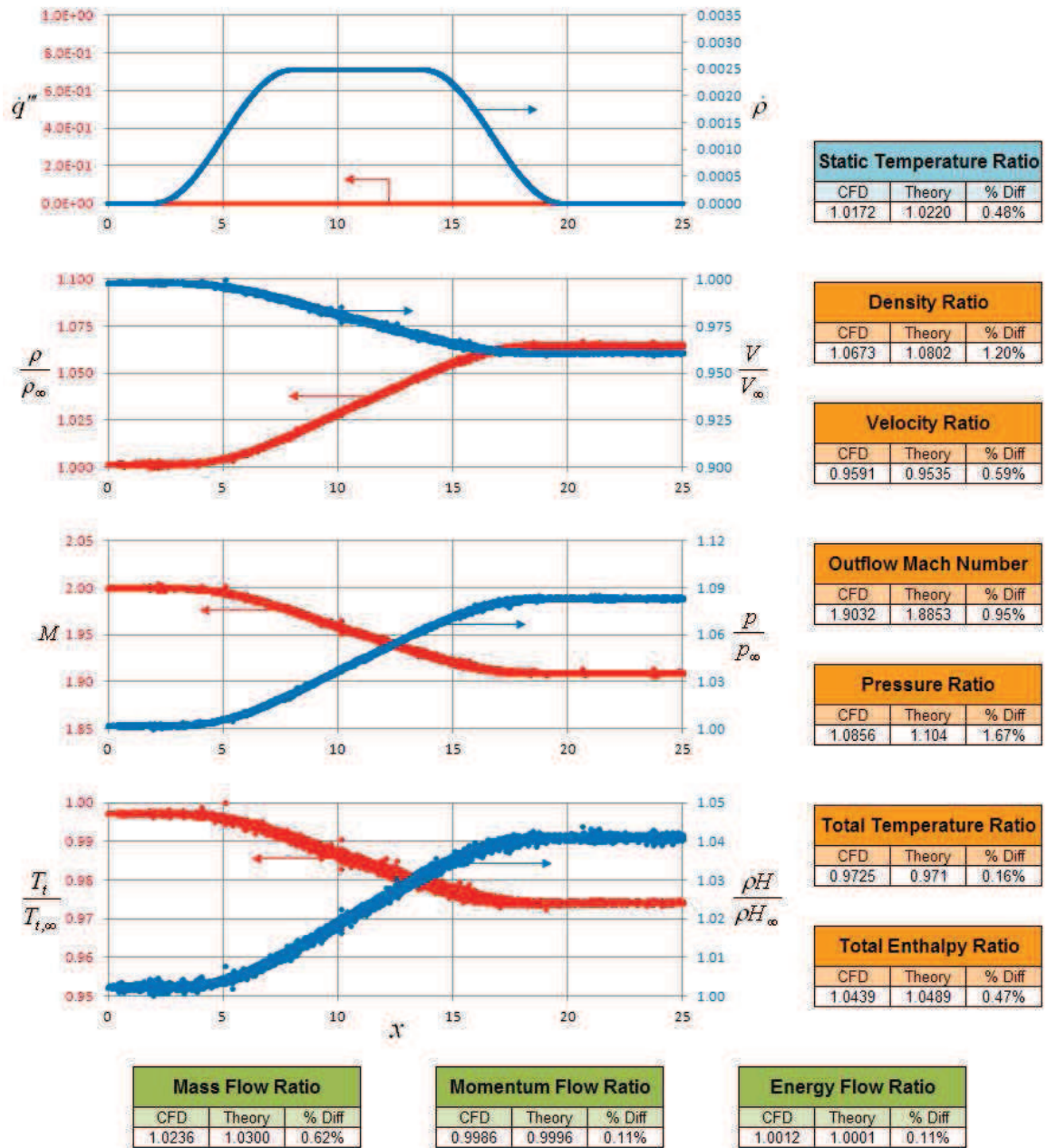


Figure 7.68: Supersonic (Mach 2.0) Cosine Mass Generation.



Figure 7.69: Supersonic (Mach 2.0) Constant Mass and Heat Generation.

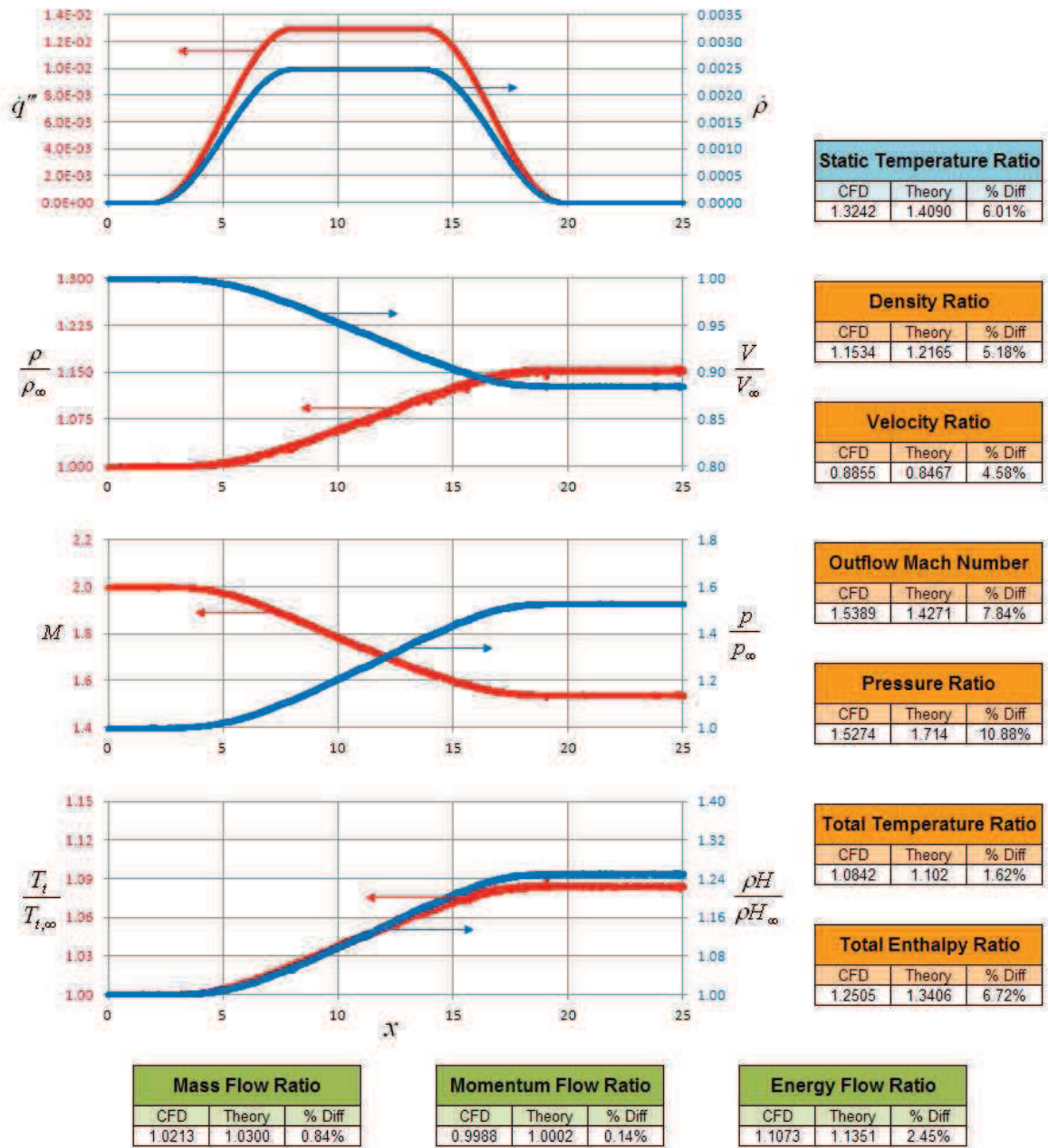


Figure 7.70: Supersonic (Mach 2.0) Cosine Mass and Heat Generation.

7.2.1.3 Hypersonic Linear Afterburner (Mach 7)

Heat was added to the flow in a supersonic combustor to simulate the effects of combustion. The outflow boundary conditions do not need to be modified because the characteristics are taken from upstream (domain elements). The hypersonic cases were built using the baseline CFDsol solver. The maximum error for the Mach 7 cases is 3.7%. Density fluctuations become excessive at hypersonic speeds.

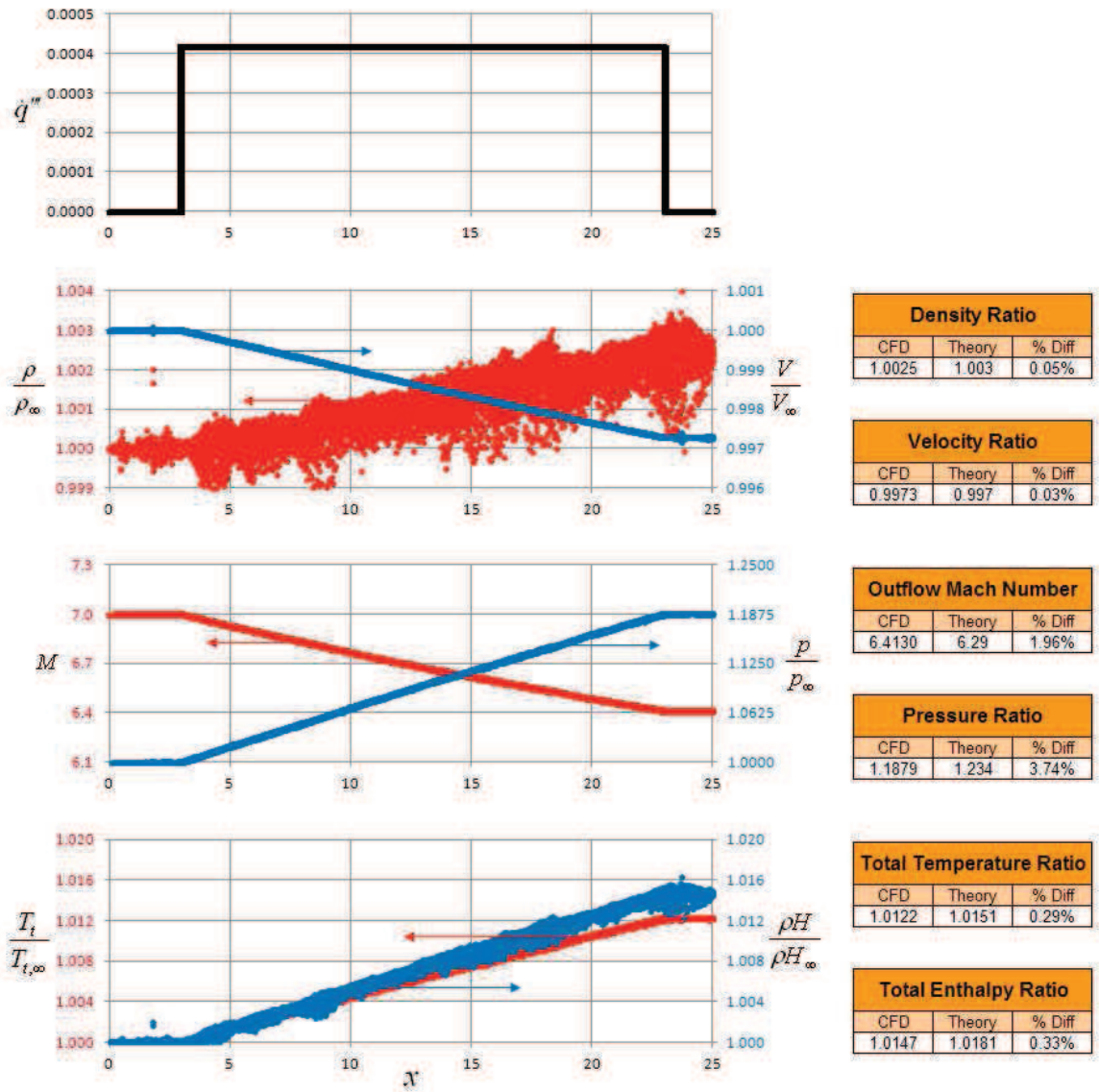


Figure 7.71: Supersonic (Mach 7.0) Constant Heat Generation.

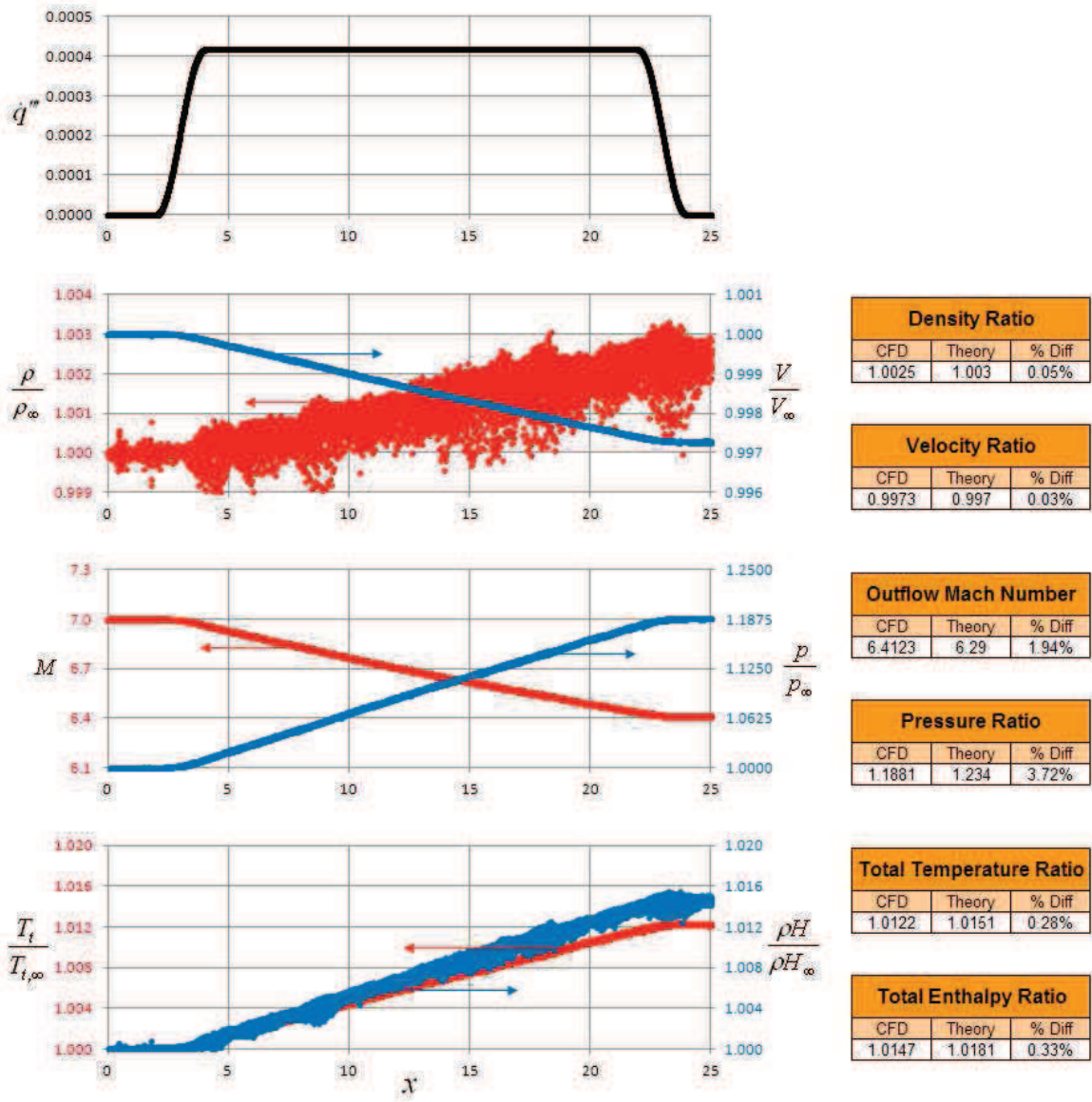


Figure 7.72: Supersonic (Mach 7.0) Cosine Heat Generation.

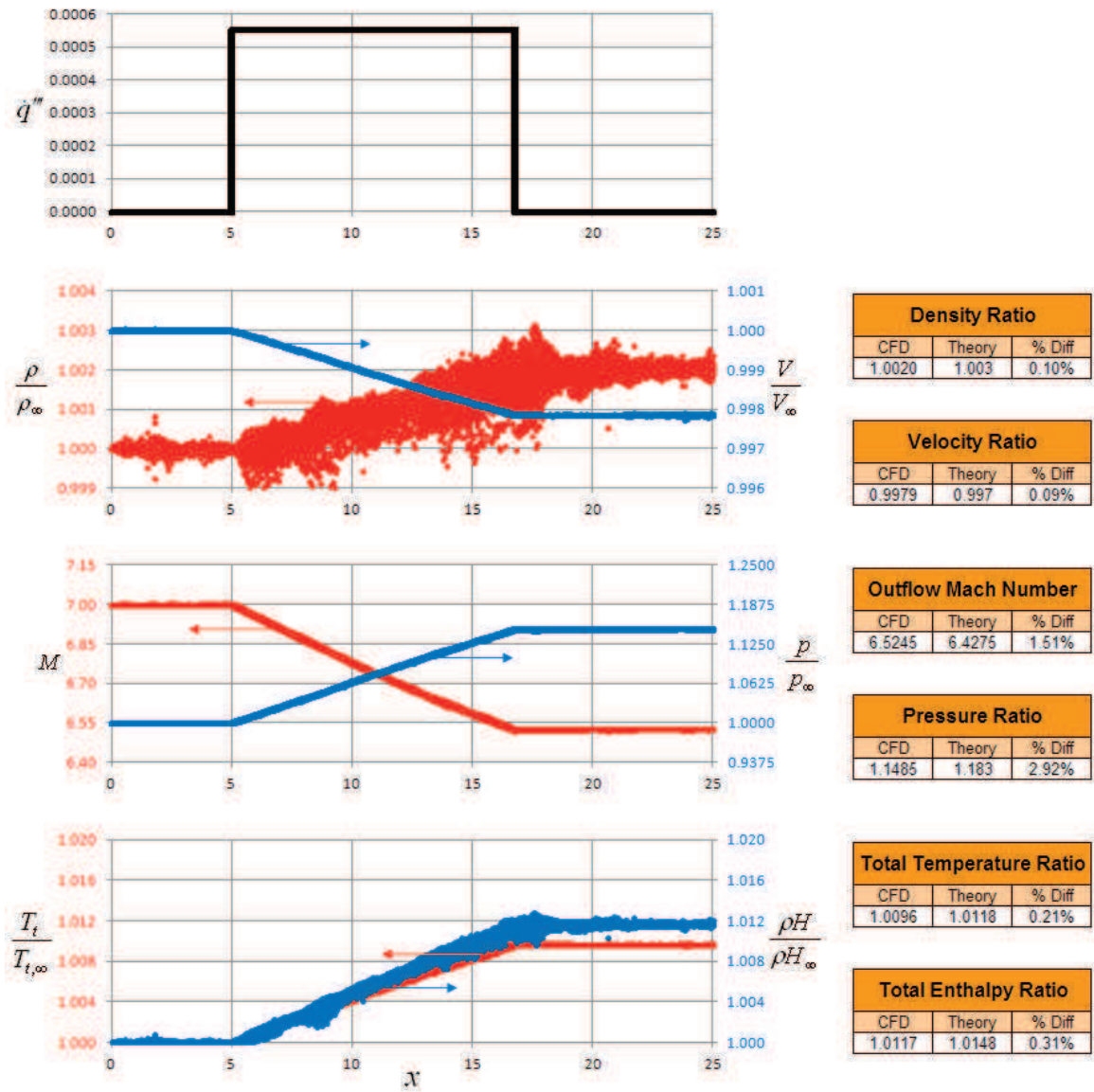


Figure 7.73: Supersonic (Mach 7.0) Constant Heat Generation.

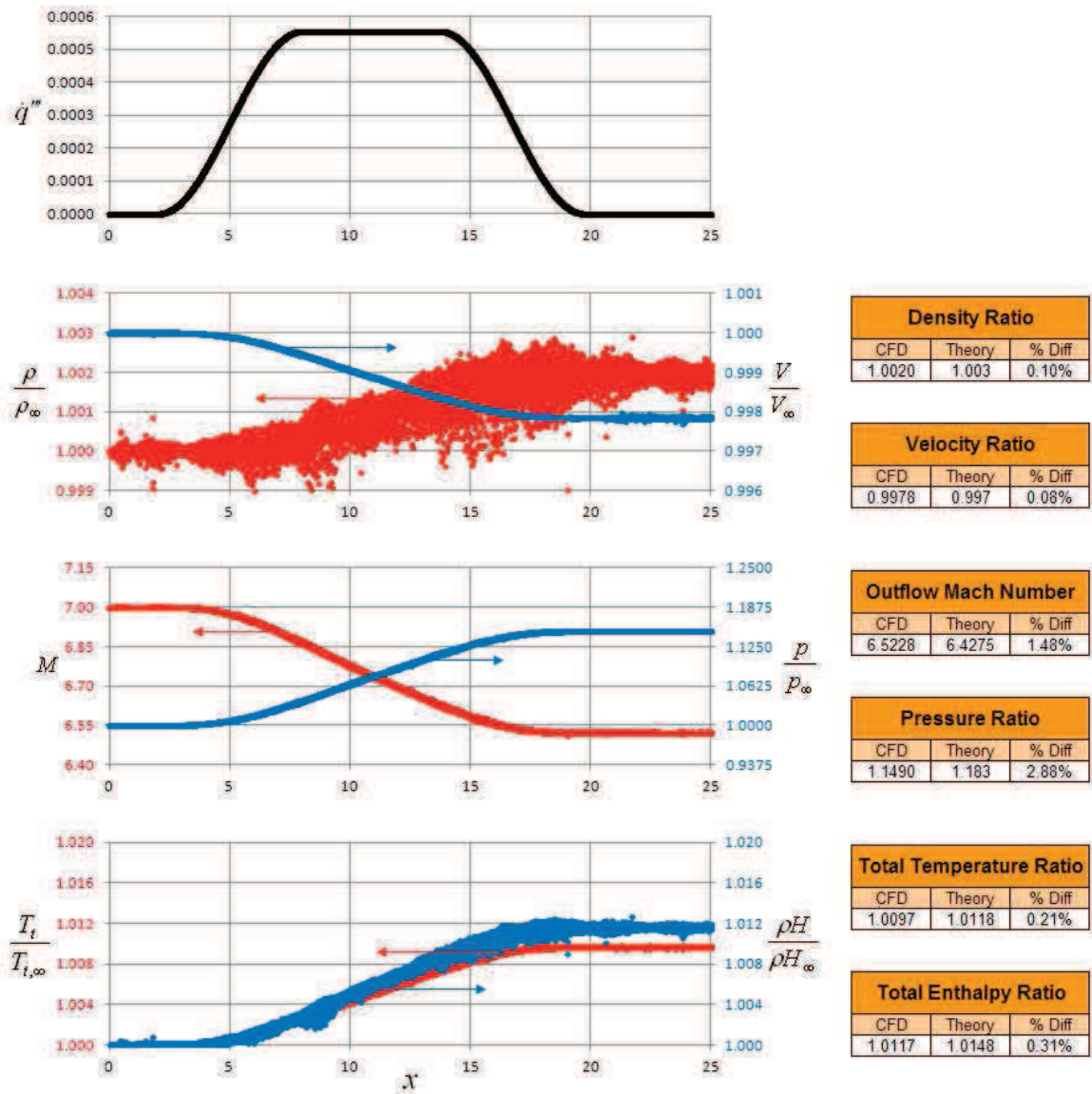


Figure 7.74: Supersonic (Mach 7.0) Cosine Heat Generation.

7.2.2 Rocket BC

A simple nozzle was designed using the method of characteristics (Emmons, 1958) to have a design Mach number of 2.2. The nozzle geometry can be seen in Figure 7.75. A sharp corner at the throat forms an expansion fan to help maintain isentropic flow at the design condition. The nozzle was tested at three back pressures: 84, 90, and 100% of the design pressure. Below the design condition, a normal shock appears in the diverging section. The position of the normal shock in Figure 7.76 is upstream of that predicted by theory because the solution has not fully converged. Shortly after the cuts shown in Figure 7.76 were made, the solution diverges along the shock plane. The properties in Figure 7.76 just upstream of the shock plane become unstable; the Mach number jumps up before dropping through the shock. Increasing the artificial dissipation *FACTOR* and decreasing the relaxation factor *tau* (or time step *dtfix*) did not help the problem. Refining the mesh within this problem region only exacerbated the divergence. These are inherent instability problems with the implementation of CFDsol.

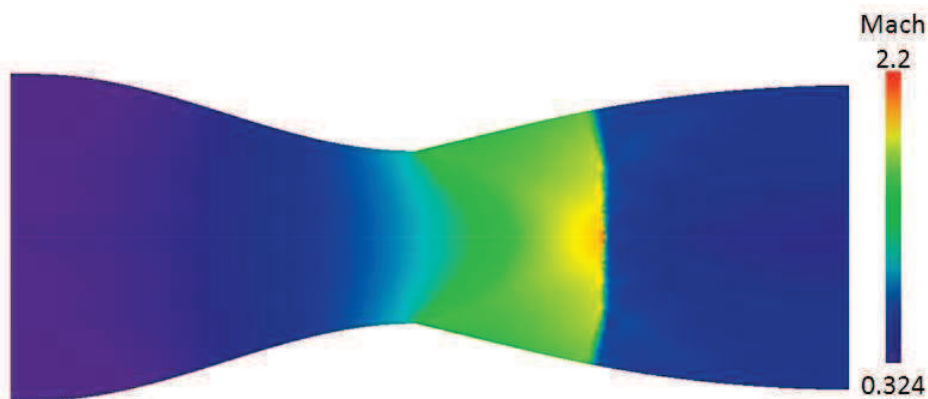


Figure 7.75: Local Mach Number within Rocket Nozzle at $p_t = 1.67 p_{inf} = 0.84 p_{design}$.

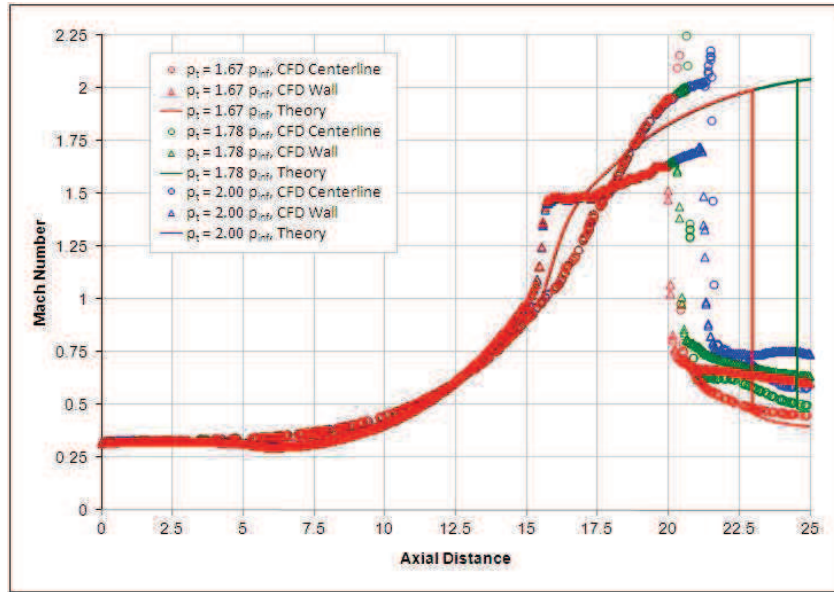


Figure 7.76: Local Mach Number along Centerline and Wall Surface of Rocket Nozzle.

7.3 Viscous Aerodynamics

Viscous flows are divided into two groups here: Laminar and turbulent. An incompressible boundary layer, circular cylinder, and ellipse are demonstrated for the laminar flows. The incompressible boundary layer was used to validate the viscous terms in CFDsol before moving forward. An incompressible turbulent boundary layer is demonstrated for higher Reynolds number flows, showing the capabilities of the new Spalart-Allmaras model.

7.3.1 Laminar

Four cases are presented here to demonstrate how well CFDsol represents laminar flows. An incompressible laminar boundary layer was used as an initial validation of the viscous terms. Because the artificial dissipation plays such a dominant role in near-wall flows modeled in CFDsol, the viscous terms were first verified by showing that the effective Reynolds number

of the boundary layer approaches the actual Reynolds number as artificial dissipation is removed from the solution. The viscous terms were then substituted into NS3D for a final check.

7.3.1.1 Laminar (Incomp) Boundary Layer

A flat plate boundary layer is demonstrated at a moderate (laminar) Reynolds number ($Re = 3600$, Mach 0.3). A mesh with medium-refinement is shown in Figure 7.77 for the laminar boundary layer case: The bottom half of the boundary layer is spanned by three elements; the outer mesh grows sparser in the external flow. Large elements are purposefully used in the external flow so that the boundary layer is not dependent on the external flow, which is near the free-stream velocity. The element spacing varies along the length of the plate according to the theoretical boundary layer thickness.

Figure 7.78 shows the results from CFDsol for the laminar flat plate boundary layer. The dimensionless velocity profile (Figure 7.78a) compares the velocity profiles at 100 locations along the plate to Blasius' solution. The profile shows a near linear section near the wall and curvature transitioning to the external flow. The gradient near the wall is lessened due to the presence of artificial dissipation, which tries to eliminate high gradients in the field. The lower wall gradient results in a lower skin friction along the length of the plate. Figure 7.78b shows the skin friction to be approximately 50% of that predicted by theory. The lower gradient also thickens the boundary layer (Figure 7.78c). The total percent error for each of the 100 cuts is shown along the length of the plate in Figure 7.78d (6 to 20%).

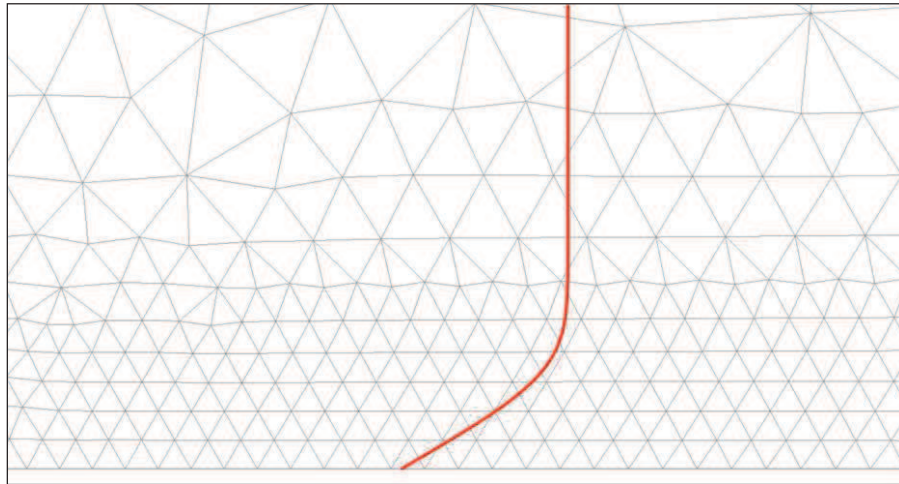


Figure 7.77: Moderate Mesh for Laminar Boundary Layer ($Re = 3600$).

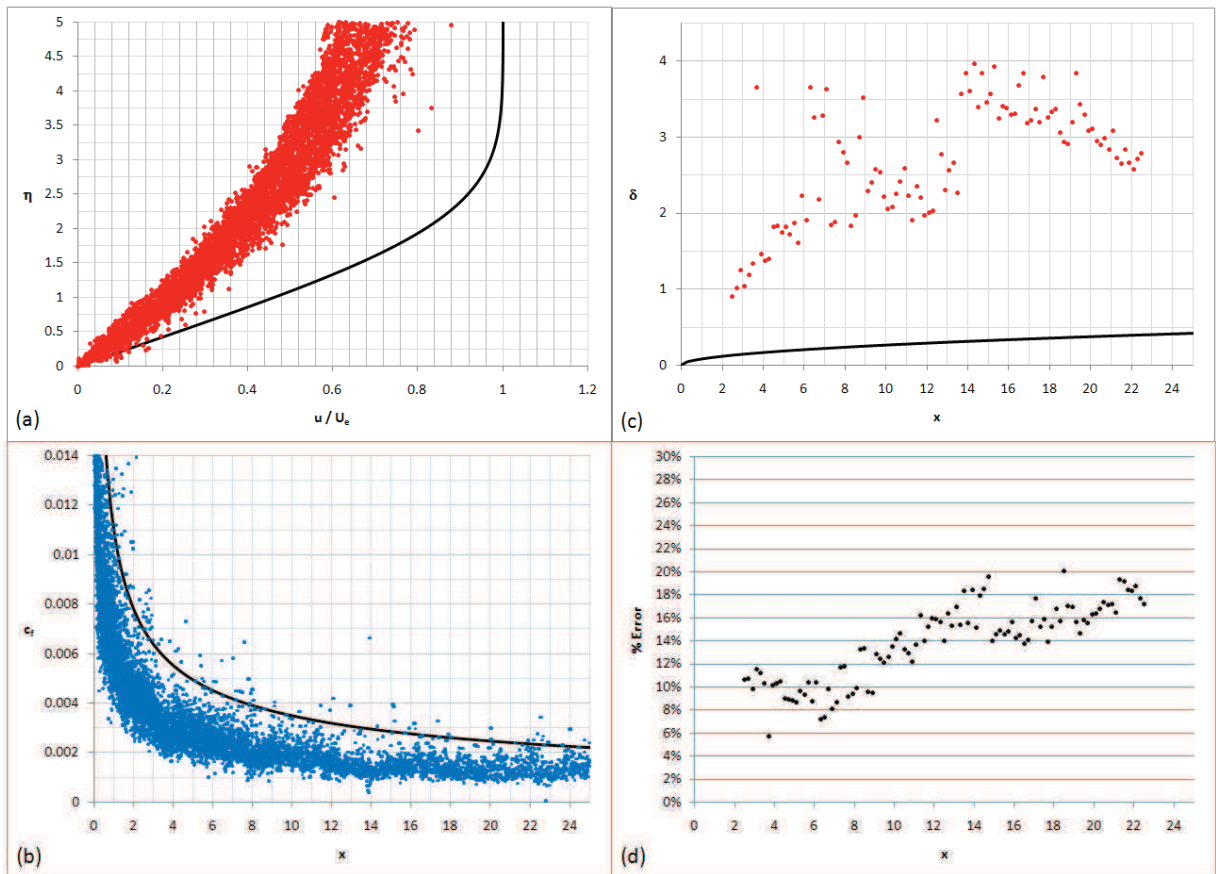


Figure 7.78: Laminar Boundary Layer Results from CFDsol:
(a) Dimensionless Boundary Layer Profile; (b) Skin Friction along Plate;
(c) Boundary Layer Thickness along Plate; (d) Percent Error in Profile along Plate.

Several meshes were used in attempts to approach the analytical solution. Figure 7.79 shows a coarse and fine mesh. The coarse mesh has half as many elements in the lower 50% of the boundary layer. In both meshes the elements increase in size in the external flow. The velocity profiles for these two meshes are also shown in Figure 7.79. Compared to the moderate mesh in Figure 7.78, the course mesh has up to 30% error while the fine mesh has less than 18% error. The skin friction and boundary layer thickness predicted by the finer mesh is also improved.

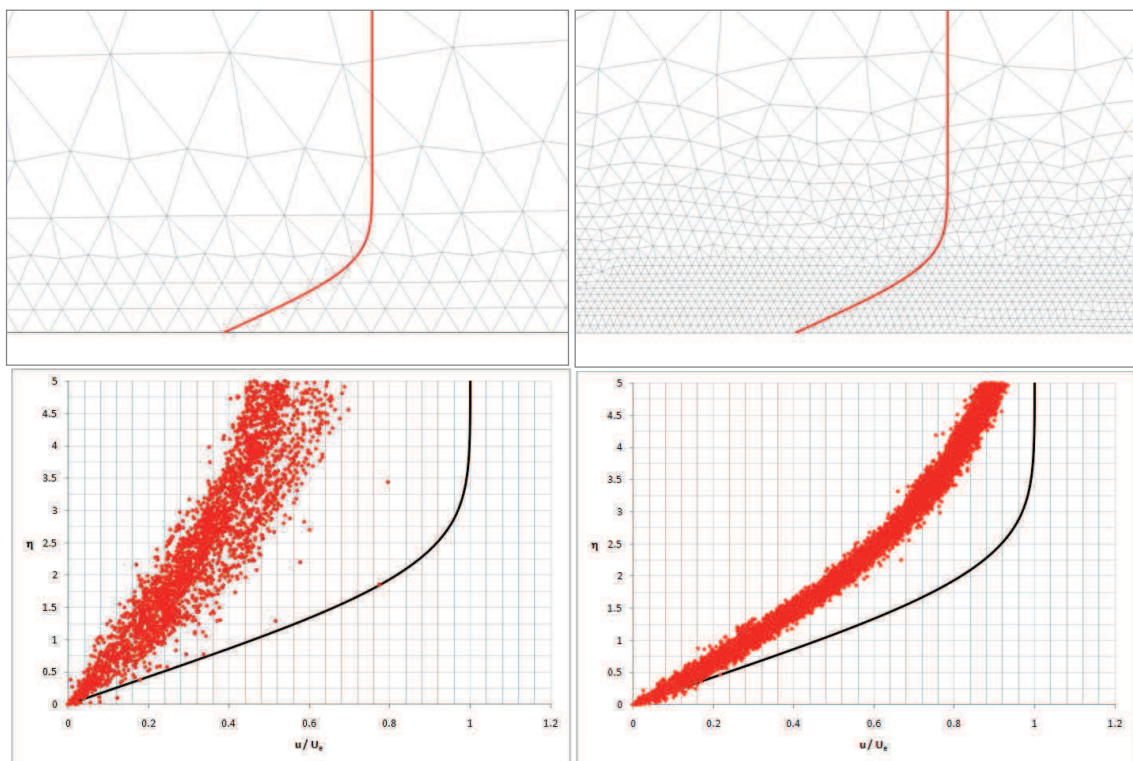


Figure 7.79: Course (Left) and Fine (Right) Meshes and Dim'less Velocity Profiles.

The artificial dissipation is necessary in the external flow for capturing shocks and other flow features. Within the boundary layer, the opposite is true. Artificial dissipation reduces the skin friction, thickens the boundary layer, and adds error to the near wall solution. We desire to have artificial dissipation in the external flow but no extra dissipation in the boundary

layer, especially near the wall. To demonstrate that the viscous equations are modeling the boundary layer correctly, aside from the influences of artificial dissipation, an effective Reynolds number Re_{eff} was calculated for the mesh and dissipation *FACTOR*.

Blasius' solution is plotted according to the similarity distance η from the wall:

$$\eta = y\sqrt{\frac{Re}{2x}}$$

This distance can be scaled, creating an effective velocity profile:

$$\eta_{eff} = y\sqrt{\frac{Re_{eff}}{2x}} = y\sqrt{\frac{Re}{2x}}\sqrt{\frac{Re_{eff}}{Re}} = \eta\sqrt{\frac{Re_{eff}}{Re}}$$

The effective Reynolds number Re_{eff} represents the velocity profile given in the CFD data, chosen to best fit the velocities near the wall. Data from CFDsol and NS3D was compared for the three meshes and various dissipation scalars. CFDsol utilizes the dissipation model scaled by *FACTOR* outlined in previous sections, while NS3D applies a segment-weighted dissipation model scaled by *diss*. The results are shown in Figure 7.80. The two dissipation models create different magnitudes in the effective Reynolds number with similar trends: The effective Reynolds number decreases with the amount of artificial dissipation used, by diminishing amounts; and, as the artificial dissipation approaches zero, the effective Reynolds number approaches the actual Reynolds number used in the solution.

As a repetition, the viscous terms from CFDsol were transferred to NS3D, using the inviscid terms and artificial dissipation model in NS3D. The effective Reynolds number matches that created by the original terms (shown as triangles in Figure 7.80). The repetition shows that the viscous terms in CFDsol are modeling the laminar boundary layer with good accuracy

while the artificial dissipation in either solver increases Re_{eff} at the bottom of the boundary layer. For this reason, the least amount of artificial dissipation should be used for any viscous solution. The results also show that as Re_{eff} approaches the actual Re , the error is minimized if three or more elements are used to discretize the lower half of the boundary layer. Four elements in the lower 60% is suggested as a precaution.

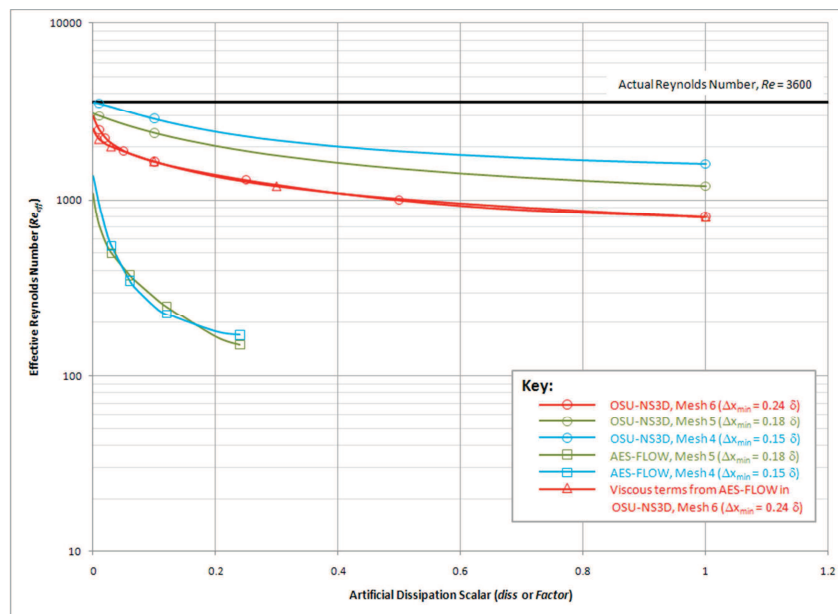


Figure 7.80: Effective Reynolds Number vs. Artificial Dissipation Scalar.

7.3.1.2 Ellipse

An ellipse with 6:1 ratio was tested under viscous conditions, representing a more complex geometry resembling the leading edge of airfoils. The inviscid mesh was generated using three mesh sources so that the mesh was finer at the leading and trailing edges. The spacing was chosen to resemble the radius of curvature of the ellipse surface. The viscous mesh in Figure 7.12 was generated by adding a boundary layer refinement to the inviscid mesh, using the laminar guidelines. The pressure and velocity near the viscous ellipse ($Re = 4000$) are

shown in Figure 7.81. The surface pressure distributions for both the inviscid and viscous ellipses are shown in Figure 7.13. The two pressure distributions are similar, on average, but the viscous pressure distribution is much noisier. The noise is created by variations in velocity (kinetic energy) that feeds back into the pressure distribution.

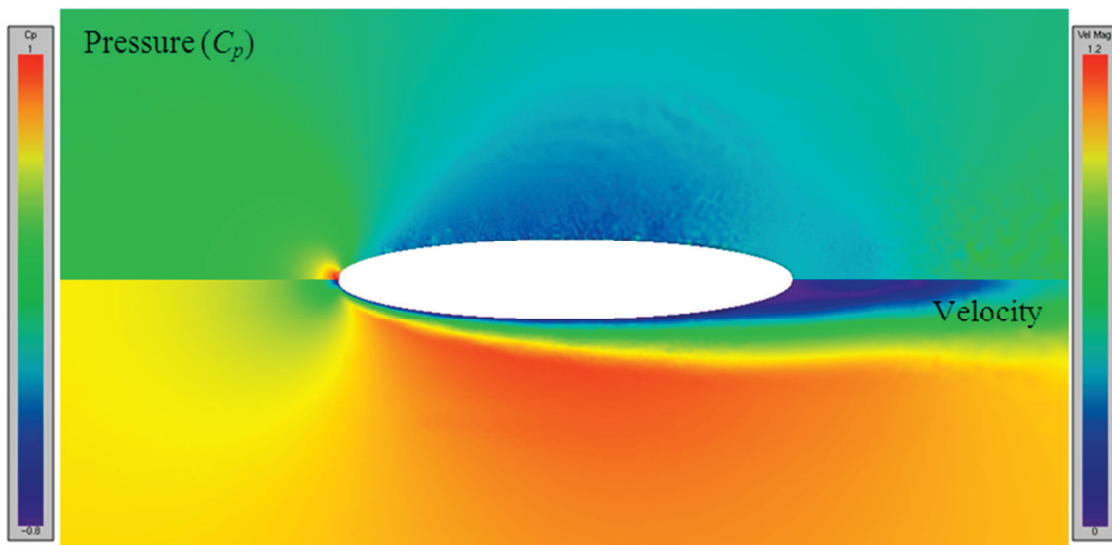


Figure 7.81: Pressure (top) and Velocity (bottom) around Viscous Ellipse ($Re = 4000$).

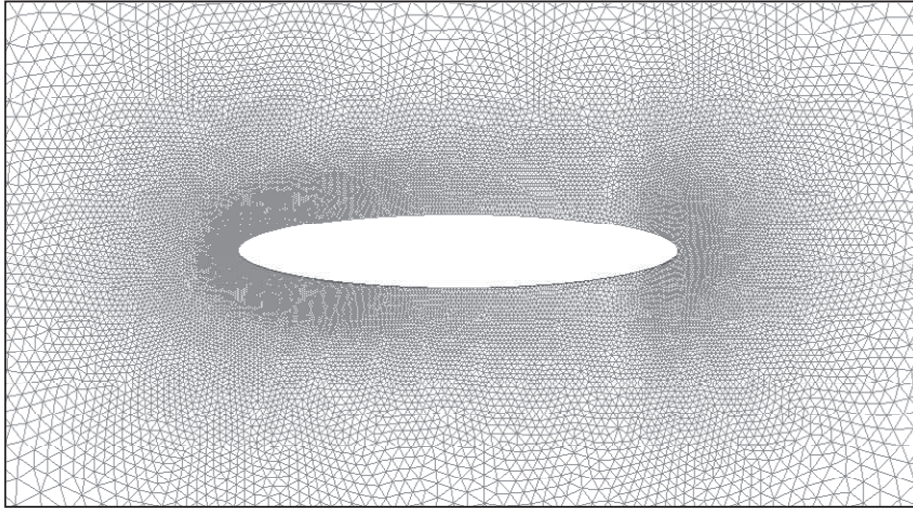


Figure 7.82: Ellipse Mesh.

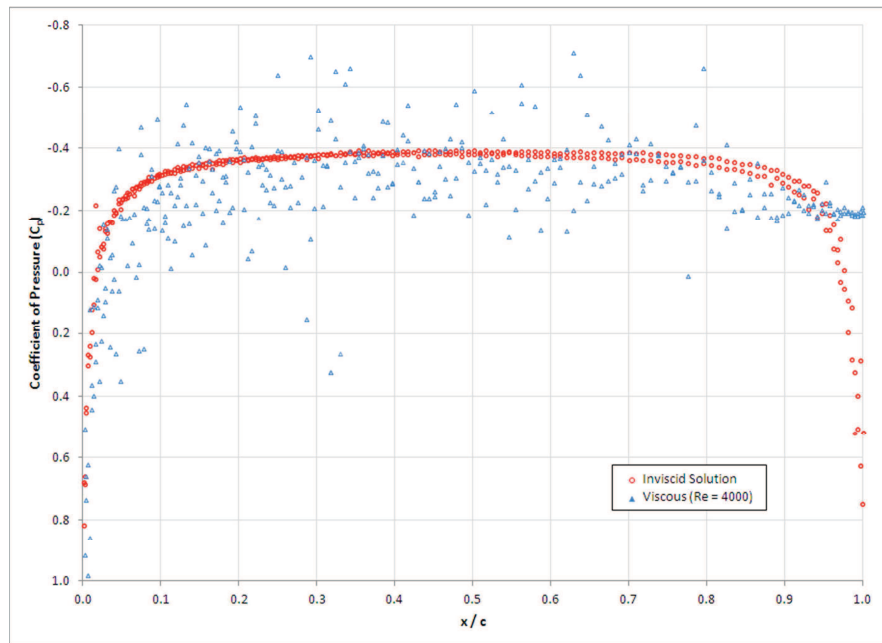


Figure 7.83: Pressure over Inviscid (left) & Viscous (right) Ellipse (Mach 0.3).

7.3.1.3 Cylinder

The cylinder was tested at two low Reynolds numbers: $Re_D = 9.6$ and 41. The velocity and pressure distributions are shown in Figure 7.85 and Figure 7.86 for Reynolds numbers of 9.6 and 41, respectively. These distributions were calculated using the meshes shown in Figure 7.84 and Figure 7.87. Each mesh was generated, starting with the inviscid mesh and adding viscous refinement. For $Re_D = 41$, the boundary layer thickness was estimated using a crude mesh. Four elements were spaced evenly across the lower 60% of the boundary layer. The mesh spacing was allowed to double at the top of the boundary layer. For $Re_D = 9.6$, the inviscid mesh was refined progressing until the viscous solution converged.

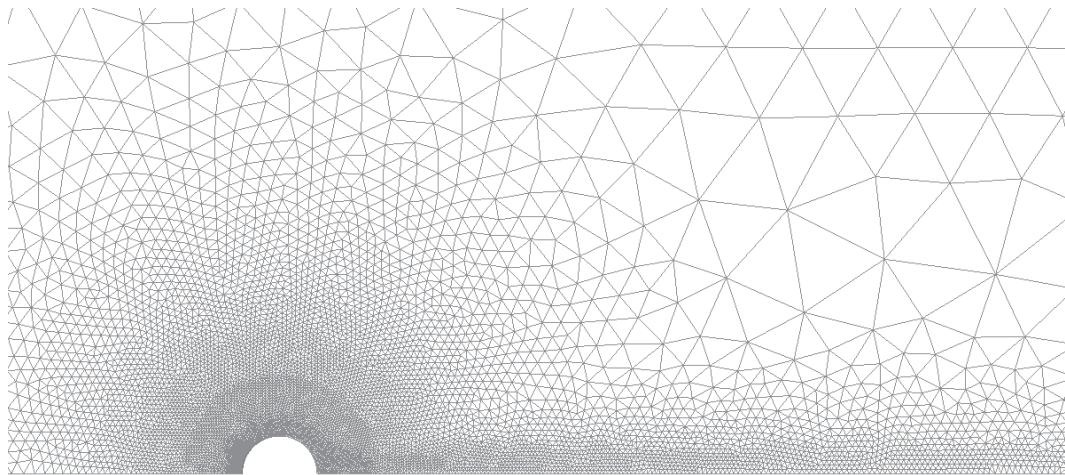


Figure 7.84: Viscous Cylinder Mesh ($Re_D = 9.6$).

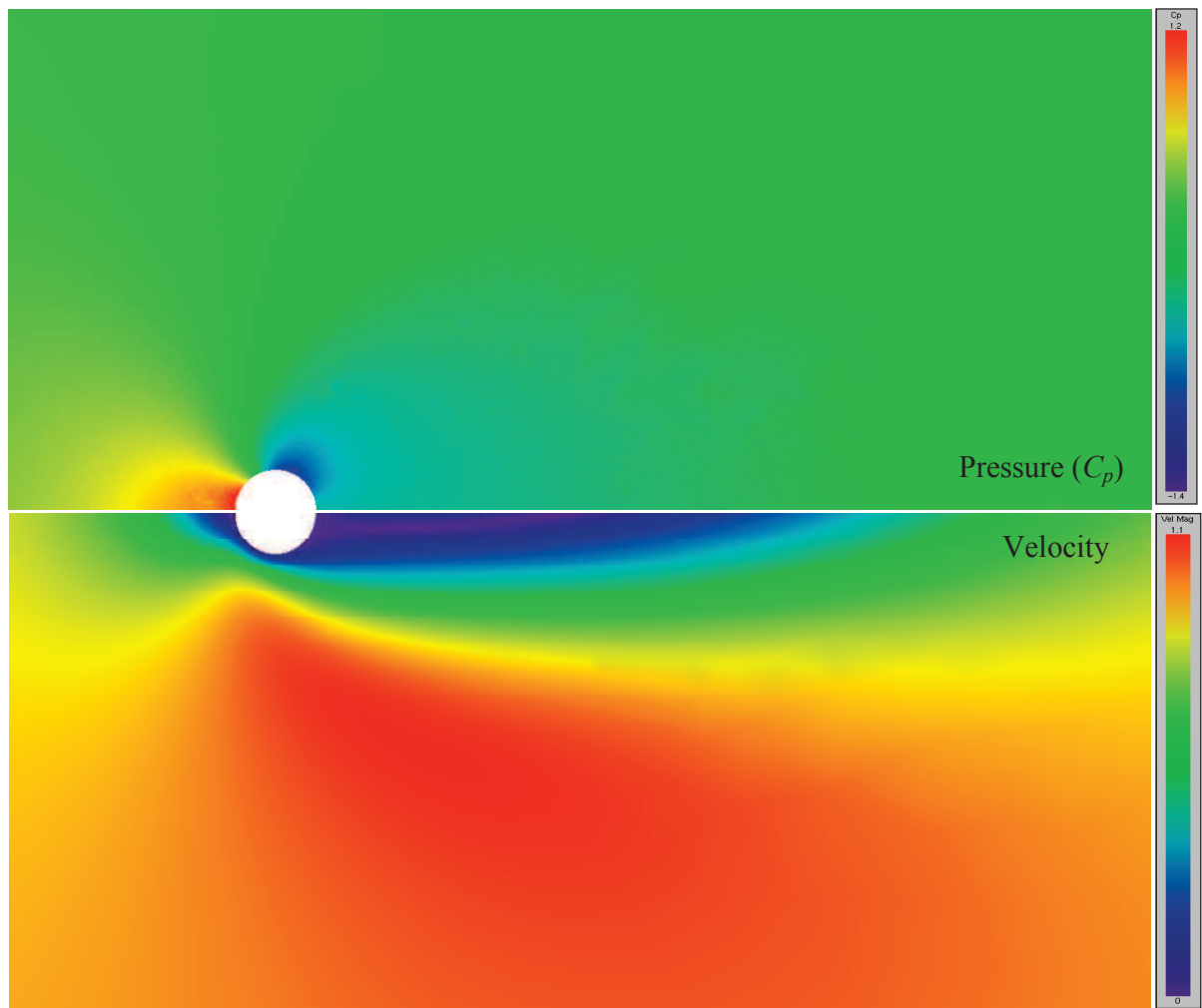


Figure 7.85: Pressure (top) and Velocity (bottom) around Viscous Cylinder ($Re_D = 9.6$).

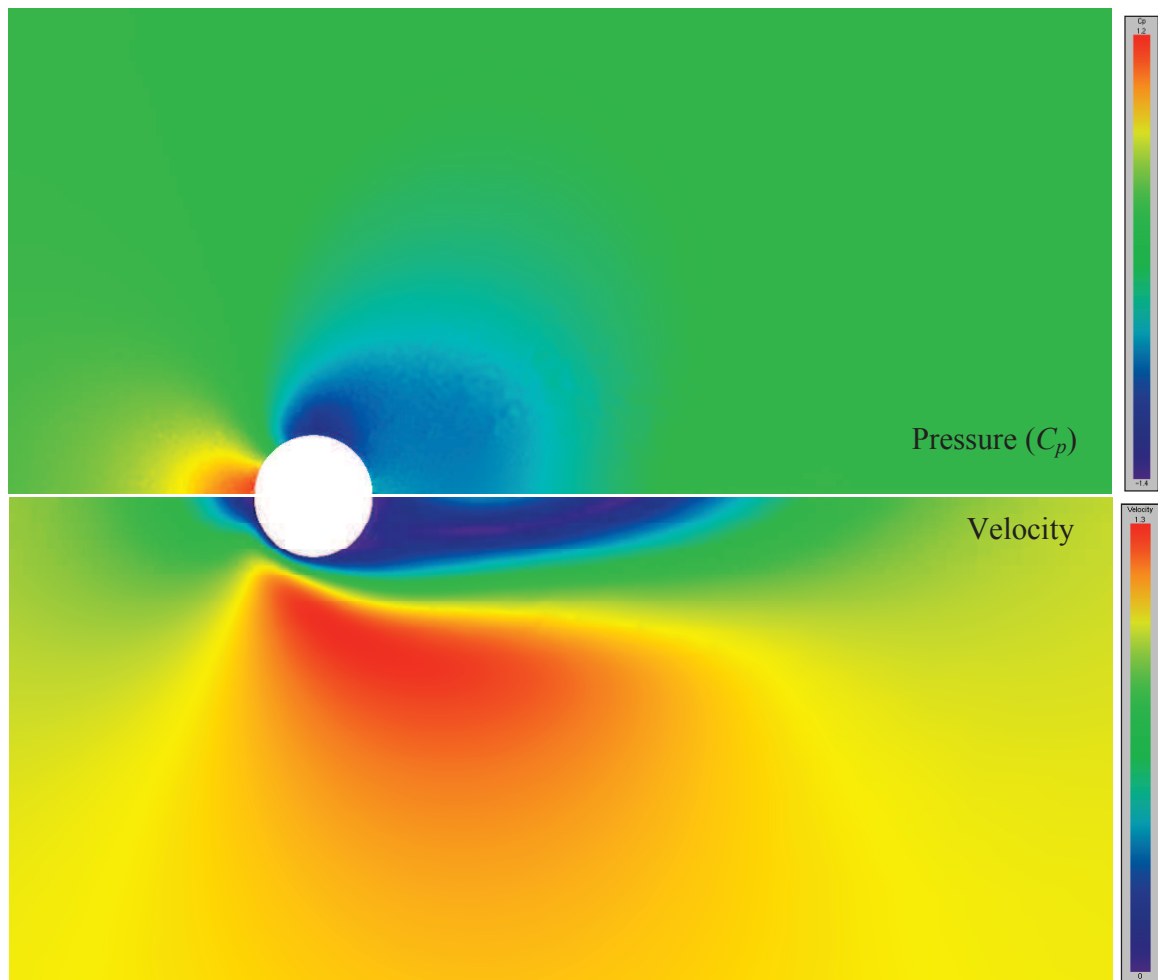


Figure 7.86: Pressure (top) and Velocity (bottom) around Viscous Cylinder ($Re_D = 41$).

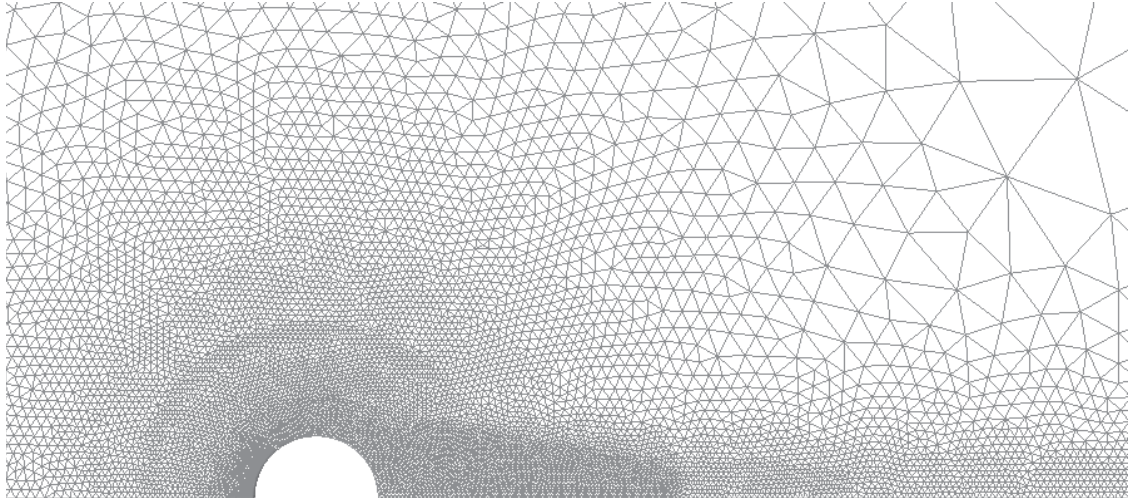


Figure 7.87: Viscous Cylinder Mesh ($Re_D = 41$).

7.3.2 Turbulent

Two turbulent cases are demonstrated here. The first case was used to demonstrate the effectiveness of the Spalart-Allmaras model to model a turbulent boundary layer in a fully developed turbulent section. The second case models a boundary layer from leading edge (laminar) through transition to fully developed turbulence. The growth of turbulence is demonstrated over the length of the plate.

7.3.2.1 Turbulent Section

The first case demonstrates how well the SA model works for a fully developed turbulent boundary layer. A section of length 7 units was tested 15 units from the leading edge of the plate. The Reynolds number Re_x ranged from 3.45×10^6 to 5.06×10^6 across the section, representing a fully developed boundary layer at the inflow plane. The inflow and outflow planes were specified using the velocity and eddy viscosity profiles outlined in the two-dimensional viscous solutions. The resulting solution, shown in Figure 7.88 and Figure 7.89, demon-

states that the SA model is stable for a fully developed turbulent boundary layer. Figure 7.89 shows the eddy viscosity profile at three locations evenly spaced across the section, which compare well with the inflow and outflow profiles.

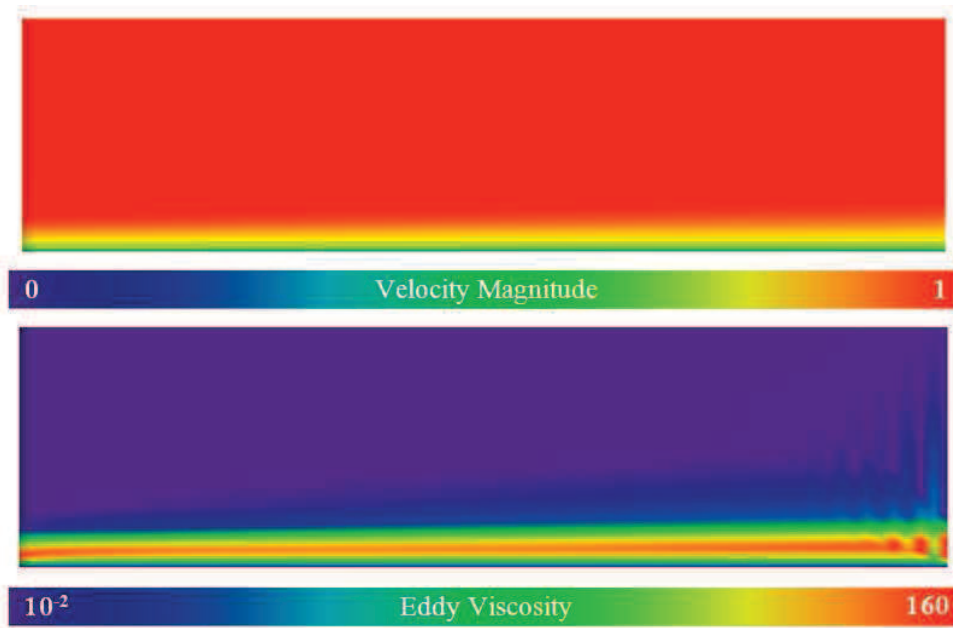


Figure 7.88: Velocity (top) and Eddy Viscosity (bottom) for Turbulent Section.

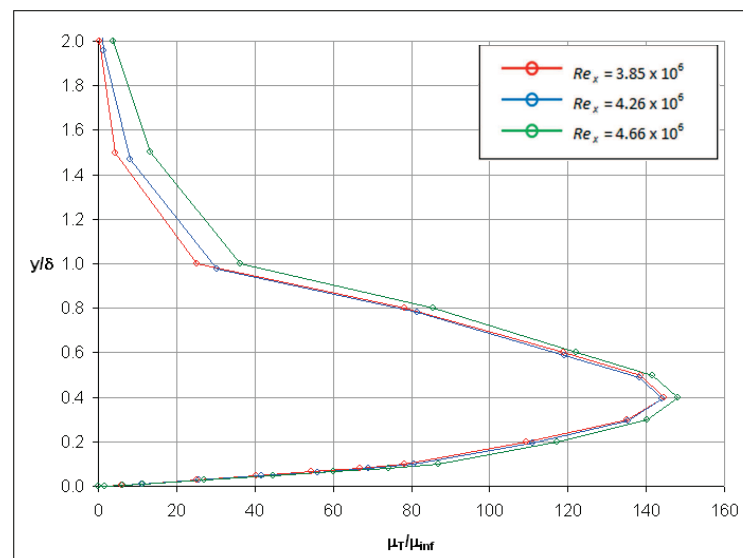


Figure 7.89: Eddy Viscosity Profiles at Quarter Locations across Turbulent Section.

7.3.2.2 Incompressible Turbulent Boundary Layer

The second case demonstrates how turbulence, in the form of μ_T , grows along a transitional boundary layer. A plate of length 1605 units was tested from leading to trailing edges, where $Re_x = 6.4 \times 10^6$ at the trailing edge of the plate. (To minimize run time, the laminar and turbulent sections were started from theoretical profiles, and the transitional section was Interpolated using a laminar profile.) The solution was expected to grow a laminar boundary layer over the upstream 3% of the plate. The remaining solution should begin to transition to turbulence, growing from near-zero to a substantial amount of turbulence. Figure 7.90 shows the velocity and eddy viscosity distributions along the plate. The eddy viscosity grows along the length of the plate, driven strongly by the advection properties near the wall, not purely by the source terms. The eddy viscosity profile grows in parallel with the velocity profile, as expected, since the SA source term is a function of the strain in the flow, which is established by the velocity profile.

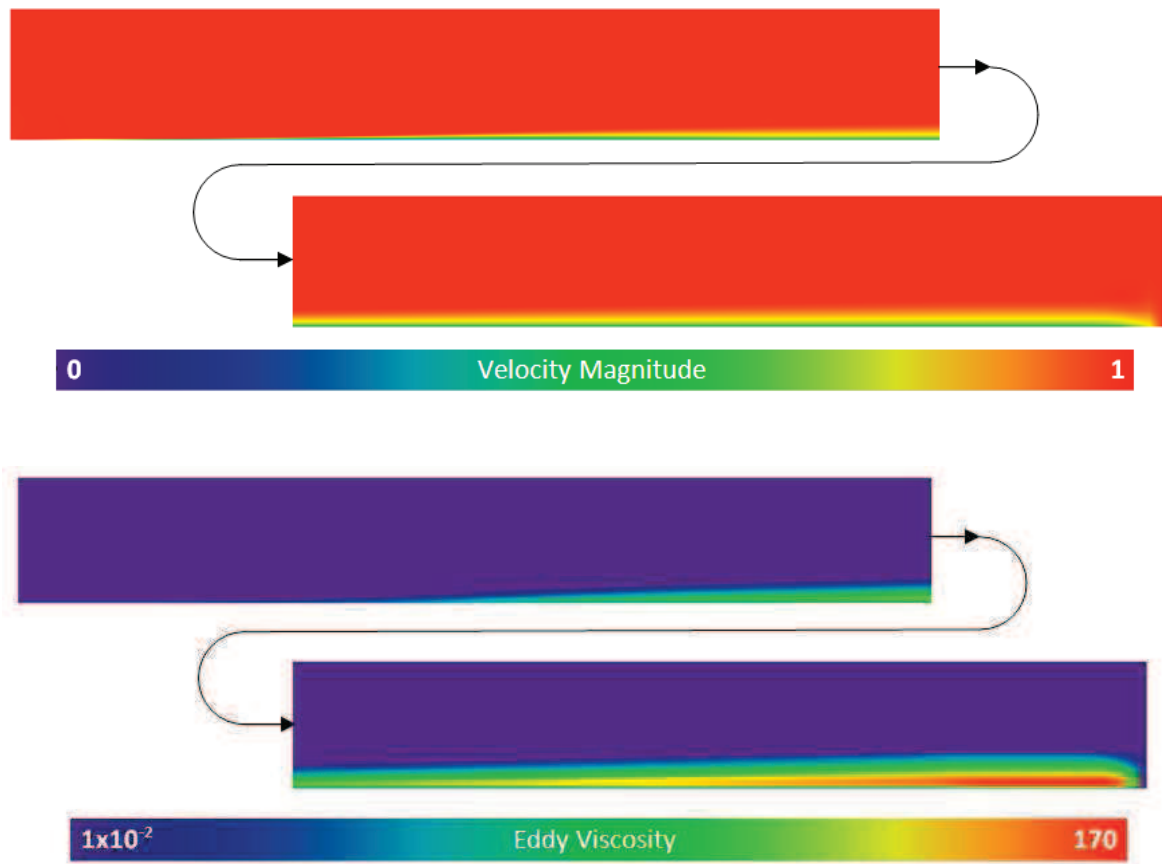


Figure 7.90: Velocity (Top) and Eddy Viscosity (Bottom) for Trans. Boundary Layer.

Five velocity profiles along the plate are shown in Figure 7.91. The boundary layer shows transition from laminar to fully turbulent across the length of the plate. The distribution of eddy viscosity across the boundary layer is shown at the same five locations (also in Figure 7.91). The fully-developed eddy viscosity profile takes on the shape of the theoretical profile illustrated in the two-dimensional viscous solutions. The maximum value of eddy viscosity $\mu_{T,max}$ in the profile grows along the length of the plate, as shown in Figure 7.92.

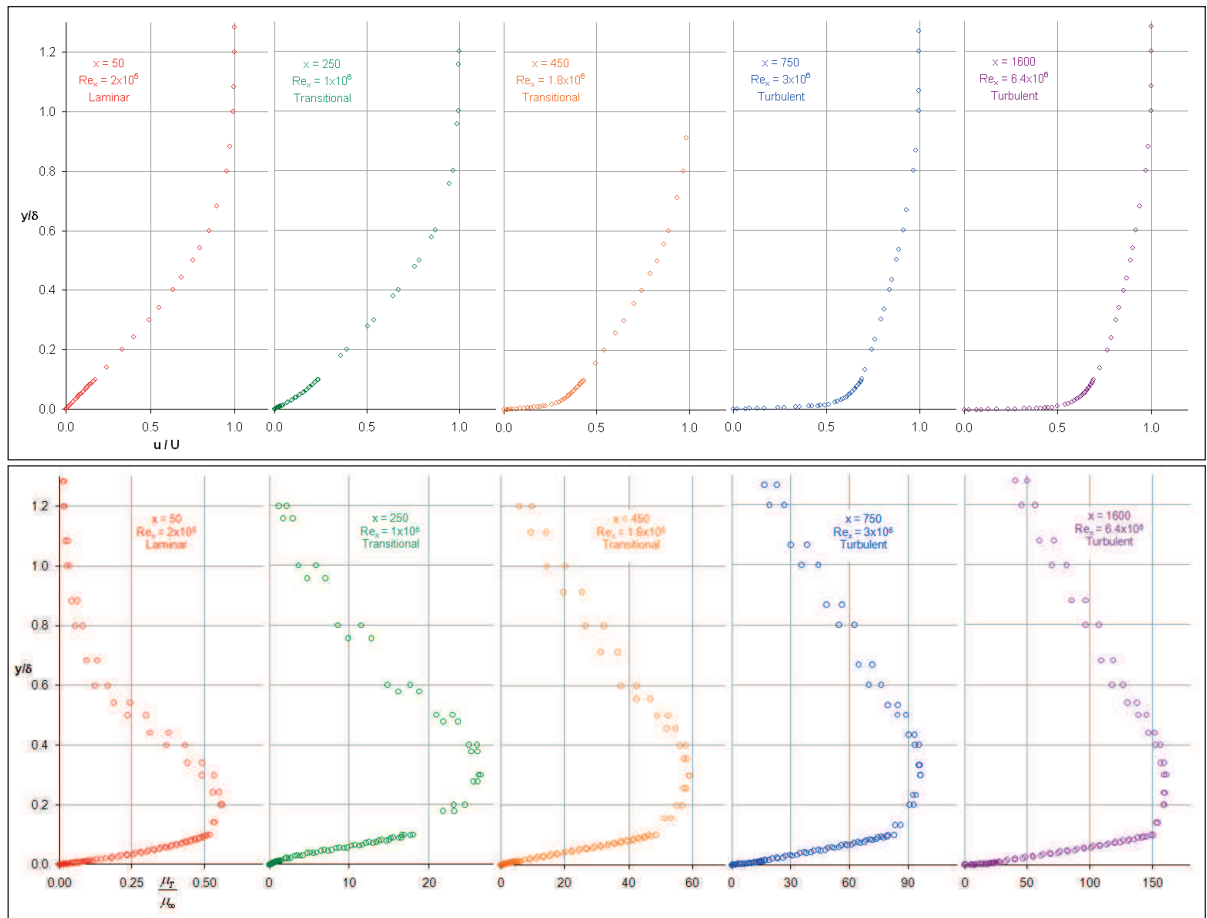


Figure 7.91: Velocity (top) and Eddy Viscosity (bottom) Profiles along a Flat Plate Boundary Layer.

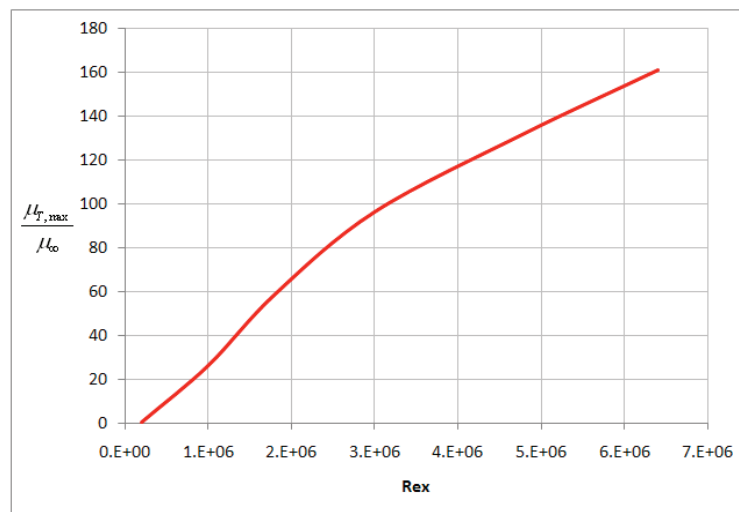


Figure 7.92: Growth of Turbulence (Eddy Viscosity) along Length of Plate.

CHAPTER VIII

CONCLUSIONS

8.1 Conclusions

The conclusions of this work are given in five parts: The objectives from Chapter 2 are revisited and their completion addressed. The five solvers are evaluated using the test data presented in the previous two chapters. Many pitfalls were encountered during the course of this work. The most important (and most memorable) of these pitfalls are discussed here as a warning to future users. Standard practices are also discussed to guide future users through the difficult decisions of mesh spacing, time step size, and selection of other controls. Five different solvers were used during this work. Converting between those solvers is often confusing and requires several support codes to handle the process.

8.1.1 Objectives

Five objectives were outlined in Chapter 2. These objectives are revisited here along with their progress and completion:

Objective 1. The two-dimensional OSU codes Euler2D and NS2D were enhanced to include propulsion models, viscous terms, and turbulence models. Mass and heat generation were

added to the governing equations in Euler2D to create quasi-combustion terms. A rocket exhaust model and coupled turbojet boundaries were implemented in Euler2D. Acoustic output files were also created in Euler2D to study the properties at specified locations in the field. Viscous terms were added to Euler2D, which was renamed NS2D. Viscous local time stepping was implemented in NS2D to increase stability with the added viscous terms.

Compressible Spalart-Allmarus (SA) and Menter's SST models were implemented in NS2D along with rotation correction terms to adapt the production of turbulence in the presence of non-inertial rotation. Heat transfer boundary conditions were also implemented in NS2D.

Objective 2. The three-dimensional OSU codes Euler3D and NS3D were enhanced using Euler2D and NS2D as a proof of concept and fundamental basis. Quasi-combustion terms were added through mass and heat source terms. The rocket and turbojet boundary conditions were applied to inflow and outflow boundaries. Acoustic output files were created in Euler3D using similar interpolations found in Euler2D. Viscous terms were added to Euler3D, which was renamed NS3D. The viscous local time steps were expanded to 3D using NS2D as a pattern. Compressible SA and SST models were implemented in NS3D by expanding the two-dimensional models to include the additional terms and indices of the three-dimensional code. Heat transfer conditions were implemented in NS3D to mirror those in NS2D. All four codes were implemented using similar variable and array names to reduce the workload of future researchers and facilitate transfer of models between the OSU solvers.

Objective 3. CFDsol was enhanced during a NASA contract. The quasi-combustion terms and rocket boundary conditions from Euler3D were adapted to work in CFDsol. To give CFDsol the ability to model elastic deformation (inviscid) and non-inertial motion, the inviscid transpiration, non-inertial source terms, modal elastics and rigid body dynamics

models were also adapted from Euler3D into CFDsol. The viscous terms in CFDsol were corrected to match its derivation and equations implementation in NS3D. The SA turbulence model in NS3D was transferred to CFDsol, and the arrays and variables were adapted to interconnect with those already present in CFDsol.

Objective 4. The capabilities of the four in-house OSU codes were demonstrated on a variety of cases and regimes: Subsonic, transonic, supersonic, and hypersonic; inviscid and viscous. Some of those demonstrations were used as verification (comparison with analytical or numerical models) or validation (comparison with experimental data). The demonstrations showed that the inviscid codes are accurately modeling subsonic, supersonic, and hypersonic fields. Questionable results arose from several transonic airfoil comparisons. The geometry representation and meshes have been improved by another OSU researcher. These enhancements in turn improved the transonic solutions. The SA model has been verified with a handful of tests. This verification needs to be continued. Concerns have been raised for modeling turbulence over curved walls, and derivatives were shown to be decoupled when using orthogonal elements. The SST and heat transfer models have not yet been verified.

Objective 5. Prior to the completion of the NASA contract, several cases from Objective 4 were repeated using CFDsol. These cases were used to compare and improve the performance of CFDsol. In the end, the accuracy of CFDsol is comparable with the OSU codes and other numerical solutions. The use of explicit derivatives in CFDsol often limits the applications or mesh convergence. The stability is supplemented through an overly aggressive shock capturing (artificial dissipation) routine and inviscid local time stepping. Viscous local time stepping was implemented using the one-dimensional model shown in this work, which is not sufficient to stabilize general unstructured meshes. This author would encourage those

working with CFDsol to implement stiffness matrix to calculate viscous time steps and a new artificial dissipation model as a means of enhancing stability. CFDsol should also be adapted to either avoid the use of explicit property derivatives in the inviscid terms or by including the explicit derivatives in the stiffness matrix for local time steps.

8.1.2 Evaluation

The five solvers were demonstrated in Chapters 6 and 7 on inviscid, propulsion, viscous, and time accurate test cases. This evaluation has been broken down into similar sections:

Inviscid Solutions. The inviscid verifications were performed by comparing solutions with analytical solutions, numerical models, and the other solvers. Validations were also performed by comparing the inviscid solutions with experimental data. These comparisons showed that the inviscid solutions in Euler2D, Euler3D and CFDsol are accurate within the limits of the discretization. The accuracy of subsonic and supersonic solutions in all three solvers improves with mesh convergence. Transonic solutions are the most difficult because transonic meshes must converge the subsonic regions to one standard and locally supersonic regions to a different standard. Transonic solutions are highly dependent upon the definition of the geometry and convergence of the mesh in all regimes.

NS2D and NS3D were built from Euler2D and Euler3D. The inviscid boundary conditions were retained during this process. Inviscid walls can be used with Navier-Stokes to constrain the flow without requiring boundary layer modeling. For example, a wind tunnel wall or streamlined close-out can be modeled with an inviscid wall. The inviscid wall can also be used to model an inviscid solution with viscous dissipation in place of artificial dissipation. Viscous dissipation is sufficient for a Re of 100 or less, but the viscous terms may increase

run times over artificial dissipation because of the local viscous time step. This has been done at various stages in this research. NS2D and NS3D could be used to model inviscid and viscous solutions as all encompassing solvers. This is discouraged in general because the viscous terms will increase computational costs with decreased stability. The viscous terms in NS2D and NS3D could be used to stabilize inviscid solutions more efficiently with viscous dissipation over artificial dissipation ($diss = -1$). Viscous solutions have been generated without artificial dissipation for Reynolds numbers less than 100, and the artificial dissipation in the SA and SST models was scaled to be equivalent to a Reynolds number of 300. These numbers can be used as a guide for setting Re for inviscid solution stability.

Inviscid Stability. All three solvers are stabilized with artificial dissipation and local time stepping. Local time stepping is well known throughout the literature and has been implemented in all three solvers as a means of local relaxation. Inviscid local time step is calculated using the maximum local velocity ($V + a$) and characteristic length. The OSU codes use segments between nodes to define this local velocity and length. CFDsol performs this calculation on each element and uses the minimum distance between a vertex and its opposing face as the characteristic length. The velocity vector and characteristic length are often uncoupled in CFDsol. The vectored-coupling in the OSU codes creates a much more stable influence on the local flow field.

The artificial dissipation routines in all three solvers are sufficient to control many disturbances at their nominal levels ($diss = 1.0$ and $FACTOR = M_\infty / 10$). Artificial dissipation is also a source of error for all three solvers since its contributions are *artificial* in nature. Artificial dissipation is a necessary evil in that its effects on stability are necessary but changes to the overall solution are undesirable. CFDsol contains explicit derivatives in its inviscid

terms, which create numerical instabilities similar to viscous terms. CFDsol does not attempt to handle these instabilities in its local time stepping.

Viscous Solutions. Laminar solutions were verified against analytical and numerical solutions for simple geometries. The viscous terms in NS2D and NS3D are accurate for all cases investigated when appropriate mesh spacings are used (normal and streamwise) and when the artificial dissipation is minimized in the solution. Many solutions were generated with artificial dissipation to stabilize the initial startup and then artificial dissipation was removed from the solution. When this method was not appropriate, the zero dissipation length *dislen* was used to remove artificial dissipation from the near-wall region while keeping artificial dissipation during startup and within the regions of the flow dominated by inviscid physics.

CFDsol was much more difficult to verify. Artificial dissipation is essential in the current implementation of CFDsol so its viscous routine was transferred to NS3D and named NS3Dsol. NS3Dsol did not have the same requirements for artificial dissipation and showed the viscous terms were accurate for laminar solutions in the limit as artificial dissipation was removed from the solution.

Viscous Error Sources. Several sources of error exist in all three viscous solvers: Artificial dissipation, viscous stability, and the accuracy of stresses. Artificial dissipation attempts to smooth out the velocity profile normal to the wall. This decreases the skin friction and thickens the boundary layer. A zero dissipation length *dislen* has been implemented in all three solvers and shown to be an effective means of improving solutions from NS2D and NS3D. An appropriate size for *dislen* is often hard to determine for the entire field, especially when the thickness of the boundary layer varies drastically across the body.

Alternative artificial dissipation models, like Nithiarasu (1998), were investigated in this work. These models are scaled by the gradient or Laplacian of one or more properties. Gradients are sensitive to numerical errors and scaling the artificial dissipation model by such errors makes the model equally sensitive to numerical errors. The Laplacian is calculated using second derivatives, which are even more sensitive to numerical errors and less accurate on a piece-wise linear field. Finally, the greatest sensitivity and instability in the gradients was found near boundaries, which kept the artificial dissipation on near boundaries. The entire exploration into artificial dissipation models was searching for less dissipation near walls.

Viscous stability was also an issue in all three solvers, especially in cross flow velocities in the 3D solvers. The cross flow velocity is more susceptible because the magnitude is very small and does not wash out the numerical errors that feedback into the viscous stresses. NS2D showed signs of viscous instability for very small meshes used to test the turbulence models. The number of degrees-of-freedom is increased for 3D meshes, which heightens their sensitivity. The viscous stiffness matrices were used to calculate the viscous local time steps in NS2D and NS3D. The viscous local time steps were shown to stabilize the viscous terms. In fact, tests were performed on a cylinder in NS3D using local time steps calculated using the x - and y -derivatives (2D model). When the cross flow velocity was aligned with x or y , the solution showed much greater stability. When the cross flow aligned with z , numerical instabilities were dominated w .

The accuracy of stresses and heat fluxes is affected by the accuracy of the gradients used to calculate them. The properties are piecewise linear, so the gradients are piecewise constant. This means the viscous fluxes are piecewise constant and apply a different influence depending on the element. Orthogonal elements decouple their derivatives, which is good for a flat plate but problematic for flows along curved boundaries. The accuracy of stresses and heat fluxes would be greatly increased by smoothing out the derivatives. Hybrid elements (linearly distributing the

derivative) were investigated and found to apply appropriate smoothing to the derivatives near the center of the domain, but the elements near the boundary showed even more degradation in derivative accuracy. O'Neill (2011) shows that the mass matrix is singular in the hybrid system and applies a plate stiffness term to smooth any additional problems. The plate stiffness acts like an artificial dissipation for derivatives. Plate stiffness was applied to the derivatives in NS2D and showed to be ineffective until the stiffness began to dissipate all of the derivatives.

Heat Transfer. The heat flux terms have been implemented in all three viscous solvers, and heat transfer boundary conditions have been implemented in the OSU solvers along with input files. These conditions have been tested for stability and to show that the solution changes based on the boundary condition. These conditions need to be verified (and validated) before being used in any application.

Turbulence Modeling. A compressible Spalart-Allmarus (SA) model was implemented in all three solvers. That model has been tested on simple test cases and verified using a flat plate boundary layer. Problems were demonstrated in the presence of wall curvature using Rumsey's bump case. The SA model needs further exploration and validation. Menter's SST model was also implemented in the OSU codes. The SST model is relatively untested. The advection, diffusion, and artificial dissipation terms have been tested on a simple advection problem. The source terms require the presence of a wall to be tested. The ω boundary condition is hard to satisfy and is very sensitive to wall normal spacing. A simple flat plate boundary layer should be used first and tested with various wall normal spacings to verify that the ω boundary condition is working properly for small spacing. For rotation in the non-inertial frame, the Spalart-Shur rotation-curvature correction has been applied to both the SA and SST models. The curvature term has been neglected for simplicity, leaving the rotation correction, which has not been tested.

Time Accuracy. Time accuracy was improved for CFDsol using the Cowan's (2003) predictor-corrector method applied in the OSU codes. The time accuracy of this method was demonstrated on the Wagner and Theodorsen cases in CFDsol. For the OSU codes, unsteady demonstrations were performed using the Wagner airfoil, double shock tube, and laminar cylinder with trailing vortex street. For reasonable time steps, the predictor-correct retained its time accuracy even when viscous terms are added to the model. A reasonable time step for advection is upper bounded by the macro flow scales (period of T): $\Delta t < T / N$, where N is the number of steps per flow period. The smallest period in the flow should dictate the time step and 20 to 100 steps should be used per period. These numbers come from experience. The very small velocities near the wall do not advect information as quickly as the external flow, and little investigation has been done in this research to determine the effects of time step near the wall. For now, the time step can be controlled by the external, macro flow features. URANS could add another complication if the SA or SST models are used in time accurate solutions. URANS assumes that the properties are time-averaged over some period T , which is assumed to be much larger the period of fluctuations in the flow. The time-averaged properties are then considered to change "slowly" in time. How this will affect the time step in NS2D or NS3D is not yet known. If the turbulence models are treated as differential models supplementing the RANS equations (as this author intended), then the time accuracy of the solution is still governed by the macro scales in the flow. When the turbulence models are verified, the time accuracy of the models should be investigated. If the URANS models are found to be less than time-accurate, LES or DES models should be added to the suite. LES models are based on spatial averaging rather than time-averaging. DES models are a combination of URANS near the wall and LES away from the wall.

Propulsion Modeling. Propulsions models were created in all five solvers. The models were created to be simple, physics-based means of representing the near-field around propulsion sys-

tems while generating thrust. Before this work, turbojets and rockets were modeled as far field (freestream) boundaries and vehicles were pulled along with “magic strings”. Scramjets were modeled in the engine off condition. Three models were implied in the OSU codes through the boundary fluxes: Rocket exhaust, turbojet inflow, and turbojet exhaust boundaries. The rocket exhaust was modeled using total properties, as an equivalent to a combustion chamber. The turbojet inflow applied a pressure at the inflow to induce a specified mass flow into the turbojet. The other properties were pulled into the engine naturally by the mass flow. The outflow turbojet boundary was coupled to its inflow using conservation principles. Quasi-combustion, or mass and heat generation terms, were added to the model to simulate ramjet, scramjet, or afterburner combustion. The rocket and quasi-combustion terms were added to CFDsol and implemented through explicit boundary conditions.

Several sources of error could be produced during propulsion modeling. All three of the models were created to be simple and easy to apply. The rocket boundary condition assumes that all combustion occurs upstream of the exhaust boundary, which may not be the case in reality. The turbojet boundary conditions do not model the swirling flow induced up and downstream of the rotating turbomachinery. The rocket and turbojet exhausts also apply a constant flux to all boundary elements in the inertial frame. The real exhaust profile contains a boundary layer along the nozzle walls and wakes from upstream components (turbine blades). The propulsion models were implemented in all five models but only tested in inviscid fields to avoid the complex modeling of boundary layers in inlets and nozzles. This author would advise future users to utilize the inviscid wall boundaries to simplify inlets and nozzles and only use viscous walls where necessary to build boundary layers around the inlet leading edge and nozzle lip.

MDA Readiness. The original intent of this research was to create a multi-disciplinary analysis (MDA) suite using the OSU codes and/or CFDsol. The stability problems limit the use of

CFDsol, so this research was focused on the OSU codes. The available connections with CFDsol (post-contract) can be seen in Figure 2.3. The in-house codes are much closer to implemented in such as MDA environment. Figure 2.4 shows the current connections available with the OSU codes. This section outlines that readiness. The accuracy of the solvers has been addressed in the previous evaluation sub-sections. The next step is to evaluate the ability of the solvers to model the various interfaces necessary to incorporate the models into a MDA environment (Figure 2.1 and 2.2). Those connections can later be made more efficient.

The lower order system model represents the simplest connections between the CFD solvers and external modules. Euler2D/3D and CFDsol can handle modal elastics, which are preprocessed in a structural FEA such as STARS. Elastic deformations are modeled through inviscid transpiration. NS2D/3D contain similar connections but only velocity transpiration is available for the no-slip wall. Elastic boundaries can only be incorporated into the viscous solutions by moving the boundaries and deforming / remeshing the internal mesh. This topic is a complex research area in and of itself and has been left for future researchers. These mode shapes can be calculated using the solid temperature distribution created using the CFD boundaries as inputs to the thermal FEA. This temperature distribution is readily available between iterations in the safe mode restart case.rst file. The lower order propulsion models have been implemented in the CFD solvers. (Turbojet boundaries still need to be implemented in CFDsol.) These models are taken from user inputs, which can be constructed using the current CFD solution in the case.rst. The rigid body dynamics model has been incorporated in the five codes in full non-linear form. A subroutine has been set aside in the OSU codes to hard-code controls routines directly into the CFD solution. (A similar routine can be added to CFDsol.) The lower order systems model can be readily created using any of the five CFD solvers, FEA, and propulsion models through inter-connecting modules to transfer information between the routines.

The higher order systems model can be created by passing CFD boundaries temperatures (case.rst) to the thermal FEA, which returns boundary heat fluxes to the CFD solver (case.tbc). (Different boundary conditions must be passed between the two codes. If the heat flux is passed to the structure, then the problem is actually singular. One essential condition (temperature) is required to correctly specify the problem. If heat flux is pass to the CFD solution, the far field acts as the essential condition.) The temperature distribution in the structure can be passed to the structural FEA along with the surface traction Eq. 4.237 from the CFD solution (case.rst). The viscous stresses are calculated from the velocity gradients, which requires knowledge of the mesh (case.g2d, case.g3d, or case.cfs). This process could be simplified by including the stress components of the traction in the restart file (case.rst) or another file that is over-written between iterations. The structural deformations can be processed using the temperature and traction distributions and passed back to the CFD solutions using the modal format in the case.vec. The higher order combustor is modeled similar to the higher order turbojet, and the higher order rocket boundary is the same as the turbojet exhaust boundary. The higher order turbojet can be created by passing properties for entire boundary between the CFD solvers used in this work and external CFD models. The inflow plane can be modeled by passing the flux and properties on the central CFD solution to the external module, which returns the pressure distribution across the inflow plane. The exhaust plane can be modeled using the opposite process. The central CFD gets fluxes and properties from the external solver and passes back the pressure along the boundary. If the external routine only passes fluxes, *flux_props* can be used to calculate the properties to construct the Riemann invariant matrix. Again, the rigid body and controls routines are fully implemented as non-linear models with the central CFD code.

8.1.3 Precaution against Pitfalls

The tests in this research were performed with unit values to keep the tests simple where possible. Common Mach and Reynolds numbers were also used. Simple cases were repeated for different Reynolds numbers to ensure the independence of results. More tests should be run with dimensional numbers to properly prove that those numbers are being applied properly and their dimensions are removed properly.

Non-Dimensional Numbers. Following Cowan's (2003) implementation as a standard, all input files are dimensional except for the controls file. The controls file contains three values used to non-dimensionalize all input values: *refdim* contains the length dimension, *ainf* contains the length and time dimensions, and *rhoinf* contains the mass and length (volume) dimensions. The length dimensions should be consistent in all three values. For instance, if the length dimension is in *meters* (SI), then the mass and time dimensions should be in *kilograms* and *seconds*. The use of g_c should be avoided altogether. In other words, *slugs* are used as the English mass unit for *feet*, and *slinches* are used as the English mass unit for *inches*. Dimensionless values are specified in the controls file for Mach number *mach*, Reynolds number *Re*, and time step *dt* so that *ainf* and *rhoinf* can be ignored for unsteady simulations. Finally, a dimensional value for gravity has been included in the controls file instead of the dynamics input file (*case.dyn*) to facilitate later expansion of its use as a body force in the CFD solution.

Care must be taken when applying the freestream Mach and Reynolds numbers. Non-inertial solutions can be specified using the velocity of the non-inertial frame instead of a freestream (*ifree* = .true.). The "freestream velocity" is still used to non-dimensionalize all inputs from

files. The “freestream velocity” is calculated in the same manner as when a freestream is actually present: $u_{inf} = mach \ a_{inf}$.

The Reynolds number can be tricky to specify if precautions are not taken. For instance, a wing of a UAV has a dimensional chord of 3 in and will be modeled at a Reynolds number of 3.5×10^5 . The mesh should be generated in units of inches, where the wing chord has a length of 3 mesh units. If *refdim* is specified to be 3 mesh units (3 inches), *Re* should be 3.5×10^5 because *refdim* is equal to the chord length. *refdim* is often specified as 1 mesh unit (in this case, 1 inch). In this case, *Re* should be set to 1.166×10^5 so that the Reynolds number based on the 3-inch chord is appropriately 3.498×10^5 . Finally, someone else generates the mesh in millimeters (3 in = 76.2 mm). If the simulation is steady or unsteady without the presence of other models, the mesh can be left in terms of millimeters (*refdim* = 76.2, *Re* = 3.5×10^5 or *refdim* = 1, *Re* = 4.593×10^3 so that Reynolds number based on chord is 3.4999×10^5). If additional models are being used, the mesh can be scaled by 0.03937 to put the mesh units in terms of inches, or the mesh can be redrawn.

Demonstration vs. Verification vs. Validation. Verification is the process of comparing solutions from a method with those produced by other analytical or numerical methods. Validation is a similar process, where experimental data is used as a means of comparison (Roache, 1998a and 1998b). All verifications and validations are demonstrations of capability, but demonstrations are not verifications or validations without comparison. Verification proves that the simulation is properly representing the desired physics model. The CFD solution is verified when the solution matches the desired physics model (analytical or numerical distribution that has been verified) within a reasonable limit or in the limit as the mesh approaches a continuous discretization. Validation is performed on a verified model to

demonstrate how *valid* (accurate) that model is for a particular purpose, which the validation data encompasses.

The temperature boundary conditions and SST model (particularly the accuracy of the ω boundary condition) have not yet been verified. This is the first step in *verifying* that all of the equations have been implemented in the CFD codes correctly. Acoustics have been also been seen in the demonstrations. An acoustic verification can be constructed using the quasi-combustion terms to oscillate mass production on a single element (very small) to simulate a single monopole source. The resulting distribution of density should resemble radiating waves with magnitude decreasing according to r^{-2} in 2D and r^{-3} in 3D. The acoustics output file can be used to track the position and strength of the waves. Several new features have been added to the rigid body model: Fully inertia matrix I , stiffness K , and damping C matrices. These later two matrices can be verified simple linear and rotational motion compared with analytical models or the modal elastics model. The inertial matrix can be verified using an asymmetric vehicle compared to an analytical or numerical systems model. Finally, the heat transfer, SA, SST, and acoustics need to validated before their use. In fact, the solver(s) should be validated using experimental data within the regime on a similar geometry before assuming that the CFD solution is extendable to any generic cases.

8.1.4 Standards and Good Practices

Several standards or good practices have been created by this research, Cowan (2003), Brown (2009), O'Neill (2011), and other OSU researchers. These best practices will help future modelers and researchers to create appropriate meshes, time steps, stability, etc.

Meshing Practices. Brown (2009) and others have found an appropriate inviscid spacing is less than 8% of the radius of curvature or less than 5% rotation of the boundary element normals. This work has found similarity trends for the number of elements necessary to model shocks and expansion fans with a given accuracy: Figure 6.50 and 6.53. The number of elements increases with *diss* and *mach* and decreases with the displacement angle for shocks. The artificial dissipation model smears the shock or expansion over a given number of elements, found in the figures. Viscous spacing is defined by the boundary layer thickness. Laminar boundary layers can be represented using six elements at a minimum, where the normal spacing at the wall should be at most 15% of the boundary layer thickness δ . The lateral spacing along the wall can be as large as 0.5δ . The turbulent boundary layer can be modeled using the laminar guidelines as a basis. The spacing nearest the wall is refined across three y^+ regions: at least 1 element are used across the viscous sublayer ($y^+ < 5$), at least 3 elements across the buffer layer ($5 < y^+ < 30$), and at least 8 elements across the log-law region ($30 < y^+ < \delta^+$). Of course, more elements can be used.

Time Stepping and Solution Stability. The time accuracy and stability are both tightly controlled by the time step size dt . A reasonable time step for advection is upper bounded by the macro flow scales (period of T): $\Delta t < T / N$, where N is 20 to 100. For example, a vortex street is created downstream of a body, and the vortex street is the dominant frequency in the flow. The frequency of the vortex shedding is 1000 Hz (or period $T = 1$ ms). An appropriate time step would be 0.01 ms. For $refdim = 3$ in, $ainf = 13,400$ in/s, and $mach = 0.2$, the dimensions are removed from the time step: $dt^* = dt \text{ mach ainf} / refdim = 8.93 \times 10^{-3}$. The inviscid local time step is calculated using the spacing in the wake:

$$dt^* = \frac{\Delta t U_\infty}{L} = CFL \frac{\Delta x_{wake}}{V+a} \frac{U_\infty}{L} \leq 0.8 \frac{M_\infty}{M_\infty + 1} \frac{\Delta x_{wake}}{L} \quad (8.1)$$

where $CFL = 0.8$ is an empirical upper limit for inviscid solutions in Euler2D/3D. $CFL = 0.5$ is suggested as a general guideline for steady and unsteady flows. For a freestream Mach number *mach* of 0.2 and wake spacing that is 2% of the reference length $\Delta x_{wake} / L = 0.02$, the time step is limited to 2.67×10^{-3} , which is the limiting case here. Stability requires the time step to be 30% smaller or 333 steps per flow period.

The previous discussion does not include the viscous local time step, which is more difficult to calculate in general. Viscous local time stepping is specified through *itime*. If the mesh is an orthogonal grid, then *itime* = -1 can be used to quickly calculate the viscous local time step. For unstructured triangular and tetrahedral meshes, *itime* = 0, 1, and 2 are used to specify three levels of fidelity: Heat transfer, momentum, and turbulence model stability. Each level performs an additional stiffness matrix check. The levels are specified so that the first is the most frequently used, and last is least frequent. *irsds* can be set to *.true.* to create several files that can help tune the local and global time steps. The *case.time* file gives the ratio of local to global time (first column) and ratio of local times across the different fidelity levels. The first column will always be smaller than unity because the upper bound of the local time step is the global, keeping the solver in a relaxed state. The remaining columns are less than unity when that level of fidelity is being used. If a column is equal to unity (after a short startup), then *itime* can be reduced to remove this matrix from the local time stepping calculation. *itime* = 0 should always be kept at a minimum for unstructured meshes. The first column of the *case.time* relates the local to global time steps. If this column is equal to 0.1, then at least 10 inner cycles *ncyc* should be used to give the solution enough “local

time” to converge a given time step. This does not mean all of these cycles need to be used. *rsdtol* can be used to cause check the residual convergence and exit the inner cycles for the time step. Different levels of *rsdtol* are appropriate for different applications: *rsdtol* < 10^{-6} for class projects, < 10^{-8} for thesis research, and < 10^{-10} for published research, as general guides. These limits can be increased to reduce run times to meet delivery schedules. (*rsdmax* can be used to early terminate the program in case of divergence: *rsdmax* = 10^2 is typical.) Using *rsdtol*, the number of cycles can be set large (*ncyc* = 100) and exit before all of those cycles are used. When *irsds* = .true., the number of cycles for each time step is written to the case.cyc, which can be used to ensure that enough inner cycles are being used. A general rule of thumb for efficiency is to use 5 to 20 cycles per step (after a reasonable startup period). If less than 5 cycles are being used, then the time step is too small and the solution will eat up time stepping forward (nearly explicit). If the cycles are greater than 20, then the time step is too large and the time will be spent converging the given solution before stepping. The residual at the end of each time step is written to case.rsd. When *irsds* = .true., the residual at the end of each cycle is written to case.rsd2, which shows how quickly the residual converges on each time step. The energy residual is generally the largest so the energy residual is tested against *rsdtol* and *rsdmax*.

Beyond local and global time step size, stability is augmented with artificial dissipation. If the Reynolds number is less than 100, the artificial dissipation can become optional. The artificial dissipation model can be turned off for *idiss* = -1, which is much more efficient. Otherwise, dissipation must be added to the solution to stabilize the inviscid flux terms. A nominal value of unity is appropriate for all dissipation *diss* in the OSU codes, and one tenth the freestream Mach number can be used for dissipation *FACTOR* in CFDsol. For

supersonic solutions, the lower order model ($idiss = 0$) is appropriate in the OSU codes. For subsonic or rotating fields, the higher order model ($idiss = 1$) introduces a minimum gradient detection, which allows the non-inertial and vortex rotations to persist in the solution. For viscous solutions, artificial dissipation changes the boundary layer region of the flow. The zero dissipation length $dislen$ can be used to minimize dissipation over a set thickness. $dislen$ works best when it is set equal to half of the boundary layer thickness, but the boundary layer grows over the length of the surface. The boundary layer thickness at 70% of the body length (chord, etc.) can be used as a single measurement. Artificial dissipation is also applied in the turbulence models and can be scaled down using $disst$, where the artificial dissipation applied in the turbulence models is scaled by $diss*disst$. A nominal value of unity is appropriate for the turbulent scalar $disst$. If the turbulence model is more or less stable for a given situation, $disst$ can be changed without affecting dissipation in the RANS equations.

Solution Modes. Several different solution modes can be used in the OSU codes. $isol = 0$ uses the steady solution, which assumes no time accuracy. $isol = 1$ and 2 produces time accurate solutions (with reasonably small time steps) that are first and second order accurate in time. The first order scheme can be restarted using a case.unk file ($irstrt = .true.$). The second order scheme requires two previous solution steps, the later of which is not recorded in the case.un#. All data needed for a smooth restart is recorded in the safe mode ($isafe$) restart file (case.rst). Safe mode should be used, at the cost of efficiency, for second order solutions. The restart file cannot be used for plotting like the unknowns file case.un# or case.unk, but the restart file allows for smooth and efficient restart of a solution of any order.

The number of Gauss points used for numerical integration $ipnt$ can be increased in Euler2D/3D, but Cowan (2003) and Brown (2009) have shown that $ipnt$ is sufficient for the

current implementation. If alternative dissipation models are implemented in Euler2D/3D, the number of Gauss points should be retested, particularly the analytical integration (Appendix B), which perfectly represents an integral in a more efficient manner than third order Gauss quadrature. NS2D/3D are restricted to $ipnt = 1$ at all times.

Finally, “incompressible” solutions can be created using $mach = 0.2$ or less. Incompressible flow is defined by an infinite acoustic speed, which is not possible in any of the codes tested in this research. In the limit as Mach number approaches zero, the density fluctuations become very small and the compressible implementation of the continuity equation becomes ineffective, limiting convergence. This does not create a limit on capability but rather presents a change in efficient calculations – longer run times.

8.1.5 Converting between Solvers

Five different solvers have been used during this research. The solvers are very similar, especially the OSU solvers, but the files used by each differ, if only in number of properties. Several support codes have been written just to convert between solutions and solvers.

Converting between Modes. The inviscid solvers Euler2D/3D are Reynolds number dependent. When converting to the viscous solvers NS2D/3D, the Reynolds number must be added to the restart file case.unk. Reynolds number and several other values should simultaneously be specified in the controls file. Inviscid solutions can be used to quickly converge the pressure and external velocity before slowing the velocity near the wall in a viscous solution. The conversion between viscous to inviscid solutions has not been used in this research.

The viscous solvers can be used to model laminar, SA, or SST based solutions. Conversion between these solutions is detected by the *iturb* flag in the case.unk file and happens auto-

matically upon restart. Laminar properties are retained when moving from laminar to turbulent or SA to SST models. The SST properties k and ω can be used to calculate the eddy viscosity and distribution of SA variable $\hat{\nu}$, so the turbulent properties are retained during this conversion.

The conversion between inertial and non-inertial frames can be deceptively tricky, especially if the freestream velocity u_{inf} in the inertial frame is replaced by the mesh velocity (v_x, v_y, v_z) in the non-inertial frame. For example, the solution field is converged in the inertial frame using a freestream Mach number of 0.5. The solution is used as the initial conditions for a non-inertial solution, where the velocity is induced by the motion of the frame. The motion of the frame is equal in magnitude and opposite in direction of the velocity induced on the flow: $v_x = -u_{inf}$. To ensure that the initial conditions match the desired non-inertial solution, the mesh velocity must be specified in terms of $mach$ and $ainf$. If $ainf = 1116$ ft/s, then a mesh velocity of -558 ft/s is required for smooth conversion. If an angle of attack is present in the inertial solution, then the incidence should be reflected in the initial pitch angle θ .

Tools Useful in Conversion. Several support codes have been created to facilitate the conversion of files and their values. EditCon reads a controls file (case.con or case.data), edits the control data, and writes that data back to a controls file (original or different format). EditCon was originally created to allow the controls file to be edited during batch processing. EditCon can also be used to produce a default controls file using the case.con with “&control” on the first line and “/” on a following line. EditCon produces a table of values with short descriptions and prompts to assist the user in setting up the given controls file. The user is prompted for the solver to be used before the file is written, and only the appropriate controls are written to the file.

Geometry and unknowns files can be converted from 2D to 3D solvers using 2DExtrude. (Several other support codes are available to convert between 2D and 3D, but 2DExtrude is the most comprehensive.) 2DExtrude can be used to convert the geometry file alone (case.g2d to case.g3d), a single restart file (case.unk), or multiple solution files (case.un#) for plotting in Glplot3D. The later is used when the additional features are needed that are only available in Glplot3D: Velocity magnitude, total energy, internal energy, or vorticity. Total energy and velocity magnitude are necessary to plot the SST distributions in the caset.un#. After extrusion, the mesh and velocity can be aligned with different axes using Rotate3D. This process can also be used to make a 2D solution available to Remesh3D.

Remesh3D can be used to refine the unstructured mesh using the gradient or Hessian of various properties. Remesh3D creates a new caseR.bac file which defines the spacing as direction vectors at all nodes in the original mesh. The spacing and direction is linearly interpolated during the meshing process. Surface and Volume are used to create the new mesh. The original geometry definitions case.sur can be used in the 2D or 3D domain.

The original solution can be interpolated onto a remeshed or different user-defined mesh to decrease convergence time. UnkInterp2D/3D linearly interpolates the properties on each element using the shape function as a basis. In the infrequent case that the new node lies outside of the original mesh, the distribution along the boundary element is extrapolated normal to the boundary element. At the end of the interpolation, the user is given the ability to adapt the header of the new case.unk file. The header can also be quickly changed using UnkAdapt2D/3D. The Mach number, Reynolds number, solution time, orientation angles or ratio of specific heats can be adapted in the header. The other values are purposefully untouchable. UnkAdapt2D/3D can also be used to write out an ASCII copy of the case.unk.

Interfacing CFDsol with In-House Codes. After the NASA contract, switching between CFDsol and the OSU in-house codes is purely a matter of file format. The geometry files can be converted using g3d2cfs or cfs2g3d. g3d2cfs can be used to convert a OSU geometry to CFDsol geometry, which stores less information. The reverse process in cfs2g3d is not a complete process and should only be used for viewing case.un# files in Glplot3D.

The controls files can be converted using EditCon, similar to that described above. The only other conversion involves the restart file. CFDsol works best for explicit solutions although implicit cycling can be used in certain cases. The surfaces must be specified for the calculations of loads. EditCon reads the case.bco file for solid walls and includes these surfaces in the case.data file. EditCon presents the surfaces in tabular format to be included or excluded. Finally, two output file formats are available in CFDsol; only *itrans* = 1 produces case.un# files that can be plotted in Glplot3D.

CFDsol uses a case.out file, which is over written after each iteration. Unk2Out can be used to convert a OSU case.unk file or CFDsol output file case.un# to the restart format case.out. The case.unk should always be retained after conversion because CFDsol over writes the case.out. If the restart does not happen properly, CFDsol has already changed the restart file. The case.unk is necessary to recreate the file.

8.2 Recommendations

Several recommendations have been given in the previous descriptions. Some of these recommendations are repeated here with more detail. The recommendations given in this section are listed in order from most to least importance, in the opinion of this author.

Improve Mesh Generation. Mesh generation has been a constant problem throughout this work. Many of the cases in this document were generated in the 2D plane and extruded to 3D, if necessary. Meshes were also generated using Pave2D/3D. Both processes were used to avoid meshing 3D domains using Volume, which shows weird quirks. The run time error reporting is difficult to interpret and often misleading. Volume can fail to generate a mesh because the surfaces do not intersect within a given tolerance, the routine decides to remove and replace the same element repeatedly, or the mesh fails any of various qualifications. Surface can fail for the same reasons, but working with Surface is much more tolerable.

Many of the features in Surface and Volume are desirable. Both codes use Delauney's marching front method to produce a smooth, nearly isotropic mesh distribution. Surface can be used to produce 2D meshes or the boundaries for 3D meshes. The boundary mesh can be view and evaluated before Volume is used to fill in the internal mesh. The spacing throughout the domain is specified using the background mesh spacing and three types of source distributions. The background mesh is used by Remesh3D in its mesh refinement. Switching to a very new process could also keep future users from utilizing Remesh3D, although generating a mesh with Remesh3D is often much easier than producing the original mesh. (This is most likely due to the eradication of bugs during the creation of the original mesh.)

The new meshing software should use a robust and repeatable method, like Delauney's marching front method, to generate isotropic meshes within the domain. Anisotropic, or stretched, meshes are also nice for efficiency when dealing with shocks and boundary layers. The new software should incorporate different methods of refining the mesh that can utilize the meshing practices outlined earlier in this chapter: Surface curvature, refinement along shocks and expansions, and viscous wall-normal spacing. The later requires the presence of

rectangular elements along boundaries and prismatic elements normal to 3D walls. Triangle and tetrahedral elements are the only options available in Euler2D/3D, NS2D/3D, or CFDsol. Rectangles can be split into triangles, and prisms can be split into tetrahedra to convert efficient meshes into their least complicated denominator. Later, the 2D codes can be amended to include quadrilateral elements, and the prisms can be added to the 3D codes by adding a shape function to interpolate between two linear triangles.

Ideally, the new meshing software should be robust and efficient, open source and free, and include separate surface and volume meshing into a single GUI. The surface meshing routine could be used to produce 2D meshes, while the combination could be used to produce 3D domains. Ideally, the source, background, and surface definitions used in the case.bac and case.sur could be extended into the new input files so that the new and old software are interchangeable. This would also allow the user to use Remesh3D to refine meshes.

Improved Wall Elements. After near-wall meshing has been introduced into the mesh, the next logical step is to utilize those elements instead of breaking the elements down into triangle and tetrahedra. The 2D domain would utilize quadrilateral elements to represent the boundary layer mesh, and the 3D domain would apply a shape function normal to a linear triangle element to create prisms. (Pyramids would also be needed in 3D to connect the quad faces of prisms together with tetrahedra.) Since the elements are new, different wall normal distributions should be tested for efficiency and stability. Some of the lessons learned by O'Neill (2011) should be applied during this process. Higher order distributions (second or third order) could be used normal to the wall minimizing the number of elements required to model the boundary layer. A quadrilateral element can be constructed with a linear

streamwise and higher order normal distribution. The higher order distribution ($0 \leq \eta \leq 1$) can be used to interpolate the coefficient used by the linear shape functions ($0 \leq \xi \leq 1$):

$$\Phi(\eta, \xi) = (1 - \xi)p_{123} + \xi p_{456} \quad (8.2)$$

$$p_{123} = p_1(2\eta - 1)(\eta - 1) + p_2 4\eta(1 - \eta) + p_3 \eta(2\eta - 1) \quad (8.3)$$

$$p_{456} = p_4(2\eta - 1)(\eta - 1) + p_5 4\eta(1 - \eta) + p_6 \eta(2\eta - 1) \quad (8.4)$$

Similarly, the prism elements can be created by distributing the element along the surface of the wall using the linear element. The higher order function is used to extrude the triangle away from the wall. Using the same conventions, the prism element is created using Eqs. 8.3 and 8.4 for two vertices of the triangle:

$$\Phi(\eta, \xi_1, \xi_2) = \xi_1 p_{123} + \xi_2 p_{456} + (1 - \xi_1 - \xi_2) p_{789} \quad (8.5)$$

$$p_{789} = p_7(2\eta - 1)(\eta - 1) + p_8 4\eta(1 - \eta) + p_9 \eta(2\eta - 1) \quad (8.6)$$

Parallel Processing. The OSU codes and CFDsol have been setup to accommodate a future switch from single core to parallel processing. The mathematics can be broken apart into loops over the domain elements, boundary elements, and segments. Parallel processing can be created in two ways: Multiple platforms across network, or multiple cores on a platform. The later can be accomplished using OpenMP with minor adjustments to the source code. Each loop that will be parallel processed using OpenMP is bounded by “\$OMP ...” and “\$OMP END ...”. The OpenMP statements contain the loop construct, variables to be shared across all threads, variables to be isolated between threads, and variables to be assembled commonly between the threads. All variables used in the loop must be subdivided properly into these three groups so that steady and time-accurate solutions are possible. If

the variables are over written by one core before being used by another, or if residual contributions depend on the order of assembly, then the accuracy of solutions is no longer ensured.

Several of the cases from this body of work should be used to represent subsonic, transonic, supersonic, and time accurate solutions for the inviscid and viscous solvers. OpenMP can be applied to Euler2D and checked versus the single core solution. If OpenMP is applied correctly, the solutions should be exactly the same. The OpenMP statements can be expanded to include viscous variables and transferred to NS2D, then all OpenMP statements can be expanded to include the third dimension (z , w , etc.) and transferred to Euler3D and NS3D. Similar procedures could be performed in CFDsol. Each solver should be checked against itself with and without OpenMP.

Visualization. Pinkerman (2010) extended the capabilities of the OSU codes for plotting 3D meshes and solutions. Pinkerman added four new plotting variables to Gplot3D and created a new visualization suite that utilizes the netCDF file formats. Additional variables need to be added to both codes for plotting turbulent properties. This will greatly enhance turbulent debugging and support later use of the turbulence models. Pinkerman's new implementation has the capability to model streamlines, streamtubes, and crinkle plots. These feature and other enhanced visualization needs to be implemented in a manner that can applied to any geometry (case.nc3d) or solution files (single case.unk or multiple case.un#).

Inertial and Non-Inertial Turbulence Modeling. The SA model has been verified in the inertial frame using Rumsey's flat plate case. This process should be repeated for the SST model. More complex inertial cases should be modeled using the SA and SST models. Then both turbulence models should be tested in the non-inertial frame. Schiestel and Elena

(1997) present a simple cavity with rotating walls to test turbulence in rotating conditions.

One wall can be modeled using rotation in the non-inertial frame, and the opposing wall and boundary can be modeled as “stationary” using the velocity transpiration condition.

Verify Heat Transfer Model. The temperature boundary conditions and heat conduction terms need to be verified using simple conduction cases. Simple analytical solutions for planar walls can be found in Incropera and DeWitt (2002). The three boundary conditions (known temperature, known heat flux, and adiabatic wall) can be applied in different combinations and combined with the heat generation (quasi-combustion) terms to verify the heat transfer model in the presence of no fluid flow. The model can then be tested for external forced convection over a flat plate. This will fully verify the heat transfer model.

Improved Artificial Dissipation. A better artificial dissipation model needs to be found. The current dissipation model in the OSU codes is sufficient for inviscid solutions but overly diffuse in the near-wall region. The result is a stable solution with a thickened boundary layer and reduced skin friction. Neither of these influences is acceptable. A zero dissipation length has been applied to NS2D/3D to keep the current model until a new model is found. CFDsol applies a shock-capturing model that is overly diffuse everywhere within the flow, and the zero dissipation length cannot be applied to CFDsol because artificial dissipation is required to stabilize the solution, even in the near-wall region.

Both artificial dissipation models are *ad hoc* attempts at creating positive entropy growth everywhere in the solution. A more appropriate model would incorporate the Second Law of Thermodynamics to ensure a positive entropy growth, using entropy. Guermond (2011) develops an entropy-base artificial viscosity model for general conservation laws. Olson

(2012) uses the bulk viscosity to increase the second coefficient of viscosity in a direction fashion. Olson is able to accurately capture oblique shocks without causing excess dissipation within the boundary layer. Appendix I combines the concepts seen in Guermond and Olson into an efficient artificial dissipation scheme, where entropy production is assembled as a combination of the Navier-Stokes equations and artificial dissipation is added to cancel any entropy destruction calculated. After this or similar dissipation model has been implemented in the solvers, Brown's (2009) tests should be repeated to evaluate the effects of the new model on solution convergence.

Expand Turbulence Modeling Pallet. The turbulence modeling pallet should be enlarged to include Wilcox's k-w model (Wilcox, 1998 and 2002) and several k-e models (Jones and Launder, 1972 and 1973; Yakhot and Orszag, 1986 and 1992; Zhang and Orszag, 1998; Shih, et al., 1994 and 1995; KY-Chien, 1982). For improved time accuracy, DDES models are suggested (Mocket, 2009; Spalart, 2009). DDES often builds off of existing RANS models near the wall, such as SA or SST. DDES uses LES in the external flow, which requires a small change to the existing RANS model and an additional interpolation function.

Coupling between Structural, Rigid Body, and Propulsion. The current implementation of the structural (model elastic), rigid body, and propulsion models are isolated from each other, except through the fluid flow. The structure should feel the influence of inertial motion and propulsion system. For instance, the thrust created by turbojets and rockets should create bending and torsional deformations in the structure, and the vertical oscillations of the vehicle, say short period oscillations, should induce bending deformations. The structural connections can be created in the modal elastics by summing the engine and rocket momentum into the generalized forces (*get_force*) and creating mass matrices to

couple the elastic and rigid body dynamic systems. The introduction of mass matrices also couples the inertia of the structural deformations back into the rigid body dynamics. The forces and moments created by the propulsion models already feed into the rigid body.

The propulsion boundaries need to feel the structural deflections and velocities and rigid body motion. The mesh velocity has already been used to correct the propulsion boundaries on a per-element basis. The overall properties and fluxes are calculated with taking the relative motion of the vehicle into account. The mesh velocity and kinetic energy should be taken into account on average, integrating the overall property exchange over the entire boundary. This includes applying the average kinetic energy due to mesh velocity in the flux to properties conversion *flux_props*. Finally, the deformations and velocities induced by structural motion can be applied along the propulsion boundaries similar to the adaptations to the flow tangency equation. The mode shapes can be interpolated across the propulsion boundary to determine deformations at all nodes on the boundary. The normal velocity at the boundary can be corrected similar to flow tangency, and the boundary normals can be rotated locally and applied without actually deforming the boundary surface.

More Sophisticated Modeling. Improvements were discussed along with the assessment of the readiness of a MDA integration. The lower order MDA environment can be created after the following upgrades have been completed:

- Elastic boundaries need to be incorporated into the viscous solvers by moving the boundaries and deforming / remeshing the internal mesh.
- The structural and thermal were interconnected during the NASA contract, but the connections need to be made to complete the loop with the CFD solvers. The temper-

ature distribution is readily available between iterations in the safe mode restart case.rst file, and the modal displacement can be returned through the case.vec file for now.

- Turbojet boundaries still need to be implemented in CFDsol.
- A subroutine needs to be set aside in CFDsol to hard-code controls routines directly into the CFD solution.

The first bullet being the largest hurdle.

The higher order MDA environment will require much more work. The following requirements need to be met before assembling the higher order environment:

- The CFD temperatures (case.rst) were passed to the thermal FEA in the lower order model, but the FEA needs to return boundary heat fluxes to the CFD solver (case.tbc).
- A module needs to be implemented in the CFD solvers or externally to calculate the surface traction (viscous stresses and pressure) to pass to the structural FEA. The case.vec can still be used to pass back structural deformation as a single mode.
- The higher order turbojet can be created by passing properties for entire boundary between the CFD solvers used in this work and external CFD models. The inflow plane can be modeled by passing the flux and properties on the central CFD solution to the external module, which returns the pressure distribution across the inflow plane. For the exhaust plane, the central CFD gets fluxes and properties from the external solver and passes back the pressure along its boundary. If the external routine only passes fluxes, *flux_props* can be used to calculate the properties to construct the Riemann matrix.
- Higher order scramjet and rocket exhaust models can be integrated together using the above turbojet model as a guide.

- Hypersonic accuracy can be increased by adding variable specific heats, pressure dilation (turbulence), and real gas equations. Specific heats can be calculated on regions that advect downstream from boundaries, each corresponding to different temperature tables.

Improved Gradients. Only piece-wise constant gradients were tested in this work. Other gradients are also available but their accuracy needs to be tested. O'Neill (2011) suggests using a hybrid element approach to calculate a piece-wise linear gradient distribution. O'Neill uses a "plate stiffness" term to remove oscillations created by the near singular nature of the system. Edge-based gradients are used in *smooth1* and *smooth2*. These gradients are constructed using the weighting function for each segment. The weighted segments are constructed from the element inverse Jacobians and boundary normals in a similar manner to the hybrid method, so edge-based gradients may suffer from similar oscillations near the boundaries.

Many of the other methods require additional storage to pre-calculate, store, and track the gradients throughout the solution. Such storage would reduce the overall run time (eliminate repetitive calculation of stresses and heat flux) at the cost of additional memory requirement. If something is stored, this author suggests stored the strain tensor and divergence of velocity to reassemble into viscous stresses, while the entire heat flux and turbulent kinetic energy terms are stored in their final form. The strain tensor could then be used to assemble the curvature correction term in the Spalart-Shur correction in the turbulence models. The curvature term is currently neglected because the unsteady term is expensive and the derivative of piece-wise constant strain is zero on the current implementation. Storing the strain tensor (like the unknowns) would allow for its gradient to be calculated on each element and the unsteady terms to be quickly assembled.

REFERENCES

- Abbott, I.H.; A.E. von Doenhoff. *Theory of Wing Sections*. Dover Publications. New York. 1959.
- AGARD. "Validation of Computational Fluid Dynamics." Lisbon, Portugal, May 2-5, 1988. NATO Advisory Group for Aeronautical Research and Development. *AGARD-CP-437*. Vol. I. December 1988.
- Allen, D.H.; W.E. Heisler. *Introduction to Aerospace Structural Analysis*. John Wiley and Sons, Inc. 1985.
- Anderson, J.D. *Fundamentals of Aerodynamics*. 3rd Ed. McGraw-Hill. Boston. 2001.
- Arena, A.S. Unpublished Email Transmission. Smith-Hess Panel Code. March 2012.
- Arminjon, P.; A. Madrane. "Staggered Mixed Finite Volume/Finite Element Method for the Navier-Stokes Equations." *AIAA Journal*. Vol. 37. No. 12. Dec 1999. pp. 1558-1571.
- Babcock, D.; A.S. Arena. "Estimating Aircraft Stability Derivatives Through Finite Element Analysis." *AIAA Paper 2004-5174*. AIAA Atmospheric Flight Mechanics Conference and Exhibit, Providence, RI. 2004.
- Baker, A.J. *Finite Element Computational Fluid Mechanics*. Hemisphere Publishing Company. New York. 1983.
- Baldwin, B.S.; T.J. Barth. "A One-Equation Turbulence Transport Model for High Reynolds Number Wall-Bounded Flows." *NASA-TM-102847*. 1990.
- Barche, J. (Chair) "Experimental Data Base for Computer Program Assessment." *AGARD-AR-138*. Report of the Fluid Dynamics Panel, Work Group 04. Specialized Printing Services Limited. Loughton, Essex. May 1979.
- Barsoum, M.E.; A.N. Alexandrou. "Stable Finite Element Solutions for Fully Viscous Compressible Flows." *Finite Elements in Analysis and Design*. Vol. 19. 1995. pp. 69-87.

- Bathie, W.W. *Fundamentals of Gas Turbines*. 2nd Ed. John Wiley and Sons, Inc. New York. 1996.
- Baumann, C.E.; J.T. Oden. "A Discontinuous hp-Finite Element Method for the Euler and Navier-Stokes Equations." *International Journal for Numerical Methods in Fluids*. Vol. 31. 1999. pp. 79-95.
- Bercovier, M.; M. Engelman. "A Finite Element for the Numerical Solution of Viscous Incompressible Flows." *Journal of Computational Physics*. Vol. 30. 1979. pp. 181-201.
- Bers, L. "On the Circulatory Subsonic Flow of a Compressible Fluid Past a Circular Cylinder." *NACA TN 970*. 1945.
- Bertin, J.J.; R.M. Cummings. "Critical Hypersonic Aerothermodynamic Phenomena." *Annual Review of Fluid Mechanics*. Vol. 38. 2006. pp. 129-157.
- Bertin, J.J.; M.L. Smith. *Aerodynamics for Engineers*. 3rd Ed. Prentice-Hall. Upper Saddle River, NJ. 1998.
- Bonhaus, D.L. "A Higher Order Accurate Finite Element Method for Viscous Compressible Flow." *Ph.D. Dissertation*. Virginia Polytechnic Institute and State University. Nov. 1998.
- Bristeau, M.O.; R. Glowinski; L. Duto; J. Periaux; G. Roge. "Compressible Viscous Flow Calculations Using Compatible Finite Element Approximations." *International Journal for Numerical Methods in Fluids*. Vol. 11. 1990. pp. 719-749.
- Brenner, S.C.; L.R. Scott. *The Mathematical Theory of Finite Element Methods*. 2nd Ed. Springer-Verlag. New York. 2002.
- Brooks, A.N.; T.J.R. Hughes. "Streamline Upwind/Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equations." *Computer Methods in Applied Mechanics and Engineering*. Vol. 32. 1982. pp. 199-259.
- Brown, C.W. "Evaluating Integration Methods and Examining Sensitivity to Temporal and Spatial Variations in Euler2D." *Masters' Creative Component*. Oklahoma State University. 2009.
- Burbeau, A.; P. Sagaut. "Simulation of a Viscous Compressible Flow Past a Circular Cylinder with High-Order Discontinuous Galerkin Methods." *Computers & Fluids*. Vol. 31. 2002. pp. 867-889.

- Burnsnall, W.J.; L.K. Loftin. "Experimental Investigation of the Pressure Distribution about a Yawed Circular Cylinder in the Critical Reynolds Number Range." *NACA TN 2463*. 1951.
- Burnett, D.S. *Finite Element Analysis: From Concepts to Applications*. Addison-Wesley Publishing Company. Reading, Massachusetts. 1987.
- Catris, S.; B. Aupoix. "Density Corrections for Turbulence Models." *Aerospace Science and Technology*. Vol. 4. 2000. pp. 1-11.
- Cebeci, T.; J.P. Shao; F. Kafyeke; E. Laurendeau. *Computational Fluid Dynamics for Engineers*. Horizons Publishing, Inc. Long Beach, CA. 2005.
- Cheng, H.K. "Perspectives on Hypersonic Viscous Flow Research." *Annual Review of Fluid Mechanics*. Vol. 25. 1993. pp. 445-84.
- Cheng, L.; R.D. White; K. Gresh. "Three-Dimensional Viscous Finite Element Formulation for Acoustic Fluid-Structure Interaction." *Computer Methods in Applied Mechanics and Engineering*. Vol. 197. 2008. pp. 4160-4172.
- Chien, K.Y. "Predictions of Channel and Boundary Layer Flows with a Low-Reynolds-Number Turbulence Models." *AIAA Journal*. Vol. 20. No. 1. 1982. pp. 33-38.
- Cowan, T. "Finite Element CFD Analysis of Super-Maneuvering and Spinning Structures." *Ph.D. Dissertation*. Oklahoma State University. 2003.
- Cowan, T.J.; C.R. O'Neill; A.S. Arena. "Transpiration Boundary Condition for Computational Fluid Dynamics Solutions in a Non-Inertial Reference Frame." *Journal of Aircraft*. Vol. 41, No. 5. Sept 2004. pp. 1252-1255.
- Curran, E.T.; W.H. Heiser; D.T. Pratt. "Fluid Phenomena in Scramjet Combustion Systems." *Annual Review of Fluid Mechanics*. Vol. 28. 1996. pp. 323-60.
- Dacles-Mariani, J.; G.G. Zilliac; J.S. Chow; P. Bradshaw. "Numerical/Experimental Study of a Wingtip Vortex in the Near Field." *AIAA Journal*. Vol. 33. No. 9. 1995. pp. 1561-1568.
- de Sampaio, P.A.B. "A Finite Element Formulation for Transient Incompressible Viscous Flows Stabilized by Local Time-Steps." *Computer Methods in Applied Mechanics and Engineering*. Vol. 194. 2005. pp. 2095-2108.
- Deck, S.; P. Duveau; P. d'Espiney; P. Guillen, "Development and Application of Spalart-Allmaras One Equation Turbulence Model to Three-Dimensional Supersonic Complex Configurations," *Aerospace Science and Technology*. Vol. 6. 2002. pp. 171-183.

- Dolejsi, V. "On the Discontinuous Galerkin Method for the Numerical Solution of the Navier-Stokes Equations." *International Journal for Numerical Methods in Fluids*. Vol. 45. 2004. pp. 1083-1106.
- Drela, M. "XFOIL: An Analysis and Design System for Low Reynolds Number Airfoils." *Low Reynolds Number Airfoils*. Lectures Notes in Engineering. Vol. 54. Springer-Verlag. 1989.
- Drela, M. *XFOIL: Subsonic Airfoil Development System*. <http://web.mit.edu/drela/Public/web/xfoil/>. April 7, 2008.
- Dubois, F.; G. Mehlamn. "Nonparameterized 'Entropy Fix' for Roes Method." *AIAA Journal*. Vol 31. No. 1. 1993. p.199-200.
- Elmiligui, A.; K.S. Abdol-Hamid; S.J. Massey; S.P. Pao. "Numerical Study of Flow Past a Circular Cylinder using RANS, Hybrid RANS/LES, and PANS Formulations." *AIAA Paper 2004-4959*. 2004.
- Emmons, H. W. *Fundamentals of Gas Dynamics*. Princeton University Press. 1958.
- Fan, S.; B. Lakshminarayana; M. Barnett. "A Low Reynolds Number $k-\varepsilon$ Model for Unsteady Turbulent Boundary Layer Flows." *AIAA Journal*. Vol. 31. No. 10. 1993. pp. 1777-1784.
- Farokhi, S. *Aircraft Propulsion*. John Wiley & Sons, Inc. New Jersey. 2009.
- Fernandez, G.; M. Hafez. "Finite Element Solution of Compressible Navier-Stokes Equations." *Advances in Finite Element Analysis in Fluid Dynamics*. FED-Vol. 123. ASME. 1991.
- Ferri, A.; A. Libby; V. Zakkay. "Theoretical and Experimental Investigation of Supersonic Combustors." *High Temperatures in Aeronautics*. Editor: C. Ferrari. Pergamon Press, Oxford. 1964.
- Fisher, C.C.; A. S. Arena. "On The Transpiration Method for Efficient Aeroelastic Analysis Using an Euler Solver." *AIAA Paper 96-3436*. AIAA Atmospheric Flight Mechanics Conference, San Diego, CA. July 1996.
- Fluent 6.3 User Guide*. "Chapter 12: Modeling Turbulence." Fluent, Inc. 2006.
- Gai, S.L.; The, S.L. "Interaction Between a Conical Shock Wave and a Plane Turbulent Boundary Layer." *AIAA Journal*. Vol. 38, No. 5. 2000. pp. 804-811.
- Givoli, D. "Non-Reflecting Boundary Conditions." Review Article. *Journal of Computational Physics*. Vol. 94. 1991. pp. 1-29.

- Godon, P.; S. Giora. "A Two-Dimensional Time Dependent Chebyshev Method of Collocation for the Study of Astrophysical Flows." *Computer Methods in Applied Mechanics and Engineering*. Vol. 110. 1993. pp. 171-194.
- Gorski, J.J. "Application of the David Taylor Navier-Stokes (DTNS) Code in Non-Inertial Reference Frames." *CDNSWC-SHD-1362-01*. 1992.
- Gowan, F.E.; E.W. Perkins. "Drag of Circular Cylinders for a Wide Range of Reynolds Numbers and Mach Numbers." *NACA TN 2960*. 1953.
- Guermond, J.; R. Pasquetti; B. Popov. "Entropy Viscosity Method for Nonlinear Conservation Laws." *Journal of Computational Physics*. Vol. 230. No. 11. May 2011. pp. 4248-4267.
- Gulcat, U.; A.R. Aslan. "Accurate 3D Viscous Incompressible Flow Calculations with the FEM." *International Journal for Numerical Methods in Fluids*. Vol. 25. 1997. pp. 985-1001.
- Gupta, K.K.; C. Bach. "Computational Fluid Dynamics-Based Aeroservoelastic Analysis with Hyper-X Applications." *AIAA Journal*. Vol. 45. No. 7. 2007. pp. 1459-1471.
- Hassett, A. *Thesis*. Oklahoma State University. Not yet published.
- Heinrich, J.C.; R.S. Marshall. "Viscous Incompressible Flow by a Penalty Function Finite Element Method." *Computers and Fluids*. Vol. 9. 1981. pp. 73-83.
- Hellsten, A. "Some Improvements in Menter's k- ω SST Turbulence Model." *AIAA-98-2554*. 29th AIAA Fluid Dynamics Conference, Albuquerque, NM. June 1997
- Hoffman, G.H. "Improved Form of the Low Reynolds Number k- ϵ Turbulence Model." *Physics of Fluids*. Vol. 18. 1975. pp. 309-312.
- Incropera, F.P.; D.P. DeWitt. *Introduction to Heat Transfer*. 4th Ed. John Wiley and Sons. New York. 2002.
- Ivings, M.J.; D.M. Causon; E.F. Toro. "On Riemann Solvers for Compressible Liquids." *International Journal for Numerical Methods in Fluids*. Vol. 28. 1998. pp. 395-418.
- Jakobsen, L.A.; H. Moller; E. Lund. "Sensitivity Analysis of Convection Dominated Steady 2D Fluid Flow Using SUPG FEM." *AIAA-2000-4822*. 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization. Long Beach, Ca. 2000.
- Jiang, B.-N.; L.A. Povinelli. "Least-Squares Finite Element Method for Fluid Dynamics." *Computer Methods in Applied Mechanics and Engineering*. Vol. 81. 1990. pp. 13-37.

- Jinyun, Yu. "Symmetric Gaussian Quadrature Formulae for Tetrahedral Regions." *Computer Methods in Applied Mechanics and Engineering*. Vol. 43. 1984. pp. 349-353.
- John, J.E.A. *Gas Dynamics*. 2nd Ed. Prentice Hall. Englewood Cliffs, New Jersey. 1984.
- Johnson, D.A.; L.S. King. "A Mathematically Simple Turbulence Closure Model for Attached and Separated Turbulent Boundary Layers." *AIAA Journal*. Vol 23. 1985. pp. 1684-1692.
- Jones, G.W.; J.J. Cincotta; R.W. Walker. "Aerodynamic Forces on a Stationary and Oscillating Circular Cylinder at High Reynolds Numbers." *NASA TR R-300*. 1969.
- Jones, R.T. "The Unsteady Lift of a Wing of Finite Aspect Ratio." *NACA Report 681*. 1940.
- Jones, W.P.; B.E. Launder. "The Prediction of Laminarization with a Two-Equation Model of Turbulence." *International Journal of Heat and Mass Transfer*. Vol. 15. 1972. pp. 301-314.
- Jones, W.P.; Launder, B.E. "The Calculation of Low-Reynolds-Number Phenomena with a Two-Equation Model of Turbulence." *International Journal of Heat and Mass Transfer*. Vol. 16. 1973. pp. 1119-1130.
- Kallinderis, Y.; K. Nakajima. "Finite Element Method for Incompressible Viscous Flows with Adaptive Hybrid Grids." *AIAA Journal*. Vol. 32. No. 8. 1994. pp.1617-1625.
- Kashiyama, K.; T. Tamai; W. Inomata; S. Yamaguchi. "A Parallel Finite Element Method for Incompressible Navier-Stokes Flow Based on Unstructured Grids." *Computer Methods in Applied Mechanics and Engineering*. Vol. 190. 2000. pp. 333-344.
- Katz, J.; A. Plotkin. *Low-Speed Aerodynamics: From Wing Theory to Panel Methods*. McGraw-Hill. New York. 2001.
- Kays, W.; M. Crawford; B. Weigand. *Convective Heat and Mass Transfer*. 4th Ed. McGraw-Hill. Boston. 2005.
- Kim, S.D.; D.J. Song. "Modified Shear-Stress Transport Turbulence Model for Supersonic Flows." *Journal of Aircraft*. Vol. 42, No. 5. Sept-Oct 2005. pp. 1118-1125.
- Kinsler, L.E.; A.R. Frey; A.B. Coppens; J.V. Sanders. *Fundamentals of Acoustics*. 4th Ed. John Wiley & Sons, Inc. New York. 2000.
- Kok, J.C. "Resolving the Dependence on Freestream Values for the $k-\omega$ Turbulence Model." *NLR-TP-99295*. 1999.

- Krishnamurthy, V.S.; W. Shyy. "Study of Compressibility Modifications to the $k-\varepsilon$ Turbulence Model." *Physics of Fluids*. Vol 9. No. 9. 1997. pp. 2769-2788.
- Kuhn, G.D. "Calculation of Compressible Nonadiabatic Boundary Layers in Laminar, Transitional, and Turbulent Flow by the Method of Integral Relations." *NASA CR 1797*. 1971.
- Lam, C.K.G.; K. Bremhorst. "Modified Form of $k-\varepsilon$ Model for Predicting Wall Turbulence." *Journal of Fluids Engineering*. Vol. 103. 1981. pp. 456-460.
- Launder, B.E.; G.J. Reece; W. Rodi. "Progress in the Development of a Reynolds-Stress Turbulence Closure." *Journal of Fluid Mechanics*. Vol. 68. Pt. 3. 1975. pp. 527-566.
- Launder, B.E.; B.I. Sharma. "Application of the Energy Dissipation Model of Turbulence to the Calculation of Flow near a Spinning Disc." *Letters in Heat and Mass Transfer*. Vol. 1. No. 2. 1974. pp. 131-138.
- Lee, B.H.K. "Oscillatory Shock Motion Caused by Transonic Shock Boundary-Layer Interaction." *AIAA Journal*. Vol. 28. No. 5. 1990. pp. 942-944.
- LeVeque, R.J. *Numerical Methods for Conservation Laws*. 2nd Ed. Birkhauser Verlag, Basel, Germany. 1992.
- Li, B.Q. *Discontinuous Finite Element Fluid Dynamics and Heat Transfer*. Springer. London. 2006.
- Lorin, E.; A.B.H. Ali; A. Soulaïmani. "A Positivity Preserving Finite Element – Finite Volume Solver for the Spalart-Allmaras Turbulence Model." *Computational Methods in Applied Mechanical Engineering*. Vol. 196. 2007. pp. 2097-2116.
- Luo, H.; J.D. Baum; R. Lohner. "A Fast, Matrix-free Implicit Method for Compressible Flows on Unstructured Grids." *Journal of Computational Physics*. Vol. 146. 1998. pp. 664-690.
- Mani, M.; J.A. Ladd; W. Bower. "Rotation and Curvature Correction Assessment for One- and Two-Equation Turbulence Models." *Journal of Aircraft*. Vol. 41, No. 2. March-April 2004. pp. 268-273.
- Marcum, D.L.; R.K. Agarwal. "Finite Element Navier-Stokes Solver for Unstructured Grids." *AIAA Journal*. Vol. 30. No. 3. 1992. pp. 648-654.
- Massarotti, N.; P. Nithiarasu; O.C. Zienkiewicz. "Characteristic-Based-Split (CBS) Algorithm for Incompressible Flow Problems with Heat Transfer." *International*

- Journal of Numerical Methods for Heat & Fluid Flow*. Vol. 8. No. 8. 1998. pp. 969-990.
- Masud, A.; T.J.R. Hughes. "A Space-Time Galerkin Least-Squares Finite Element Formulation of the Navier-Stokes Equations for Moving Domain Problems." *Computational Methods in Applied Mechanics and Engineering*. Vol. 146. 1997. pp. 91-126.
- Mattingly, J.D. *Elements of Gas Turbine Propulsion*. McGraw-Hill, Inc. New York. 1996.
- McCormick, B.W. *Aerodynamics, Aeronautics, and Flight Mechanics*. John Wiley & Sons. New York. 1994.
- Menter, F.R. "Improved Two-Equation $k-\omega$ Turbulence Models for Aerodynamics Flows." *NASA-TM-103975*. 1992.
- Menter, F.R. "Influence of Freestream Values on $k-\omega$ Turbulence Model Predictions." *AIAA Journal*. Vol 30. No. 6. 1992. pp. 1657-1659.
- Menter, F.R. "Zonal Two Equation $k-\omega$ Turbulence Models for Aerodynamic Flows." *AIAA Paper 93-2906*. July 1993.
- Menter, F.R. "Two-Equation Eddy-Viscosity Turbulence Models for Engineering Applications." *AIAA Journal*. Vol. 32. No. 8. 1994. pp. 1598-1605.
- Menter, F.R.; M. Kuntz; R. Langtry. "Ten Years of Industrial Experience with the SST Turbulence Model." *Turbulence, Heat, and Mass Transfer*. Vol. 4. 2003. pp. 625-632.
- Mittal, S. "Finite Element Computation of Unsteady Viscous Compressible Flows." *Computer Methods in Applied Mechanics and Engineering*. Vol. 157. 1996. pp. 151-175.
- Mocket, C. *A Comprehensive Study of Detached-Eddy Simulation*. Univerlagtuberlin. 2009.
- Moffitt, N.J. "First Stages of a Viscous Finite Element Solver for Non-Inertial and Aeroelastic Problems." *Masters' Thesis*. Oklahoma State University. 2004.
- Mohr, G.A. "Finite Element Analysis of Viscous Flows." *Computational Techniques & Applications: CTAC-83*. (J. Noye & C. Fletcher, Eds.) Elsevier Science Publishers. North-Holland. 1984.
- Moran, M.J.; H.N. Shapiro. *Fundamentals of Engineering Thermodynamics*. 4th Ed. John Wiley and Sons, Inc. New York. 2000.

- Munson, B.R.; D.F. Young; T.H. Okiishi. *Fundamentals of Fluid Mechanics*. 3rd Ed. John Wiley & Sons, Inc. New York. 1998.
- N'dri, D.; A. Garon; A. Fortin. "Stable Space-Time Formulation for the Navier-Stokes Equations." *AIAA-2000-0394*. 38th AIAA Aerospace Sciences Meeting and Exhibit. 2000.
- Nelson, R.C. *Flight Stability and Automatic Control*. 2nd Ed. McGraw-Hill Companies, Inc. Boston. 1998.
- Nesliturk, A.I.; S.H. Aydin; M. Tezer-Sezgin. "Two-Level Finite Element Method with a Stabilizing Subgrid for the Incompressible Navier-Stokes Equations." *International Journal for Numerical Methods in Fluids*. Vol. 58. 2008. pp. 551-572.
- Nicoud, F.; F. Ducros. "Subgrid-Scale Modeling Based on the Square of the Velocity Gradient Tensor." *Flow, Turbulence and Combustion*. Vol 62. 1999. pp. 183-200.
- Nithiarasu, P.; O.C. Zienkiewicz; B.V.K. Satya Sai; K. Morgan; R. Codina; M. Vazquez. "Shock Capturing Viscosity for the General Fluid Mechanics Algorithm." *International Journal for Numerical Methods in Fluids*. Vol 28. 1998. pp. 1325-1353.
- Oliver, T.A. "A High-Order, Adaptive Discontinuous Galerkin Finite Element Method for the Reynolds-Averaged Navier-Stokes Equations." *Ph.D. Thesis*. Massachusetts Institute of Technology. Sept. 2008.
- Olson, B.J.; S.K. Lele. "Directional Artificial Bulk Viscosity for Shock Capturing on High Aspect Ratio Grids." *Computational Science and Discovery*. Vol. 5. July 2012.
- O'Neill, C.R. "Higher Order and Dynamic CFD for Aeroelastic Simulations." *Dissertation*. Oklahoma State University. 2011.
- O'Neill, C.R.; A.S. Arena. "Comparison of Time Domain Training Signals for CFD Based Aerodynamic Identification." *AIAA Paper 04-0209*. AIAA 42th Aerospace Sciences Meeting and Exhibit. Reno, NV. January 2004.
- O'Neill, C.R.; A.S. Arena. "Aircraft Flight Dynamics with a Non-Inertial CFD Code." *AIAA Paper 2005-0230*. AIAA 43th Aerospace Sciences Meeting and Exhibit. Reno, NV. January 2005.
- Ou, Y.-R.; J.A. Burns. "Optimal Control of Lift/Drag Ratio on a Rotating Cylinder." *NASA CR 187586 (or ICASE Report 91-49)*. 1991.
- Panaras, A.G. "Shear-Layer-Edge Interaction: Simulation by Finite-Area Vortices." *AIAA Journal*. Vol. 28. No. 8. 1990. pp. 1557-1564.

- Papadopoulos, P.; E. Venkatapathy; D. Prabhu; M.P. Loomis; D. Olynick. "Current Grid-Generation Strategies and Future Requirements in Hypersonic Vehicle Design, Analysis and Testing." *Applied Mathematical Modeling*. Vol. 23. 1999. pp. 705-735.
- Peiro, J.; J. Peraire; K. Morgan. *FELISA System Reference Manual: Basic Theory*. December 3, 1993.
- Pier, B. "Finite-Amplitude Crossflow Vortices, Secondary Instability and Transition in the Rotating-Disk Boundary Layer." *Journal of Fluid Mechanics*. Vol. 287. 2003. pp. 315-343.
- Pinkerman, C.W. "Advancement of Computational Fluid Dynamics Visualization for Finite Element Applications." *Thesis*. Oklahoma State University. 2010.
- Pontaza, J.P.; J.N. Reddy. "Least-Squares Finite Element Formulations for Viscous Incompressible and Compressible Fluid Flows." *Computer Methods in Applied Mechanics and Engineering*. Vol. 195. 2006. pp. 2454-2494.
- Pope, S.B. *Turbulent Flows*. Cambridge University Press. 2000.
- Reid, E.G. "Tests on Rotating Cylinders." *NACA TN 209*. 1924.
- Roache, P.J. *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers, Albuquerque, New Mexico. 1998.
- Roache, P.J. "Verification of Codes and Calculations." *AIAA Journal*. Vol. 36, No. 5. May 1998. pp. 696-702.
- Rubesin, M.W. "Extra Compressibility Terms for Favre-Averaged Two-Equation Models of Inhomogeneous Turbulent Flows." *NASA-CR-177556*. 1990.
- Rumsey, C.L. "Compressibility Considerations for $k-\omega$ Turbulence Models in Hypersonic Boundary Layer Applications." *NASA-TM-215705*. Apr 2009.
- Rumsey, C. <http://turbmodels.larc.nasa.gov> NASA Langley Research Center: Turbulence Modeling Resource. January 2012.
- Rumsey, C.L.; T. Nishino. "Numerical Study Comparing RANS and LES Approaches on a Circulation Control Airfoil." *International Journal of Heat and Fluid Flow*. Vol. 32. 2011. pp.847-864.
- Sadeh, W.Z.; H.J. Brauer. "A Visual Investigation of Turbulence in Stagnation Flow about a Circular Cylinder." *NASA CR 3019*. 1978.

- Sarkar, S.; G. Erlebacher; M.Y. Hussaini; H.O. Kreiss. "The Analysis and Modeling of Dilatational Terms in Compressible Turbulence." *Journal of Fluid Mechanics*. Vol 227. 1991. p. 473.
- Sarkar, S. "The Pressure Dilatation Correlation in Compressible Flows." *Physics of Fluids A*. Vol. 4. 1992. pp. 2674-2682.
- Schiestel, R.; L. Elena. "Modeling of Anisotropic Turbulence in Rapid Rotation." *Aerospace Science and Technology*. No. 7. 1997. pp. 441-451.
- Selig, M. *UIUC Airfoil Data Site*. <http://www.aae.uiuc.edu/m-selig/ads.html>. March 16, 2004.
- Schlichting, H. *Boundary Layer Theory*. 7th Ed. McGraw-Hill. New York. 1987.
- Shih, T.-H.; W.W. Liou; A. Shabbir; Z. Yang; J. Zhu. "A New k - ϵ Eddy-Viscosity Model for High Reynolds Number Turbulent Flows: Model Development and Validation." *Computers & Fluids*. Vol. 24. No. 3. 1995. p. 227-238.
- Shih, T.-H.; J. Zhu; J.L. Lumley. "A New Reynolds Stress Algebraic Equation Model." *NASA-TM-106644*. 1994.
- Shur, M.L.; M.K. Strelets; A.K. Travin; P.R. Spalart. "Turbulence Modeling in Rotating and Curved Channels: Assessing the Spalart-Shur Correction." *AIAA Journal*. Vol. 38, No. 5. 2000. pp. 784-792.
- Smagorinsky, J. "General Circulation Experiments with the Primitive Equations: I. The Basic Experiment." *Monthly Weather Review*. Vol. 91. 1963. pp. 99-164.
- Smirnov, P.E.; F.R. Menter. "Sensitization of the SST Turbulence Model to Rotation and Curvature by Applying the Spalart-Shur Correction Term." *Journal of Turbo-machinery*. Vol. 131. Oct 2009. pp. 1-9.
- Soulaimani, A.; M. Fortin. "Finite Element Solution of Compressible Viscous Flows Using Conservative Variables." *Computer Methods in Applied Mechanics and Engineering*. Vol. 118. 1994. pp. 319-350.
- Spalart, P.R. "Detached-Eddy Simulation." *Annual Review of Fluid Mechanics*. Vol. 41. 2009. pp. 181-202.
- Spalart, P.R. "Trends in Turbulence Treatments." *AIAA 2000-2306*. June 2000.
- Spalart, P.R.; S.R. Allmaras. "A One-Equation Turbulence Model for Aerodynamic Flows." *AIAA Paper 92-0439*. 30th AIAA Aerospace Sciences Meeting. Reno, NV. 1992.

- Spalart, P.R.; C.L. Rumsey. "Effective Inflow Conditions for Turbulence Models in Aerodynamic Calculations." *AIAA Journal*. Vol. 45. No. 10. 2007. pp. 2544-2553.
- Spalart, P.R.; M. Shur. "On the Sensitization of Turbulence Models to Rotation and Curvature." *Aerospace Science and Technology*. No. 5. 1997. pp. 297-302.
- Speziale, C.G.; S. Sarkar; T.B. Gatski. "Modeling the Pressure-Strain Correlation of Turbulence: An Invariant Dynamical Systems Approach." *Journal of Fluid Mechanics*. Vol. 227. 1991. pp. 245-272.
- Speziale, C.G. "Turbulence Modeling in Non-Inertial Frames of Reference." *Theoretical and Computational Fluid Dynamics*. Vol. 1. 1989. pp. 3-19.
- Stephens, C.H.; A.S. Arena. "Application of the Transpiration Method for Aeroservoelastic Prediction using CFD." *AIAA Paper 98-2071*. 39th AIAA/ ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Long Beach, CA. April 1998.
- Stewart, M.E.M.; A. Suresh; M.S. Liou; A.K. Owen; D.G. Messitt. "Multidisciplinary Analysis of a Hypersonic Engine." *NASA TM-2002-211971* or *AIAA-2002-5127*. 2002.
- Sukraw, M.R. "Validation of NS2D." *Masters' Creative Component*. Oklahoma State University. 2008.
- Swanson, R.C.; C.L. Rumsey. "Computation of Circulation Control Airfoil Flows." *Computers & Fluids*. Vol. 38. 2009. pp. 1925-1942.
- Tanahashi, T.; H. Okanaga; T. Saito. "GSMAC Finite Element Method for Unsteady Incompressible Navier-Stokes Equations at High Reynolds Numbers." *International Journal of Numerical Methods in Fluids*. Vol. 11. 1990. pp. 479-499.
- Tay, A.O.; G.D. Davis. "Application of the Finite Element Method to Convection Heat Transfer Between Parallel Planes." *International Journal of Heat and Mass Transfer*. Vol. 14. 1971. pp. 1057-1069.
- Theodorsen, T.; A. Regier. "Experiments on Drag of Revolving Disks, Cylinders, and Streamlined Rods at High Speeds." *NACA Report 793*. 1944.
- Thomasset, F. *Implementation of Finite Element Method for Navier-Stokes Equations*. Springer-Verlag. New York. 1981.
- Thomee, V. *Galerkin Finite Element Methods for Parabolic Problems*. Springer-Verlag. Berlin. 2006.

- Toro, E.F. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. Springer-Verlag, Berlin. 1997.
- Tworzydło, W.W.; J.T. Oden; E.A. Thornton. "Adaptive Implicit/Explicit Finite Element Method for Compressible Viscous Flow." *Computer Methods in Applied Mechanics and Engineering*. Vol. 95. 1992. pp. 397-440.
- Vanyo, J.P. *Rotating Fluids in Engineering and Science*. Butterworth-Heinemann. Boston. 1993.
- Wagner, H. "Über die Entstehung des Dynamischen Auftriebes von Tragflugeln." *ZAMM*. Vol. 5. 1925. pp. 17-35.
- Walters, B. "2D and 3D CFD Cases: A Test of 2D and 3D CFD Codes at Hypersonic Mach Numbers." *Independent Study*. (A.S. Arena.) Oklahoma State University. Spring 2009.
- Weick, F.E. *Aircraft Propeller Design*. McGraw-Hill Book Company, Inc. New York. 1930.
- White, F.M. *Viscous Fluid Flow*. 2nd Ed. McGraw-Hill. Boston. 1991.
- Whiting, C.H. "Stabilized Finite Element Methods for Fluid Dynamics using a Hierarchical Basis." *Ph.D. Dissertation*. Rensselaer Polytechnic University. Troy, New York. 1999.
- Wieselsberger, C. "New Data on the Laws of Fluid Resistance." NACA TN-84. March 1922. (Republished from: "Neuere Feststellungen Über die Gesetze des Flüssigkeits Luftwiderstands." *Phys. Z.* Vol. 22. 1921. pp. 321-328.)
- Wilcox, D.C.; M.W. Rubesin. "Progress in Turbulence Modeling for Complex Flow Fields including Effects of Compressibility." *NASA-TP-1517*. 1980.
- Wilcox, D.C. "Dilatation Dissipation Corrections for Advanced Turbulence Models." *AIAA Journal*. Vol. 30. No. 11. 1992. pp. 2639-2646.
- Wilcox, D.C. "Simulation of Transition with a Two Equation Turbulence Model." *AIAA Journal*. Vol. 32. No. 2. 1994. pp. 247-255.
- Wilcox, D.C. "Reassessment of the Scale Determining Equation for Advanced Turbulence Models." *AIAA Journal*. Vol. 26. No. 11. 1988. pp. 1299-1310.
- Wilcox, D.C. *Turbulence Modeling for CFD*. 2nd Ed. DCW Industries, Inc., 5354 Palm Drive, La Canada, CA. 2002.

- Williamson, C.H.K. "Vortex Dynamics in the Cylinder Wake." *Annual Review of Fluid Dynamics*. Vol. 28. 1996. pp. 477-539.
- Wood, B.; W. Loth; P. Geubelle; S. McIlwain. "A Numerical Methodology for an Aeroelastic SBLI Flow." *AIAA-2000-0552*. 38th Aerospace Sciences Meeting and Exhibit. Reno, Nevada. 2000.
- Yakhot, V.; Orszag, S.A. "Renormalization Group Analysis of Turbulence: 1. Basic Theory." *Journal of Scientific Computing*. Vol. 1. 1986. pp. 3-51.
- Yakhot, V.; S.A. Orszag; S. Thangam; T.B. Gatski; C.G. Speziale. "Development of Turbulence Models for Shear Flows by a Double Expansion Technique." *Physics of Fluids A*. Vol. 4. No. 7. 1992. pp. 1510-1520.
- Zhang, H.S.; R.M.C. So; C.G. Speziale; Y.G. Lai. "Near Wall Two-Equation Model for Compressible Turbulent Flows." *AIAA Journal*. Vol. 31. 1993. pp. 196-199.
- Zhang, Y.; S.A. Orszag. "Two-Equation RNG Transport Modeling of High Reynolds Number Pipe Flow." *Journal of Scientific Computing*. Vol. 13. No. 4. 1998. pp. 471-483.
- Zienkiewicz, O.C.; R.L. Taylor. *The Finite Element Method: Volume 3, Fluid Dynamics*. 5th Ed. Butterworth-Heinemann, Oxford. 2000.

APPENDIX A

STOKES' HYPOTHESIS

The dissipation term in the energy equation represents the energy dissipated by viscous effects. Since viscous effects cannot create energy, but instead can only dissipate energy, then the viscous dissipation term must be positive semi-definite. This term is given by:

$$\Phi = \mu [2u_x^2 + 2v_y^2 + 2w_z^2 + (v_x + u_y)^2 + (u_z + w_x)^2 + (v_z + w_y)^2] + \lambda (u_x + v_y + w_z)^2$$

Stokes' hypothesis says that second viscosity can be represented as a direction function of kinetic viscosity: $\lambda = -\frac{2}{3}\mu$. Recent findings of Karim and Rosenhear¹ include values of bulk viscosity greater than zero and some even greater than their corresponding kinetic viscosity. These experiments are a matter of controversy themselves (Truesdell²), but they seem to suggest that Stokes hypothesis should be $\lambda \geq -\frac{2}{3}\mu$. This can be shown to satisfy Stokes hypothesis by proving that viscous dissipation is non-negative ($\Phi \geq 0$) when:

$$\mu \geq 0 \quad \text{and} \quad 3\lambda + 2\mu \geq 0$$

¹ Karim, S. M., and L. Rosenhead (1952), "The Second Coefficient of Viscosity of Liquids and Gases," *Rev. Modern Phys.*, vol. 24, pp. 108-116.

² Truesdell, C. (1954), "The Present Status of the Controversy Regarding the Bulk Viscosity of Liquids," *Proc. Roy. Soc. London Ser. A*, vol. 226, pp. 1-69.

Note: This investigation stems from a discussion by White³ on the topic and a rough proof on the web by Steve Baum⁴. Further background should be sought on the dependency of bulk viscosity on the frequencies occurring in the flow. Landau and Lifshitz⁵ also give a discussion of the subject of second viscosity.

Investigation:

$$\Phi = \mu(2u_x^2 + 2v_y^2 + 2w_z^2 + (v_x + u_y)^2 + (u_z + w_x)^2 + (v_z + w_y)^2) + \lambda(u_x + v_y + w_z)^2 \geq 0$$

The fourth through sixth terms can be positive or negative (or zero for that matter). But after squaring the terms, all three terms are assured to be non-negative, as long as the kinetic viscosity is positive. This is a safe physical assumption to make. Therefore, considering $\mu \geq 0$, the previous viscous dissipation term reduces to:

$$\mu[2u_x^2 + 2v_y^2 + 2w_z^2] + \lambda(u_x + v_y + w_z)^2 \geq 0$$

or

$$(2\mu + \lambda)u_x^2 + (2\mu + \lambda)v_y^2 + (2\mu + \lambda)w_z^2 + 2\lambda u_x v_y + 2\lambda u_x w_z + 2\lambda v_y w_z \geq 0$$

This suggests the following as a general form:

$$A[u_x + \alpha(v_y + w_z)]^2 + B[v_y + \beta w_z]^2 + Cw_z^2 \geq 0$$

The first two terms in brackets and the final term (without the square) can again take on any value: positive, negative, or zero. But once the terms are squared, none of the three terms can be non-negative. Therefore, this general form is ensured to be non-negative, for any

³ White, F. M. (1991), *Viscous Fluid Flow*, McGraw-Hill, Boston, pp 67-68.

⁴ <http://stommel.tamu.edu/~baum/reid/book1/book/node102.html#SECTION00630000000000000000>

⁵ Landau, L. D., and E. M. Lifshitz (1959), *Fluid Mechanics*, Pergamon, London, sec. 78.

values of u_x , v_y , and w_z when A , B , and C are all non-negative. Expanding the general form to one similar to the modified viscous dissipation:

$$Au_x^2 + 2A\alpha u_x v_y + 2A\alpha u_x w_z + A\alpha^2 v_y^2 + 2A\alpha^2 v_y w_z + A\alpha^2 w_z^2 + Bv_y^2 + 2B\beta v_y w_z + B\beta^2 w_z^2 + Cw_z^2 \geq 0$$

or

$$Au_x^2 + (A\alpha^2 + B)v_y^2 + (A\alpha^2 + B\beta^2 + C)w_z^2 + 2A\alpha u_x v_y + 2A\alpha u_x w_z + (2A\alpha^2 + 2B\beta)v_y w_z \geq 0$$

Associating the general terms with those specific to the modified viscous dissipation:

$$\begin{aligned} A = 2\mu + \lambda & & A\alpha^2 + B = 2\mu + \lambda & & A\alpha^2 + B\beta^2 + C = 2\mu + \lambda \\ 2A\alpha = 2\lambda & & 2A\alpha^2 + 2B\beta = 2\lambda & & \end{aligned}$$

Using these five equations to solve for the five coefficients:

$$\begin{aligned} A = 2\mu + \lambda & & \alpha = \frac{\lambda}{A} = \frac{\lambda}{2\mu + \lambda} \\ B = 2\mu + \lambda - A\alpha^2 = \frac{4\mu(\mu + \lambda)}{2\mu + \lambda} & & \beta = \frac{\lambda}{B} - \frac{A}{B}\alpha^2 = \frac{\lambda}{2(\mu + \lambda)} \\ C = 2\mu + \lambda - A\alpha^2 - B\beta^2 = \frac{\mu(2\mu + 3\lambda)}{\mu + \lambda} & & \end{aligned}$$

In order for $A \geq 0$, then $2\mu + \lambda \geq 0$, which also makes the denominator of B non-negative. Therefore $B \geq 0$, when $\mu + \lambda \geq 0$, which, in turn, makes the denominator of C non-negative. Therefore $C \geq 0$, for $2\mu + 3\lambda \geq 0$. These three inequalities must all be true in order for the coefficients A , B , and C to be non-negative, and the viscous dissipation term ensured to be

non-negative. (The other two coefficients are not important because they exist within the squared terms of the general form of the equation. Being in the squared term, these two coefficients can take on any form and not affect the positive nature of the viscous dissipation term.) Solving all three inequalities for bulk viscosity, in terms of kinetic viscosity, the following three inequalities emerge:

$$\lambda \geq -2\mu \qquad \lambda \geq -\mu \qquad \lambda \geq -\frac{2}{3}\mu$$

The third inequality is the most restrictive. Therefore, if the third condition is met, both of the remaining conditions will also be met: Stokes' hypothesis. Stokes chose the lower limit of the possible values for his bulk viscosity. In conclusion, the viscous dissipation is positive semi-definite when kinetic viscosity is positive semi-definite and the second coefficient is greater than $-\frac{2}{3}$ times the first coefficient. Mathematically,

$$\Phi = \mu \left[2u_x^2 + 2v_y^2 + 2w_z^2 + (v_x + u_y)^2 + (u_z + w_x)^2 + (v_z + w_y)^2 \right] + \lambda (u_x + v_y + w_z)^2 \geq 0$$

$$\text{if and only if} \qquad \mu \geq 0 \qquad \text{and} \qquad \lambda \geq -\frac{2}{3}\mu$$

APPENDIX B

ANALYTICAL INTEGRALS

Fourth order Gauss quadrature should provide a perfect solution of the flux integrals, since the flux integrals are at most fourth order polynomials of the shape function vectors (in the boundary integrals). Instead of utilizing a fourth order Gauss quadrature, the FEM equations are evaluated here using exact, analytical integration methods¹.

For the triangle elements, the integral is written in terms of factorials of powers:

$$\int_0^1 \int_0^{(1-\xi_2)} (\xi_1^m \xi_2^n \xi_3^p) d\xi_1 d\xi_2 = \frac{m!n!p!}{(m+n+p+2)!} \quad (\text{B.1})$$

For the line elements, the integral can similarly be written:

$$\int_0^1 (\xi_1^m \xi_2^n) d\xi_1 = \frac{m!n!}{(m+n+1)!} \quad (\text{B.2})$$

Now, the flux integrals need to be written in a manner that utilizes Eqs. B.1 and B.2. The flux vectors **F** and **G** are represented as a combination of first, second, and third order terms. In other words, the mass and enthalpy fluxes are a series of second order values,

¹ Baker, A.J. *Finite Element Computational Fluid Mechanics*. Hemisphere Publishing Company. 1983.

while the momentum fluxes is a third order terms, with a first order pressure term at the appropriate location. The flux terms are represented in a shorthand notation:

$$\mathbf{F}(\phi) = \Phi_e \alpha_1 + \Phi_e \beta_1 \Phi_e \beta_2 + \Phi_e \theta_1 \Phi_e \theta_2 \Phi_e \theta_3, \quad (\text{B.3})$$

where α_i , β_i , and θ_i represent different primitive variables (i.e., $(\rho)_e$, $(u)_e$, $(v)_e$, $(p)_e$, and $(\rho e)_e$). Now, the flux integrals are calculated using the shorthand notation, and the correct primitive variables are then substituted back into the equations. The element flux integrals can be written:

$$\sum_e \int_{A_e} \left(\frac{\partial \Phi_e^T}{\partial x} \mathbf{F} \right) d\Omega = \sum_e \int_0^1 \int_0^{(1-\xi_2)} \left(\frac{\partial \Phi_e^T}{\partial x} \mathbf{F} \right) (2A_e) d\xi_1 d\xi_2 = \sum_e \begin{Bmatrix} A_{11} \\ A_{12} \\ A_{13} \end{Bmatrix} \int_0^1 \int_0^{(1-\xi_2)} \mathbf{F}(\phi) d\xi_1 d\xi_2 \quad (\text{B.4})$$

$$\sum_e \int_{A_e} \left(\frac{\partial \Phi_e^T}{\partial y} \mathbf{G} \right) d\Omega = \sum_e \int_0^1 \int_0^{(1-\xi_2)} \left(\frac{\partial \Phi_e^T}{\partial y} \mathbf{G} \right) (2A_e) d\xi_1 d\xi_2 = \sum_e \begin{Bmatrix} A_{21} \\ A_{22} \\ A_{23} \end{Bmatrix} \int_0^1 \int_0^{(1-\xi_2)} \mathbf{G}(\phi) d\xi_1 d\xi_2 \quad (\text{B.5})$$

Because the inverse Jacobian matrix A_{ij} is constant across a given element, these terms can be moved outside of the element integrals, so that only the flux term remains in the integral. Eq. B.3 is substituted into the integrals:

$$\int_0^1 \int_0^{(1-\xi_2)} \mathbf{F}(\phi) d\xi_1 d\xi_2 = \int_0^1 \int_0^{(1-\xi_2)} (\Phi_e \alpha_1 + \Phi_e \beta_1 \Phi_e \beta_2 + \Phi_e \theta_1 \Phi_e \theta_2 \Phi_e \theta_3) d\xi_1 d\xi_2 \quad (\text{B.6})$$

The first order term is easily integrated using the factorial-power method:

$$\int_0^1 \int_0^{(1-\xi_2)} (\Phi_e \alpha_1) d\xi_1 d\xi_2 = \left(\int_0^1 \int_0^{(1-\xi_2)} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix}^T d\xi_1 d\xi_2 \right) \alpha_1 = \frac{1}{6} \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix}^T \alpha_1 = \frac{1}{2} \bar{\alpha}_1 \quad (\text{B.7})$$

where the over bar represents the average property. The second order term requires some manipulation: $\Phi_e \beta_1 \Phi_e \beta_2 = (\beta_1^T \Phi_e^T)^T \Phi_e \beta_2$. The transpose of a scalar is the same scalar, so that $\Phi_e \beta_1 \Phi_e \beta_2 = \beta_1^T \Phi_e^T \Phi_e \beta_2$. Substituting this notation into the integral:

$$\int_0^1 \int_0^{(1-\xi_2)} (\Phi_e \beta_1 \Phi_e \beta_2) d\xi_1 d\xi_2 = \beta_1^T \left(\int_0^1 \int_0^{(1-\xi_2)} \begin{bmatrix} \xi_1^2 & \xi_1 \xi_2 & \xi_1 \xi_3 \\ \xi_1 \xi_2 & \xi_2^2 & \xi_2 \xi_3 \\ \xi_1 \xi_3 & \xi_2 \xi_3 & \xi_3^2 \end{bmatrix} d\xi_1 d\xi_2 \right) \beta_2 \quad (\text{B.8})$$

$$\int_0^1 \int_0^{(1-\xi_2)} (\Phi_e \beta_1 \Phi_e \beta_2) d\xi_1 d\xi_2 = \frac{1}{24} \beta_1^T \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \beta_2 = \frac{9\bar{\beta}_1 \bar{\beta}_2 + \beta_1 \cdot \beta_2}{24} \quad (\text{B.9})$$

The third order term requires similar manipulation. The transpose of the scalar remains the scalar, so that $\Phi_e \theta_1 \Phi_e \theta_2 \Phi_e \theta_3 = (\theta_1^T \Phi_e^T)^T \Phi_e \theta_2 \Phi_e \theta_3 = \theta_1^T \Phi_e^T \Phi_e \theta_2 \Phi_e \theta_3$. The transpose identity does not help any further, so a compact notation must be created:

$$\theta_1^T \Phi_e^T \Phi_e \theta_2 \Phi_e \theta_3 = \theta_1^T [\Phi_{e3}] (\theta_2) \theta_3 \quad (\text{B.10})$$

$$[\Phi_{e3}] = (\Phi_e^T \Phi_e) \Phi_e = \begin{bmatrix} \xi_1^2 \Phi_e & \xi_1 \xi_2 \Phi_e & \xi_1 \xi_3 \Phi_e \\ \xi_1 \xi_2 \Phi_e & \xi_2^2 \Phi_e & \xi_2 \xi_3 \Phi_e \\ \xi_1 \xi_3 \Phi_e & \xi_2 \xi_3 \Phi_e & \xi_3^2 \Phi_e \end{bmatrix} \quad (\text{B.11})$$

Each term of the matrix is then a vector in itself – a third-order tensor. The notation represents the multiplication of θ_3 times every inner vector, and then pre-multiplying the matrix by θ_1 and post-multi-plying by θ_2 . The compact notation is written in expanded form:

$$\theta_1^T [\Phi_{e3}] (\theta_2) \theta_3 = \theta_1^T \begin{bmatrix} \xi_1^2 \Phi_e \theta_3 & \xi_1 \xi_2 \Phi_e \theta_3 & \xi_1 \xi_3 \Phi_e \theta_3 \\ \xi_1 \xi_2 \Phi_e \theta_3 & \xi_2^2 \Phi_e \theta_3 & \xi_2 \xi_3 \Phi_e \theta_3 \\ \xi_1 \xi_3 \Phi_e \theta_3 & \xi_2 \xi_3 \Phi_e \theta_3 & \xi_3^2 \Phi_e \theta_3 \end{bmatrix} \theta_2 \quad (\text{B.12})$$

The third order integral term is evaluated:

$$\int_0^1 \int_0^{(1-\xi_2)} (\Phi_e \theta_1 \Phi_e \theta_2 \Phi_e \theta_3) d\xi_1 d\xi_2 = \theta_1^T \left(\int_0^1 \int_0^{(1-\xi_2)} [\Phi_{e3}] d\xi_1 d\xi_2 \right) (\theta_2) \theta_3 \quad (\text{B.13})$$

where the integral of $[\Phi_{e3}]$ is expanded below to show every term.

$$\int_0^1 \int_0^{(1-\xi_2)} [\Phi_{e3}] d\xi_1 d\xi_2 = \begin{bmatrix} \left\{ \begin{matrix} \xi_1^3 \\ \xi_1^2 \xi_2 \\ \xi_1^2 \xi_3 \end{matrix} \right\}^T & \left\{ \begin{matrix} \xi_1^2 \xi_2 \\ \xi_1 \xi_2^2 \\ \xi_1 \xi_2 \xi_3 \end{matrix} \right\}^T & \left\{ \begin{matrix} \xi_1^2 \xi_3 \\ \xi_1 \xi_2 \xi_3 \\ \xi_1 \xi_3^2 \end{matrix} \right\}^T \\ \left\{ \begin{matrix} \xi_1^2 \xi_2 \\ \xi_1 \xi_2^2 \\ \xi_1 \xi_2 \xi_3 \end{matrix} \right\}^T & \left\{ \begin{matrix} \xi_1 \xi_2^2 \\ \xi_2^3 \\ \xi_2^2 \xi_3 \end{matrix} \right\}^T & \left\{ \begin{matrix} \xi_1 \xi_2 \xi_3 \\ \xi_2^2 \xi_3 \\ \xi_1 \xi_3^2 \end{matrix} \right\}^T \\ \left\{ \begin{matrix} \xi_1^2 \xi_3 \\ \xi_1 \xi_2 \xi_3 \\ \xi_1 \xi_3^2 \end{matrix} \right\}^T & \left\{ \begin{matrix} \xi_1 \xi_2 \xi_3 \\ \xi_2^2 \xi_3 \\ \xi_1 \xi_3^2 \end{matrix} \right\}^T & \left\{ \begin{matrix} \xi_1 \xi_3^2 \\ \xi_2 \xi_3^2 \\ \xi_3^3 \end{matrix} \right\}^T \end{bmatrix} \quad (\text{B.14})$$

$$\int_0^1 \int_0^{(1-\xi_2)} [\Phi_{e3}] d\xi_1 d\xi_2 = \frac{1}{120} \begin{bmatrix} \left\{ \begin{matrix} 6 \\ 2 \\ 2 \end{matrix} \right\}^T & \left\{ \begin{matrix} 2 \\ 2 \\ 1 \end{matrix} \right\}^T & \left\{ \begin{matrix} 2 \\ 1 \\ 2 \end{matrix} \right\}^T \\ \left\{ \begin{matrix} 2 \\ 2 \\ 1 \end{matrix} \right\}^T & \left\{ \begin{matrix} 2 \\ 6 \\ 2 \end{matrix} \right\}^T & \left\{ \begin{matrix} 1 \\ 2 \\ 2 \end{matrix} \right\}^T \\ \left\{ \begin{matrix} 2 \\ 1 \\ 2 \end{matrix} \right\}^T & \left\{ \begin{matrix} 1 \\ 2 \\ 2 \end{matrix} \right\}^T & \left\{ \begin{matrix} 2 \\ 2 \\ 6 \end{matrix} \right\}^T \end{bmatrix} \quad (\text{B.15})$$

The tensor (matrix of vectors) above is introduced back into equation:

$$\begin{aligned} & \int_0^1 \int_0^{(1-\xi_2)} (\Phi_e \theta_1 \Phi_e \theta_2 \Phi_e \theta_3) d\xi_1 d\xi_2 \\ &= \frac{1}{120} \left(27 \bar{\theta}_1 \bar{\theta}_2 \bar{\theta}_3 + 3 \bar{\theta}_1 (\theta_2 \cdot \theta_3) + 3 \bar{\theta}_2 (\theta_1 \cdot \theta_3) + 3 \bar{\theta}_3 (\theta_1 \cdot \theta_2) + 2 \sum_3 \theta_{1,i} \theta_{2,i} \theta_{3,i} \right) \end{aligned} \quad (\text{B.16})$$

Using the shorthand integrals above, the two element integrals containing the flux vectors \mathbf{F} and \mathbf{G} are written:

$$\sum_e \int_{A_e} \left(\frac{\partial \Phi_e^T}{\partial x} \mathbf{F} \right) dV = \sum_e \left\{ \begin{matrix} A_{11} \\ A_{12} \\ A_{13} \end{matrix} \right\} \int_0^1 \int_0^{(1-\xi_2)} \left\{ \begin{matrix} \Phi_e \rho_e \Phi_e u_e \\ \Phi_e \rho_e (\Phi_e u_e)^2 + \Phi_e p_e \\ \Phi_e \rho_e \Phi_e u_e \Phi_e v_e \\ (\Phi_e (\rho e)_e + \Phi_e p_e) \Phi_e u_e \end{matrix} \right\} d\xi_1 d\xi_2 = \sum_e \left\{ \begin{matrix} A_{11} \\ A_{12} \\ A_{13} \end{matrix} \right\} \left\{ \begin{matrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{matrix} \right\} \quad (\text{B.17})$$

$$\sum_e \int_{A_e} \left(\frac{\partial \Phi_e^T}{\partial y} \mathbf{G} \right) dV = \sum_e \left\{ \begin{matrix} A_{21} \\ A_{22} \\ A_{23} \end{matrix} \right\} \int_0^1 \int_0^{(1-\xi_2)} \left\{ \begin{matrix} \Phi_e \rho_e \Phi_e v_e \\ \Phi_e \rho_e \Phi_e u_e \Phi_e v_e \\ \Phi_e \rho_e (\Phi_e v_e)^2 + \Phi_e p_e \\ (\Phi_e (\rho e)_e + \Phi_e p_e) \Phi_e v_e \end{matrix} \right\} d\xi_1 d\xi_2 = \sum_e \left\{ \begin{matrix} A_{21} \\ A_{22} \\ A_{23} \end{matrix} \right\} \left\{ \begin{matrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{matrix} \right\} \quad (\text{B.18})$$

where

$$F_1 = \frac{9\bar{\rho}\bar{u} + \rho_e \cdot u_e}{24} \quad F_4 = \frac{9(\bar{\rho e} + \bar{p})\bar{u} + ((\rho e)_e + p_e) \cdot u_e}{24} \quad (\text{B.19})$$

$$G_1 = \frac{9\bar{\rho}\bar{v} + \rho_e \cdot v_e}{24} \quad G_4 = \frac{9(\bar{\rho e} + \bar{p})\bar{v} + ((\rho e)_e + p_e) \cdot v_e}{24} \quad (\text{B.20})$$

$$F_2 = \frac{1}{120} \left(27\bar{\rho}\bar{u}^2 + 3\bar{\rho}(u_e \cdot u_e) + 18\bar{u}\bar{\rho u} + 2\sum_3 \rho_i u_i^2 \right) + \frac{1}{2}\bar{p} \quad (\text{B.21})$$

$$G_3 = \frac{1}{120} \left(27\bar{\rho}\bar{v}^2 + 3\bar{\rho}(v_e \cdot v_e) + 18\bar{v}\bar{\rho v} + 2\sum_3 \rho_i v_i^2 \right) + \frac{1}{2}\bar{p} \quad (\text{B.22})$$

$$F_3 = G_2 = \frac{1}{120} \left(27\bar{\rho}\bar{u}\bar{v} + 3\bar{\rho}(u_e \cdot v_e) + 9\bar{u}\bar{\rho v} + 9\bar{v}\bar{\rho u} + 2\sum_3 \rho_i u_i v_i \right) \quad (\text{B.23})$$

The boundary integral equations can be evaluated in one pass because the flux terms are combined into a single normal flux term. The boundary integrals do not simply follow the pattern shown above because their integrands include one more shape function vector making each term one order higher:

$$\sum_{be} \int_{l_{be}} \Phi_{be}^T \mathbf{F}_n(\phi_{be}) d\Gamma = \sum_{be} l_{be} \int_0^1 \Phi_{be}^T \mathbf{F}_n(\phi_{be}) d\xi_1 \quad (\text{B.24})$$

where

$$\mathbf{F}_n(\phi_{be}) = \begin{Bmatrix} \Phi_{be} \rho_{be} \Phi_{be} V_{n,be} \\ \Phi_{be} \rho_{be} \Phi_{be} u_{be} \Phi_{be} V_{n,be} + \Phi_{be} p_{be} \hat{n}_{x,be} \\ \Phi_{be} \rho_{be} \Phi_{be} v_{be} \Phi_{be} V_{n,be} + \Phi_{be} p_{be} \hat{n}_{y,be} \\ \Phi_{be} V_{n,be} (\Phi_{be} (\rho e)_{be} + \Phi_{be} p_{be}) \end{Bmatrix} \quad (\text{B.25})$$

$$V_{n,be} = u_{be} \hat{n}_x + v_{be} \hat{n}_y \quad (\text{B.26})$$

Like the element integrals above, the boundary flux is rewritten using shorthand:

$$\mathbf{F}_n(\phi_{be}) = \Phi_{be} \alpha_1 + \Phi_{be} \beta_1 \Phi_{be} \beta_2 + \Phi_{be} \theta_1 \Phi_{be} \theta_2 \Phi_{be} \theta_3 \quad (\text{B.27})$$

The boundary integrals are evaluated using the shorthand:

$$\int_0^1 \Phi_{be}^T \mathbf{F}_n(\phi_{be}) d\xi_1 = \int_0^1 \Phi_{be}^T (\Phi_{be} \alpha_1 + \Phi_{be} \beta_1 \Phi_{be} \beta_2 + \Phi_{be} \theta_1 \Phi_{be} \theta_2 \Phi_{be} \theta_3) d\xi_1 \quad (\text{B.28})$$

The second order term is evaluated:

$$\int_0^1 (\Phi_{be}^T \Phi_{be} \alpha_1) d\xi_1 = \left(\int_0^1 \begin{bmatrix} \xi_1^2 & \xi_1 \xi_2 \\ \xi_1 \xi_2 & \xi_2^2 \end{bmatrix} d\xi_1 \right) \alpha_1 = \frac{1}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \alpha_1 \quad (\text{B.29})$$

The third order term is evaluated, using the 3rd-order compact notation:

$$\int_0^1 (\Phi_{be}^T \Phi_{be} \beta_1 \Phi_{be} \beta_2) d\xi_1 = \int_0^1 ([\Phi_{be3}] (\beta_1) \beta_2) d\xi_1 = \int_0^1 [\Phi_{be3}] d\xi_1 (\beta_1) \beta_2 \quad (\text{B.30})$$

$$[\Phi_{be3}] = (\Phi_{be}^T \Phi_{be}) \Phi_{be} = \begin{bmatrix} \xi_1^2 \Phi_{be} & \xi_1 \xi_2 \Phi_{be} \\ \xi_1 \xi_2 \Phi_{be} & \xi_2^2 \Phi_{be} \end{bmatrix} \quad (\text{B.31})$$

The notation represents the multiplication of β_2 times every inner vector, and then post-multiplying the matrix by β_1 . The compact notation is written in expanded form:

$$[\Phi_{be3}] (\beta_1) \beta_2 = \begin{bmatrix} \xi_1^2 \Phi_{be} \beta_2 & \xi_1 \xi_2 \Phi_{be} \beta_2 \\ \xi_1 \xi_2 \Phi_{be} \beta_2 & \xi_2^2 \Phi_{be} \beta_2 \end{bmatrix} \beta_1 \quad (\text{B.32})$$

The remaining integral becomes:

$$\int_0^1 [\Phi_{be3}] d\xi_1 = \int_0^1 \begin{bmatrix} \left\{ \begin{matrix} \xi_1^3 \\ \xi_1^2 \xi_2 \end{matrix} \right\} & \left\{ \begin{matrix} \xi_1^2 \xi_2 \\ \xi_1 \xi_2^2 \end{matrix} \right\} \\ \left\{ \begin{matrix} \xi_1^2 \xi_2 \\ \xi_1 \xi_2^2 \end{matrix} \right\} & \left\{ \begin{matrix} \xi_1 \xi_2^2 \\ \xi_2^3 \end{matrix} \right\} \end{bmatrix} d\xi_1 = \frac{1}{12} \begin{bmatrix} \left\{ \begin{matrix} 3 \\ 1 \\ 1 \\ 1 \end{matrix} \right\} & \left\{ \begin{matrix} 1 \\ 1 \\ 1 \\ 3 \end{matrix} \right\} \end{bmatrix} \quad (\text{B.33})$$

Again, the tensor above is introduced back into the integral, and written in a much simpler manner:

$$\int_0^1 (\Phi_{be}^T \Phi_{be} \beta_1 \Phi_{be} \beta_2) d\xi_1 = \frac{1}{6} \left\{ \begin{matrix} \beta_{1,1} \beta_{2,1} \\ \beta_{1,2} \beta_{2,2} \end{matrix} \right\} + \frac{1}{3} \left\{ \begin{matrix} 1 \\ 1 \end{matrix} \right\} \overline{\beta_1 \beta_2} \quad (\text{B.34})$$

The fourth order term is evaluated, first by applying the transpose identity and then by creating a 4th- order compact notation:

$$\Phi_{be}^T \Phi_{be} \theta_1 \Phi_{be} \theta_2 \Phi_{be} \theta_3 = \Phi_{be}^T \Phi_{be} \theta_1 \theta_2^T \Phi_{be}^T \Phi_{be} \theta_3 = [\Phi_{be4}] (\theta_1) (\theta_2^T) \theta_3 \quad (\text{B.35})$$

$$[\Phi_{be4}] = (\Phi_{be}^T \Phi_{be}) (\Phi_{be}^T \Phi_{be}) = \begin{bmatrix} \xi_1^2 \Phi_{be}^T \Phi_{be} & \xi_1 \xi_2 \Phi_{be}^T \Phi_{be} \\ \xi_1 \xi_2 \Phi_{be}^T \Phi_{be} & \xi_2^2 \Phi_{be}^T \Phi_{be} \end{bmatrix} \quad (\text{B.36})$$

Here, each term of the matrix is a matrix in itself – fourth-order tensor. The notation represents the pre-multiplication of θ_2^T and post-multiplication of θ_3 times every inner matrix, and then post-multiplying the matrix by θ_1 . The compact notation is expanded:

$$[\Phi_{be4}] (\theta_1) (\theta_2^T) \theta_3 = \begin{bmatrix} \xi_1^2 \theta_2^T \Phi_{be}^T \Phi_{be} \theta_3 & \xi_1 \xi_2 \theta_2^T \Phi_{be}^T \Phi_{be} \theta_3 \\ \xi_1 \xi_2 \theta_2^T \Phi_{be}^T \Phi_{be} \theta_3 & \xi_2^2 \theta_2^T \Phi_{be}^T \Phi_{be} \theta_3 \end{bmatrix} \theta_1 \quad (\text{B.37})$$

The fourth order term is evaluated:

$$\int_0^1 (\Phi_{be}^T \Phi_{be} \theta_1 \Phi_{be} \theta_2 \Phi_{be} \theta_3) d\xi_1 = \left(\int_0^1 [\Phi_{be4}] d\xi_1 \right) (\theta_1) (\theta_2^T) \theta_3 \quad (\text{B.38})$$

The remaining integral becomes:

$$\int_0^1 [\Phi_{be4}] d\xi_1 = \int_0^1 \begin{bmatrix} \xi_1^2 \Phi_{be}^T \Phi_{be} & \xi_1 \xi_2 \Phi_{be}^T \Phi_{be} \\ \xi_1 \xi_2 \Phi_{be}^T \Phi_{be} & \xi_2^2 \Phi_{be}^T \Phi_{be} \end{bmatrix} d\xi_1 \quad (\text{B.39})$$

$$\int_0^1 [\Phi_{be4}] d\xi_1 = \int_0^1 \begin{bmatrix} \begin{bmatrix} \xi_1^4 & \xi_1^3 \xi_2 \\ \xi_1^3 \xi_2 & \xi_1^2 \xi_2^2 \end{bmatrix} & \begin{bmatrix} \xi_1^3 \xi_2 & \xi_1^2 \xi_2^2 \\ \xi_1^2 \xi_2^2 & \xi_1 \xi_2^3 \end{bmatrix} \\ \begin{bmatrix} \xi_1^3 \xi_2 & \xi_1^2 \xi_2^2 \\ \xi_1^2 \xi_2^2 & \xi_1 \xi_2^3 \end{bmatrix} & \begin{bmatrix} \xi_1^2 \xi_2^2 & \xi_1 \xi_2^3 \\ \xi_1 \xi_2^3 & \xi_2^4 \end{bmatrix} \end{bmatrix} d\xi_1 \quad (\text{B.40})$$

$$\int_0^1 [\Phi_{be4}] d\xi_1 = \frac{1}{60} \begin{bmatrix} \begin{bmatrix} 12 & 3 \\ 3 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix} \\ \begin{bmatrix} 3 & 2 \\ 3 & 2 \end{bmatrix} & \begin{bmatrix} 2 & 3 \\ 3 & 12 \end{bmatrix} \end{bmatrix} \quad (\text{B.41})$$

The tensor is introduced back into the integral, and written in a much simpler manner:

$$\begin{aligned} & \int_0^1 (\Phi_{be}^T \Phi_{be} \theta_1 \Phi_{be} \theta_2 \Phi_{be} \theta_3) d\xi_1 \\ &= \frac{1}{60} \left(8 \begin{Bmatrix} \theta_{1,1} \theta_{2,1} \theta_{3,1} \\ \theta_{1,2} \theta_{2,2} \theta_{3,2} \end{Bmatrix} + 2 \bar{\theta}_3 \begin{Bmatrix} \theta_{1,1} \theta_{2,1} \\ \theta_{1,2} \theta_{2,2} \end{Bmatrix} + 2 \bar{\theta}_2 \begin{Bmatrix} \theta_{1,1} \theta_{3,1} \\ \theta_{1,2} \theta_{3,2} \end{Bmatrix} + \begin{Bmatrix} \theta_{1,2} \\ \theta_{1,1} \end{Bmatrix} (\theta_2 \cdot \theta_3) + 16 \bar{\theta}_1 \bar{\theta}_2 \bar{\theta}_3 \begin{Bmatrix} 1 \\ 1 \end{Bmatrix} \right) \end{aligned} \quad (\text{B.42})$$

Using the shorthand integrals above, the boundary integral containing the normal flux vector

\mathbf{F}_n is written:

$$\sum_{be} \int_{A_{be}} \Phi_{be}^T \mathbf{F}_n(\phi_{be}) d\Gamma = \sum_{be} l_{be} \begin{Bmatrix} F_{n,1} \\ F_{n,2} \\ F_{n,3} \\ F_{n,4} \end{Bmatrix} \quad (\text{B.43})$$

where

$$F_{n,1} = \frac{1}{6} \begin{Bmatrix} \rho_{be,1} V_{n,be,1} \\ \rho_{be,2} V_{n,be,2} \end{Bmatrix} + \frac{1}{3} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix} \bar{\rho} \bar{V}_n \quad (\text{B.44})$$

$$F_{n,4} = \frac{1}{6} \begin{Bmatrix} ((\rho e)_{be,1} + p_{be,1}) V_{n,be,1} \\ ((\rho e)_{be,2} + p_{be,2}) V_{n,be,2} \end{Bmatrix} + \frac{1}{3} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix} (\bar{\rho e} + \bar{p}) \bar{V}_n \quad (\text{B.45})$$

$$F_{n,2} = \frac{1}{60} \left(8 \begin{Bmatrix} \rho_{be,1} u_{be,1} V_{n,be,1} \\ \rho_{be,2} u_{be,2} V_{n,be,2} \end{Bmatrix} + 2\bar{V}_n \begin{Bmatrix} \rho_{be,1} u_{be,1} \\ \rho_{be,2} u_{be,2} \end{Bmatrix} + 2\bar{u} \begin{Bmatrix} \rho_{be,1} V_{n,be,1} \\ \rho_{be,2} V_{n,be,2} \end{Bmatrix} \right) + \frac{1}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} p_{be} \hat{n}_{x,be} \quad (B.46)$$

$$+ \begin{Bmatrix} \rho_{be,2} \\ \rho_{be,1} \end{Bmatrix} (u_{be} \cdot V_{n,be}) + 16\bar{\rho}\bar{u}\bar{V}_n \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

$$F_{n,3} = \frac{1}{60} \left(8 \begin{Bmatrix} \rho_{be,1} v_{be,1} V_{n,be,1} \\ \rho_{be,2} v_{be,2} V_{n,be,2} \end{Bmatrix} + 2\bar{V}_n \begin{Bmatrix} \rho_{be,1} v_{be,1} \\ \rho_{be,2} v_{be,2} \end{Bmatrix} + 2\bar{v} \begin{Bmatrix} \rho_{be,1} V_{n,be,1} \\ \rho_{be,2} V_{n,be,2} \end{Bmatrix} \right) + \frac{1}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} p_{be} \hat{n}_{y,be} \quad (B.47)$$

$$+ \begin{Bmatrix} \rho_{be,2} \\ \rho_{be,1} \end{Bmatrix} (v_{be} \cdot V_{n,be}) + 16\bar{\rho}\bar{v}\bar{V}_n \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

For the far field condition uses Riemann invariants to correct the local flow to include necessary freestream components, and then the corrected fluxes are input directly into the boundary integrals. In this way, the boundary integrals differ from those demonstrated above. The boundary integrals are calculated using a linear distribution of Riemann flux across the boundary element:

$$\sum_{be} \int_{l_{be}} \Phi_{be}^T (\Phi_{be} \mathbf{F}_{n,be}) d\Gamma = \sum_{be} l_{be} \left(\int_0^1 \begin{bmatrix} \xi_1^2 & \xi_1 \xi_2 \\ \xi_1 \xi_2 & \xi_2^2 \end{bmatrix} d\xi_1 \right) \mathbf{F}_{n,be} = \sum_{be} \frac{l_{be}}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \mathbf{F}_{n,be} \quad (B.48)$$

The FEM governing equations is written in terms of the exact integrals using the shorthand flux notation:

$$[\mathbf{M}_c] \frac{\partial \mathbf{U}}{\partial t} - \sum_e \left(\begin{Bmatrix} A_{11} \\ A_{12} \\ A_{13} \end{Bmatrix} \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{Bmatrix} + \begin{Bmatrix} A_{21} \\ A_{22} \\ A_{23} \end{Bmatrix} \begin{Bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{Bmatrix} \right) + \sum_{be_{IWS}} l_{be} \begin{Bmatrix} F_{n,1} \\ F_{n,2} \\ F_{n,3} \\ F_{n,4} \end{Bmatrix} + \sum_{be_{FF}} \frac{l_{be}}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \mathbf{F}_{n,be} = 0 \quad (B.49)$$

where be_{IWS} represents the Inviscid Wall and Symmetry plane boundary elements and be_{FF} represents the Far Field boundary elements.

Non-Inertial Source Term. For continuous integration, the non-inertial source integrals can

be represented:

$$-\int_{A_e} \Phi_e^T \mathbf{S} d\Omega = \int_{A_e} \Phi_e^T \Phi_e \rho_e \begin{pmatrix} 0 \\ \Phi_e (a_{t,x,e} - \omega v_{r,e}) \\ \Phi_e (a_{t,y,e} + \omega u_{r,e}) \\ \Phi_e \vec{a}_{t,e} \cdot \Phi_e (V_{t,e} + \vec{V}_{r,e}) \end{pmatrix} d\Omega = \int_{A_e} \Phi_e^T \Phi_e \rho_e (\Phi_e \alpha_1 + \Phi_e \beta_1 \Phi_e \beta_2) d\Omega \quad (\text{B.50})$$

$$-\int_{A_e} \Phi_e^T \mathbf{S} d\Omega = \int_{A_e} [\Phi_{e3}] (\rho_e) \alpha_1 d\Omega + \int_{A_e} [\Phi_{e4}] (\rho_e) (\beta_1^T) \beta_2 d\Omega \quad (\text{B.51})$$

$$-\int_{A_e} \Phi_e^T \mathbf{S} d\Omega = \int_{A_e} [\Phi_{e3}] d\Omega (\rho_e) \alpha_1 + \int_{A_e} [\Phi_{e4}] d\Omega (\rho_e) (\beta_1^T) \beta_2 \quad (\text{B.52})$$

where

$$[\Phi_{e3}] = \begin{bmatrix} \xi_1^2 \Phi_e & \xi_1 \xi_2 \Phi_e & \xi_1 \xi_3 \Phi_e \\ \xi_1 \xi_2 \Phi_e & \xi_2^2 \Phi_e & \xi_2 \xi_3 \Phi_e \\ \xi_1 \xi_3 \Phi_e & \xi_2 \xi_3 \Phi_e & \xi_3^2 \Phi_e \end{bmatrix} \quad (\text{B.53})$$

$$[\Phi_{e4}] = \begin{bmatrix} \xi_1^2 \Phi_e^T \Phi_e & \xi_1 \xi_2 \Phi_e^T \Phi_e & \xi_1 \xi_3 \Phi_e^T \Phi_e \\ \xi_1 \xi_2 \Phi_e^T \Phi_e & \xi_2^2 \Phi_e^T \Phi_e & \xi_2 \xi_3 \Phi_e^T \Phi_e \\ \xi_1 \xi_3 \Phi_e^T \Phi_e & \xi_2 \xi_3 \Phi_e^T \Phi_e & \xi_3^2 \Phi_e^T \Phi_e \end{bmatrix} \quad (\text{B.54})$$

$$\Phi_e^T \Phi_e = \begin{bmatrix} \xi_1^2 & \xi_1 \xi_2 & \xi_1 \xi_3 \\ \xi_1 \xi_2 & \xi_2^2 & \xi_2 \xi_3 \\ \xi_1 \xi_3 & \xi_2 \xi_3 & \xi_3^2 \end{bmatrix} \quad (\text{B.55})$$

The first term is simplified:

$$\int_{A_e} [\Phi_{e3}] d\Omega = \int_0^{1-\xi_2} \int_0^0 \left[\begin{array}{c} \left\{ \begin{array}{c} \xi_1^3 \\ \xi_1^2 \xi_2 \\ \xi_1 \xi_2^2 \\ \xi_1 \xi_3 \end{array} \right\}^T \\ \left\{ \begin{array}{c} \xi_1^2 \xi_2 \\ \xi_1 \xi_2^2 \\ \xi_1 \xi_2 \xi_3 \end{array} \right\}^T \\ \left\{ \begin{array}{c} \xi_1 \xi_2^2 \\ \xi_1 \xi_2 \\ \xi_1 \xi_2 \xi_3 \end{array} \right\}^T \\ \left\{ \begin{array}{c} \xi_1 \xi_2 \\ \xi_2 \\ \xi_2 \xi_3 \end{array} \right\}^T \\ \left\{ \begin{array}{c} \xi_1 \xi_2 \xi_3 \\ \xi_2 \xi_3 \\ \xi_2 \xi_3 \end{array} \right\}^T \\ \left\{ \begin{array}{c} \xi_1 \xi_2 \\ \xi_1 \xi_3 \\ \xi_2 \xi_3 \\ \xi_3 \end{array} \right\}^T \end{array} \right] |2A_e| d\xi_1 d\xi_2 = \frac{A_e}{60} \left[\begin{array}{c} \left\{ \begin{array}{c} 6 \\ 2 \\ 2 \end{array} \right\}^T \\ \left\{ \begin{array}{c} 2 \\ 2 \\ 1 \end{array} \right\}^T \\ \left\{ \begin{array}{c} 2 \\ 2 \\ 2 \end{array} \right\}^T \\ \left\{ \begin{array}{c} 2 \\ 6 \\ 2 \end{array} \right\}^T \\ \left\{ \begin{array}{c} 2 \\ 1 \\ 2 \end{array} \right\}^T \\ \left\{ \begin{array}{c} 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{array} \right\}^T \end{array} \right] \quad (\text{B.56})$$

$$\int_{A_e} [\Phi_{e3}] d\Omega (\rho_e) \alpha_1 = \frac{A_e}{60} \left\{ \begin{array}{c} 2\bar{\rho}\bar{\alpha}_1 + 4\rho_1\alpha_{1,1} - \rho_2\alpha_{1,3} - \rho_3\alpha_{1,2} \\ 2\bar{\rho}\bar{\alpha}_1 - \rho_1\alpha_{1,3} + 4\rho_2\alpha_{1,2} - \rho_3\alpha_{1,1} \\ 2\bar{\rho}\bar{\alpha}_1 - \rho_1\alpha_{1,2} - \rho_2\alpha_{1,1} + 4\rho_3\alpha_{1,3} \end{array} \right\} \quad (\text{B.57})$$

The first term is used to define the integrals for the x- and y-momentum source term:

$$\int_{A_e} \Phi_e^T \Phi_e \rho_e (a'_{t,x,e} - \omega v_{r,e}) d\Omega = \frac{A_e}{60} \left\{ \begin{array}{c} 2\bar{\rho}(\bar{a}_{t,x} - \omega \bar{v}_r) + 4\rho_1(a_{t,x,1} - \omega v_{r,1}) - \rho_2(a_{t,x,3} - \omega v_{r,3}) - \rho_3(a_{t,x,2} - \omega v_{r,2}) \\ 2\bar{\rho}(\bar{a}_{t,x} - \omega \bar{v}_r) - \rho_1(a_{t,x,3} - \omega v_{r,3}) + 4\rho_2(a_{t,x,2} - \omega v_{r,2}) - \rho_3(a_{t,x,1} - \omega v_{r,1}) \\ 2\bar{\rho}(\bar{a}_{t,x} - \omega \bar{v}_r) - \rho_1(a_{t,x,2} - \omega v_{r,2}) - \rho_2(a_{t,x,1} - \omega v_{r,1}) + 4\rho_3(a_{t,x,3} - \omega v_{r,3}) \end{array} \right\} \quad (\text{B.58})$$

$$\int_{A_e} [\Phi_{e4}] d\Omega (\rho_e) (\beta_1^T) \beta_2 = \frac{A_e}{180} \left\{ \begin{array}{l} K + (6\rho_1\beta_{1,1} + \bar{\rho}\bar{\beta}_1)\beta_{2,1} + (3\rho_1\beta_{1,1} - \rho_e \cdot \beta_1)\bar{\beta}_2 \\ K + (6\rho_2\beta_{1,2} + \bar{\rho}\bar{\beta}_1)\beta_{2,2} + (3\rho_2\beta_{1,2} - \rho_e \cdot \beta_1)\bar{\beta}_2 \\ K + (6\rho_3\beta_{1,3} + \bar{\rho}\bar{\beta}_1)\beta_{2,3} + (3\rho_3\beta_{1,3} - \rho_e \cdot \beta_1)\bar{\beta}_2 \end{array} \right\} \quad (\text{B.62})$$

$$K = \bar{\rho}\bar{\beta}_1\bar{\beta}_2 + \bar{\rho}(\beta_1 \cdot \beta_2) + \bar{\beta}_1(\rho_e \cdot \beta_2) \quad (\text{B.63})$$

The second term is used to define the integrals for the energy source term:

$$\begin{aligned} \int_{A_e} \Phi_e^T \Phi_e \rho_e \Phi_e \bar{a}_{t,e} \cdot \Phi_e (\bar{V}_{t,e} + \bar{V}_{r,e}) d\Omega = \\ = \frac{A_e}{180} \left\{ \begin{array}{l} K_x + (6\rho_1 a_{t,x,1} + \bar{\rho} \bar{a}_{t,x}) \tilde{u}_1 + (2\rho_1 a_{t,x,1} - \rho_2 a_{t,x,3} - \rho_3 a_{t,x,2}) \tilde{u} \\ K_x + (6\rho_2 a_{t,x,2} + \bar{\rho} \bar{a}_{t,x}) \tilde{u}_2 + (-\rho_1 a_{t,x,3} + 2\rho_2 a_{t,x,2} - \rho_3 a_{t,x,1}) \tilde{u} \\ K_x + (6\rho_3 a_{t,x,3} + \bar{\rho} \bar{a}_{t,x}) \tilde{u}_3 + (-\rho_1 a_{t,x,2} - \rho_2 a_{t,x,3} + 2\rho_3 a_{t,x,3}) \tilde{u} \end{array} \right\} \\ + \frac{A_e}{180} \left\{ \begin{array}{l} K_y + (6\rho_1 a_{t,y,1} + \bar{\rho} \bar{a}_{t,y}) \tilde{v}_1 + (2\rho_1 a_{t,y,1} - \rho_2 a_{t,y,3} - \rho_3 a_{t,y,2}) \tilde{v} \\ K_y + (6\rho_2 a_{t,y,2} + \bar{\rho} \bar{a}_{t,y}) \tilde{v}_2 + (-\rho_1 a_{t,y,3} + 2\rho_2 a_{t,y,2} - \rho_3 a_{t,y,1}) \tilde{v} \\ K_y + (6\rho_3 a_{t,y,3} + \bar{\rho} \bar{a}_{t,y}) \tilde{v}_3 + (-\rho_1 a_{t,y,2} - \rho_2 a_{t,y,3} + 2\rho_3 a_{t,y,3}) \tilde{v} \end{array} \right\} \end{aligned} \quad (\text{B.64})$$

where

$$K_x = \bar{\rho} \bar{a}_{t,x} \tilde{u} + \bar{\rho} (a_{t,x} \cdot \tilde{u}) + \bar{a}_{t,x} (\rho_e \cdot \tilde{u}) \quad \tilde{u}_i = V_{t,x,i} + u_{r,i} \quad (\text{B.65})$$

$$K_y = \bar{\rho} \bar{a}_{t,y} \tilde{v} + \bar{\rho} (a_{t,y} \cdot \tilde{v}) + \bar{a}_{t,y} (\rho_e \cdot \tilde{v}) \quad \tilde{v}_i = V_{t,y,i} + v_{r,i} \quad (\text{B.66})$$

Using the shorthand integrals above, the source term is written:

$$-\sum_e \int_{A_e} \Phi_e^T S d\Omega = \sum_e \int_{A_e} \Phi_e^T \Phi_e \rho_e \left\{ \begin{array}{l} 0 \\ \Phi_e (a_{t,x,e} - \omega v_{r,e}) \\ \Phi_e (a_{t,y,e} + \omega u_{r,e}) \\ \Phi_e \bar{a}_{t,e} \Phi_e (\bar{V}_{t,e} + \bar{V}_{r,e}) \end{array} \right\} d\Omega = \sum_e \left\{ \begin{array}{l} 0 \\ \{S_{21} \ S_{22} \ S_{23}\}^T \\ \{S_{31} \ S_{32} \ S_{33}\}^T \\ \{S_{41} \ S_{42} \ S_{43}\}^T \end{array} \right\} \quad (\text{B.67})$$

where

$$S_{21} = \frac{A_e}{60} (K_{xm} + 4\rho_1 (a_{t,x,1} - \omega v_{r,1}) - \rho_2 (a_{t,x,3} - \omega v_{r,3}) - \rho_3 (a_{t,x,2} - \omega v_{r,2})) \quad (\text{B.68})$$

$$S_{22} = \frac{A_e}{60} (K_{xm} - \rho_1(a_{t,x,3} - \omega v_{r,3}) + 4\rho_2(a_{t,x,2} - \omega v_{r,2}) - \rho_3(a_{t,x,1} - \omega v_{r,1})) \quad (\text{B.69})$$

$$S_{23} = \frac{A_e}{60} (K_{xm} - \rho_1(a_{t,x,2} - \omega v_{r,2}) - \rho_2(a_{t,x,1} - \omega v_{r,1}) + 4\rho_3(a_{t,x,3} - \omega v_{r,3})) \quad (\text{B.70})$$

$$S_{31} = \frac{A_e}{60} (K_{ym} + 4\rho_1(a_{t,y,1} + \omega u_{r,1}) - \rho_2(a_{t,y,3} + \omega u_{r,3}) - \rho_3(a_{t,y,2} + \omega u_{r,2})) \quad (\text{B.71})$$

$$S_{32} = \frac{A_e}{60} (K_{ym} - \rho_1(a_{t,y,3} + \omega u_{r,3}) + 4\rho_2(a_{t,y,2} + \omega u_{r,2}) - \rho_3(a_{t,y,1} + \omega u_{r,1})) \quad (\text{B.72})$$

$$S_{33} = \frac{A_e}{60} (K_{ym} - \rho_1(a_{t,y,2} + \omega u_{r,2}) - \rho_2(a_{t,y,1} + \omega u_{r,1}) + 4\rho_3(a_{t,y,3} + \omega u_{r,3})) \quad (\text{B.73})$$

$$S_{41} = \frac{A_e}{180} (K_{xe} + (6\rho_1 a_{t,x,1} + \bar{\rho} \bar{a}_{t,x}) \tilde{u}_1 + (2\rho_1 a_{t,x,1} - \rho_2 a_{t,x,3} - \rho_3 a_{t,x,2}) \bar{\tilde{u}}) \\ + \frac{A_e}{180} (K_{ye} + (6\rho_1 a_{t,y,1} + \bar{\rho} \bar{a}_{t,y}) \tilde{v}_1 + (2\rho_1 a_{t,y,1} - \rho_2 a_{t,y,3} - \rho_3 a_{t,y,2}) \bar{\tilde{v}}) \quad (\text{B.74})$$

$$S_{42} = \frac{A_e}{180} (K_{xe} + (6\rho_2 a_{t,x,2} + \bar{\rho} \bar{a}_{t,x}) \tilde{u}_2 + (-\rho_1 a_{t,x,3} + 2\rho_2 a_{t,x,2} - \rho_3 a_{t,x,1}) \bar{\tilde{u}}) \\ + \frac{A_e}{180} (K_{ye} + (6\rho_2 a_{t,y,2} + \bar{\rho} \bar{a}_{t,y}) \tilde{v}_2 + (-\rho_1 a_{t,y,3} + 2\rho_2 a_{t,y,2} - \rho_3 a_{t,y,1}) \bar{\tilde{v}}) \quad (\text{B.75})$$

$$S_{43} = \frac{A_e}{180} (K_{xe} + (6\rho_3 a_{t,x,3} + \bar{\rho} \bar{a}_{t,x}) \tilde{u}_3 + (-\rho_1 a_{t,x,2} - \rho_2 a_{t,x,3} + 2\rho_3 a_{t,x,3}) \bar{\tilde{u}}) \\ + \frac{A_e}{180} (K_{ye} + (6\rho_3 a_{t,y,3} + \bar{\rho} \bar{a}_{t,y}) \tilde{v}_3 + (-\rho_1 a_{t,y,2} - \rho_2 a_{t,y,3} + 2\rho_3 a_{t,y,3}) \bar{\tilde{v}}) \quad (\text{B.76})$$

$$K_{xm} = 2\bar{\rho}(\bar{a}_{t,x} - \omega \bar{v}_r) \quad \tilde{u}_i = V_{t,x,i} + u_{r,i} \quad (\text{B.77})$$

$$K_{xe} = \bar{\rho} \bar{a}_{t,x} \bar{\tilde{u}} + \bar{\rho} (a'_{t,x} \cdot \tilde{u}) + \bar{a}_{t,x} (\rho_e \cdot \tilde{u}) \quad (\text{B.78})$$

$$K_{ym} = 2\bar{\rho}(\bar{a}_{t,y} + \omega \bar{u}_r) \quad \tilde{v}_i = V_{t,y,i} + v_{r,i} \quad (\text{B.79})$$

$$K_{ye} = \bar{\rho} \bar{a}_{t,y} \bar{\tilde{v}} + \bar{\rho} (a'_{t,y} \cdot \tilde{v}) + \bar{a}_{t,y} (\rho_e \cdot \tilde{v}) \quad (\text{B.80})$$

APPENDIX C

EULER2D FILE FORMATS

The file formats included in this section are used to create the input files required to operate Euler2D and interpret the files that are written by Euler2D.

Euler2D

Summary of File Formats

Input Files:

- `case.g2d` (required) contains the geometry data structures representing the computational mesh as required by the flow solver (ASCII)
- `case.con` (required) contains values for the solver control parameters and flow conditions (ASCII)
- `case.unk` (optional) contains the values of the primitive flow variables (density, velocity, pressure, and total enthalpy) for each node of the computational mesh to be used as the initial conditions for the flow solution (Binary)
- `case.dyn` (optional) contains the non-inertial matrices and initial conditions as required for a non-inertial solution (ASCII)
- `case.vec` (optional) contains the elastic mode matrices, initial conditions, and mode shape vectors for the solid wall surfaces as required for an aero-elastic solution (ASCII)
- `case.frc` (optional) contains external forces to be applied to each solution step in a dynamic or aeroelastic solution (ASCII)
- `case.cmb` (optional) contains information to represent the influence of combustion on the flow solution (ASCII)
- `case.eng` (optional) contains the properties needed to represent rocket and engine boundary conditions (ASCII)
- `case.acst` (optional) contains the acoustic output data (ASCII)

Output Files:

- `case.un#` contains the values of the primitive flow variables (density, velocity, pressure, and total enthalpy) for each node of the computational mesh; # is iterated as more files are produced so the progress of the solution can be followed (Binary)
- `case.rsd` contains a history of the solution residuals for the conservation variables (density, momentum (2), and total energy) (ASCII)

- `case.rs2` contains a history of the solution residuals for the conservation variables for each inner cycle (ASCII)
- `case.cyc` contains a history of the number of inner cycles used to converge each iteration (ASCII)
- `case.time` contains a history of the local time step ratios (ASCII)
- `case.lfs` contains a history of the dimensionless aerodynamic forces acting on the solid walls of the geometry (ASCII)
- `xd.dat` contains a history of the non-inertial displacements, velocities, and accelerations for a dynamic solution (ASCII)
- `xn.dat` contains a history of the generalized displacements, velocities, and forces for an unsteady, aeroelastic solution (ASCII)
- `case.rst` and `case.rs2` contain information on up to two sets of unknowns data, elastic system data, and dynamic motion data (binary)
- `case.pac` contains a history of pressure data at prescribed nodes (ASCII)
- `case.rac` contains a history of density data at prescribed nodes (ASCII)
- `case.uac` contains a history of u-velocity data at prescribed nodes (ASCII)
- `case.vac` contains a history of v-velocity data at prescribed nodes (ASCII)

Geometry Input File (case.g2d)

Basic File Format

```
Line of Text
  nnd nel nsg nbe nbp nwl nsd
Line of Text
  (LBE(i), i = 1, 8)
Line of Text
  (COOR(i,j), j = 1,2) (i = 1,nnd)
Line of Text
  (IELM(i,j), j = 1,3) (i = 1,nel)
Line of Text
  (ISEG(i,j), j = 1,2) (i = 1,nsg)
Line of Text
  (IBEL(i,j), j = 1,3) (i = 1,nbe)
```

Definition of Terms

nnd: int number of nodes
nel: int number of elements
nsg: int number of segments
nbe: int number of boundary elements
nbp: int number of boundary points
nwl: int number of wall nodes
nsd: int number of singular nodes

LBE(i): int start/ stop index for 4 BC types

COOR(i,1): real x-coordinate for node *i*
COOR(i,2): real y-coordinate for node *i*

IELM(i,1): int node 1 for element *i*
IELM(i,2): int node 2 for element *i*
IELM(i,3): int node 3 for element *i*

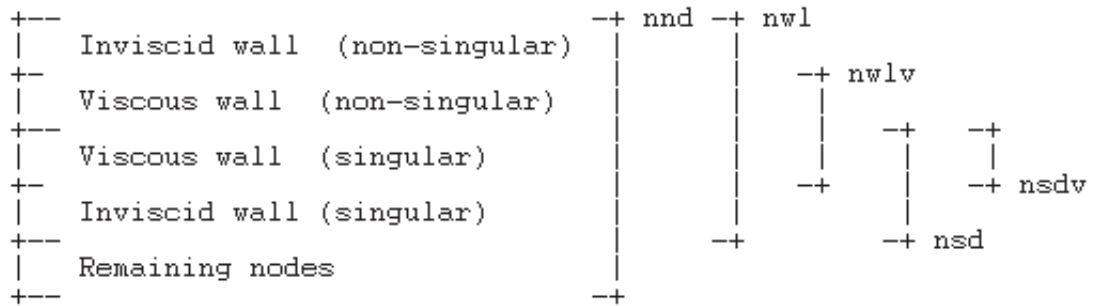
ISEG(i,1): int node 1 for segment *i*
ISEG(i,2): int node 2 for segment *i*

IBEL(i,1): int node 1 for boundary elem. *i*
IBEL(i,2): int node 2 for boundary elem. *i*
IBEL(i,3): int surface index in case.sur

Comments

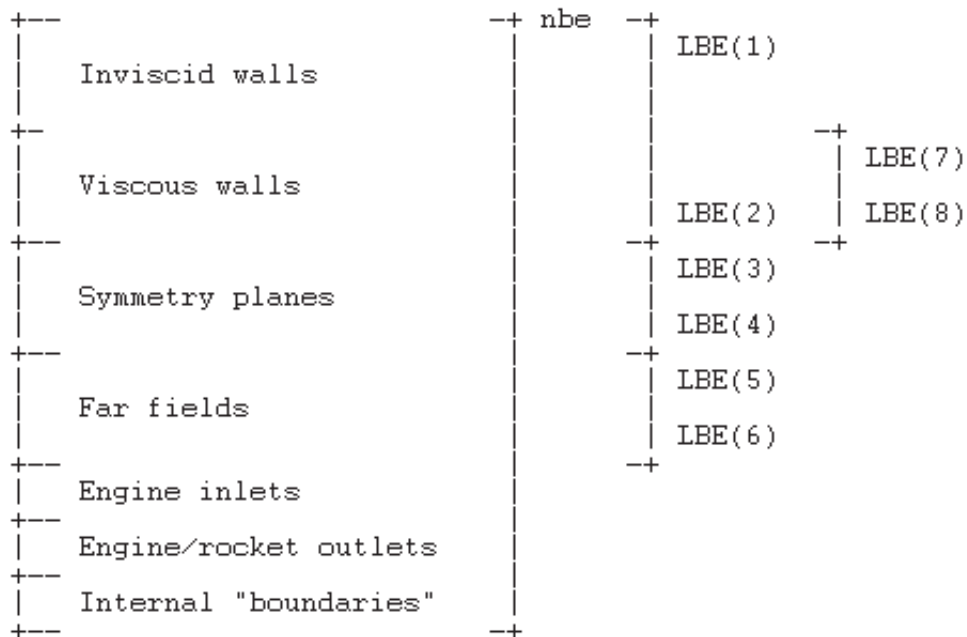
- This is a plain text (ASCII) file.
- The nodal coordinates in this file are treated as dimensional values and are non-dimensionalized using the reference dimension `refdim` specified in the control file.
- The element connectivity data `IELM` and `IBEL` define clockwise oriented elements.
- The program `makeg2d` is used to convert a standard STARS surface triangulation file `case.fro` and modified boundary conditions file `case.bco` into an appropriately sorted two-dimensional geometry file.
- Nodal data `COOR` is sorted such that the first `nwl` nodes are defined as solid wall nodes. Out of the first `nwl` nodes, the last `nsd` nodes are defined as singular nodes. The viscous nodes are placed in the middle (`nwlv` and `nsdv`), according to the following diagram:

Nodes:



- Boundary element data is sorted based on the starting/stopping indexes for the three BC types, i.e. boundary elements LBE(1) through LBE(2) are solid wall elements, LBE(3) through LBE(4) are symmetry elements, LBE(5) through LBE(6) are far-field elements, and LBE(7) through LBE(8) are viscous solid wall elements, where the two solid wall elements are restricted to $LBE(1) \leq LBE(7) < LBE(8) \leq LBE(2)$. (In other words, the viscous solid walls must exist within the limits of the viscous walls.)

Boundary elements:



Sample File

```
$ nnd, nel, nsg, nbe, nbp, nwl, nsd
   8     6    13     8     8     3     0
$ LBE(8)
   1     2     3     2     3     8     1     2
$ Nodal coordinates
-.100000E+01 -.100000E+01
 0.100000E+01 -.100000E+01
 0.000000E+00 -.100000E+01
 0.100000E+01 0.100000E+01
-.100000E+01 0.100000E+01
 0.100000E+01 0.000000E+00
 0.000000E+00 0.100000E+01
-.100000E+01 0.000000E+00
$ Element connectivity
   1     3     8
   3     2     6
   5     8     7
   6     4     7
   8     3     6
   6     7     8
$ Segment connectivity
   1     3
   1     8
   2     3
   2     6
   3     8
   3     6
   4     6
   4     7
   5     8
   5     7
   6     7
   6     8
   7     8
$ Boundary edge data
   1     3     1
   3     2     1
   2     6     2
   6     4     2
   4     7     3
   7     5     3
   5     8     4
   8     1     4
```

Solver Control Input File (case . con)

Basic File Format

```
&control
  dt          = 0.1d0,
  gamma       = 1.4d0,
  diss        = 1.0d0,
  cfl         = 0.5d0,

  mach        = 0.6d0,
  alpha       = 0.0d0,
  refdim      = 1.0d0,

  nstp        = 100,
  nout        = 50,
  ncyc        = 4,

  rsdtol      = 1.0d-20,
  rsdmax      = 10.0d0,

  isol        = 0,
  ipnt        = 1,
  idiss       = 0,
  icomb       = 0,

  istrtr      = .false.,
  iaero       = .true.,
  idynm       = .false.,
  ielast      = .false.,
  iprop       = .false.,
  ifree       = .true.,
  iforce      = .false.,
  isafe       = .false.,
  irsds       = .false.,
  iacoust     = .false.,

  nr          = 0,
  ainfnf      = 1.0d0,
  rhoinf      = 1.0d0,
  gravity     = 0.0d0,
/
```

Definition of Terms

dt:	real	dimensionless global time step
gamma:	real	ratio of specific heats
diss:	real	dissipation factor
cfl:	real	local time step stability factor
mach:	real	freestream Mach number
alpha:	real	freestream angle of attack (deg)
refdim:	real	reference length (dim'l)
nstp:	int	total solution steps
nout:	int	output frequency, steps/output
ncyc:	int	iterative cycles per solution step
rsdtol:	real	energy residual converg tolerance
rsdmax:	real	energy residual divergence criteria
isol:	int	CFD solution type
ipnt:	int	number of Gauss points
idiss:	int	dissipation type
icomb:	int	combustion model type
istrtr:	log	restart flag
iaero:	log	aerodynamic forces flag
idynm:	log	dynamic/non-inertial flag
ielast:	log	elastic flag
iprop:	log	propulsion flag
ifree:	log	free-stream velocity flag
iforce:	log	external forces flag
isafe:	log	safe-mode flag
irsds:	log	residual watching flag
iacoust:	log	acoustics output flag
nr:	int	number of elastic modes
ainfnf:	real	free-stream sonic speed (dim'l)
rhoinf:	real	free-stream density (dim'l)
gravity:	real	gravity (dim'l)

Comments

- This is a plain text (ASCII) file formatted as a Fortran namelist.
 - The namelist must begin with the line “&control” and end with the line “/”.
 - The remaining lines can be listed in any order or omitted, if desired.
 - The intermediate lines work like variable assignments with the loose format: *variable_name = value*, followed by a comma.
 - Integers (*int*) are listed as whole numbers.
 - Real numbers (*real*) are listed in double precision, scientific notation: *###d±###*.
 - Logical variables (*log*) are listed as either “.true.” or “.false.”.

- Lines can be commented out by including an exclamation point “!” prior to any other information on the line.
- The default values, shown above, are used for variables omitted or commented out of the namelist.
- Any information listed after the last line of the namelist “/” are not read by the program and can be used to store notes and other calculations.
- The global time step Δt is only used for unsteady solutions. Δt is a dimensionless value calculated: $\Delta t = \Delta t U / L$, where Δt is the dimensional time step, U is the free-stream velocity ($= \text{mach} \text{ ainf}$), and L is the reference length refdim .
- Appropriate values for the dissipation factor are in the range $0.0 < \text{diss} \leq 2.0$. Some dissipation is required to stabilize the solution, but too much dissipation will corrupt the solution and possibly be a destabilizing influence.
- The local time step stability factor is a safety factor used to compute local time steps for each solution step. For steady solutions, a stability factor of 0.8 is typical for most problems. For unsteady solutions, the stability factor is typically $0.3 \leq \text{cfl} \leq 0.8$.
- The values of refdim , mach , ainf , and rhoinf are used to non-dimensionalize all values read into the flow solver.
- The free-stream angle of attack is ignored for dynamic (non-inertial) problems.
- The number of iterative cycles should be set to 4 for steady solutions. For unsteady solutions, use a sufficient number of cycles to allow for an appropriate level of convergence at each step. A sufficient number can be estimated as $N = \Delta t / \Delta t_{\text{loc,min}}$.
- The following is a good practice for finding a sufficient number of iterations for unsteady solutions:
 1. Select initial values for Δt , ncyc , and rsdtol .
 2. Step the solution forward 20-50 iterations.
 3. Check the *.cyc file for the number of cycles required per iteration. The number of cycles should level off after 10 iterations. If not, run enough iterations for the number of required cycles to level off.
 4. If the last 10 iterations require more than 20 cycles, lower the time step.
 5. If the last 10 iterations require less than 8 iterations, increase the time step.
 6. The sweet spot is 10-15 iterations.
- The residual tolerance can be used to exit the iterative cycles if the energy residual meets a specified criteria rsdtol . (The energy residual is used because the other residuals normally converge faster than energy.) This feature can be used to set the number of iterations to a very large number with a residual tolerance rsdtol . When the residual drops below the tolerance, the solution will progress to the next time step. Lower rsdtol values require more iterations.
- The divergence tolerance rsdmax creates an upper tolerance on the energy residual. If the solution is diverging, the energy residual will grow larger than rsdmax and terminate the run. The solution also terminates if the residuals become NAN or INFINITY. Larger rsdmax values will allow the solution to diverge further and ensure that the solution is in fact diverging.

- There are four available CFD solution types defined as follows:
 - `isol = 0` is a steady solution (not time accurate)
 - `isol = 1` is a first-order unsteady solution
 - `isol = 2` is a second-order unsteady solution
 - `isol = 3` is a supersonic piston perturbation solution
- There are four types of integration defined as follows:
 - `ipnt = 0` uses analytical equations (no Gauss quadrature)
 - `ipnt = 1` uses a one-point Gauss quadrature
 - `ipnt = 3` uses a three-point symmetric Gauss quadrature
 - `ipnt = 4` uses a four-point symmetric Gauss quadrature
- There are three available dissipation types defined as follows:
 - `idiss = 0` is a low order dissipation
 - `idiss = 1` is a high order dissipation with gradient limiters
- The lower order dissipation is typically overly diffuse and should be used in conjunction with low values of the dissipation factor. Low-order dissipation works best for problems without strong vortices and for supersonic/hypersonic flows.
- The higher order dissipation is more CPU intensive than the low-order dissipation and less stable. Larger values for the dissipation factor are typically required for stabilization. The high-order dissipation works best for subsonic to transonic flows with strong gradients or vortices. Rotating domains will typically require high-order dissipation to resolve the circulating pattern of the relative flow velocities.
- Combustion properties are specified in the `case.cmb` file. The mass and heat generation are distributed throughout the domain using the following distributions:
 - `icomb = 0`, no combustion (`case.cmb` not read)
 - `icomb = 1`, combustion properties are piece-wise linear (specified at the nodes)
 - `icomb = 2`, combustion properties are constant (specified) on the elements
- When the restart flag `istrt` is set to `.true.`, the solver will read one set of solution unknowns from the `case.unk` file and apply this set of unknowns as the initial conditions for the new iterative solution.
- A restarted solution assumes that the time gradient of the initial state is zero, i.e. the solution stored in the `case.unk` file is a converged, steady state solution. This has a significant impact on the second-order unsteady solution since it relies on two sets of solution unknowns for advancement to the next time step, i.e. a second-order unsteady solution should not be restarted from the last time step of a similar unsteady solution that was stopped because both sets of unsteady data from the last solution step are not available for accurate evaluation of the time gradients in the flow.
- If the aerodynamics flag `iaero` is set to `.true.`, the aerodynamic forces are applied to the dynamic and elastic motion. If the flag is set to `.false.`, the dynamic and elastic motion must be forced externally or occur as free-response vibrations.
- The non-inertial dynamics routine is turned on when `idynm` is set to `.true.`. Euler2D will then read in the `case.dyn` file for dynamic inputs and write out dynamic motion to the `xd.dat`.
 - If the free-stream velocity flag `ifree` is set to `.false.`, the free-stream velocity is set to zero, and relative flow velocities must be generated through dynamic rotation or translation of the non-inertial coordinate system.

- If `ifree = .true.`, the freestream starts aligned with the global fixed x-direction (i.e., `alpha = beta = 0`) but can be rotated using the initial orientation of the body in `case.dyn`.
- The elastic deflection routine is turned on when `ielast` is set to `.true.`. Euler2D will read in the `case.vec` file for modal elastic inputs and write out modal deformations and forces to the `xn.dat`. The number of modes `nr` must match that shown in the elastic file `case.vec`.
- For steady solutions (`isol = 0`), the dynamics flags for each degree of freedom in the `case.dyn` and `case.vec` should be set to 1 (clamped condition).
- The propulsion boundary conditions are turned on when `iprop` is set to `.true.`. Euler2D will read in the `case.eng` file for rocket and engine inputs.
- If the external forces flag is set to `.true.`, the solver will read the user defined external force vector for each global time step from the input file `case.frc`. If the solver reaches the end of the input file before completing the solution, the last force vector in the file carries over to each of the remaining time steps if it was non-zero.
- If the safe-mode flag is set to `.true.`, Euler2D writes two files per step that are used to restart the solution: `case.rst` and `case.rs2`. Two files are used so while one file is being over-written, the other file is still preserved. Each file stores the previous two values of all unknowns, elastic mode shapes, and generalized elastic forces. Safe-mode can be used for fast restarts for very long runs that are not time sensitive.
- When the safe-mode flag is set to `.true.`, Euler2D checks for both restart files. If the `case.rst` exists, but has an error, the error is reported to the user. The `case.rst` can be moved, renamed, or deleted. The solution will then be restarted from the `case.rs2` file. (Euler2D does not skip between files to avoid overwriting files that contain correctable errors.)
- If the residual watching flag is set to `.true.`, residuals are written out at each inner iteration to the `case.rsd2` file. This option can be used to check the residual convergence within steps. The number of inner cycles used at each iteration is written to the `case.cyc` file for plotting and comparison of convergence.
- If the acoustic output flag is set to `.true.`, the acoustic input file `case.acst` is read for controls, and one or more of the acoustic output files (pressure – `case.pac`; density – `case.rac`; u-velocity – `case.uac`; v-velocity – `case.vac`) are written.
- Gravity is assumed to act on the vehicle along the inertial y-axis. In the non-inertial reference frame, the body force vector rotates so that gravity is always pointed down in the positive y-direction. The value `gravity` is non-dimensionalized using `refdim` (L), `mach` (M), and `ainf`, so the dimensions of these variables should be consistent:

$$g^* = \frac{gL}{M^2 a_\infty^2}$$

Unknowns (Initial Conditions) Input File (`case.unk`)

Basic File Format

```
np gam xmi alp ref t
((UN(i,j), i = 1,nnd ), j = 1,5)
```

Definition of Terms

np:	int	number of nodes
gam:	real	ratio of specific heats
xmi:	real	free-stream Mach number
alp:	real	free-stream angle of attack
ref:	real	reference dimension
t:	real	dimensionless time

UN(i,1):	real	density for node i
UN(i,2):	real	x-velocity for node i
UN(i,3):	real	y-velocity for node i
UN(i,4):	real	pressure for node i
UN(i,5):	real	total enthalpy for node i

Comments

- This is an unformatted (binary) file.
- The solution unknowns stored in this file are dimensionless quantities.
- For dynamic (non-inertial) problems, the solution unknowns stored in the file are relative quantities referenced to the body-fixed coordinate system.
- The quantities `nnd` must match the values in the geometry file `case.g2d` as `nnd`.
- The quantities `gam` and `xmi` must match the values in the control file `case.con` as `gamma` and `mach`.
- When restarting a solution, the most recent unknowns output file `case.un#` can be renamed as the initial conditions file `case.unk`.

Dynamic Mesh Input File (`case.dyn`)

Basic File Format

```
Line of Text
  (R0(i), i = 1, 2)
Line of Text
  ((RM1(i,j), j = 1,3), i = 1,3)
Line of Text
  ((RC1(i,j), j = 1,3), i = 1,3)
Line of Text
  ((RK1(i,j), j = 1,3), i = 1,3)
Line of Text
  x, y, q, vx, vy, vq, ax, ay, aq
Line of Text
  (IBXD(i), i = 1,3)
```

Definition of Terms

R0(1):	real	x-coord. for origin of rotation
R0(2):	real	y-coord. for origin of rotation
RM1(i,j):	real	dimensional mass matrix
RC1(i,j):	real	dimensional damping matrix
RK1(i,j):	real	dimensional stiffness matrix
x:	real	initial x-position of system
y:	real	initial y-position of system
q:	real	initial orientation of system (deg)
vx:	real	initial x-velocity of system
vy:	real	initial y-velocity of system
vq:	real	initial angular velocity (deg/s)
ax:	real	initial x-acceleration of system
ay:	real	initial y-acceleration of system
aq:	real	initial angular acceleration (deg/s ²)
IBXD(1):	int	dynamics flag for x-DOF
IBXD(2):	int	dynamics flag for y-DOF
IBXD(3):	int	dynamics flag for rotational DOF

Comments

- This is a plain text (ASCII) file.
- All values entered into this file should be dimensional. The solver will automatically non-dimensionalize the values using the reference conditions specified in the solver control file. The units of mass, length, and time in this file should match those in the controls file `case.con`.
- The vector defining the origin of rotation `R0` is subtracted directly from the nodal coordinates defined in the geometry input file `case.g2d` after it is non-dimensionalized by the reference dimension.
- The mass matrix `RM1` defined in this file cannot be singular.
- Initial conditions for the two translational degrees of freedom are specified relative to the inertial coordinate system, i.e. as seen by a stationary observer on the ground.
- The dynamics of each degree of freedom is controlled separately using the following values for `IBXD`:
 - `IBXD = 0` is a free / forced response calculation, i.e. uses mass, stiffness, and damping to compute position, velocity, and acceleration of system.
 - `IBXD = 1` is a clamped condition, i.e. hold at initial position and velocity with zero acceleration.
 - `IBXD = 2` is a constant acceleration, uncoupled response, i.e. integrates acceleration and velocity to compute new position.
 - `IBXD = 3` is a forced mulistep response used for system identification purposes.

- The acceleration of gravity is scaled by gravity so that:

$$\bar{a}_g = \bar{g} = -gravity \hat{j}$$

where j points in the positive y -direction in the inertial frame.

- To convert from `xd.dat` to ICs:

$$XD(1) \times refdim \rightarrow x$$

$$XD(2) \times refdim \rightarrow y$$

$$XD(3) \times 180/\pi \rightarrow q$$

$$XD(4) \times mach \times ainf \rightarrow vx$$

$$XD(5) \times mach \times ainf \rightarrow vy$$

$$XD(6) \times 180/\pi \times mach \times ainf / refdim \rightarrow vq$$

The accelerations ax , ay , and aq are not used to restart the system and do not need to be converted from the `xd.dat` file.

Sample File

```
$ Position vector to origin of non-inertial frame (rx, ry)
0.0d0 0.0d0
$ Mass matrix for non-inertial frame (3 x 3)
1.0d0 0.0d0 0.0d0
0.0d0 1.0d0 0.0d0
0.0d0 0.0d0 1.0d0
$ Damping matrix for non-inertial frame (3 x 3)
1.0d0 0.0d0 0.0d0
0.0d0 1.0d0 0.0d0
0.0d0 0.0d0 1.0d0
$ Stiffness matrix for non-inertial frame (3 x 3)
1.0d0 0.0d0 0.0d0
0.0d0 1.0d0 0.0d0
0.0d0 0.0d0 1.0d0
$ IC's for non-inertial frame (x, y, q, vx, vy, vq, ax, ay, aq)
0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0
$ IBXD for non-inertial frame (3)
1 1 1
```

Elastic Vectors Input File (`case.vec`)

Basic File Format

```

Line of Text
  nr
Line of Text
  ((RM(i,j), j = 1,nr), i = 1,nr)
Line of Text
  ((RC(i,j), j = 1,nr), i = 1,nr)
Line of Text
  ((RK(i,j), j = 1,nr), i = 1,nr)
Line of Text
  (XN(i), i = 1,nr*2)
Line of Text
  (IBXN(i), i = 1,nr)
Line of Text
  ((PHIA(i,j), i = 1,nwl*2), j = 1,nr)

```

Definition of Terms

nr:	int	number of elastic modes
RM(i,j):	real	dimensional mass matrix
RC(i,j):	real	dimensional damping matrix
RK(i,j):	real	dimensional stiffness matrix
XN(i):	real	initial gen. displ. for mode i
XN(i+nr):	real	initial gen. vel. for mode i
IBXD(i):	int	dynamics flag for i^{th} mode
PHIA(i*2-1,j):	real	x-displacement vector for mode j at node i
PHIA(i*2,j):	real	y-displacement vector for mode j at node i

Comments

- This is a plain text (ASCII) file.
- All values entered into this file should be dimensional. The solver will automatically non-dimensionalize the values using the reference conditions specified in the solver control file. The units of mass, length, and time in this file should match those in the controls file `case.con`.
- The number of modes `nr` listed in this file must match the number listed in the controls file `case.con`.
- The mass matrix `RM` defined in this file cannot be singular.
- The dynamics of each degree of freedom is controlled separately using the following values for `IBXN`:
 - `IBXN = 0` is a free / forced response calculation, i.e. uses mass, stiffness, and damping to compute position, velocity, and acceleration of system.
 - `IBXN = 1` is a clamped condition, i.e. hold at initial position with zero velocity and acceleration.
 - `IBXN = 2` is a constant velocity, uncoupled response, i.e. integrates velocity to compute new displacement.
 - `IBXN = 3` is a forced multistep response used for system identification purposes.
- Do not combined zero `IBXN` values with non-zero values for different modes if there are coupling or off-diagonal terms in the mass, damping, or stiffness matrices.
- A limited set of simple model vectors representing standard rigid-body degrees of freedom can be generated using the program `makevec2d`.

Sample File

```
$ Number of elastic modes
3
$ Mass matrix for elastic modes (nr x nr)
1.0d0    0.0d0    0.0d0
0.0d0    1.0d0    0.0d0
0.0d0    0.0d0    1.0d0
$ Damping matrix for elastic modes (nr x nr)
1.0d0    0.0d0    0.0d0
0.0d0    1.0d0    0.0d0
0.0d0    0.0d0    1.0d0
$ Stiffness matrix for elastic modes (nr x nr)
1.0d0    0.0d0    0.0d0
0.0d0    1.0d0    0.0d0
0.0d0    0.0d0    1.0d0
$ IC's for elastic modes (x1...xn, vx1...vxn)
0.0d0    0.0d0    0.0d0    0.0d0    0.0d0    0.0d0
$ IBXN for elastic modes (nr)
1      1      1
$ Elastic modes vectors ((nwl*2) x nr)
0.0d0    1.0d0
0.0d0    1.0d0
0.0d0    1.0d0
0.0d0    1.0d0
0.0d0    1.0d0
0.0d0    1.0d0
0.0d0    1.0d0
  ⋮      ⋮
```

External Forces File (`case.frc`)

Basic File Format

```

0      (FD(i), i = 1, 3)
      (FA(i), i = 1, nr)
  ⋮    ⋮          ⋮    ⋮
istp  (FD(i), i = 1, 3)
      (FA(i), i = 1, nr)
  ⋮    ⋮          ⋮    ⋮
nstp  (FD(i), i = 1, 3)
      (FA(i), i = 1, nr)

```

Definition of Terms

istp:	int	current solution step
nstp:	int	total or last solution step
FD(1):	real	x-force applied at <i>istp</i>
FD(2):	real	y-force applied at <i>istp</i>
FD(3):	real	pitch moment applied at <i>istp</i>
FA(i):	real	gen. force applied to mode <i>i</i>

Comments

- This is a plain text (ASCII) file.
- All values entered into this file should be dimensional. The solver will automatically non-dimensionalize the values using the reference conditions specified in the solver control file. The units of mass, length, and time in this file should match those in the controls file `case.con`.
- The forces applied to the three translational degrees of freedom are specified relative to the inertial coordinate system, i.e. as seen by a stationary observer on the ground.
- The specified forces are read one line at a time following each solution step.
- Up to `nstp` forces may be specified, but are not required. The last force read in by the solver will be applied for all remaining solution steps.
- The force coefficients `FD` and generalized forces `FA` in this input file are non-dimensionalized using the reference conditions specified in the control file `case.con`:

$$FD(1) = \frac{F'_x}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L} \quad FD(2) = \frac{F'_y}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L}$$

$$FD(3) = \frac{M'_0}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^2} \quad FA(i) = \frac{F_i}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^2}$$

where M is the free-stream Mach number and L is the reference dimension, both appearing in `case.con`.

Sample File (*nr* = 3)

```
0  0.00d+00  0.00d+00  0.00d+00  0.00d+00  0.00d+00  0.00d+00
1  0.00d+00
2  0.00d+00
3  0.00d+00
4  0.00d+00
5  0.00d+00
6  0.00d+00
7  0.00d+00
8  0.00d+00
9  0.00d+00
10 0.00d+00
11 0.00d+00
12 0.00d+00
13 0.00d+00
14 0.00d+00
15 0.00d+00
16 0.00d+00
17 0.00d+00
18 0.00d+00
19 0.00d+00
20 0.00d+00
⋮  ⋮           ⋮           ⋮           ⋮           ⋮
```

Combustion Input File (case.cmb)

Basic File Format

Line of Text

nn

Line of Text

(AMDOT(i), AMDOTE(i)) (i = 1, nn)

Definition of Terms

nn: int number of nodes or elements

AMDOT(i): real mass generation at node or element i (dim¹)

AMDOTE(i): real enthalpy generation at node or element i (dim¹)

Comments

- This is a plain text (ASCII) file.
- The value of nn must match either the number of nodes ($i_{\text{comb}} = 1$) or the number of element ($i_{\text{comb}} = 2$), specified by the combustion option. Values for mass and heat generation must be specified at all nodes or elements in the domain. Nodes or elements that do not contain influences from combustion should be written as zeros.
- The mass and heat generations are non-dimensionalized using $\text{refdim}(L)$, $\text{mach}(M)$, and ainf , so the dimensions of these variables should be consistent:

$$\dot{\rho}^* = \frac{\dot{\rho} L}{\rho_{\infty} M a_{\infty}}$$

$$\dot{\rho}H^* = \frac{\dot{\rho}H L}{\rho_{\infty} M^3 a_{\infty}^3}$$

Sample File

```
$ Combustion input file: nn
  30
$ Nodal combustion values
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0130507508 0.2871165176
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0130507508 0.2871165176
0.0130507508 0.2871165176
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0130507508 0.2871165176
0.0130507508 0.2871165176
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0130507508 0.2871165176
0.0130507508 0.2871165176
0.0000000000 0.0000000000
0.0130507508 0.2871165176
0.0130507508 0.2871165176
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0000000000 0.0000000000
0.0000000000 0.0000000000
```

Engine Conditions Input File (case.eng)

Basic File Format

```

Line of Text
  nrbc nebc
Line of Text
  nrstp gain
Line of Text
  (i isrf ptin Hin) (i = 1,nrbc)
Line of Text
  (i nis nos
   isrf(1) isrf(2) ... isrf(nis)
   jsrf(1) jsrf(2) ... jsrf(nos)
   mdotin
   mdotf thrust mdotHf )
                                     (i = 1,nebc)

```

Definition of Terms

nrbc:	int	number of rocket conditions
nebc:	int	number of engine conditions
nrstp:	int	number of iterations to converge the rocket total pressure / enthalpy
gain:	real	gain for engine controllers
isrf:	int	surface index in case.sur
ptin:	real	total press. at inflow plane
Hin:	real	total enthalpy at inflow plane
nis:	int	number of inflow surfaces
nos:	int	number of outflow surfaces
isrf(i):	real	inflow surface indices (*.sur)
jsrf(i):	real	outflow surface indices (*.sur)
mdotin:	real	design mass flow rate (inflow)
mdotf:	real	fuel flow rate
thrust:	real	uninstalled thrust
mdotHf:	real	fuel heat generation rate

Comments

- This is a plain text (ASCII) file.
- Rocket boundary conditions: nrbc is the number of surfaces that will be used to generate rocket outflow boundary conditions.
 - Each rocket BC corresponds to a surface (isrf) in case.sur.
 - Each rocket BC is specified using three values: Total pressure, total enthalpy, and number of iterations. The Mach number is calculated using the specified total pressure and downstream static pressure. The total properties are increased linearly from freestream total properties to the specified totals in nter iterations.
 - The total pressure and enthalpy in this file are dimensional quantities. The flow solver non-dimensionalizes these values using mach (M) and ainF, so the dimensions of these variables should be consistent:

$$p_t^* = \frac{p_t}{\rho_\infty M^2 a_\infty^2} \qquad H_t^* = \frac{H_t}{M^2 a_\infty^2} = \frac{c_p T_t}{M^2 a_\infty^2}$$

- The total pressure and enthalpy along the rocket boundary is increased linearly over the course of nrstp iterations. After nrstp iterations, the total properties are held at their specified values. The properties are held constant for the entirety of

restarted solutions (`istrt = .true.` then `nrstp = 0`). The following guidelines are given for `nrstp`: $5,000 < nrstp < 10,000$.

- **Engine boundary conditions:** `nebc` is the number of combinations of inflow and outflow engine surfaces. For example, one combination may be the inflow and outflow plane of the core of a turbofan engine, while a second combination is the inflow and outflow planes of the bypass flow. Or, a combination can be used to model a dual inlet that exhausts through a single nozzle.
 - Each engine combination corresponds to a number of inflow surfaces (`nis`, `isrf`) and outflow surfaces (`nos`, `jsrf`) in `case.sur`.
 - The inflow condition is specified using a design mass flow rate and static pressure. The mass flow rate is calculated by integrating the normal mass flux over the inflow plane. If the mass flow rate is lower than the design value, the static pressure is lowered. If the mass flow rate is too high, the static pressure at the inflow plane is increased.
 - The outflow condition is specified using the fuel flow rate, uninstalled thrust, and fuel heat generation rate. The inflow mass flow rate and fuel flow rate are added together to obtain the outflow mass flow rate. Likewise, the uninstalled thrust and fuel heat generation rate are added to the momentum and energy fluxes, respectively to obtain properties at the outflow plane.
 - The uninstalled thrust is calculated between the two specified planes, not including any inlets or nozzles upstream or downstream of the engine boundaries.
 - The mass flow rate, pressures, uninstalled thrusts, and heat generation rates in this file are dimensional quantities. The flow solver non-dimensionalizes these values using `refdim` (L), `mach` (M), and `ainf`, so the dimensions of these variables should be consistent:

$$\dot{m}_{in,D}^* = \frac{\dot{m}_{in,D}}{\rho_{\infty} M a_{\infty} L^2} \quad p_o^* = \frac{p_o}{\rho_{\infty} M^2 a_{\infty}^2}$$

$$\dot{m}_f^* = \frac{\dot{m}_f}{\rho_{\infty} M a_{\infty} L} \quad F^* = \frac{F}{\rho_{\infty} M^2 a_{\infty}^2 L} \quad \dot{m}H_f^* = \frac{\dot{m}H_f}{\rho_{\infty} M^3 a_{\infty}^3 L}$$

- The mass flow rate entering an engine inflow plane is controlled by adjusting the pressure along the inflow boundary. The pressure is adjusted according to the control scheme:

$$p^{(n+1)} = p^{(n)} + k \left(\frac{\dot{m}_{in}^{(n)}}{\dot{m}_{in,D}} - 1 \right)$$

where k is the controller gain (`gain`). The following guidelines are given for `gain`: $0.0003 < gain < 0.001$.

- Each surface in `case.sur` should only be connected to one rocket boundary condition or one engine combination. Surfaces connected to a rocket or engine BC should be indicated by a 5, 6, 7 or 8 in `case.bco` so not to be confused with the other boundary conditions. Euler2D checks to see that all boundary elements listed in the geometry file (`case.g2d`) are used in either the LBE (wall, symmetry, or far field) or propulsion boundaries (rocket or engine planes). The program terminates if the element count used does not match that predicted by the numbers in the `case.g2d` and `case.eng`.

Sample File

```

$ Engine conditions input file
  2      2
$ Overall engine controls (nrstp, gain)
 10000  0.0004
$ Rocket conditions
  1  4   792816.51701   1302510.0375
  2  6   792816.51701   1302510.0375
$ Engine conditions
  1  2  1
      8  9
      2
      193.53073
      0.1562459  122326.09  3.4374098
  1  2  1
      18  19
      12
      580.59219
      0.0000000  48930.436  0.0000000

```

Acoustic Input File (case.acst)

Basic File Format

```
Line of Text
  ipres idens iuvel ivvel
Line of Text
  nacp nacl
Line of Text
  x(1)    y(1)
  :      :
  x(nacp) y(nacp)
Line of Text
  x1(1)   y1(1)   x2(1)   y2(1)
  :      :      :      :
  x1(nacl) y1(nacl) x2(nacl) y2(nacl)
```

Definition of Terms

ipres:	int	pressure output flag
idens:	int	density output flag
iuvel:	int	u-velocity output flag
ivvel:	int	v-velocity output flag
nacp:	int	number of acoustic points
nacl:	int	number of acoustic lines
x(:):	real	x-coordinate of acoustic point
y(:):	real	y-coordinate of acoustic point
x1(:):	real	x-coord - starting point of line
y1(:):	real	y-coord - starting point of line
x2(:):	real	x-coord - ending point of line
y2(:):	real	y-coord - ending point of line

Comments

- This is a plain text (ASCII) file.
- One of the four acoustics flags must be on, in order for the output files to be created and data written out.
 - If `ipres = 1`, then coefficient of pressure data is written to the `case.pac` file.
 - If `idens = 1`, then density data is written to the `case.rac` file.
 - If `iuvel = 1`, then u-velocity data is written to the `case.uac` file.
 - If `ivvel = 1`, then v-velocity data is written to the `case.vac` file.
 - If none of the four flags are on, `ipres` is set to 1 (on).
- The header lines must still be written, even if no data is given in the corresponding section. For instance, if `nacp = 0`, the point header must still exist in the file at the 5th line. If `nacl = 0`, the line header must still exist at the end of the file. The header lines exist as place holders, so anything can be written on these lines, but the lines themselves must exist.
- Acoustic points can exist anywhere inside of the domain. These points are described by (x,y) coordinates. If the coordinates do not exist within the field, the first node of the first element is used to calculate properties written to the file(s). A warning is written in the header of these files, and the (x,y) coordinates of this point will not match those in the input file. This data should not be used but treated as a place holder.
- The acoustic lines are represented by drawing a line from a starting point (x_1,y_1) to an ending point (x_2,y_2) . Any segment in the solution domain that crosses this line is used to calculate properties at the intersection (of the acoustic line and segment). Intersections along a particular line are listed in order from starting to ending points.

Sample File

```
$ Acoustic output file flags
  1  0  1  0
$ Number of points and lines
  4  2
$ Coordinates of acoustic points
  0.3  0.0
  0.7  0.1
  0.1  0.0
  0.8  0.8
$ Coordinates of acoustic lines
  0.5  0.0  0.5  1.0
  0.0  0.0  1.0  0.0
```

Unknowns Output File (case.un#)

Basic File Format

```
np gam xmi alp ref t  
( (UN(i,j), i = 1, nnd ), j = 1, 5)
```

Definition of Terms

np:	int	number of nodes
gam:	real	ratio of specific heats
xmi:	real	free-stream Mach number
alp:	real	free-stream angle of attack
ref:	real	reference dimension
t:	real	dimensionless time

UN(i,1):	real	density for node <i>i</i>
UN(i,2):	real	x-velocity for node <i>i</i>
UN(i,3):	real	y-velocity for node <i>i</i>
UN(i,4):	real	pressure for node <i>i</i>
UN(i,5):	real	total enthalpy for node <i>i</i>

Comments

- This is an unformatted (binary) file.
- The solution unknowns stored in this file are dimensionless quantities.
- For dynamic (non-inertial) problems, the solution unknowns stored in the file are relative quantities referenced to the body-fixed coordinate system. The fluid velocities (in the inertial frame) can be calculated by subtracting out the translational and rotational components of the body-fixed coordinate system.

Residuals Output File (`case.rsd`)

Basic File Format

```
1      (RSD(i), i = 1,4)
:
:
istp   (RSD(i), i = 1,4)
:
:
nstp   (RSD(i), i = 1,4)
```

Definition of Terms

istp:	int	current solution step
nstp:	int	total or last solution step
RSD(1):	real	density solution residual
RSD(2):	real	x-momentum solution residual
RSD(3):	real	y-momentum solution residual
RSD(4):	real	energy solution residual

Comments

- This is a plain text (ASCII) file.
- For steady problems, the solution residuals indicate the degree of convergence to the final steady state solution. All four solution residuals should converge to approximately the same order of magnitude.
- For unsteady problems, the solution residuals indicate the degree of convergence for each global step of the solution, or the degree of convergence for the steady solution that is solved at each step.

Sample File

```
1 0.38320E-05 0.10743E-04 0.69854E-05 0.10598E-03
2 0.20317E-05 0.50694E-05 0.40436E-05 0.56307E-04
3 0.12024E-05 0.35187E-05 0.26241E-05 0.32195E-04
4 0.91334E-06 0.25166E-05 0.23637E-05 0.24240E-04
5 0.73183E-06 0.19442E-05 0.22228E-05 0.19376E-04
6 0.59870E-06 0.16179E-05 0.20889E-05 0.15963E-04
7 0.51663E-06 0.14311E-05 0.19719E-05 0.13946E-04
8 0.44924E-06 0.12989E-05 0.18536E-05 0.12398E-04
9 0.39510E-06 0.12095E-05 0.17283E-05 0.11156E-04
10 0.34726E-06 0.11478E-05 0.15878E-05 0.99450E-05
11 0.30775E-06 0.10746E-05 0.14329E-05 0.88159E-05
12 0.26207E-06 0.98700E-06 0.12833E-05 0.76280E-05
13 0.22418E-06 0.87924E-06 0.11245E-05 0.65113E-05
14 0.18904E-06 0.77764E-06 0.98148E-06 0.54617E-05
15 0.15809E-06 0.69345E-06 0.84471E-06 0.44739E-05
16 0.13411E-06 0.62203E-06 0.72991E-06 0.37422E-05
17 0.11564E-06 0.55717E-06 0.64350E-06 0.32661E-05
18 0.10516E-06 0.50502E-06 0.57520E-06 0.30152E-05
19 0.10101E-06 0.46193E-06 0.53100E-06 0.29279E-05
20 0.98711E-07 0.43618E-06 0.49934E-06 0.28901E-05
  ..
  ..
```

Residuals Output File (`case.rsd2`)

Basic File Format

```
1 (RSD(i), i = 1,4) 1
1 (RSD(i), i = 1,4) 2
: : :
1 (RSD(i), i = 1,4) icyc
: : :
istp (RSD(i), i = 1,4) 1
: : :
nstp (RSD(i), i = 1,4) 1
: : :
```

Definition of Terms

istp:	int	current solution step
icyc:	int	iteration of current residual
nstp:	int	total or last solution step
RSD(1):	real	density solution residual
RSD(2):	real	x-momentum solution residual
RSD(3):	real	y-momentum solution residual
RSD(4):	real	energy solution residual

Comments

- This is a plain text (ASCII) file.
- This file is output when `irsds = .true.` in the controls `case.con` file. The residuals shown in this file represent the RMS changes at all nodes in the domain for this inner cycle. The convergence of residuals within any iteration can be seen in the trend in the residuals through the cycles used.

Sample File

1	0.38320E-05	0.10743E-04	0.69854E-05	0.10598E-03	1
1	0.20317E-05	0.50694E-05	0.40436E-05	0.56307E-04	2
1	0.12024E-05	0.35187E-05	0.26241E-05	0.32195E-04	3
1	0.91334E-06	0.25166E-05	0.23637E-05	0.24240E-04	4
1	0.73183E-06	0.19442E-05	0.22228E-05	0.19376E-04	5
1	0.59870E-06	0.16179E-05	0.20889E-05	0.15963E-04	6
1	0.51663E-06	0.14311E-05	0.19719E-05	0.13946E-04	7
1	0.44924E-06	0.12989E-05	0.18536E-05	0.12398E-04	8
2	0.39510E-06	0.12095E-05	0.17283E-05	0.11156E-04	1
2	0.34726E-06	0.11478E-05	0.15878E-05	0.99450E-05	2
2	0.30775E-06	0.10746E-05	0.14329E-05	0.88159E-05	3
2	0.26207E-06	0.98700E-06	0.12833E-05	0.76280E-05	4
2	0.22418E-06	0.87924E-06	0.11245E-05	0.65113E-05	5
2	0.18904E-06	0.77764E-06	0.98148E-06	0.54617E-05	6
2	0.15809E-06	0.69345E-06	0.84471E-06	0.44739E-05	7
2	0.13411E-06	0.62203E-06	0.72991E-06	0.37422E-05	8
3	0.11564E-06	0.55717E-06	0.64350E-06	0.32661E-05	1
3	0.10516E-06	0.50502E-06	0.57520E-06	0.30152E-05	2
3	0.10101E-06	0.46193E-06	0.53100E-06	0.29279E-05	3
3	0.98711E-07	0.43618E-06	0.49934E-06	0.28901E-05	4
∴	∴	∴	∴	∴	∴

Cycles Output File (case.cyc)

Basic File Format

```
1      icyc(1)
:      :
istp   icyc(istp)
:      :
nstp   icyc(nstp)
```

Definition of Terms

istp:	int	current solution step
nstp:	int	total or last solution step
icyc():	int	number of cycles used to converge this iteration

Comments

- This is a plain text (ASCII) file.
- When the energy residual becomes lower than the residual tolerance `rsdtol`, then the iteration is said to be converged. The number of cycles to converge the solution is written to this file.
- The number of cycles can be used to rate the effectiveness of the unsteady time step `dt`. If too many cycles are being required, the time step can be lowered, decreasing the changes to the field. (This assumes that an appropriate residual tolerance has been chosen.)
- The following is a good practice for finding a sufficient number of iterations for unsteady solutions:
 - Select initial values for `dt`, `ncyc`, and `rsdtol`.
 - Step the solution forward 20-50 iterations.
 - Check the *.cyc file for the number of cycles required per iteration. The number of cycles should level off after 10 iterations. If not, run enough iterations for the number of required cycles to level off.
 - If the last 10 iterations require more than 20 cycles, lower the time step.
 - If the last 10 iterations require less than 8 iterations, increase the time step.
 - The sweet spot is 10-15 iterations.

Sample File

1	20
2	20
3	20
4	20
5	19
6	19
7	20
8	19
9	19
10	19
11	19
12	19
13	18
14	19
15	18
16	18
17	18
18	19
19	18
20	18
⋮	⋮

Time Step Output File (`case.time`)

Basic File Format

```
utime  
:  
:
```

Definition of Terms

utime: real unsteady time step ratio

Comments

- This is a plain text (ASCII) file.
- The time step file `case.time` is written out when the residuals watching flag is turned on (`irsds = .true.`).
- The ratio is calculated using the following equation:

$$utime = MIN\left(\frac{(\Delta t_{inv})_n}{dt}, 1\right)$$

where $(\Delta t_{inv})_n$ is the inviscid local time step on node n and dt is the global time step.

- The first column should be less than unity (1). The ratio of local to global time step acts like a relaxation factor in the solver. The solver limits this ratio to a maximum of unity, so unity in the first column shows that the global time step is smaller than all of the local time steps on the domain. The global time step should be increased until a value less than unity is reached somewhere on the domain.
- The number of cycles `ncyc` can also be gauged using the first column. The number of cycles should be greater than $dt / \Delta t_{min}$ to maintain a reasonable assumption of time accuracy. Column represents the minimum of $\Delta t_{min} / dt$ on the domain, or the maximum cycles needed on the domain.

Sample File

```
0.85620E+00
0.87930E+00
0.78200E+00
0.79380E+00
0.79380E+00
0.79381E+00
0.79380E+00
0.79380E+00
0.79379E+00
0.79380E+00
0.79380E+00
0.79380E+00
0.79378E+00
0.79380E+00
0.79380E+00
0.79381E+00
0.79380E+00
0.79380E+00
0.79382E+00
0.79382E+00
:
```

Aerodynamic Loads Output File (`case.lds`)

Basic File Format

```

0      0.0      (FD(i), i = 1,3)
:      :      :
:      :      :
istp   t_istp   (FD(i), i = 1,3)
:      :      :
:      :      :
nstp   t_nstp   (FD(i), i = 1,3)

```

Definition of Terms

```

istp:  int    current solution step
nstp:  int    total or last solution step
t:     real   dimensionless time at step i

FD(1): real   x-force coefficient
FD(2): real   y-force coefficient
FD(3): real   moment coefficient

```

Comments

- This is a plain text (ASCII) file.
- The force coefficients `FD` in this output file are dimensionless values based on the reference conditions specified in the solver control file `case.con`:

$$FD(1) = \frac{F'_x}{\frac{1}{2}\rho_\infty M^2 a_\infty^2 L} \quad FD(2) = \frac{F'_y}{\frac{1}{2}\rho_\infty M^2 a_\infty^2 L} \quad FD(3) = \frac{M'_0}{\frac{1}{2}\rho_\infty M^2 a_\infty^2 L^2}$$

where M is the free-stream Mach number and L is the reference dimension, both appearing in `case.con`.

- The moment coefficient is calculated in reference to the origin of the mesh.
- For dynamic (non-inertial) problems, the force coefficients stored in this file are referenced to the body-fixed coordinate system.

Sample File

0	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
1	0.10000E+00	0.29186E+01	0.40893E-04	0.20386E-04
2	0.20000E+00	0.53878E+01	0.74036E-04	0.36920E-04
3	0.30000E+00	0.73426E+01	0.10798E-03	0.53867E-04
4	0.40000E+00	0.87754E+01	0.14611E-03	0.72920E-04
5	0.50000E+00	0.97221E+01	0.18835E-03	0.94033E-04
6	0.60000E+00	0.10250E+02	0.23574E-03	0.11772E-03
7	0.70000E+00	0.10444E+02	0.26257E-03	0.13112E-03
8	0.80000E+00	0.10391E+02	0.25348E-03	0.12657E-03
9	0.90000E+00	0.10173E+02	0.25302E-03	0.12632E-03
10	0.10000E+01	0.98630E+01	0.23377E-03	0.11668E-03
11	0.11000E+01	0.95136E+01	0.20482E-03	0.10219E-03
12	0.12000E+01	0.91543E+01	0.19370E-03	0.96640E-04
13	0.13000E+01	0.88118E+01	0.23365E-03	0.11662E-03
14	0.14000E+01	0.85116E+01	0.28553E-03	0.14256E-03
15	0.15000E+01	0.82553E+01	0.37539E-03	0.18747E-03
16	0.16000E+01	0.80367E+01	0.55544E-03	0.27749E-03
17	0.17000E+01	0.78461E+01	0.76662E-03	0.38306E-03
18	0.18000E+01	0.76747E+01	0.10095E-02	0.50449E-03
19	0.19000E+01	0.75147E+01	0.12664E-02	0.63292E-03
20	0.20000E+01	0.73607E+01	0.15058E-02	0.75262E-03
⋮	⋮	⋮	⋮	⋮

Dynamic Output File (xd.dat)

Basic File Format

```

0      0.0      (XD(i), i = 1,6)
              (FD(i), i = 1,3)
:      :      :      :
:      :      :      :
istp   t_istp   (XD(i), i = 1,6)
              (FD(i), i = 1,3)
:      :      :      :
:      :      :      :
nstp   t_nstp   (XD(i), i = 1,6)
              (FD(i), i = 1,3)

```

Definition of Terms

istp:	int	current solution step
nstp:	int	total or last solution step
t:	real	dimensionless time at step i
XD(1):	real	x- position
XD(2):	real	y- position
XD(3):	real	pitch angle (rad)
XD(4):	real	x- velocity
XD(5):	real	y- velocity
XD(6):	real	pitch rate (rad/s)
FD(1):	real	x- force
FD(2):	real	y- force
FD(3):	real	pitch moment

Comments

- This is a plain text (ASCII) file.
- The dynamic data in this output file is dimensionless based on the reference conditions specified in the solver control file `case.con`:

$$\begin{aligned}
 XD(1) &= \frac{x_0}{L} & XD(2) &= \frac{y_0}{L} & XD(3) &= \theta_0 \text{ (rad)} \\
 XD(4) &= \frac{V_x}{M a_\infty} & XD(5) &= \frac{V_y}{M a_\infty} & XD(6) &= \frac{L}{M a_\infty} q_0 \text{ (rad / s)} \\
 FD(1) &= \frac{F'_x}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L} & FD(2) &= \frac{F'_y}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L} & FD(3) &= \frac{M'_0}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^2}
 \end{aligned}$$

where M is the Mach number and L is the reference dimension in `case.con`.

- The position, velocity, and acceleration vectors in this file are defined relative to the global coordinate system, while the rotational quantities are defined as rotations about the local or body-fixed coordinate system.

Sample File

```
1 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
2 0.20317E-05 0.50694E-05 0.40436E-05 0.40436E-05 0.56307E-04 0.56307E-04 0.32195E-04 0.32195E-04 0.24240E-04 0.24240E-04
3 0.12024E-05 0.35187E-05 0.26241E-05 0.26241E-05 0.23637E-05 0.23637E-05 0.22228E-05 0.22228E-05 0.19376E-04 0.19376E-04
4 0.91334E-06 0.25166E-05 0.25166E-05 0.25166E-05 0.20889E-05 0.20889E-05 0.19719E-05 0.19719E-05 0.15963E-04 0.15963E-04
5 0.73183E-06 0.19442E-05 0.19442E-05 0.19442E-05 0.18536E-05 0.18536E-05 0.17283E-05 0.17283E-05 0.13946E-04 0.13946E-04
6 0.59870E-06 0.16179E-05 0.16179E-05 0.16179E-05 0.15878E-05 0.15878E-05 0.14329E-05 0.14329E-05 0.11566E-04 0.11566E-04
7 0.51663E-06 0.14311E-05 0.14311E-05 0.14311E-05 0.14329E-05 0.14329E-05 0.12989E-05 0.12989E-05 0.99450E-05 0.99450E-05
8 0.44924E-06 0.12989E-05 0.12989E-05 0.12989E-05 0.12095E-05 0.12095E-05 0.11478E-05 0.11478E-05 0.88159E-05 0.88159E-05
9 0.39510E-06 0.11478E-05 0.11478E-05 0.11478E-05 0.10746E-05 0.10746E-05 0.10746E-05 0.10746E-05 0.76280E-05 0.76280E-05
10 0.34726E-06 0.10746E-05 0.10746E-05 0.10746E-05 0.98700E-06 0.98700E-06 0.98700E-06 0.98700E-06 0.65113E-05 0.65113E-05
11 0.30775E-06 0.98700E-06 0.98700E-06 0.98700E-06 0.87924E-06 0.87924E-06 0.87924E-06 0.87924E-06 0.54617E-05 0.54617E-05
12 0.26207E-06 0.87924E-06 0.87924E-06 0.87924E-06 0.84471E-06 0.84471E-06 0.84471E-06 0.84471E-06 0.44739E-05 0.44739E-05
13 0.22418E-06 0.84471E-06 0.84471E-06 0.84471E-06 0.77764E-06 0.77764E-06 0.77764E-06 0.77764E-06 0.37422E-05 0.37422E-05
14 0.18904E-06 0.77764E-06 0.77764E-06 0.77764E-06 0.69345E-06 0.69345E-06 0.69345E-06 0.69345E-06 0.32661E-05 0.32661E-05
15 0.15809E-06 0.69345E-06 0.69345E-06 0.69345E-06 0.62203E-06 0.62203E-06 0.62203E-06 0.62203E-06 0.30152E-05 0.30152E-05
16 0.13411E-06 0.62203E-06 0.62203E-06 0.62203E-06 0.57520E-06 0.57520E-06 0.57520E-06 0.57520E-06 0.29279E-05 0.29279E-05
17 0.11564E-06 0.57520E-06 0.57520E-06 0.57520E-06 0.53100E-06 0.53100E-06 0.53100E-06 0.53100E-06 0.28991E-05 0.28991E-05
18 0.10516E-06 0.50502E-06 0.50502E-06 0.50502E-06 0.49934E-06 0.49934E-06 0.49934E-06 0.49934E-06 0.28991E-05 0.28991E-05
19 0.10101E-06 0.46193E-06 0.46193E-06 0.46193E-06 0.43618E-06 0.43618E-06 0.43618E-06 0.43618E-06 0.28991E-05 0.28991E-05
20 0.98711E-07 0.43618E-06 0.43618E-06 0.43618E-06 0.43618E-06 0.43618E-06 0.43618E-06 0.43618E-06 0.28991E-05 0.28991E-05
... ..
... ..
... ..
```

Dynamic Output File (xn.dat)

Basic File Format

```

0      0.0      (XN(i), i = 1,nr)
              (VN(i), i = 1,nr)
              (FA(i), i = 1,nr)

:      :      :      :
istp   t_istp   (XN(i), i = 1,nr)
              (VN(i), i = 1,nr)
              (FA(i), i = 1,nr)

:      :      :      :
nstp   t_nstp   (XN(i), i = 1,nr)
              (VN(i), i = 1,nr)
              (FA(i), i = 1,nr)

```

Definition of Terms

```

istp:  int    current solution step
nstp:  int    total or last solution step
t:     real   dimensionless time at step i

XN(i): real   generalized displ. on mode i
VN(i): real   generalized velocity on mode i
FA(i): real   generalized force on mode i

```

Comments

- This is a plain text (ASCII) file.
- The number of modes nr is given in the control file `case.con`.
- The elastic data in this output file is dimensionless based on the reference conditions specified in the solver control file `case.con`:

$$XN(i) = x_{n,i} \qquad VN(i) = \frac{L}{M a_\infty} \dot{x}_{n,i} \qquad FA(i) = \frac{F_{n,i}}{\rho_\infty M^2 a_\infty^2 L^2}$$

where M is the Mach number and L is the reference dimension in `case.con`.

- To reassemble the elastic displacements at all points along the surface of the model, use the modal deflections (mode shapes) `PHIA` in the elastics file (`case.vec`) and $x_{n,i}$. The displacement vector at the k^{th} node is:

$$\vec{\delta}(k) = \sum_{i=1}^N x_{n,i} \begin{Bmatrix} PHIA(2k-1, i) \\ PHIA(2k, i) \end{Bmatrix}$$

where N is the number of mode shapes in the elastic system. The boundary velocity at the k^{th} node is:

$$\vec{V}_b(k) = \sum_{i=1}^N \dot{x}_{n,i} \begin{Bmatrix} PHIA(2k-1, i) \\ PHIA(2k, i) \end{Bmatrix}$$

Restart Files (case.rst and case.rs2)

Basic File Format

```
istp nnd gam xmi ref dt
((UN(i,j), i = 1,nnd), j = 1,5)
((UNO(i,j), i = 1,nnd), j = 1,5)
(XN(i,j), i = 1,2*nr),
      (XN1(i), i = 1,2*nr)
(FA(i,j), i = 1,nr),
      (FA2(i), i = 1,nr)
(XD(i,j), i = 1,6),
      (XD1(i), i = 1,6)
(FD(i,j), i = 1,3),
      (FD2(i), i = 1,3)
```

Definition of Terms

istp:	int	step in global solution
nnd:	int	number of nodes
gam:	real	ratio of specific heats
xmi:	real	free-stream Mach number
ref:	real	reference dimension
dt:	real	global time step
UN:	real	unknowns at previous step
UNO:	real	unknowns at two steps prior
XN:	real	elastic deflect / velocity
XN1:	real	prev. elastic deflect / velocity
FA:	real	generalized aero. forces
FA2:	real	external forcing on modes
XD:	real	rigid body position / velocity
XD1:	real	prev. rigid body pos. / vel.
FD:	real	aero. forces on vehicle
FD2:	real	external forcing on vehicle

Comments

- This is an unformatted (binary) file.
- The solution unknowns, deflections, and forces stored in this file are dimensionless quantities.
- Unknowns properties vector UNO is only read/written for 2nd order unsteady solutions (isol = 2).
- Elastic properties XN, XN1, FA, and FA2 is only read/written when the elastics flag ielast is set to .true.
- Non-inertial properties XD, XD1, FD, and FD2 is only read/written when the non-inertial flag idynm is set to .true.

Acoustic Pressure Output File (case.pac)

Basic File Format

Controls:

```
dt = "dt"
Lref = "refdim"
Uinf = "uinf"
ainf = "ainf"
mach = "mach"
gam = "gam"
```

Data Layout:

```
nacp = "nacp"
nacl = "nacl"
-- "NN" intersections with Line #"N"
  :           :           :
-- "NN" intersections with Line #"N"

x-coord ==> "x1" "x2" || "x3" . . . .
y-coord ==> "y1" "y2" || "y3" . . . .
--- Time --- ---- ++ ---- . . . .
  "t"      "Cp1" "Cp2" || "Cp3" . . . .
  :           :   :   ||   :
  :           :   :   ||   :
```

Definition of Terms

dt:	real	current solution step
refdim:	real	total or last solution step
uinf:	real	total or last solution step
ainf:	real	total or last solution step
mach:	real	total or last solution step
gam:	real	total or last solution step
nacp:	int	number of acoustic points
nacl:	int	number of acoustic lines
NN:	int	number of intersections along a particular acoustic line
N:	int	index of this acoustic lines
x#:	real	x-coordinate at node
y#:	real	y-coordinate at node
t:	real	solution time
Cp#:	real	coefficient of pressure at node

Comments

- This is a plain text (ASCII) file.
- Text shown in the file format above without quotes (“ ”) is used directly in the output file. Text shown above in quotes represents a variable, or number written to the file. These variables are defined above on the right.
- Descriptions may be included in the output file to the right of line of header data.
- The first section represents solution controls used to generate the data file.
- The second section describes the data in the third section. If any of the acoustic points are not found within the solution domain, a warning is written before the number of acoustic lines is written. The number of intersections is listed for each acoustic line in the input file.
- The data is written out so that each column represents a node or intersection within the domain. The first two rows give the (x,y) coordinate of the node or intersection. The rows under the dashed divider line are the coefficient of pressure at the node or intersection point at the solution time designated in the first column.
- A double vertical line breaks the acoustic point and acoustic line data. Subsequent single vertical lines break data from acoustic lines.

- The intersections along a particular acoustic line are ordered parametrically along the length of the acoustic line, from starting to end points, as listed in the input file.

Sample File

Controls:

```

dt = 0.00100000 Dimensionless time step
Lref = 1.00000000 Reference dimension
Uinf = 781.20000000 Freestream velocity
ainf = 1116.00000000 Freestream acoustic speed
mach = 0.70000000 Freestream Mach number
gam = 1.40000000 Ratio of specific heats
  
```

Data Layout:

```

nacp = 2 Number of acoustic points
nacl = 2 Number of acoustic lines
  
```

```

-- 2 intersections with Line # 1
-- 1 intersections with Line # 2
  
```

```

x-coord ==> 0.0000000 0.2000000 0.4000000 0.6000000 0.8000000 1.0000000 1.2000000 1.4000000
Y-coord ==> 0.0000000 0.7000000 1.3000000 1.9000000 2.5000000 3.1000000 3.7000000 4.3000000
-----
Time -----
0.0010000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
0.0020000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
0.0030000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
0.0040000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
:
:
:
  
```

Acoustic Density Output File (case.rac)

Basic File Format

Controls:

```
dt = "dt"
Lref = "refdim"
Uinf = "uinf"
ainf = "ainf"
mach = "mach"
gam = "gam"
```

Data Layout:

```
nacp = "nacp"
nacl = "nacl"
-- "NN" intersections with Line #"N"
  :           :           :
-- "NN" intersections with Line #"N"

x-coord ==> "x1" "x2" || "x3" . . . .
y-coord ==> "y1" "y2" || "y3" . . . .
--- Time --- ---- ++ ---- . . . .
  "t"      "rh1" "rh2" || "rh3" . . . .
  :           :   :   ||   :
  :           :   :   ||   :
```

Definition of Terms

dt:	real	current solution step
refdim:	real	total or last solution step
uinf:	real	total or last solution step
ainf:	real	total or last solution step
mach:	real	total or last solution step
gam:	real	total or last solution step
nacp:	int	number of acoustic points
nacl:	int	number of acoustic lines
NN:	int	number of intersections along a particular acoustic line
N:	int	index of this acoustic lines
x#:	real	x-coordinate at node
y#:	real	y-coordinate at node
t:	real	solution time
rh#:	real	dimensionless density at node

Comments

- This is a plain text (ASCII) file.
- Text shown in the file format above without quotes (" ") is used directly in the output file. Text shown above in quotes represents a variable, or number written to the file. These variables are defined above on the right.
- Descriptions may be included in the output file to the right of line of header data.
- The first section represents solution controls used to generate the data file.
- The second section describes the data in the third section. If any of the acoustic points are not found within the solution domain, a warning is written before the number of acoustic lines is written. The number of intersections is listed for each acoustic line in the input file.
- The data is written out so that each column represents a node or intersection within the domain. The first two rows give the (x,y) coordinate of the node or intersection. The rows under the dashed divider line are the dimensionless density at the node or intersection point at the solution time designated in the first column.
- Density data is presented in dimensionless form:

$$rh\# = \frac{\rho}{\rho_{\infty}}$$

- A double vertical line breaks the acoustic point and acoustic line data. Subsequent single vertical lines break data from acoustic lines.
- The intersections along a particular acoustic line are ordered parametrically along the length of the acoustic line, from starting to end points, as listed in the input file.

Sample File

Controls:

```

dt = 0.00100000 Dimensionless time step
Lref = 1.00000000 Reference dimension
Uinf = 781.20000000 Freestream velocity
ainf = 1116.00000000 Freestream acoustic speed
mach = 0.70000000 Freestream Mach number
gam = 1.40000000 Ratio of specific heats
  
```

Data Layout:

```

nacp = 2 Number of acoustic points
nacl = 2 Number of acoustic lines
  
```

```

-- 2 intersections with Line # 1
-- 1 intersections with Line # 2
  
```

```

x-coord ==> 0.0000000 0.2000000 0.4000000 0.6000000 0.7000000 0.7000000
y-coord ==> 0.0000000 0.0000000 0.4000000 1.0000000 1.3000000 1.3000000
----- Time -----
0.0010000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
0.0020000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
0.0030000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
0.0040000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
:
:
:
  
```

Acoustic u-Velocity Output File (case.uac)

Basic File Format

Controls:

```
dt = "dt"
Lref = "refdim"
Uinf = "uinf"
ainf = "ainf"
mach = "mach"
gam = "gam"
```

Data Layout:

```
nacp = "nacp"
nacl = "nacl"
-- "NN" intersections with Line #"N"
  :           :           :
-- "NN" intersections with Line #"N"

x-coord ==> "x1" "x2" || "x3" . . . .
y-coord ==> "y1" "y2" || "y3" . . . .
--- Time --- ---- ++ ---- . . . .
  "t"      "uu1" "uu2" || "uu3" . . . .
  :         :     : ||   :
  :         :     : ||   :
```

Definition of Terms

dt:	real	current solution step
refdim:	real	total or last solution step
uinf:	real	total or last solution step
ainf:	real	total or last solution step
mach:	real	total or last solution step
gam:	real	total or last solution step
nacp:	int	number of acoustic points
nacl:	int	number of acoustic lines
NN:	int	number of intersections along a particular acoustic line
N:	int	index of this acoustic lines
x#:	real	x-coordinate at node
y#:	real	y-coordinate at node
t:	real	solution time
uu#:	real	dimensionless u-velocity at node

Comments

- This is a plain text (ASCII) file.
- Text shown in the file format above without quotes (“ ”) is used directly in the output file. Text shown above in quotes represents a variable, or number written to the file. These variables are defined above on the right.
- Descriptions may be included in the output file to the right of line of header data.
- The first section represents solution controls used to generate the data file.
- The second section describes the data in the third section. If any of the acoustic points are not found within the solution domain, a warning is written before the number of acoustic lines is written. The number of intersections is listed for each acoustic line in the input file.
- The data is written out so that each column represents a node or intersection within the domain. The first two rows give the (x,y) coordinate of the node or intersection. The rows under the dashed divider line are the dimensionless u-velocity at the node or intersection point at the solution time designated in the first column.
- u-Velocity data is presented in dimensionless form:

$$uu\# = \frac{u}{U_\infty}$$

- A double vertical line breaks the acoustic point and acoustic line data. Subsequent single vertical lines break data from acoustic lines.
- The intersections along a particular acoustic line are ordered parametrically along the length of the acoustic line, from starting to end points, as listed in the input file.

Sample File

Controls:

```

dt = 0.00100000 Dimensionless time step
Lref = 1.00000000 Reference dimension
Uinf = 781.20000000 Freestream velocity
ainf = 1116.00000000 Freestream acoustic speed
mach = 0.70000000 Freestream Mach number
gam = 1.40000000 Ratio of specific heats

```

Data Layout:

```

nacp = 2 Number of acoustic points
nacl = 2 Number of acoustic lines

```

```

-- 2 intersections with Line # 1
-- 1 intersections with Line # 2

```

```

x-coord ==> 0.0000000 0.2000000 0.4000000 0.6000000 0.7000000 1.0000000 1.3000000 1.0000000
Y-coord ==> 0.0000000 0.7000000 0.4000000 0.6000000 0.4000000 1.0000000 1.3000000 1.0000000
----- Time -----
0.0010000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
0.0020000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
0.0030000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
0.0040000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
:
:
:

```

Acoustic v-Velocity Output File (case.vac)

Basic File Format

Controls:

```
dt = "dt"
Lref = "refdim"
Uinf = "uinf"
ainf = "ainf"
mach = "mach"
gam = "gam"
```

Data Layout:

```
nacp = "nacp"
nacl = "nacl"
-- "NN" intersections with Line #"N"
  :           :           :
-- "NN" intersections with Line #"N"

x-coord ==> "x1" "x2" || "x3" . . . .
y-coord ==> "y1" "y2" || "y3" . . . .
--- Time --- ---- ++ ---- . . . .
  "t"      "vv1" "vv2" || "vv3" . . . .
  :         :     : ||   :
  :         :     : ||   :
```

Definition of Terms

dt:	real	current solution step
refdim:	real	total or last solution step
uinf:	real	total or last solution step
ainf:	real	total or last solution step
mach:	real	total or last solution step
gam:	real	total or last solution step
nacp:	int	number of acoustic points
nacl:	int	number of acoustic lines
NN:	int	number of intersections along a particular acoustic line
N:	int	index of this acoustic lines
x#:	real	x-coordinate at node
y#:	real	y-coordinate at node
t:	real	solution time
vv#:	real	dimensionless v-velocity at node

Comments

- This is a plain text (ASCII) file.
- Text shown in the file format above without quotes (“ ”) is used directly in the output file. Text shown above in quotes represents a variable, or number written to the file. These variables are defined above on the right.
- Descriptions may be included in the output file to the right of line of header data.
- The first section represents solution controls used to generate the data file.
- The second section describes the data in the third section. If any of the acoustic points are not found within the solution domain, a warning is written before the number of acoustic lines is written. The number of intersections is listed for each acoustic line in the input file.
- The data is written out so that each column represents a node or intersection within the domain. The first two rows give the (x,y) coordinate of the node or intersection. The rows under the dashed divider line are the dimensionless v-velocity at the node or intersection point at the solution time designated in the first column.
- v-Velocity data is presented in dimensionless form:

$$vv\# = \frac{v}{U_\infty}$$

- A double vertical line breaks the acoustic point and acoustic line data. Subsequent single vertical lines break data from acoustic lines.
- The intersections along a particular acoustic line are ordered parametrically along the length of the acoustic line, from starting to end points, as listed in the input file.

Sample File

Controls:

```

dt = 0.00100000 Dimensionless time step
Lref = 1.00000000 Reference dimension
Uinf = 781.20000000 Freestream velocity
ainf = 1116.00000000 Freestream acoustic speed
mach = 0.70000000 Freestream Mach number
gam = 1.40000000 Ratio of specific heats
  
```

Data Layout:

```

nacp = 2 Number of acoustic points
nacl = 2 Number of acoustic lines
  
```

```

-- 2 intersections with Line # 1
-- 1 intersections with Line # 2
  
```

x-coord ==>	0.00000000	0.20000000	0.40000000	0.60000000	0.70000000	1.00000000	1.30000000	1.00000000	0.70000000
Y-coord ==>	0.00000000	0.70000000	0.40000000	0.60000000	0.60000000	1.30000000	1.30000000	1.00000000	0.60000000
Time	0.00100000	0.00200000	0.00300000	0.00400000	0.00100000	0.00200000	0.00300000	0.00400000	0.00100000
	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000

APPENDIX D

NS2D FILE FORMATS

The file formats included in this section are used to create the input files required to operate NS2D and interpret the files that are written by NS2D. In order to make this appendix more compact, file formats that are already outlined in Appendix C are not repeated here. If additional terms or options are added to model viscous or turbulent effects, the file formats are called out in full detail here to show the differences.

NS2D

Summary of File Formats

Input Files:

- `case.g2d` (required) contains the geometry data structures representing the computational mesh as required by the flow solver (ASCII)
- `case.con` (required) contains values for the solver control parameters and flow conditions (ASCII)
- `case.unk` (optional) contains the values of the primitive flow variables (density, velocity, pressure, and total enthalpy) and the turbulent flow variables (turbulent viscosity, turbulent kinetic energy, turbulent dissipation, and model variables) for each node of the computational mesh to be used as the initial conditions for the flow solution (Binary)
- `case.dyn` (optional) contains the non-inertial matrices and initial conditions as required for a non-inertial solution (ASCII) [see Appendix C]
- `case.vec` (optional) contains the elastic mode matrices, initial conditions, and mode shape vectors for the solid wall surfaces as required for an aero-elastic solution (ASCII) [see Appendix C]
- `case.frc` (optional) contains external forces to be applied to each solution step in a dynamic or aeroelastic solution (ASCII) [see Appendix C]
- `case.cmb` (optional) contains information to represent the influence of combustion on the flow solution (ASCII) [see Appendix C]
- `case.eng` (optional) contains the properties needed to represent rocket and engine boundary conditions (ASCII) [see Appendix C]
- `case.acst` (optional) contains the acoustic output data (ASCII) [see Appendix C]
- `case.tbc` (optional) contains temperature boundary condition data (ASCII)

Output Files:

- `case.un#` contains the values of the primitive flow variables (density, velocity, pressure, and total enthalpy) and the turbulent flow variables (turbulent viscosity, turbulent kinetic energy, turbulent dissipation, and model variables) for each node

of the computational mesh; # is iterated as more files are produced so the progress of the solution can be followed (Binary)

- `caset.un#` contains the values of the turbulent flow variables (turbulent viscosity, turbulent kinetic energy, turbulent dissipation, and model variables) for each node of the computational mesh; # is iterated as more files are produced so the progress of the solution can be followed (Binary)
- `case.rsd` contains a history of the solution residuals for the conservation variables (density, momentum (2), and total energy) (ASCII)
- `case.rsd2` contains a history of the solution residuals for the conservation variables at each inner cycle (ASCII)
- `case.cyc` contains a history of the inner cycles used to converge each iteration (ASCII) [see Appendix C]
- `case.time` contains a history of the local time step ratios (ASCII)
- `case.lds` contains a history of the dimensionless aerodynamic forces acting on the solid walls of the geometry (ASCII) [see Appendix C]
- `xd.dat` contains a history of the non-inertial displacements, velocities, and accelerations for a dynamic solution (ASCII) [see Appendix C]
- `xn.dat` contains a history of the generalized displacements, velocities, and forces for an unsteady, aeroelastic solution (ASCII) [see Appendix C]
- `case.rst` and `case.rs2` contains information on up to two sets of unknowns data, elastic system data, and dynamic motion data (ASCII)
- `case.pac` contains a history of pressure data at prescribed nodes (ASCII) [see Appendix C]
- `case.rac` contains a history of density data at prescribed nodes (ASCII) [see Appendix C]
- `case.uac` contains a history of u-velocity data at prescribed nodes (ASCII) [see Appendix C]
- `case.vac` contains a history of v-velocity data at prescribed nodes (ASCII) [see Appendix C]

Geometry Input File (*case.g2d*)

Basic File Format

```
Line of Text
nnd nel nsg nbe nbp nwl nsd nwlv nsdv
Line of Text
(LBE(i), i = 1, 8)
Line of Text
(COOR(i,j), j = 1,2) (i = 1,nnd)
Line of Text
(IELM(i,j), j = 1,3) (i = 1,nel)
Line of Text
(ISEG(i,j), j = 1,2) (i = 1,nsg)
Line of Text
(IBEL(i,j), j = 1,4) (i = 1,nbe)
```

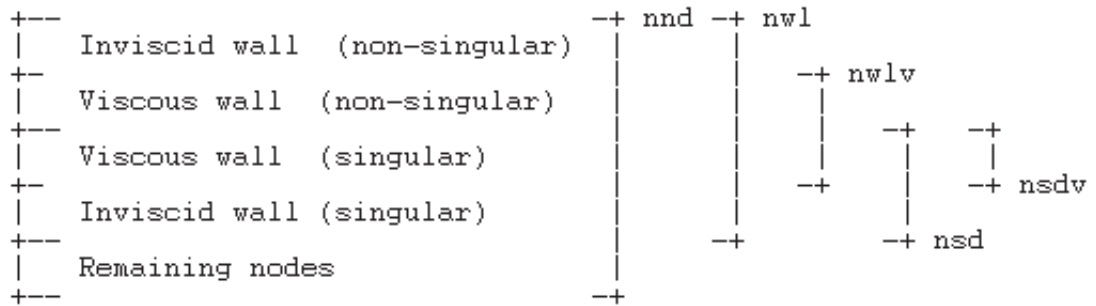
Definition of Terms

nnd:	int	number of nodes
nel:	int	number of elements
nsg:	int	number of segments
nbe:	int	number of boundary elements
nbp:	int	number of boundary points
nwl:	int	number of wall nodes
nsd:	int	number of singular nodes
nwlv:	int	number of viscous wall nodes
nsdv:	int	number of singular viscous nodes
LBE(i):	int	start/ stop index for 4 BC types
COOR(i,1):	real	x-coordinate for node <i>i</i>
COOR(i,2):	real	y-coordinate for node <i>i</i>
IELM(i,1):	int	node 1 for element <i>i</i>
IELM(i,2):	int	node 2 for element <i>i</i>
IELM(i,3):	int	node 3 for element <i>i</i>
ISEG(i,1):	int	node 1 for segment <i>i</i>
ISEG(i,2):	int	node 2 for segment <i>i</i>
IBEL(i,1):	int	node 1 for boundary elem. <i>i</i>
IBEL(i,2):	int	node 2 for boundary elem. <i>i</i>
IBEL(i,3):	int	surface index in <i>case.sur</i>
IBEL(i,4):	int	domain elem. attached to boundary elem. <i>i</i>

Comments

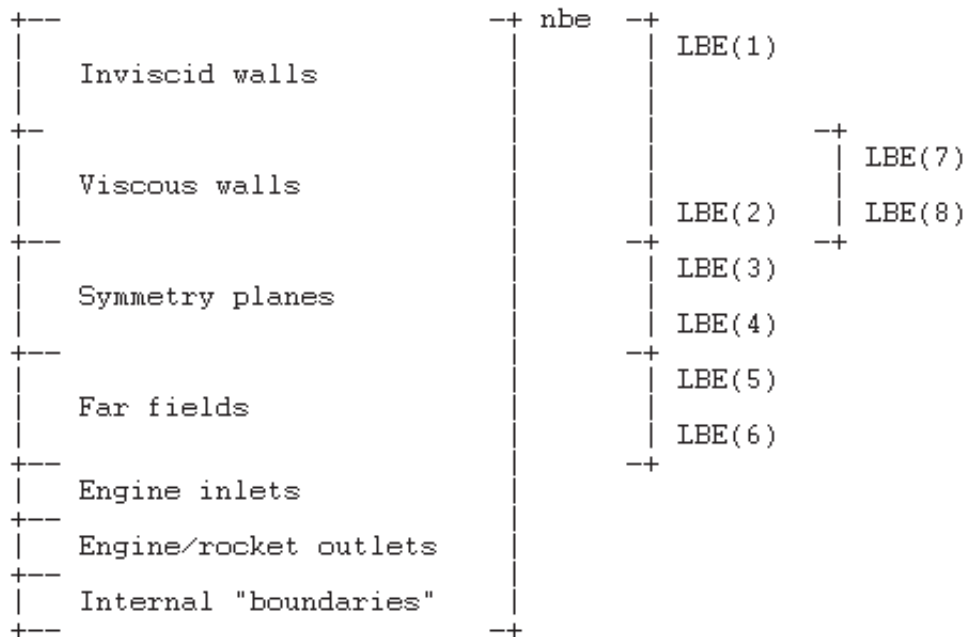
- This is a plain text (ASCII) file.
- The nodal coordinates in this file are treated as dimensional values and are non-dimensionalized using the reference dimension *refdim* specified in the control file.
- The element connectivity data *IELM* and *IBEL* define clockwise oriented elements.
- The program *makeg2d* is used to convert a standard STARS surface triangulation file *case.fro* and modified boundary conditions file *case.bco* into an appropriately sorted two-dimensional geometry file.
- Nodal data *COOR* is sorted such that the first *nwl* nodes are defined as solid wall nodes. Out of the first *nwl* nodes, the last *nsd* nodes are defined as singular nodes. The viscous nodes are placed in the middle (*nwlv* and *nsdv*), according to the following diagram:

Nodes:



- Boundary element data is sorted based on the starting/stopping indexes for the three BC types, i.e. boundary elements LBE(1) through LBE(2) are solid wall elements, LBE(3) through LBE(4) are symmetry elements, LBE(5) through LBE(6) are far-field elements, and LBE(7) through LBE(8) are viscous solid wall elements, where the two solid wall elements are restricted to $LBE(1) \leq LBE(7) < LBE(8) \leq LBE(2)$. (In other words, the viscous solid walls must exist within the limits of the viscous walls.)

Boundary elements:



Sample File

```
$ nnd, nel, nsg, nbe, nbp, nwl, nsd, nwlv, nsdv
   8     6    13     8     8     3     0     2     0
$ LBE(8)
   1     2     3     2     3     8     1     2
$ Nodal coordinates
-.100000E+01 -.100000E+01
 0.100000E+01 -.100000E+01
 0.000000E+00 -.100000E+01
 0.100000E+01  0.100000E+01
-.100000E+01  0.100000E+01
 0.100000E+01  0.000000E+00
 0.000000E+00  0.100000E+01
-.100000E+01  0.000000E+00
$ Element connectivity
   1     3     8
   3     2     6
   5     8     7
   6     4     7
   8     3     6
   6     7     8
$ Segment connectivity
   1     3
   1     8
   2     3
   2     6
   3     8
   3     6
   4     6
   4     7
   5     8
   5     7
   6     7
   6     8
   7     8
$ Boundary edge data
   1     3     1     1
   3     2     1     2
   2     6     2     2
   6     4     2     4
   4     7     3     4
   7     5     3     3
   5     8     4     3
   8     1     4     1
```

Solver Control Input File (case . con)

Basic File Format

```

&control
  dt           = 0.1d0,
  gamma       = 1.4d0,
  diss        = 1.0d0,
  cfl         = 0.5d0,
  lamb       = -0.666d0,
  Smod        = 0.0d0,
  dislen      = 1.0d-20,

  mach        = 0.6d0,
  Re          = 2.0d5,
  Pr          = 0.7d0,
  alpha       = 0.0d0,
  refdim      = 1.0d0,

  iturb       = 0,
  turbI       = 0.01d0,
  muTinf      = 0.01d0,
  rnuinf      = 0.0d0,
  rhoKinf     = 0.0d0,
  rhoWinf     = 0.0d0,
  PrT         = 0.9d0,
  disst       = 1.0d0,

  nstp        = 100,
  nout        = 50,
  ncyc        = 4,

  rsdtol      = 1.0d-20,
  rsdmax      = 10.0d0,

  isol        = 0,
  idiss       = 0,
  icomb       = 0,
  itime       = 0,

  istrtr      = .false.,
  iaero       = .true.,
  idynm       = .false.,
  ielast      = .false.,
  iprop       = .false.,
  ifree       = .true.,
  iforce      = .false.,
  isafe       = .false.,
  irsds       = .false.,
  iacoust     = .false.,
  itempbc     = .false.,

```

Definition of Terms

```

  dt: real    dimensionless global time step
  gamma: real  ratio of specific heats
  diss: real   dissipation factor
  cfl: real    local time step stability factor
  lamb: real   ratio of 2nd over 1st viscosity ( $\lambda/\mu$ )
  Smod: real   modified Sutherland's constant
  dislen: real distance from wall where no
               artificial dissipation is used

  mach: real   freestream Mach number
  Re: real     freestream Reynolds number
  Pr: real     Prandtl number
  alpha: real   free-stream angle of attack (deg)
  refdim: real  reference length (dim'l)

  iturb: int   turbulence model flag
  turbI: real  freestream turbulence intensity
  muTinf: real freestream eddy viscosity
  rnuinf: real turbulence IC and FF BC for  $\rho\nu$ 
  rhoKinf: real turbulence IC and FF BC for  $\rho K$ 
  rhoWinf: real turbulence IC and FF BC for  $\rho\omega$ 
  PrT: real    turbulent Prandtl number
  disst: real  turbulent dissipation factor

  nstp: int    total solution steps
  nout: int    output frequency, steps/output
  ncyc: int    iterative cycles per solution step

  rsdtol: real energy residual converg tolerance
  rsdmax: real energy residual divergence criteria

  isol: int    CFD solution type
  idiss: int   dissipation type
  icomb: int   combustion model type
  itime: int   viscous local time stepping flag

  istrtr: log  restart flag
  iaero: log   aerodynamic forces flag
  idynm: log   dynamic/non-inertial flag
  ielast: log  elastic flag
  iprop: log   propulsion flag
  ifree: log   free-stream velocity flag
  iforce: log  external forces flag
  isafe: log   safe-mode flag
  irsds: log   residual watching flag
  iacoust: log acoustics output flag
  itempbc: log temperature BC flag

```

```
nr = 0,
```

<pre> ainf = 1.0d0, rhoinf = 1.0d0, gravity = 0.0d0, / </pre>		<pre> nr: int number of elastic modes ainf: real free-stream sonic speed (dim'l) rhoinf: real free-stream density (dim'l) gravity: real gravity (dim'l) </pre>
---	--	--

Comments

- This is a plain text (ASCII) file formatted as a Fortran namelist.
 - The namelist must begin with the line “&control” and end with the line “/”.
 - The remaining lines can be listed in any order or omitted, if desired.
 - The intermediate lines work like variable assignments with the loose format: *variable_name* = *value*, followed by a comma.
 - Integers (*int*) are listed as whole numbers.
 - Real numbers (*real*) are listed in double precision, scientific notation: *###d+###*.
 - Logical variables (*log*) are listed as either “.true.” or “.false.”.
 - Lines can be commented out by including an exclamation point “!” prior to any other information on the line.
 - The default values, shown above, are used for variables omitted or commented out of the namelist.
 - Any information listed after the last line of the namelist “/” are not read by the program and can be used to store notes and other calculations.
- The global time step Δt is only used for unsteady solutions. Δt is a dimensionless value calculated: $\Delta t = \Delta t U / L$, where Δt is the dimensional time step, U is the free-stream velocity ($= \text{mach } a_{inf}$), and L is the reference length $refdim$.
- Appropriate values for the dissipation factor are in the range $0.0 < diss \leq 2.0$. Some dissipation is required to stabilize the solution, but too much dissipation will corrupt the solution and possibly be a destabilizing influence.
- The local time step stability factor is a safety factor used to compute local time steps for each solution step. For steady solutions, a stability factor of 0.8 is typical for most problems. For unsteady solutions, the stability factor is typically $0.3 \leq cfl \leq 0.8$.
- The coefficient λ is the ratio of second to first viscosity λ/μ , which is considered constant across the domain. λ is restricted to be greater than or equal to -2/3 to maintain positive viscous dissipation in the energy equation (Stoke’s hypothesis).
- The coefficient S_{mod} is the modified Sutherland’s coefficient: $S_{mod} = c_p S / U_{inf}^2$, where c_p is the specific heat at constant pressure, S is Sutherland’s constant, and U_{inf} is the freestream velocity ($= \text{mach } a_{inf}$). If S_{mod} is set to zero, the viscosity is constant, equal to the freestream viscosity throughout the domain. If $S_{mod} > 0$, Sutherland’s equation is used to vary viscosity with temperature (enthalpy).
- The values of $refdim$, $mach$, $ainf$, and $rhoinf$ are used to non-dimensionalize all values read into the flow solver.
- Prandtl number Pr is used to relate viscosity and thermal conductivity k . The freestream Reynolds number Re is used to relate the density and velocity in the freestream to the viscosity in the freestream, using L as $refdim$:

$$\text{Pr} = \frac{c_p \mu}{k} \quad \text{Re} = \frac{\rho_\infty U_\infty L}{\mu_\infty} \quad U_\infty = M_\infty a_\infty$$

- The free-stream angle of attack is ignored for dynamic (non-inertial) problems.
- Turbulence is represented by the addition of differential equations. The order and accuracy differs by turbulence model:
 - $\text{iturb} = 0$, no turbulence model (N-S only)
 - $\text{iturb} = 1$, Spalart-Allmarus model (one-equation; good for streamlined bodies)
 - $\text{iturb} = 2$, Menter's SST model (two-equation; good for more complex flows)
- Initial conditions, far field (freestream), and lowest allowable values for turbulence are set through rnuinf , rhoKinf , rhoWinf , turbI , and muTinf . The first three variables are used to set turbulent quantities directly, while the other two variables can be used to back-calculate the turbulent quantities from more user-friendly values:
 - For the SA model ($\text{iturb} = 1$), rnuinf represents the SA variable $\rho\nu$, which is used to directly calculate the local eddy viscosity. The amount of turbulence in the freestream affects the laminar performance and growth rate of turbulence along the wall. $\text{rnuinf} = 10^{-4}$ is suggested to create "natural transition", and $\text{rnuinf} = 3$ is suggested to create a wholly turbulent solution. If $\text{rnuinf} \leq 0$, then $\rho\nu$ is calculated from the eddy viscosity muTinf , using:

$$\mu_{T,\infty}^* = f_{\nu 1} \rho \hat{\nu}_\infty^* \quad \text{or} \quad f_{\nu 1} = \frac{(\rho \hat{\nu}_\infty^*)^3}{(\rho \hat{\nu}_\infty^*)^3 + 7.1^3}$$

- For the SST model ($\text{iturb} = 2$), rhoKinf represents the turbulent kinetic energy ρK , and rhoWinf represents the rate of dissipation of turbulent kinetic energy $\rho\omega$. The amount of turbulence in the freestream affects the laminar performance and growth rate of turbulence along the wall. If $\text{rhoKinf} \leq 0$, then ρK is calculated from turbulent intensity turbI , shown as T' :

$$T'_\infty = \sqrt{\frac{2}{3} \frac{K_\infty}{U_\infty^2}} \quad \text{or} \quad \rho K_\infty^* = 1.5 \rho_\infty^* (T'_\infty)^2$$

If $\text{rhoWinf} \leq 0$, then $\rho\omega$ is calculated from eddy viscosity muTinf , shown as μ_T :

$$\mu_{T,\infty}^* = \rho_\infty^* \frac{\rho K_\infty^*}{\rho \omega_\infty^*} \text{Re}_L \quad \text{or} \quad \rho \omega_\infty^* = \rho_\infty^* \frac{\rho K_\infty^*}{\mu_{T,\infty}^*} \text{Re}_L$$

muTinf and turbI are more user-friendly and therefore the suggested means of specifying ICs for the SST model. Turbulent intensity has a physical meaning:

- For internal flows, turbI is 1% to 5%.
 - For external (quiescent) flows and high-quality wind tunnel, $\text{turbI} < 1\%$.
- Freestream eddy viscosity muTinf represents additional diffusion throughout the solution. To minimize additional diffusion with a non-zero value:

$$\text{muTinf} = \mu_{T,\infty}^* = \frac{\mu_T}{\mu_\infty} \approx 1\%$$

- The turbulent Prandtl number Pr_T is used to model the transport of heat through turbulence, using Reynolds analogy. The turbulent Prandtl number for air is 0.9.
- Artificial dissipation is added to each turbulence model using the same algorithms applied to the N.S. equations. The N.S. dissipation factor has been combined with another scalar to allow the user to adapt the dissipation used in the turbulence models. The artificial dissipation in the turbulence models is scaled by $(disst * diss)$.
- The number of iterative cycles should be set to 4 for steady solutions. For unsteady solutions, use a sufficient number of cycles to allow for an appropriate level of convergence at each step. A sufficient number can be estimated as $N = dt / \Delta t_{loc,min}$.
- The following is a good practice for finding a sufficient number of iterations for unsteady solutions:
 1. Select initial values for dt , $ncyc$, and $rsdtol$.
 2. Step the solution forward 20-50 iterations.
 3. Check the *.cyc file for the number of cycles required per iteration. The number of cycles should level off after 10 iterations. If not, run enough iterations for the number of required cycles to level off.
 4. If the last 10 iterations require more than 20 cycles, lower the time step.
 5. If the last 10 iterations require less than 8 iterations, increase the time step.
 6. The sweet spot is 10-15 iterations.
- The residual tolerance can be used to exit the iterative cycles if the energy residual meets a specified criteria $rsdtol$. (The energy residual is used because the other residuals normally converge faster than energy.) This feature can be used to set the number of iterations to a very large number with a residual tolerance $rsdtol$. When the residual drops below the tolerance, the solution will progress to the next time step. Lower $rsdtol$ values require more iterations.
- The divergence tolerance $rsdmax$ creates an upper tolerance on the energy residual. If the solution is diverging, the energy residual will grow larger than $rsdmax$ and terminate the run. The solution also terminates if the residuals become NAN or INFINITY. Larger $rsdmax$ values will allow the solution to diverge further and ensure that the solution is in fact diverging.
- There are four available CFD solution types defined as follows:
 - $isol = 0$ is a steady solution (not time accurate)
 - $isol = 1$ is a first-order unsteady solution
 - $isol = 2$ is a second-order unsteady solution
 - $isol = 3$ is a supersonic piston perturbation solution
- There are three available options for viscous local time stepping:
 - $itime = -1$ uses the minimum distance across each element (algebraic)
 - $itime = 0$ uses diagonals of the stiffness matrix (heat transfer only)
 - $itime = 1$ uses diagonals of the stiffness matrices (momentum & heat transfer)
 - $itime = 2$ uses diagonals of the stiffness matrices (mom, heat trans & turb model)
- Euler2D uses various orders of numerical integration, specified by $ipnt$. NS2D only uses single point (or first order) Gauss quadrature ($ipnt = 1$), which is hard-coded into the controls. If a file from Euler2D is used to create the NS2D file `case.con`, then the integration order $ipnt$ must be removed or commented out (“!”).
- There are three available dissipation types defined as follows:

- `idiss = -1` is no artificial dissipation (only viscous dissipation)
- `idiss = 0` is a low order dissipation
- `idiss = 1` is a high order dissipation with gradient limiters
- The lower order dissipation is typically overly diffuse and should be used in conjunction with low values of the dissipation factor. Low-order dissipation works best for problems without strong vortices and for supersonic/hypersonic flows.
- The higher order dissipation is more CPU intensive than the low-order dissipation and less stable. Larger values for the dissipation factor are typically required for stabilization. The high-order dissipation works best for subsonic to transonic flows with strong gradients or vortices. Rotating domains will typically require high-order dissipation to resolve the circulating pattern of the relative flow velocities.
- `dislen` is the distance, near walls, where no artificial dissipation is used. The artificial dissipation model is scaled by $f(d)$, where d is the distance to the nearest wall:

$$f(d) = \begin{cases} 0 & \text{if } d \leq \text{dislen} \\ 1 & \text{if } d \geq 2 \text{dislen} \\ \frac{1}{2} \left(1 - \cos\left(\pi \left(\frac{d}{\text{dislen}} - 1\right)\right) \right) & \text{otherwise} \end{cases}$$

- Combustion properties are specified in the `case.cmb` file. The mass and heat generation are distributed throughout the domain using the following distributions:
 - `icomb = 0`, no combustion (`case.cmb` not read)
 - `icomb = 1`, combustion properties are piece-wise linear (specified at the nodes)
 - `icomb = 2`, combustion properties are constant (specified) on the elements
- When the restart flag `istrt` is set to `.true.`, the solver will read one set of solution unknowns from the `case.unk` file and apply this set of unknowns as the initial conditions for the new iterative solution.
- A restarted solution assumes that the time gradient of the initial state is zero, i.e. the solution stored in the `case.unk` file is a converged, steady state solution. This has a significant impact on the second-order unsteady solution since it relies on two sets of solution unknowns for advancement to the next time step, i.e. a second-order unsteady solution should not be restarted from the last time step of a similar unsteady solution that was stopped because both sets of unsteady data from the last solution step are not available for accurate evaluation of the time gradients in the flow.
- If the aerodynamics flag `iaero` is set to `.true.`, the aerodynamic forces are applied to the dynamic and elastic motion. If the flag is set to `.false.`, the dynamic and elastic motion must forced externally or occur as free-response vibrations.
- The non-inertial dynamics routine is turned on when `idynm` is set to `.true.`. NS2D will then read in the `case.dyn` file for dynamic inputs and write out dynamic motion to the `xd.dat`.
 - If the free-stream velocity flag `ifree` is set to `.false.`, the free-stream velocity is set to zero, and relative flow velocities must be generated through dynamic rotation or translation of the non-inertial coordinate system.
 - If `ifree = .true.`, the freestream starts aligned with the global fixed x-direction (i.e., $\alpha = \beta = 0$) but can be rotated using the initial orientation of the body in `case.dyn`.

- The elastic deflection routine is turned on when `ielast` is set to `.true.` NS2D will then read in the `case.vec` file for modal elastic inputs and write out modal deformations and forces to the `xn.dat`. The number of modes `nr` must match that shown in the elastic file `case.vec`.
- For steady solutions (`isol = 0`), the dynamics flags for each degree of freedom in the `case.dyn` and `case.vec` should be set to 1 (clamped condition).
- The propulsion boundary conditions are turned on when `iprop` is set to `.true.` NS2D will read in the `case.eng` file for rocket and engine inputs.
- If the external forces flag is set to `.true.`, the solver will read the user defined external force vector for each global time step from the input file `case.frc`. If the solver reaches the end of the input file before completing the solution, the last force vector in the file carries over to each of the remaining time steps if it was non-zero.
- If the safe-mode flag is set to `.true.`, NS2D writes two files per step that are used to restart the solution: `case.rst` and `case.rs2`. Two files are used so while one file is being over-written, the other file is still preserved. Each file stores the previous two values of all unknowns, elastic mode shapes, and generalized elastic forces. Safe-mode can be used for fast restarts for very long runs that are not time sensitive.
- When the safe-mode flag is set to `.true.`, NS2D checks for both restart files. If the `case.rst` exists, but has an error, the error is reported to the user. The `case.rst` can be moved, renamed, or deleted. The solution will then be restarted from the `case.rs2` file. (NS2D does not skip between files to avoid overwriting files that contain correctable errors.)
- If the residual watching flag is set to `.true.`, residuals are written out at each inner iteration to the `case.rsd2` file. This option can be used to check the residual convergence within steps. The number of inner cycles used at each iteration is written to the `case.cyc` file for plotting and comparison of convergence.
- If the acoustic output flag is set to `.true.`, the acoustic input file `case.acst` is read for controls, and one or more of the acoustic output files (pressure – `case.pac`; density – `case.rac`; u-velocity – `case.uac`; v-velocity – `case.vac`) are written.
- If the temperature boundary conditions flag is set to `.true.`, then the temperature boundary conditions are read in through the `case.tbc` file. *The temperature boundary conditions have not yet been verified.*
- Gravity is assumed to act on the vehicle along the inertial y -axis. In the non-inertial reference frame, the body force vector rotates so that gravity is always pointed down in the positive y -direction. The value `gravity` is non-dimensionalized using `refdim` (L), `mach` (M), and `ainf`, so the dimensions of these variables should be consistent:

$$g^* = \frac{g L}{M^2 a_\infty^2}$$

Unknowns (Initial Conditions) Input File (`case.unk`)

Basic File Format

```
np gam xmi alp ref t Re iturb
((UN(i,j), i = 1,nnd ), j = 1,5)
((UNT(i,j), i = 1,nnd ), j = 1,4)
```

Definition of Terms

<code>np</code> :	int	number of nodes
<code>gam</code> :	real	ratio of specific heats
<code>xmi</code> :	real	free-stream Mach number
<code>alp</code> :	real	free-stream angle of attack
<code>ref</code> :	real	reference dimension
<code>t</code> :	real	dimensionless time
<code>Re</code> :	real	free-stream Reynolds number
<code>iturb</code> :	int	turbulence model type
<code>UN(i,1)</code> :	real	density for node <i>i</i>
<code>UN(i,2)</code> :	real	x-velocity for node <i>i</i>
<code>UN(i,3)</code> :	real	y-velocity for node <i>i</i>
<code>UN(i,4)</code> :	real	pressure for node <i>i</i>
<code>UN(i,5)</code> :	real	total enthalpy for node <i>i</i>
<code>UNT(i,1)</code> :	real	eddy viscosity for node <i>i</i>
<code>UNT(i,2)</code> :	real	turb. KE for node <i>i</i>
<code>UNT(i,3)</code> :	real	turb. diss. for node <i>i</i>
<code>UNT(i,4)</code> :	real	model variable for node <i>i</i>

Comments

- This is an unformatted (binary) file.
- The solution unknowns stored in this file are dimensionless quantities.
- For dynamic (non-inertial) problems, the solution unknowns stored in the file are relative quantities referenced to the body-fixed coordinate system.
- The quantities `nnd` must match the values in the geometry file `case.g2d` as `nnd`.
- The quantities `gam`, `xmi`, and `Re` must match the values in the control file `case.con` as `gamma`, `mach`, and `Re`.
- The turbulent variables `UNT` are only read if a turbulence model (`iturb > 0`) has been used to construct the initial conditions file. The turbulence variables are then used to construct the variables needed for the turbulence model specified in the `case.con` file. The turbulence model used to specify the initial conditions and to step the solution after the initial conditions can be different.
- When restarting a solution, the most recent unknowns output file `case.un#` can be renamed as the initial conditions file `case.unk`.

Temperature Boundary Conditions Input File (`case.tbc`)

Basic File Format

```

Line of Text
  nwlt nbeq ntemp
Line of Text
  ITWL(1)   TWALL(1)
  ⋮         ⋮
  ITWL(nwlt) TWALL(nwlt)
Line of Text
  IQWL(1)   QWALL(1)
  ⋮         ⋮
  IQWL(nbeq) QWALL(nbeq)

```

Definition of Terms

nwlt:	int	number of nodes with specified temperatures
nbeq:	int	number of boundary elements with specified heat fluxes
ntemp:	int	number of iterations to ramp up boundary conditions
ITWL:	int	node index for temperature BC
TWALL:	real	internal enthalpy at node (dim'l)
IQWL:	int	boundary element index for heat flux BC
QWALL:	real	heat normal flux through boundary element (dim'l)

Comments

- This is a plain text (ASCII) file.
- The temperature boundary conditions are specified at `nwlt` wall nodes, listed in `ITWL`. Each node index `ITWL` is matched with a dimensional enthalpy value `TWALL` representing the temperature at that wall node. The temperature boundary conditions are strictly applied at each node between iterations.
- The temperature conditions `TWALL` are linearly increased from freestream conditions over `ntemp` iterations.
- The number of temperature boundary conditions `nwlt` must be less than the number of viscous wall nodes `nwlv` specified in the geometry file `case.g2d`, and the indices `ITWL` must point to viscous wall nodes:

$$nwl - nsd + nsdv - nwlv + 1 \leq ITWL(i) \leq nwl - nsd + nsdv$$

The viscous wall file `case.vwl` written by `makeg2d` contains a list of the nodes along viscous walls along with their x - and y -coordinates.

- The heat flux boundary conditions are specified at `nbeq` boundary elements, listed in `IQWL`. Each boundary element index `IQWL` is matched with a dimensional heat flux `QWALL` through the boundary element, where a positive `QWALL` represents heat flowing into the domain. The heat flux conditions are implied to the specified boundary elements through their boundary integrals.
- The heat flux conditions `QWALL` are linearly increased from zero (adiabatic) over `ntemp` iterations.
- The number of heat flux boundary conditions `nbeq` must be less than the number of viscous wall elements `LBE(8) - LBE(7)` specified in the geometry file `case.g2d`, and the indices `IQWL` must point to viscous wall elements:

$$LBE(7) \leq IQWL(i) \leq LBE(8)$$

The viscous wall file `case.vwl` written by `makeg2d` contains a list of the boundary elements that make-up viscous walls along with the x - and y -coordinates of both end points.

- The temperature and heat flux conditions are non-dimensionalized after being read into the solver, using the freestream density and velocity:

$$h_w^* = \frac{TWALL}{U_\infty^2} \quad q_w^{**} = \frac{QWALL}{\rho_\infty U_\infty^3}$$

- The temperature boundary conditions can be grouped into three categories, which are applied according to their rank (or importance):
 - (1) The specified temperature (Dirichlet) BC strictly applies `TWALL` at the specified viscous solid wall nodes between iterations. The temperature BC supersedes all heat flux boundary condition at the nodal level, as specified by this file.
 - (2) The adiabatic (Neumann) BC is implied through the heat flux used in viscous solid wall boundary integrals. The adiabatic wall is the default condition.
 - (3) The specified heat flux (Neumann) BC implies `QWALL` through the boundary integrals, as specified by this file.

Sample File

```
$ nwlt, nbeq, ntemp
      6      5    1000
$ Temperature BCs
1    3.0
2    3.1
3    3.0
4    2.9
5    2.8
6    2.8
$ Heat flux BCs
1    0.0
2    0.0
3    0.0
4    0.0
5    0.0
```


Temperature Boundary Conditions Input File (`case.tbc`)

Basic File Format

```

Line of Text
  nwlt nbeq ntemp
Line of Text
  ITWL(1)   TWALL(1)
  ⋮         ⋮
  ITWL(nwlt) TWALL(nwlt)
Line of Text
  IQWL(1)   QWALL(1)
  ⋮         ⋮
  IQWL(nbeq) QWALL(nbeq)

```

Definition of Terms

nwlt:	int	number of nodes with specified temperatures
nbeq:	int	number of boundary elements with specified heat fluxes
ntemp:	int	number of iterations to ramp up boundary conditions
ITWL:	int	node index for temperature BC
TWALL:	real	internal enthalpy at node (dim'l)
IQWL:	int	boundary element index for heat flux BC
QWALL:	real	heat normal flux through boundary element (dim'l)

Comments

- This is a plain text (ASCII) file.
- The temperature boundary conditions are specified at `nwlt` wall nodes, listed in `ITWL`. Each node index `ITWL` is matched with a dimensional enthalpy value `TWALL` representing the temperature at that wall node. The temperature boundary conditions are strictly applied at each node between iterations.
- The temperature conditions `TWALL` are linearly increased from freestream conditions over `ntemp` iterations.
- The number of temperature boundary conditions `nwlt` must be less than the number of viscous wall nodes `nwlv` specified in the geometry file `case.g2d`, and the indices `ITWL` must point to viscous wall nodes:

$$nwl - nsd + nsdv - nwlv + 1 \leq ITWL(i) \leq nwl - nsd + nsdv$$

The viscous wall file `case.vwl` written by `makeg2d` contains a list of the nodes along viscous walls along with their x - and y -coordinates.

- The heat flux boundary conditions are specified at `nbeq` boundary elements, listed in `IQWL`. Each boundary element index `IQWL` is matched with a dimensional heat flux `QWALL` through the boundary element, where a positive `QWALL` represents heat flowing into the domain. The heat flux conditions are implied to the specified boundary elements through their boundary integrals.
- The heat flux conditions `QWALL` are linearly increased from zero (adiabatic) over `ntemp` iterations.
- The number of heat flux boundary conditions `nbeq` must be less than the number of viscous wall elements `LBE(8) - LBE(7)` specified in the geometry file `case.g2d`, and the indices `IQWL` must point to viscous wall elements:

$$LBE(7) \leq IQWL(i) \leq LBE(8)$$

The viscous wall file `case.vw1` written by `makeg2d` contains a list of the boundary elements that make-up viscous walls along with the x - and y -coordinates of both end points.

- The temperature and heat flux conditions are non-dimensionalized after being read into the solver, using the freestream density and velocity:

$$h_w^* = \frac{TWALL}{U_\infty^2} \quad q_w^{**} = \frac{QWALL}{\rho_\infty U_\infty^3}$$

- The temperature boundary conditions can be grouped into three categories, which are applied according to their rank (or importance):
 - (4) The specified temperature (Dirichlet) BC strictly applies `TWALL` at the specified viscous solid wall nodes between iterations. The temperature BC supersedes all heat flux boundary condition at the nodal level, as specified by this file.
 - (5) The adiabatic (Neumann) BC is implied through the heat flux used in viscous solid wall boundary integrals. The adiabatic wall is the default condition.
 - (6) The specified heat flux (Neumann) BC implies `QWALL` through the boundary integrals, as specified by this file.

Sample File

```
$ nwlt, nbeq, ntemp
      6      5    1000
$ Temperature BCs
1    3.0
2    3.1
3    3.0
4    2.9
5    2.8
6    2.8
$ Heat flux BCs
1    0.0
2    0.0
3    0.0
4    0.0
5    0.0
```

Unknowns Output File (case.un#)

Basic File Format

```
np gam xmi alp ref t Re iturb  
( (UN(i,j), i = 1,nnd ), j = 1,5)  
( (UNT(i,j), i = 1,nnd ), j = 1,4)
```

Definition of Terms

np:	int	number of nodes
gam:	real	ratio of specific heats
xmi:	real	free-stream Mach number
alp:	real	free-stream angle of attack
ref:	real	reference dimension
t:	real	dimensionless time
Re:	real	free-stream Reynolds number
iturb:	int	turbulence model type

UN(i,1):	int	density for node <i>i</i>
UN(i,2):	real	x-velocity for node <i>i</i>
UN(i,3):	real	y-velocity for node <i>i</i>
UN(i,4):	real	pressure for node <i>i</i>
UN(i,5):	real	total enthalpy for node <i>i</i>
UNT(i,1):	real	eddy viscosity for node <i>i</i>
UNT(i,2):	real	turb. KE for node <i>i</i>
UNT(i,3):	real	turb. diss. for node <i>i</i>
UNT(i,4):	real	model variable for node <i>i</i>

Comments

- This is an unformatted (binary) file.
- The solution unknowns stored in this file are dimensionless quantities.
- For dynamic (non-inertial) problems, the solution unknowns stored in the file are relative quantities referenced to the body-fixed coordinate system. The fluid velocities (in the inertial frame) can be calculated by subtracting out the translational and rotational components of the body-fixed coordinate system.
- The turbulent variables UNT are only written if a turbulence model (*iturb* > 0) has been used to construct the file.

Turbulent Unknowns Output File (caset.un#)

Basic File Format

```
np gam xmi alp ref t Re iturb  
( (UT(i,j), i = 1, nnd ), j = 1, 5)
```

Definition of Terms

np:	int	number of nodes
gam:	real	ratio of specific heats
xmi:	real	free-stream Mach number
alp:	real	free-stream angle of attack
ref:	real	reference dimension
t:	real	dimensionless time
Re:	real	free-stream Reynolds number
iturb:	int	turbulence model type
UT(i,1):	int	eddy viscosity for node <i>i</i>
UT(i,2):	real	turb. KE for node <i>i</i>
UT(i,3):	real	= 0.0
UT(i,4):	real	model variable for node <i>i</i> (in press. coeff. form, see below)
UT(i,5):	real	turb. diss. for node <i>i</i> (in total energy form, see below)

Comments

- This is an unformatted (binary) file.
- The solution unknowns stored in this file are dimensionless quantities.
- The turbulent variables UNT are converted to the UT variables show in this file so that this file can be coupled with a caset.g2d file and plotted using GIPlot2D. (A caset.g2d can be created by copying the case.g2d file and adding a “t” to its name.) In this way, the four turbulence variables in UNT can be plotted:
 - Eddy viscosity $UNT(:, 1)$ can be seen in density plots.
 - Turbulent kinetic energy $UNT(:, 2)$ can be seen in velocity magnitude plots.
 - Turbulent dissipation $UNT(:, 3)$ can be seen in total energy plots.
 - Model variable $UNT(:, 4)$ can be seen in pressure coefficient plots.
- Total energy is calculated in GIPlot2D by subtracting $UT(:, 4)$ from $UT(:, 5)$.
- The pressure coefficient is calculated in GIPlot2D by subtracting the dimensionless freestream pressure ($= 1.0 / \text{gam} / \text{xmi}^2$) from $UT(:, 4)$ and scaling the result by 2.

Residuals Output File (case.rsd)

Basic File Format

```
1      (RSD(i), i = 1,6)
:
:
istp   (RSD(i), i = 1,6)
:
:
nstp   (RSD(i), i = 1,6)
```

Definition of Terms

istp:	int	current solution step
nstp:	int	total or last solution step
RSD(1):	real	density solution residual
RSD(2):	real	x-momentum solution residual
RSD(3):	real	y-momentum solution residual
RSD(4):	real	energy solution residual
RSD(5):	real	1st turb. model eq. residual
RSD(6):	real	2nd turb. model eq. residual

Comments

- This is a plain text (ASCII) file.
- For steady problems, the solution residuals indicate the degree of convergence to the final steady state solution. All four solution residuals should converge to approximately the same order of magnitude.
- For unsteady problems, the solution residuals indicate the degree of convergence for each global step of the solution, or the degree of convergence for the steady solution that is solved at each step.
- RSD(5) and RSD(6) represent residuals for the equation(s) of the turbulence model:
 - If `iturb = 0`, RSD(5) and RSD(6) are omitted.
 - If `iturb = 1`, RSD(5) is written, representing the SA differential equation; RSD(6) is omitted.
 - If `iturb = 1`, RSD(5) and RSD(6) are both written, representing the SST differential equations.

Sample File (*iturb* = 0)

1	0.38320E-05	0.10743E-04	0.69854E-05	0.10598E-03
2	0.20317E-05	0.50694E-05	0.40436E-05	0.56307E-04
3	0.12024E-05	0.35187E-05	0.26241E-05	0.32195E-04
4	0.91334E-06	0.25166E-05	0.23637E-05	0.24240E-04
5	0.73183E-06	0.19442E-05	0.22228E-05	0.19376E-04
6	0.59870E-06	0.16179E-05	0.20889E-05	0.15963E-04
7	0.51663E-06	0.14311E-05	0.19719E-05	0.13946E-04
8	0.44924E-06	0.12989E-05	0.18536E-05	0.12398E-04
9	0.39510E-06	0.12095E-05	0.17283E-05	0.11156E-04
10	0.34726E-06	0.11478E-05	0.15878E-05	0.99450E-05
11	0.30775E-06	0.10746E-05	0.14329E-05	0.88159E-05
12	0.26207E-06	0.98700E-06	0.12833E-05	0.76280E-05
13	0.22418E-06	0.87924E-06	0.11245E-05	0.65113E-05
14	0.18904E-06	0.77764E-06	0.98148E-06	0.54617E-05
15	0.15809E-06	0.69345E-06	0.84471E-06	0.44739E-05
16	0.13411E-06	0.62203E-06	0.72991E-06	0.37422E-05
17	0.11564E-06	0.55717E-06	0.64350E-06	0.32661E-05
18	0.10516E-06	0.50502E-06	0.57520E-06	0.30152E-05
19	0.10101E-06	0.46193E-06	0.53100E-06	0.29279E-05
20	0.98711E-07	0.43618E-06	0.49934E-06	0.28901E-05
:	:	:	:	:

Residuals Output File (*case.rsd2*)

Basic File Format

```

1      (RSD(i), i = 1,6)      1
1      (RSD(i), i = 1,6)      2
:      :                      :
1      (RSD(i), i = 1,6)      icyc
:      :                      :
istp   (RSD(i), i = 1,6)      1
:      :                      :
nstp   (RSD(i), i = 1,6)      1
:      :                      :

```

Definition of Terms

istp:	int	current solution step
icyc:	int	iteration of current residual
nstp:	int	total or last solution step
RSD(1):	real	density solution residual
RSD(2):	real	x-momentum solution residual
RSD(3):	real	y-momentum solution residual
RSD(4):	real	energy solution residual
RSD(5):	real	1st turb. model eq. residual
RSD(6):	real	2nd turb. model eq. residual

Comments

- This is a plain text (ASCII) file.
- This file is output when `irsds = .true.` in the controls `case.con` file. The residuals shown in this file represent the RMS changes at all nodes in the domain for this inner cycle. The convergence of residuals within any iteration can be seen in the trend in the residuals through the cycles used.
- RSD(5) and RSD(6) represent residuals for the equation(s) of the turbulence model:
 - If `iturb = 0`, RSD(5) and RSD(6) are omitted.
 - If `iturb = 1`, RSD(5) is written, representing the SA differential equation; RSD(6) is omitted.
 - If `iturb = 2`, RSD(6) and RSD(7) are both written, representing the SST differential equations.

Sample File (*iturb = 0*)

1	0.38320E-05	0.10743E-04	0.69854E-05	0.10598E-03	1
1	0.20317E-05	0.50694E-05	0.40436E-05	0.56307E-04	2
1	0.12024E-05	0.35187E-05	0.26241E-05	0.32195E-04	3
1	0.91334E-06	0.25166E-05	0.23637E-05	0.24240E-04	4
1	0.73183E-06	0.19442E-05	0.22228E-05	0.19376E-04	5
1	0.59870E-06	0.16179E-05	0.20889E-05	0.15963E-04	6
1	0.51663E-06	0.14311E-05	0.19719E-05	0.13946E-04	7
1	0.44924E-06	0.12989E-05	0.18536E-05	0.12398E-04	8
2	0.39510E-06	0.12095E-05	0.17283E-05	0.11156E-04	1
2	0.34726E-06	0.11478E-05	0.15878E-05	0.99450E-05	2
2	0.30775E-06	0.10746E-05	0.14329E-05	0.88159E-05	3
2	0.26207E-06	0.98700E-06	0.12833E-05	0.76280E-05	4
2	0.22418E-06	0.87924E-06	0.11245E-05	0.65113E-05	5
2	0.18904E-06	0.77764E-06	0.98148E-06	0.54617E-05	6
2	0.15809E-06	0.69345E-06	0.84471E-06	0.44739E-05	7
2	0.13411E-06	0.62203E-06	0.72991E-06	0.37422E-05	8
3	0.11564E-06	0.55717E-06	0.64350E-06	0.32661E-05	1
3	0.10516E-06	0.50502E-06	0.57520E-06	0.30152E-05	2
3	0.10101E-06	0.46193E-06	0.53100E-06	0.29279E-05	3
3	0.98711E-07	0.43618E-06	0.49934E-06	0.28901E-05	4
..
..

Time Step Output File (case.time)

Basic File Format

```

utime  htime  vtime  stime
  ⋮      ⋮      ⋮⋮

```

Definition of Terms

```

utime:  real  unsteady time step ratio
htime:  real  heat transfer time step ratio
vtime:  real  momentum time step ratio
stime:  real  turbulence time step ratio

```

Comments

- This is a plain text (ASCII) file.
- The time step file `case.time` is written out when the residuals watching flag is turned on (`irsds = .true.`). The third column, containing `vtime`, is only written when `itime > 1`; and the fourth column, containing `stime`, is only written when `itime = 2`.
- The four ratios are calculated using the following equations:

$$htime = MIN\left(\left(\frac{\Delta t_{HT}}{\Delta t_{inv}}\right)_n, 1\right) \quad (\Delta t_{mom})_n = MIN\left(\left(\Delta t_{mom,x}\right)_n, \left(\Delta t_{mom,y}\right)_n\right)$$

$$vtime = MIN\left(\left(\frac{\Delta t_{mom}}{\Delta t_{min}^{HT}}\right)_n, 1\right) \quad (\Delta t_{min}^{HT})_n = MIN\left(\left(\Delta t_{inv}\right)_n, \left(\Delta t_{HT}\right)_n\right)$$

$$stime = MIN\left(\left(\frac{\Delta t_{turb}}{\Delta t_{min}^{mom}}\right)_n, 1\right) \quad (\Delta t_{min}^{mom})_n = MIN\left(\left(\Delta t_{min}^{HT}\right)_n, \left(\Delta t_{mom}\right)_n\right)$$

$$utime = MIN\left(\left(\frac{\Delta t_{min}^{turb}}{dt}\right)_n, 1\right) \quad (\Delta t_{min}^{turb})_n = MIN\left(\left(\Delta t_{min}^{mom}\right)_n, \left(\Delta t_{turb}\right)_n\right)$$

$$\text{For SST,} \quad (\Delta t_{turb})_n = MIN\left(\left(\Delta t_K\right)_n, \left(\Delta t_\omega\right)_n\right)$$

where Δt_{inv} , Δt_{HT} , Δt_{mom} , and Δt_{turb} are the inviscid, heat transfer, momentum, and turbulence local time steps and dt is the global time step.

- The heat transfer time step is calculated using its stiffness matrix for $itime \geq 0$. For $itime < 0$, the minimum element length is used to calculate the local time step.
- For momentum, the diagonal of both the x - and y -stiffness matrices are tested for their local time steps and then combined into a single momentum time step.
- For the SA model, the SA diffusion terms are used to calculate its local time step. For the SST model, both the k - and ω -diffusion terms are used to calculate the local time step, as shown above.

- A conservative rule of thumb is to apply $itime = 2$ to use the minimum of all local time steps. If a column of the file shows all ones, like the example below, then the corresponding equation is more stable than those tested before it, and $itime$ can be reduced. In the example below shows the turbulence model is more stable than the other equations at all time steps. The example also shows that the momentum stability is necessary at the beginning of the run but can be removed after several iterations. For this case, after 20 iterations, $itime$ can be decreased to 0 to limit testing to the heat transfer stiffness matrix.
- Maximum viscous stability is obtained when all columns except for the first take a value of unity (1).
- The first column should be less than unity (1). The ratio of local to global time step acts like a relaxation factor in the solver. The solver limits this ratio to a maximum of unity, so unity in the first column shows that the global time step is smaller than all of the local time steps on the domain. The global time step should be increased until a value less than unity is reached somewhere on the domain.
- The number of cycles $ncyc$ can also be gauged using the first column. The number of cycles should be greater than $dt / \Delta t_{min}$ to maintain a reasonable assumption of time accuracy. Column represents the minimum of $\Delta t_{min} / dt$ on the domain, or the maximum cycles needed on the domain.

Sample File

0.85620E+00	0.75632E+00	0.56382E+00	1.00000E+00
0.87930E+00	0.75555E+00	0.62853E+00	1.00000E+00
0.78200E+00	0.75334E+00	0.75932E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.88937E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.93048E+00	1.00000E+00
0.79381E+00	0.75475E+00	0.95043E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.97490E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.98403E+00	1.00000E+00
0.79379E+00	0.75475E+00	0.98738E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.99018E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.99205E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.99405E+00	1.00000E+00
0.79378E+00	0.75475E+00	0.99739E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.99993E+00	1.00000E+00
0.79380E+00	0.75475E+00	1.00000E+00	1.00000E+00
0.79381E+00	0.75475E+00	1.00000E+00	1.00000E+00
0.79380E+00	0.75475E+00	1.00000E+00	1.00000E+00
0.79380E+00	0.75475E+00	1.00000E+00	1.00000E+00
0.79380E+00	0.75475E+00	1.00000E+00	1.00000E+00
0.79382E+00	0.75475E+00	1.00000E+00	1.00000E+00
0.79382E+00	0.75475E+00	1.00000E+00	1.00000E+00
⋮	⋮	⋮	⋮

Restart Files (case.rst and case.rs2)

Basic File Format

```

istp nnd gam xmi ref dt Re iturb
((UN(i,j), i = 1,nnd), j = 1,5)
((UNO(i,j), i = 1,nnd), j = 1,5)
((UNT(i,j), i = 1,nnd), j = 1,4)
((UNTO(i,j), i = 1,nnd), j = 1,4)
(XN(i,j), i = 1,2*nr),
      (XN1(i), i = 1,2*nr)
(FA(i,j), i = 1,nr),
      (FA2(i), i = 1,nr)
(XD(i,j), i = 1,6),
      (XD1(i), i = 1,6)
(FD(i,j), i = 1,3),
      (FD2(i), i = 1,3)

```

Definition of Terms

istp:	int	step in global solution
nnd:	int	number of nodes
gam:	real	ratio of specific heats
xmi:	real	free-stream Mach number
ref:	real	reference dimension
dt:	real	global time step
Re:	real	free-stream Reynolds number
iturb:	int	turbulence model type
UN:	real	unknowns at previous step
UNO:	real	unknowns at two steps prior
UNT:	real	turb .unknowns at prev. step
UNTO:	real	turb .unk.s at two steps prior
XN:	real	elastic deflect / velocity
XN1:	real	prev. elastic deflect / velocity
FA:	real	generalized aero. forces
FA2:	real	external forcing on modes
XD:	real	rigid body position / velocity
XD1:	real	prev. rigid body pos. / vel.
FD:	real	aero. forces on vehicle
FD2:	real	external forcing on vehicle

Comments

- This is an unformatted (binary) file.
- The solution unknowns, deflections, and forces stored in this file are dimensionless quantities.
- Turbulent unknowns properties vector UNT and UNTO are only written for turbulent solutions (`iturb > 0`).
- Unknowns properties vector UNO and UNTO are only written for 2nd order unsteady solutions (`isol = 2`).
- Elastic properties XN, XN1, FA, and FA2 is only written when the elastics flag `ielast` is set to `.true`.
- Non-inertial properties XD, XD1, FD, and FD2 is only written when the non-inertial flag `idynm` is set to `.true`.

APPENDIX E

EULER3D FILE FORMATS

The file formats included in this section are used to create the input files required to operate Euler3D and interpret the files that are written by Euler3D.

Euler3D

Summary of File Formats

Input Files:

- `case.g3d` (required, if `inetcdf = .false.`) contains geometry data structures representing the computational mesh as required by the flow solver (binary)
- `case.nc3d` (required, if `inetcdf = .true.`) contains geometry data structures representing the computational mesh as required by the flow solver (netCDF)
- `case.con` (required) contains values for the solver control parameters and flow conditions (ASCII)
- `case.unk` (optional) contains the nodal values of the primitive flow variables (density, velocity, pressure, and total enthalpy) for each node of the computational mesh to be used as the initial conditions for the flow solution (binary or netCDF)
- `case.dyn` (optional) contains the non-inertial matrices and initial conditions as required for a non-inertial solution (ASCII)
- `case.vec` (optional) contains the elastic mode matrices, initial conditions, and mode shape vectors for the solid wall surfaces as required for an aeroelastic solution (ASCII)
- `case.frc` (optional) contains external forces to be applied to each solution step in a dynamic or aeroelastic solution (ASCII)
- `case.cmb` (optional) contains mass and enthalpy generation data used by the quasi-combustion model (ASCII) [see Appendix C]
- `case.eng` (optional) contains rocket and turbojet engine (boundary condition) data (ASCII) [see Appendix C]
- `case.acst` (optional) contains the acoustic output data (ASCII)

Output Files:

- `case.un#` contains the nodal values of the primitive flow variables (density, velocity, pressure, and total enthalpy) for each node of the computational mesh; # is iterated as more files are produced so the progress of the solution can be followed (binary or netCDF)

- `case.rsd` contains a history of the solution residuals for the conservation variables (density, momentum, and total energy) (ASCII)
- `case.rsd2` contains a history of the solution residuals for the conservation variables for each inner cycle (ASCII)
- `case.cyc` contains a history of the number of inner cycles used to converge each iteration (ASCII) [see Appendix C]
- `case.time` contains a history of the local time step ratios (ASCII) [see Appendix C]
- `case.lds` contains a history of the dimensionless aerodynamic forces acting on the solid walls of the geometry (ASCII)
- `xd.dat` contains a history of the non-inertial displacements, velocities, and accelerations for a dynamic solution (ASCII)
- `xn.dat` contains a history of the generalized displacements, velocities, and forces for an unsteady, aeroelastic solution (ASCII)
- `case.rst` and `case.rs2` contain information on up to two sets of unknowns data, elastic system data, and dynamic motion data (binary)
- `case.pac` contains a history of pressure data at prescribed nodes (ASCII)
- `case.rac` contains a history of density data at prescribed nodes (ASCII)
- `case.uac` contains a history of u-velocity data at prescribed nodes (ASCII)
- `case.vac` contains a history of v-velocity data at prescribed nodes (ASCII)
- `case.wac` contains a history of w-velocity data at prescribed nodes (ASCII)

Geometry Input File (case.g3d)

Basic File Format

```
nnd nel nsg nbe nbp nwl nsd nsf
(LBE(i), i = 1, 8)
(COOR(i,j), j = 1,3) (i = 1,nnd)
(IELM(i,j), j = 1,4) (i = 1,nel)
(ISEG(i,j), j = 1,2) (i = 1,nsg)
(IBEL(i,j), j = 1,5) (i = 1,nbe)
```

Definition of Terms

```
nnd: int number of nodes
nel: int number of elements
nsg: int number of segments
nbe: int number of boundary elements
nbp: int number of boundary points
nwl: int number of wall nodes
nsd: int number of singular nodes
nsf: int number of boundary surfaces
```

LBE(i): int start/ stop index for 3 BC types

```
COOR(i,1): real x-coordinate for node i
COOR(i,2): real y-coordinate for node i
COOR(i,3): real z-coordinate for node i
```

```
IELM(i,1): int node 1 for element i
IELM(i,2): int node 2 for element i
IELM(i,3): int node 3 for element i
IELM(i,4): int node 4 for element i
```

```
ISEG(i,1): int node 1 for segment i
ISEG(i,2): int node 2 for segment i
```

```
IBEL(i,1): int node 1 for boundary elem. i
IBEL(i,2): int node 2 for boundary elem. i
IBEL(i,3): int node 3 for boundary elem. i
IBEL(i,4): int surface index in case.sur
IBEL(i,5): int domain elem. associated with
            boundary elem. i
```

Comments

- This is an unformatted (binary) file.
- The nodal coordinates in this file are treated as dimensional values and are non-dimensionalized using the reference dimension `refdim` specified in the control file.
- The element connectivity data `IELM` and `IBEL` define positive element volumes \mathcal{V}_e and boundary normal vectors \hat{n} pointed into the solution domain:

$$6\mathcal{V}_e = \begin{vmatrix} x_{14} & y_{14} & z_{14} \\ x_{24} & y_{24} & z_{24} \\ x_{34} & y_{34} & z_{34} \end{vmatrix} > 0 \quad \hat{n} = \frac{\vec{x}_{21} \times \vec{x}_{31}}{|\vec{x}_{21} \times \vec{x}_{31}|} \quad \text{where} \quad \vec{x}_{ij} = \vec{x}_i - \vec{x}_j$$

- The program `makeg3d` is used to convert a standard STARS surface triangulation file `case.fro`, mesh file `case.gri`, and modified boundary conditions file `case.bco` into an appropriately sorted three-dimensional geometry file.

- Nodal data $COOR$ is sorted such that the first n_{wl} nodes are defined as solid wall nodes. Out of the first n_{wl} nodes, the last n_{sd} nodes are defined as singular nodes.
- Boundary element data is sorted based on the starting/stopping indexes for the three BC types, i.e. boundary elements $LBE(1)$ through $LBE(2)$ are solid wall elements, $LBE(3)$ through $LBE(4)$ are symmetry elements, and $LBE(5)$ through $LBE(6)$ are far-field elements.

Geometry Input File, netCDF Format (case.nc3d)

Basic File Format

File Attributes:

```
"title" :: "Euler3D Geometry File"
"TimeDay" :: TimeDay
"name" :: filename
"Version" :: VerYMD
```

Dimensions:

```
"nnd" :: nnd (IDnnd)
"nel" :: nel (IDnel)
"nsg" :: nsg (IDnsg)
"nbe" :: nbe (IDnbe)
"nbp" :: nbp (IDnbp)
"nsf" :: nsf (IDnsf)
"ncv" :: ncv (IDncv)
"mxs" :: mxs (IDmxs)
"nLBE" :: 8 (nLBE)
"Dim2" :: 2 (ID2)
"Dim3" :: 3 (ID3)
"Dim4" :: 4 (ID4)
"Dim5" :: 5 (ID5)
```

Variables:

```
"nwl" :: nwl (IDnwl) [ ]
"nsd" :: nsd (IDnsd) [ ]
"LBE" :: LBE (IDLBE) [nLBE]
"COOR" :: COOR (IDcoor) [IDnnd, ID3]
"IELM" :: IELM (IDelm) [IDnel, ID4]
"ISEG" :: ISEG (IDseg) [IDnsg, ID2]
"IBEL" :: IBEL (IDbel) [IDnbe, ID5]
```

Definition of Terms

```
TimeDay: int time and date generated
          file: char case name
VerYMD: int MakeNC3D version, written
          using YYYYMMDD notation
```

```
nnd: int number of nodes
nel: int number of elements
nsg: int number of segments
nbe: int number of boundary elements
nbp: int number of boundary points
nsf: int number of surfaces in front file
ncv: int number of curves in front file
mxs: int maximum dimension (= 2 nel)
```

```
nLBE: int = 8, used to dimension LBE
Dim2: int = 2, used to dimension arrays
Dim3: int = 3, used to dimension arrays
Dim4: int = 4, used to dimension arrays
Dim5: int = 5, used to dimension arrays
```

```
nwl: int number of wall nodes
nsd: int number of singular nodes
```

```
LBE(i): int start/ stop index for 3 BC types
```

```
COOR(i,1): real x-coordinate for node i
COOR(i,2): real y-coordinate for node i
COOR(i,3): real z-coordinate for node i
```

```
IELM(i,1): int node 1 for element i
IELM(i,2): int node 2 for element i
IELM(i,3): int node 3 for element i
IELM(i,4): int node 4 for element i
```

```
ISEG(i,1): int node 1 for segment i
ISEG(i,2): int node 2 for segment i
```

```
IBEL(i,1): int node 1 for boundary elem. i
IBEL(i,2): int node 2 for boundary elem. i
IBEL(i,3): int node 3 for boundary elem. i
IBEL(i,4): int surface index in case.sur
IBEL(i,5): int domain elem. associated with
          boundary elem. i
```

Comments

- This file is created using the netCDF library. Formatting is handled using the netCDF library file netCDF.dll. The case.nc3d file contains the same base information in the case.g3d file.

- The netCDF formatting has been represented here using four designations:
 - Names in quotes (“ ”) represent the human-name of the variable or array
 - The value following the double-colon (::) is the variable stored under this name.
 - The name in parentheses () is the handle used to recall information from netCDF.
 - The values in brackets [] are the array dimensions. The empty brackets are shown for scalar variables. Single-dimension arrays (vectors) are shown as a single value between the brackets. Multi-dimension arrays (matrices, etc.) are shown by values separated by commas.
- Data in case .nc3d can be written or read in any particular order, but for simplicity, the file has been represented here in three sections:
 - File Attributes: Values describing the file (name, date, version, etc.) are not read by Euler3D.
 - Dimensions: Values used to size the arrays (variables) that follow are read using `nf_inq_dimid` and `nf_inq_dimlen`. The first function inquires of the netCDF ID to identify the dimension, and the second is used to read the value. The file structure has been established so that all of the dimensions are positive-definite (no zeros or negatives). For example, the number of nodes (`nnd`) are read:

```
call check(nf_inq_dimid(ncid, "nnd", IDnnd))
call check(nf_inq_dimlen(ncid, IDnnd, nnd))
```

- Variables: Scalars and arrays are read using `nf_inq_varid`, `nf_get_var_int`, `nf_put_var_double`, and `nf_get_var1_int`. The first function inquires of the netCDF ID to identify the variable. The second and third read an entire integer and double-precision real array, respectively. The final function is used to read an array one part at a time, used for LBE and IBEL. For example, the number of wall nodes (`nwl`, scalar), nodal coordinates (`COOR`, double, array), and element connectivity (`IELM`, integer, array) are read:

```
call check(nf_inq_varid(ncid, "nwl", IDnwl))
call check(nf_get_var_int(ncid, IDnwl, nwl))

call check(nf_inq_varid(ncid, "COOR", IDcoor))
call check(nf_get_var_double(ncid, IDcoor, COOR))

call check(nf_inq_varid(ncid, "IELM", IDelm))
call check(nf_get_var_int(ncid, IDelm, IELM))
```

- The functions used to read and write using the netCDF format can be found in the netCDF manuals (www.unidata.ucar.edu/software/netcdf/).
- Nodal data `COOR` is sorted such that the first `nwl` nodes are defined as solid wall nodes. Out of the first `nwl` nodes, the last `nsd` nodes are defined as singular nodes.
- The nodal coordinates in this file are treated as dimensional values and are non-dimensionalized using the reference dimension `refdim` specified in the control file.
- The element connectivity data `IELM` and `IBEL` define positive element volumes V_e and boundary normal vectors \hat{n} pointed into the solution domain:

$$6V_e = \begin{vmatrix} x_{14} & y_{14} & z_{14} \\ x_{24} & y_{24} & z_{24} \\ x_{34} & y_{34} & z_{34} \end{vmatrix} > 0 \quad \hat{n} = \frac{\vec{x}_{21} \times \vec{x}_{31}}{|\vec{x}_{21} \times \vec{x}_{31}|} \quad \text{where} \quad \vec{x}_{ij} = \vec{x}_i - \vec{x}_j$$

- Boundary element data is sorted based on the starting/stopping indexes for the three BC types, i.e. boundary elements `LBE(1)` through `LBE(2)` are solid wall elements, `LBE(3)` through `LBE(4)` are symmetry elements, and `LBE(5)` through `LBE(6)` are far-field elements.
- The program `makenc3d` is used to convert a standard STARS surface triangulation file `case.fro`, mesh file `case.gri`, and modified boundary conditions file `case.bco` into an appropriately sorted three-dimensional geometry file in netCDF format.

Solver Control Input File (case . con)

Basic File Format

```

&control
  dt          = 0.1d0,
  gamma       = 1.4d0,
  diss        = 1.0d0,
  cfl         = 0.5d0,

  mach        = 0.6d0,
  alpha       = 0.0d0,
  beta        = 0.0d0,
  refdim      = 1.0d0,

  nstp        = 100,
  nout        = 50,
  ncyc        = 4,

  rsdtol      = 1.0d-20,
  rsdmax      = 10.0d0,

  isol        = 0,
  ipnt        = 1,
  idiss       = 0,
  ndiss       = 1,
  idsol       = 2,
  icomb       = 0,

  isize       = 5,
  displ       = 0.1d0,
  omega       = 0.2d0,
  ratio       = 600.0d0,

  istrtr      = .false.,
  iaero       = .true.,
  idynm       = .false.,
  ielast      = .false.,
  iprop       = .false.,
  ifree       = .true.,
  iforce      = .false.,
  isafe       = .false.,
  irsds       = .false.,
  inetcdf     = .false.,
  iacoust     = .false.,

  nr          = 0,
  ainf        = 1.0d0,
  rhoinf      = 1.0d0,
  gravity     = 0.0d0,
/

```

Definition of Terms

dt:	real	dimensionless global time step
gamma:	real	ratio of specific heats
diss:	real	dissipation factor
cfl:	real	local time step stability factor
mach:	real	free-stream Mach number
alpha:	real	free-stream angle of attack (deg)
beta:	real	side slip angle (deg)
refdim:	real	reference length (dim'l)
nstp:	int	total solution steps
nout:	int	output frequency, steps/output
ncyc:	int	iterative cycles per solution step
rsdtol:	real	energy residual converg tolerance
rsdmax:	real	energy residual divergence criteria
isol:	int	CFD solution type
ipnt:	int	number of Gauss points
idiss:	int	dissipation type
ndiss:	int	# of inner cycles / diss. calculation
idsol:	int	order of elastic forces integration
icomb:	int	combustion model type
isize:	int	width of multisteps
displ:	real	max displacement of forced mode
omega:	real	frequency scalar
ratio:	real	chirp envelop length
istrtr:	log	restart flag
iaero:	log	aerodynamic forces flag
idynm:	log	dynamic/non-inertial flag
ielast:	log	elastic flag
iprop:	log	propulsion flag
ifree:	log	free-stream velocity flag
iforce:	log	external forces flag
isafe:	log	safe-mode flag
irsds:	log	residual watching flag
inetcdf:	log	NetCDF input / output flag
iacoust:	log	acoustics output flag
nr:	int	number of elastic modes
ainf:	real	free-stream sonic speed (dim'l)
rhoinf:	real	free-stream density (dim'l)
gravity:	real	gravity (dim'l)

Comments

- This is a plain text (ASCII) file formatted as a Fortran namelist.
 - The namelist must begin with the line “&control” and end with the line “/”.
 - The remaining lines can be listed in any order or omitted, if desired.
 - The intermediate lines work like variable assignments with the loose format: *variable_name = value*, followed by a comma.
 - Integers (*int*) are listed as whole numbers.
 - Real numbers (*real*) are listed in double precision, scientific notation: *###d+###*.
 - Logical variables (*log*) are listed as either “.true.” or “.false.”.
 - Lines can be commented out by including an exclamation point “!” prior to any other information on the line.
 - The default values, shown above, are used for variables omitted or commented out of the namelist.
 - Any information listed after the last line of the namelist “/” are not read by the program and can be used to store notes and other calculations.
- The global time step Δt is only used for unsteady solutions. Δt is a dimensionless value calculated: $\Delta t = \Delta t U / L$, where Δt is the dimensional time step, U is the free-stream velocity ($= \text{mach} \text{ ainf}$), and L is the reference length refdim .
- Appropriate values for the dissipation factor are in the range $0.0 < \text{diss} \leq 2.0$. Some dissipation is required to stabilize the solution, but too much dissipation will corrupt the solution and possibly be a destabilizing influence.
- The local time step stability factor is a safety factor used to compute local time steps for each solution step. For steady solutions, a stability factor of 0.8 is typical for most problems. For unsteady solutions, the stability factor is typically $0.3 \leq \text{cfl} \leq 0.8$.
- The values of refdim , mach , ainf , and rhoinf are used to non-dimensionalize all values read into the flow solver.
- The free-stream orientation angles are ignored for dynamic (non-inertial) problems.
- The number of iterative cycles should be set to 4 for steady solutions. For unsteady solutions, use a sufficient number of cycles to allow for an appropriate level of convergence at each step. A sufficient number can be estimated as $N = \Delta t / \Delta t_{\text{loc,min}}$.
- The following is a good practice for finding a sufficient number of iterations for unsteady solutions:
 1. Select initial values for Δt , ncyc , and rsdtol .
 2. Step the solution forward 20-50 iterations.
 3. Check the *.cyc file for the number of cycles required per iteration. The number of cycles should level off after 10 iterations. If not, run enough iterations for the number of required cycles to level off.
 4. If the last 10 iterations require more than 20 cycles, lower the time step.
 5. If the last 10 iterations require less than 8 iterations, increase the time step.
 6. The sweet spot is 10-15 iterations.
- The residual tolerance can be used to exit the iterative cycles if the energy residual meets a specified criteria rsdtol . (The energy residual is used because the other residuals normally converge faster than energy.) This feature can be used to set the number of iterations to a very large number with a residual

- tolerance `rsdtol`. When the residual drops below the tolerance, the solution will progress to the next time step. Lower `rsdtol` values require more iterations.
- The divergence tolerance `rsdmax` creates an upper tolerance on the energy residual. If the solution is diverging, the energy residual will grow larger than `rsdmax` and terminate the run. The solution also terminates if the residuals become NAN or INFINITY. Larger `rsdmax` values will allow the solution to diverge further and ensure that the solution is in fact diverging.
 - There are four available CFD solution types defined as follows:
 - `isol = 0` is a steady solution (not time accurate)
 - `isol = 1` is a first-order unsteady solution
 - `isol = 2` is a second-order unsteady solution
 - `isol = 3` is a supersonic piston perturbation solution
 - There are two types of numerical integration defined as follows:
 - `ipnt = 1` uses a one-point Gauss quadrature
 - `ipnt = 4` uses a four-point symmetric Gauss quadrature
 - There are three available dissipation types defined as follows:
 - `idiss = 0` is a low order dissipation
 - `idiss = 1` is a high order dissipation with gradient limiters
 - The lower order dissipation is typically overly diffuse and should be used in conjunction with low values of the dissipation factor. Low-order dissipation works best for problems without strong vortices and for supersonic/hypersonic flows.
 - The higher order dissipation is more CPU intensive than the low-order dissipation and less stable. Larger values for the dissipation factor are typically required for stabilization. The high-order dissipation works best for subsonic to transonic flows with strong gradients or vortices. Rotating domains will typically require high-order dissipation to resolve the circulating pattern of the relative flow velocities.
 - The values of `ndiss` controls the number of iterations between dissipation calculations. For example, if `ndiss = 1`, the dissipation is recalculated at every inner cycle.; if `ndiss = 2`, the dissipation is calculated ever other inner cycle, and stored in between; and, etc. `ndiss` can only be used to control the higher-order dissipation model (`idiss = 1`).
 - There are three available elastic solution types defined as follows:
 - `idsol = 0` is a zeroth-order integration for applied forces
 - `idsol = 1` is a first-order integration for applied forces
 - `idsol = 2` is a second-order integration for applied forces
 - Combustion properties are specified in the `case.cmb` file. The mass and heat generation are distributed throughout the domain using the following distributions:
 - `icomb = 0`, no combustion (`case.cmb` not read)
 - `icomb = 1`, combustion properties are piece-wise linear (specified at the nodes)
 - `icomb = 2`, combustion properties are constant (specified) on the elements
 - Four IBXN controls are listed in the `case.con` file: `isize`, `displ`, `omega`, and `ratio`. These controls are only necessary if `ielast` is turned on and `IBXN = 3` to `9`.
 - When the restart flag `istrt` is set to `.true.`, the solver will read one set of solution unknowns from the `case.unk` file and apply this set of unknowns as the initial conditions for the new iterative solution.

- A restarted solution assumes that the time gradient of the initial state is zero, i.e. the solution stored in the `case.unk` file is a converged, steady state solution. This has a significant impact on the second-order unsteady solution since it relies on two sets of solution unknowns for advancement to the next time step, i.e. a second-order unsteady solution should not be restarted from the last time step of a similar unsteady solution that was stopped because both sets of unsteady data from the last solution step are not available for accurate evaluation of the time gradients in the flow.
- If the aerodynamics flag `iaero` is set to `.true.`, the aerodynamic forces are applied to the dynamic and elastic motion. If the flag is set to `.false.`, the dynamic and elastic motion must be forced externally or occur as free-response vibrations.
- The non-inertial dynamics routine is turned on when `idynm` is set to `.true.`. Euler2D will then read in the `case.dyn` file for dynamic inputs and write out dynamic motion to the `xd.dat`.
 - If the free-stream velocity flag `ifree` is set to `.false.`, the free-stream velocity is set to zero, and relative flow velocities must be generated through dynamic rotation or translation of the non-inertial coordinate system.
 - If `ifree = .true.`, the freestream starts aligned with the global fixed x-direction (i.e., $\alpha = \beta = 0$) but can be rotated using the initial orientation of the body in `case.dyn`.
- The elastic deflection routine is turned on when `ielast` is set to `.true.`. Euler3D will read in the `case.vec` file for modal elastic inputs and write out modal deformations and forces to the `xn.dat`. The number of modes `nr` must match that shown in the elastic file `case.vec`.
- For steady solutions (`isol = 0`), the dynamics flags for each degree of freedom in the `case.dyn` and `case.vec` should be set to 1 (clamped condition).
- The propulsion boundary conditions are turned on when `iprop` is set to `.true.`. Euler3D will read in the `case.eng` file for rocket and engine inputs.
- If the external forces flag is set to `.true.`, the solver will read the user defined external force vector for each global time step from the input file `case.frc`. If the solver reaches the end of the input file before completing the solution, the last force vector in the file carries over to each of the remaining time steps if it was non-zero.
- If the safe-mode flag is set to `.true.`, Euler3D writes two files per step that are used to restart the solution: `case.rst` and `case.rs2`. Two files are used so while one file is being over-written, the other file is still preserved. Each file stores the previous two values of all unknowns, elastic mode shapes, and generalized elastic forces. Safe-mode can be used for fast restarts for very long runs that are not time sensitive.
- When the safe-mode flag is set to `.true.`, Euler3D checks for both restart files. If the `case.rst` exists, but has an error, the error is reported to the user. The `case.rst` can be moved, renamed, or deleted. The solution will then be restarted from the `case.rs2` file. (Euler3D does not skip between files to avoid overwriting files that contain correctable errors.)
- If the residual watching flag is set to `.true.`, residuals are written out at each inner iteration to the `case.rsd2` file. This option can be used to check the residual convergence within steps. The number of inner cycles used at each iteration is written to the `case.cyc` file for plotting and comparison of convergence.

- If the NetCDF flag is set to `.true.`, then the geometry and unknowns information is passed through NetCDF formats instead of the traditional binary files. The geometry is read in through the `case.nc3d` instead of the `case.g3d`. The unknowns files (`case.unk` and `case.un#`) retain the same name, but the format is the NetCDF format instead of the traditional binary format.
- If the acoustic output flag is set to `.true.`, the acoustic input file `case.acst` is read for controls, and one or more of the acoustic output files (pressure – `case.pac`; density – `case.rac`; u-velocity – `case.uac`; v-velocity – `case.vac`; w-velocity – `case.wac`) are written.
- Gravity is assumed to act on the vehicle along the inertial z -axis. In the non-inertial reference frame, the body force vector rotates so that gravity is always pointed down in the positive z -direction. The value `gravity` is non-dimensionalized using `refdim` (L), `mach` (M), and `ainf`, so the dimensions of these variables should be consistent:

$$g^* = \frac{g L}{M^2 a_\infty^2}$$

Unknowns (Initial Conditions) Input File (`case.unk`)

Basic File Format

```
np gam xmi alp bet ref t
((UN(i,j), i = 1,nnd ), j = 1,6)
```

Definition of Terms

np:	int	number of nodes
gam:	real	ratio of specific heats
xmi:	real	free-stream Mach number
alp:	real	free-stream angle of attack
bet:	real	side slip angle
ref:	real	reference dimension
t:	real	dimensionless time

UN(i,1):	real	density for node i
UN(i,2):	real	x-velocity for node i
UN(i,3):	real	y-velocity for node i
UN(i,4):	real	z-velocity for node i
UN(i,5):	real	pressure for node i
UN(i,6):	real	total enthalpy for node i

Comments

- This is an unformatted (binary) file.
- The solution unknowns stored in this file are dimensionless quantities.
- For dynamic (non-inertial) problems, the solution unknowns stored in the file are relative quantities referenced to the body-fixed coordinate system. The fluid velocities (in the inertial frame) can be calculated by subtracting out the translational and rotational components of the body-fixed coordinate system.
- The quantities `nnd` must match the values in the geometry file `case.g3d` as `nnd`.
- The quantities `gam` and `xmi` must match the values in the control file `case.con` as `gamma` and `mach`.
- When restarting a solution, the most recent unknowns output file `case.un#` can be renamed as the initial conditions file `case.unk`.

Geometry Input File, netCDF Format (case.unk)

Basic File Format

File Attributes:

```
"title"    :: "Euler3D Unk File"  
"time"     :: TimeDay  
"name"     :: filename  
"Version"  :: VerYMD  
"gam"      :: gam  
"xmi"      :: xmi  
"alp"      :: nbe  
"bet"      :: nbp  
"refdim"   :: ref  
"time"     :: t
```

Dimensions:

```
"nnd"      :: nnd (IDnnd)  
"Dim1"     :: 1 (ID1)
```

Variables:

```
"Density"   :: UN(:,1) (IDrho)  
              [IDnnd, ID1]  
"Xvelocity" :: UN(:,2) (IDxvel)  
              [IDnnd, ID1]  
"Yvelocity" :: UN(:,3) (IDyvel)  
              [IDnnd, ID1]  
"Zvelocity" :: UN(:,4) (IDzvel)  
              [IDnnd, ID1]  
"Pressure"  :: UN(:,5) (IDp)  
              [IDnnd, ID1]
```

Definition of Terms

```
TimeDay:  int   time and date generated  
          filename: char case name  
VerYMD:   int   Euler3D version, written  
          using YYYYMMDD notation
```

```
gam:  real  ratio of specific heats  
xmi:  real  free-stream Mach number  
alp:  real  free-stream angle of attack  
bet:  real  side slip angle  
ref:  real  reference dimension  
t:    real  dimensionless time
```

```
nnd:  int   number of nodes  
Dim1: int   = 1, used to dimension arrays
```

```
UN(i,1): real  density for node i  
UN(i,2): real  x-velocity for node i  
UN(i,3): real  y-velocity for node i  
UN(i,4): real  z-velocity for node i  
UN(i,5): real  pressure for node i
```

Comments

- This file is created using the netCDF library. Formatting is handled using the netCDF library file netCDF.dll.
- The netCDF formatting has been represented here using four designations:
 - Names in quotes (" ") represent the human-name of the variable or array
 - The value following the double-colon (::) is the variable stored under this name.
 - The name in parentheses () is the handle used to recall information from netCDF.
 - The values in brackets [] are the array dimensions. Multi-dimension arrays (matrices, etc.) are shown by values separated by commas.
- Data in case.unk can be written or read in any particular order, but for simplicity, the file has been represented here in three sections:

- **File Attributes:** Values describing the file (name, date, version, etc.) are read using `nf_get_att_double`. For example, the ratio of specific heats (`gam`) and mach number (`xmi`) are read:

```
call check(nf_get_att_double(ncid, nf_global, "gam", gam_unk))
call check(nf_get_att_double(ncid, nf_global, "xmi", xmi_unk))
```

- **Dimensions:** Values used to size the arrays (variables) that follow are read using `nf_inq_dimid` and `nf_inq_dimlen`. The first function inquires of the netCDF ID to identify the dimension, and the second is used to read the value. The file structure has been established so that all of the dimensions are positive-definite (no zeros or negatives). For example, the number of nodes (`nnd`) are read:

```
call check(nf_inq_dimid(ncid, "nnd", IDnnd))
call check(nf_inq_dimlen(ncid, IDnnd, nnd))
```

- **Variables:** The IDs for arrays are identified using `nf_inq_varid`, and then the arrays are read using `nf_get_var_double` (double-precision reals). For example, the nodal density (`UN(:,1)`, double, array) are read:

```
call check(nf_inq_varid(ncid, "Density", IDrho))
call check(nf_get_var_double(ncid, IDrho, UN(:,1)))
```

- The functions used to read and write using the netCDF format can be found in the netCDF manuals (www.unidata.ucar.edu/software/netcdf/).
- The solution unknowns stored in this file are dimensionless quantities.
- For dynamic (non-inertial) problems, the solution unknowns stored in the file are relative quantities referenced to the body-fixed coordinate system. The fluid velocities (in the inertial frame) can be calculated by subtracting out the translational and rotational components of the body-fixed coordinate system.
- The quantities `nnd` must match the values in the geometry file `case.nc3d` as `nnd`.
- The quantities `gam` and `xmi` must match the values in the control file `case.con` as `gamma` and `mach`.
- When restarting a solution, the most recent unknowns output file `case.un#` can be renamed as the initial conditions file `case.unk`.

Dynamic Mesh Input File (case.dyn)

Basic File Format

```
Line of Text
(R0(i), i = 1, 3)
Line of Text
((RM1(i,j), j = 1,6), i = 1,6)
Line of Text
((RC1(i,j), j = 1,6), i = 1,6)
Line of Text
((RK1(i,j), j = 1,6), i = 1,6)
Line of Text
x, y, z, p, q, r,
vx, vy, vz, vp, vq, vr,
ax, ay, az, ap, aq, ar
Line of Text
(IBXD(i), i = 1,13)
```

Definition of Terms

```
R0(1): real x-coord. for origin of rotation
R0(2): real y-coord. for origin of rotation
R0(3): real z-coord. for origin of rotation

RM1(i,j): real dimensional mass matrix
RC1(i,j): real dimensional damping matrix
RK1(i,j): real dimensional stiffness matrix

x: real initial x-position of system
y: real initial y-position of system
z: real initial z-position of system
p: real initial roll angle of system (deg)
q: real initial pitch angle of system (deg)
r: real initial yaw angle of system (deg)
vx: real initial x-velocity of system
vy: real initial y-velocity of system
vz: real initial z-velocity of system
vp: real initial roll rate (deg/s)
vq: real initial pitch rate (deg/s)
vr: real initial yaw rate (deg/s)
ax: real initial x-acceleration of system
ay: real initial y-acceleration of system
az: real initial z-acceleration of system
ap: real initial roll acceleration (deg/s2)
aq: real initial pitch acceleration (deg/s2)
ar: real initial yaw acceleration (deg/s2)

IBXD(1): int flag for x-position-DOF
IBXD(2): int flag for y-position-DOF
IBXD(3): int flag for z-position-DOF
IBXD(4): int flag for x-velocity-DOF
IBXD(5): int flag for y-velocity-DOF
IBXD(6): int flag for z-velocity-DOF
IBXD(7): int flag for p-velocity-DOF
IBXD(8): int flag for q-velocity-DOF
IBXD(9): int flag for r-velocity-DOF
IBXD(10): int flag for first quaternion q0
IBXD(11): int flag for second quaternion q1
IBXD(12): int flag for third quaternion q2
IBXD(13): int flag for fourth quaternion q3
```

Comments

- This is a plain text (ASCII) file.
- All values entered into this file should be dimensional. The solver will automatically non-dimensionalize the values using the reference conditions specified in the solver control file. The units of mass, length, and time in this file should match those in the controls file case.con.

- To convert from `xd.dat` to ICs:

```

XD(1) x refdim → x
XD(2) x refdim → y
XD(3) x refdim → z
XD(4) x mach x ainf → vx
XD(5) x mach x ainf → vy
XD(6) x mach x ainf → vz
roll x 180/π → p
pitch x 180/π → q
yaw x 180/π → r
XD(7) x 180/π x mach x ainf / refdim → vp
XD(8) x 180/π x mach x ainf / refdim → vq
XD(9) x 180/π x mach x ainf / refdim → vr

```

The accelerations ax , ay , az , ap , aq , and ar are not used to restart the system and do not need to be converted from the `xd.dat` file.

Sample File

```

$ Position vector to origin of non-inertial frame (rx, ry, rz)
0.0d0 0.0d0 0.0d0
$ Mass matrix for non-inertial frame (6 x 6)
1.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0
0.0d0 1.0d0 0.0d0 0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 1.0d0 0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0 1.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0 0.0d0 1.0d0 0.0d0
0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 1.0d0
$ Damping matrix for non-inertial frame (6 x 6)
1.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0
0.0d0 1.0d0 0.0d0 0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 1.0d0 0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0 1.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0 0.0d0 1.0d0 0.0d0
0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 1.0d0
$ Stiffness matrix for non-inertial frame (6 x 6)
1.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0
0.0d0 1.0d0 0.0d0 0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 1.0d0 0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0 1.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0 0.0d0 1.0d0 0.0d0
0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 1.0d0
$ IC's for non-inertial frame (6 positions, 6 rates, 6 accelerations)
0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0 0.0d0 0.0d0 0.0d0
$ IBXD for non-inertial frame (13)
1 1 1 1 1 1 1 1 1 1 1 1 1

```

Elastic Vectors Input File (`case.vec`)

Basic File Format

```

Line of Text
  nr
Line of Text
  ((RM(i,j), j = 1,nr), i = 1,nr)
Line of Text
  ((RC(i,j), j = 1,nr), i = 1,nr)
Line of Text
  ((RK(i,j), j = 1,nr), i = 1,nr)
Line of Text
  (XN(i), i = 1,nr*2)
Line of Text
  (IBXN(i), i = 1,nr)
Line of Text
  ((PHIA(i,j), i = 1,nw1*3), j = 1,nr)

```

Definition of Terms

nr:	int	number of elastic modes
RM(i,j):	real	dimensional mass matrix
RC(i,j):	real	dimensional damping matrix
RK(i,j):	real	dimensional stiffness matrix
XN(i):	real	initial gen. displ. for mode i
XN(i+nr):	real	initial gen. vel. for mode i
IBXD(i):	int	dynamics flag for i^{th} mode
PHIA(i*3-2,j):	real	x-displacement vector for mode j at node i
PHIA(i*3-1,j):	real	y-displacement vector for mode j at node i
PHIA(i*3,j):	real	z-displacement vector for mode j at node i

Comments

- This is a plain text (ASCII) file.
- All values entered into this file should be dimensional. The solver will automatically non-dimensionalize the values using the reference conditions specified in `case.con`.
- The mass matrix `RM` defined in this file cannot be singular.
- The dynamics of each degree of freedom is controlled separately using the following values for `IBXN`:
 - `IBXN = 0` is a free / forced response calculation, i.e. uses mass, stiffness, and damping to compute position, velocity, and acceleration of system.
 - `IBXN = 1` is a clamped condition, i.e. hold at initial position and velocity with zero acceleration.
 - `IBXN = 2` is a constant velocity, uncoupled response, i.e. integrates acceleration and velocity to compute new displacement.
 - `IBXN = 3` is a forced 3211 multistep response used for system ID purposes.
 - `IBXN = 4` is a forced multistep response used for system ID purposes.
 - `IBXN = 5` is a forced chirp response used for system ID purposes.
 - `IBXN = 6` is a forced multi-processor chirp response used for system ID purposes.
 - `IBXN = 7` is a forced DC chirp response used for system ID purposes.
 - `IBXN = 8` is a forced Schroeder chirp response used for system ID purposes.
 - `IBXN = 9` is a forced Fresnel chirp response used for system ID purposes.
- Do not combine zero `IBXN` values with non-zero values for different modes if there are coupling or off-diagonal terms in the mass, damping, or stiffness matrices.
- The modal displacement is defined at `nw1` nodes using the `PHIA`. The corresponding nodes are specified through `IPHIA`, which point to the node indices in `case.g3d`.

- A limited set of simple model vectors representing standard rigid-body degrees of freedom can be generated using the program `makevec3d`.
- To convert from `xn.dat` to ICs:

$$\begin{aligned}
 XN(i) &\longrightarrow XN(i) \\
 VN(i) \times mach \times ainf / refdim &\longrightarrow XN(i+nr)
 \end{aligned}$$

Sample File

```

$ Number of elastic modes
3
$ Mass matrix for elastic modes (nr x nr)
1.0d0      0.0d0      0.0d0
0.0d0      1.0d0      0.0d0
0.0d0      0.0d0      1.0d0
$ Damping matrix for elastic modes (nr x nr)
1.0d0      0.0d0      0.0d0
0.0d0      1.0d0      0.0d0
0.0d0      0.0d0      1.0d0
$ Stiffness matrix for elastic modes (nr x nr)
1.0d0      0.0d0      0.0d0
0.0d0      1.0d0      0.0d0
0.0d0      0.0d0      1.0d0
$ IC's for elastic modes (x1...xn, vx1...vxn)
0.0d0      0.0d0      0.0d0      0.0d0      0.0d0      0.0d0
$ IBXN for elastic modes (nr)
1      1      1
$ Elastic modes vectors PHIA ((nwl*3) x nr)
0.0d0      0.0d0      1.0d0
0.0d0      0.0d0      1.0d0
0.0d0      0.0d0      1.0d0
0.0d0      0.0d0      1.0d0
0.0d0      0.0d0      1.0d0
0.0d0      0.0d0      1.0d0
0.0d0      0.0d0      1.0d0
0.0d0      0.0d0      1.0d0
  ⋮          ⋮          ⋮

```

External Forces File (`case.frc`)

Basic File Format

```
0      (FD(i), i = 1, 6)
      (FA(i), i = 1,nr)
:      :      : :
:      :      : :
istp   (FD(i), i = 1, 6)
      (FA(i), i = 1,nr)
:      :      : :
:      :      : :
nstp   (FD(i), i = 1, 6)
      (FA(i), i = 1,nr)
```

Definition of Terms

istp:	int	current solution step
nstp:	int	total or last solution step
FD(1):	real	x-force applied at <i>istp</i>
FD(2):	real	y-force applied at <i>istp</i>
FD(3):	real	z-force applied at <i>istp</i>
FD(4):	real	roll moment applied at <i>istp</i>
FD(5):	real	pitch moment applied at <i>istp</i>
FD(6):	real	yaw moment applied at <i>istp</i>
FA(i):	real	gen. force applied to mode <i>i</i>

Comments

- This is a plain text (ASCII) file.
- All values entered into this file should be dimensional. The solver will automatically non-dimensionalize the values using the reference conditions specified in the solver control file. The units of mass, length, and time in this file should match those in the controls file `case.con`.
- The forces applied to the three translational degrees of freedom are specified relative to the inertial coordinate system, i.e. as seen by a stationary observer on the ground.
- The specified forces are read one line at a time following each solution step.
- Up to `nstp` forces may be specified, but are not required. The last force read in by the solver will be applied for all remaining solution steps

Acoustic Input File (case.acst)

Basic File Format

```
Line of Text
  ipres  idens  iuvel  ivvel  iwvel
Line of Text
  nacp  nacl
Line of Text
  x(1)    y(1)    z(1)
  ⋮      ⋮      ⋮
  x(nacp) y(nacp) z(nacp)
Line of Text
  x1(1) y1(1) z1(1) x2(1) y2(1) z2(1)
  ⋮      ⋮      ⋮      ⋮      ⋮      ⋮
  x1(nacl) y1(nacl) z1(nacl)
  x2(nacl) y2(nacl) z2(nacl)
```

Definition of Terms

ipres:	int	pressure output flag
idens:	int	density output flag
iuvel:	int	u-velocity output flag
ivvel:	int	v-velocity output flag
iwvel:	int	w-velocity output flag
nacp:	int	number of acoustic points
nacl:	int	number of acoustic lines
x(:):	real	x-coordinate of acoustic point
y(:):	real	y-coordinate of acoustic point
z(:):	real	z-coordinate of acoustic point
x1(:):	real	x-coord - starting point of line
y1(:):	real	y-coord - starting point of line
z1(:):	real	z-coord - starting point of line
x2(:):	real	x-coord - ending point of line
y2(:):	real	y-coord - ending point of line
z2(:):	real	z-coord - ending point of line

Comments

- This is a plain text (ASCII) file.
- One of the four acoustics flags must be on, in order for the output files to be created and data written out.
 - If `ipres = 1`, then coefficient of pressure data is written to the `case.pac` file.
 - If `idens = 1`, then density data is written to the `case.rac` file.
 - If `iuvel = 1`, then u-velocity data is written to the `case.uac` file.
 - If `ivvel = 1`, then v-velocity data is written to the `case.vac` file.
 - If `iwvel = 1`, then w-velocity data is written to the `case.wac` file.
 - If none of the four flags are on, `ipres` is set to 1 (on).
- The header lines must still be written, even if no data is given in the corresponding section. For instance, if `nacp = 0`, the point header must still exist in the file at the 5th line. If `nacl = 0`, the line header must still exist at the end of the file. The header lines exist as place holders, so anything can be written on these lines, but the lines themselves must exist.
- Acoustic points can exist anywhere inside of the domain. These points are described by (x,y,z) coordinates. If the coordinates do not exist within the field, the first node of the first element is used to calculate properties written to the file(s). A warning is written in the header of these files, and the (x,y,z) coordinates of this point will not match those in the input file. This data should not be used but treated as a place holder.
- The acoustic lines are represented by drawing a line from a starting point (x_1,y_1,z_1) to an ending point (x_2,y_2,z_2) . Any element face in the solution domain that crosses

this line is used to calculate properties at the intersection (of the acoustic line and element face). Intersections along a particular line are listed in order from starting to ending points.

Sample File

```
$ Acoustic output file flags
 1 0 1 0 0
$ Number of points and lines
4 2
$ Coordinates of acoustic points
0.3 0.0
0.7 0.1
0.1 0.0
0.8 0.8
$ Coordinates of acoustic lines
0.5 0.0 0.5 1.0
0.0 0.0 1.0 0.0
```

Unknowns Output File (case.un#)

Basic File Format

```
np gam xmi alp bet ref t  
( (UN(i,j), i = 1, nnd ), j = 1, 6)
```

Definition of Terms

np:	int	number of nodes
gam:	real	ratio of specific heats
xmi:	real	free-stream Mach number
alp:	real	free-stream angle of attack
bet:	real	side slip angle
ref:	real	reference dimension
t:	real	dimensionless time

UN(i,1):	real	density for node <i>i</i>
UN(i,2):	real	x-velocity for node <i>i</i>
UN(i,3):	real	y-velocity for node <i>i</i>
UN(i,4):	real	z-velocity for node <i>i</i>
UN(i,5):	real	pressure for node <i>i</i>
UN(I,6):	real	total enthalpy for node <i>i</i>

Comments

- This is an unformatted (binary) file.
- The solution unknowns stored in this file are dimensionless quantities.
- For dynamic (non-inertial) problems, the solution unknowns stored in the file are relative quantities referenced to the body-fixed coordinate system. The fluid velocities (in the inertial frame) can be calculated by subtracting out the translational and rotational components of the body-fixed coordinate system.

Unknowns File, netCDF Format (case.un#)

Basic File Format

File Attributes:

```
"title"    :: "Euler3D Un# File"  
"time"     :: TimeDay  
"name"     :: filename  
"Version"  :: VerYMD  
"gam"      :: gam  
"xmi"      :: xmi  
"alp"      :: alp  
"bet"      :: bet  
"refdim"   :: ref  
"time"     :: t
```

Dimensions:

```
"nnd"      :: nnd (IDnnd)  
"Dim1"     :: 1 (ID1)  
"Dim4"     :: 4 (ID4)  
"Dim6"     :: 6 (ID6)
```

Variables:

```
"Density"   :: UN(:,1) (IDrho)  
              [IDnnd, ID1]  
"Xvelocity" :: UN(:,2) (IDxvel)  
              [IDnnd, ID1]  
"Yvelocity" :: UN(:,3) (IDyvel)  
              [IDnnd, ID1]  
"Zvelocity" :: UN(:,4) (IDzvel)  
              [IDnnd, ID1]  
"Pressure"  :: UN(:,5) (IDP)  
              [IDnnd, ID1]
```

Definition of Terms

```
TimeDay:   int   time and date generated  
           file: char case name  
VerYMD:    int   Euler3D version, written  
              using YYYYMMDD notation
```

```
gam:  real  ratio of specific heats  
xmi:  real  free-stream Mach number  
alp:  real  free-stream angle of attack  
bet:  real  free-stream side slip angle  
ref:  real  reference dimension  
t:    real  dimensionless time
```

```
nnd:  int   number of nodes  
Dim1: int   = 1, used to dimension arrays  
Dim4: int   = 4, used to dimension arrays  
Dim6: int   = 6, used to dimension arrays
```

```
UN(i,1): real  density for node i  
UN(i,2): real  x-velocity for node i  
UN(i,3): real  y-velocity for node i  
UN(i,4): real  z-velocity for node i  
UN(i,5): real  pressure for node i
```

Comments

- This file is created using the netCDF library. Formatting is handled using the netCDF library file netCDF.dll.
- The netCDF formatting has been represented here using four designations:
 - Names in quotes (" ") represent the human-name of the variable or array
 - The value following the double-colon (::) is the variable stored under this name.
 - The name in parentheses () is the handle used to recall information from netCDF.
 - The values in brackets [] are the array dimensions. Multi-dimension arrays (matrices, etc.) are shown by values separated by commas.

- Data in `case.un#` can be written or read in any particular order, but for simplicity, the file has been represented here in three sections:
 - File Attributes: Values describing the file (name, date, version, etc.) are written using `nf_put_att_text`, `nf_put_att_int`, or `nf_put_att_double` if the value is a string, integer, or double-precision real, respectively. For example, the date (int), case name (char), and Mach number (xmi, double) are written:


```
call check(nf_put_att_int(ncid, nf_global, "time", nf_int, 8, Day ))
call check(nf_put_att_text(ncid, nf_global, "name",
                          len(trim(filen)), trim(filen) ))
call check(nf_put_att_double(ncid, nf_global, "xmi", nf_double,
                             1, xmi ))
```
 - Dimensions: Values used to size the arrays (variables) that follow are written using `nf_def_dim`. The file structure has been established so that all of the dimensions are positive-definite (no zeros or negative numbers). For example, the number of nodes (nnd) is written:


```
call check(nf_def_dim(ncid, "nnd", nnd, IDnnd))
```
 - Variables: Arrays are defined using `nf_def_var` and written using `nf_put_var_double` (double-precision reals). For example, the nodal density (`UN(:,1)`, double, array) are written:


```
RankTwo = (/ IDnnd, ID1 /)
call check(nf_def_var(ncid, "Density", nf_double, 2, RankTwo,
                     IDrho))
call check(nf_put_var_double(ncid, IDrho, UN(:,1)))
```
 - The functions used to read and write using the netCDF format can be found in the netCDF manuals (www.unidata.ucar.edu/software/netcdf/).
- The solution unknowns stored in this file are dimensionless quantities.
- For dynamic (non-inertial) problems, the solution unknowns stored in the file are relative quantities referenced to the body-fixed coordinate system. The fluid velocities (in the inertial frame) can be calculated by subtracting out the translational and rotational components of the body-fixed coordinate system.

Residuals Output File (*case.rsd*)

Basic File Format

```
1      (RSD(i), i = 1,5)
:
:
istp   (RSD(i), i = 1,5)
:
:
nstp   (RSD(i), i = 1,5)
```

Definition of Terms

istp:	int	current solution step
nstp:	int	total or last solution step
RSD(1):	real	density solution residual
RSD(2):	real	x-momentum solution residual
RSD(3):	real	y-momentum solution residual
RSD(4):	real	z-momentum solution residual
RSD(5):	real	energy solution residual

Comments

- This is a plain text (ASCII) file.
- For steady problems, the solution residuals indicate the degree of convergence to the final steady state solution. All five solution residuals should converge to approximately the same order of magnitude.
- For unsteady problems, the solution residuals indicate the degree of convergence for each global step of the solution, or the degree of convergence for the steady solution that is solved at each step.

Sample File

1	0.38320E-05	0.10743E-04	0.69854E-05	0.69854E-05	0.10598E-03
2	0.20317E-05	0.50694E-05	0.40436E-05	0.40436E-05	0.56307E-04
3	0.12024E-05	0.35187E-05	0.26241E-05	0.26241E-05	0.32195E-04
4	0.91334E-06	0.25166E-05	0.23637E-05	0.23637E-05	0.24240E-04
5	0.73183E-06	0.19442E-05	0.22228E-05	0.22228E-05	0.19376E-04
6	0.59870E-06	0.16179E-05	0.20889E-05	0.20889E-05	0.15963E-04
7	0.51663E-06	0.14311E-05	0.19719E-05	0.19719E-05	0.13946E-04
8	0.44924E-06	0.12989E-05	0.18536E-05	0.18536E-05	0.12398E-04
9	0.39510E-06	0.12095E-05	0.17283E-05	0.17283E-05	0.11156E-04
10	0.34726E-06	0.11478E-05	0.15878E-05	0.15878E-05	0.99450E-05
11	0.30775E-06	0.10746E-05	0.14329E-05	0.14329E-05	0.88159E-05
12	0.26207E-06	0.98700E-06	0.12833E-05	0.12833E-05	0.76280E-05
13	0.22418E-06	0.87924E-06	0.11245E-05	0.11245E-05	0.65113E-05
14	0.18904E-06	0.77764E-06	0.98148E-06	0.98148E-06	0.54617E-05
15	0.15809E-06	0.69345E-06	0.84471E-06	0.84471E-06	0.44739E-05
16	0.13411E-06	0.62203E-06	0.72991E-06	0.72991E-06	0.37422E-05
17	0.11564E-06	0.55717E-06	0.64350E-06	0.64350E-06	0.32661E-05
18	0.10516E-06	0.50502E-06	0.57520E-06	0.57520E-06	0.30152E-05
19	0.10101E-06	0.46193E-06	0.53100E-06	0.53100E-06	0.29279E-05
20	0.98711E-07	0.43618E-06	0.49934E-06	0.49934E-06	0.28901E-05

Residuals Output File (`case.rsd2`)

Basic File Format

```
1 (RSD(i), i = 1,5) 1
1 (RSD(i), i = 1,5) 2
:                   :
:                   :
1 (RSD(i), i = 1,5) icyc
:                   :
:                   :
istp (RSD(i), i = 1,5) 1
:                   :
:                   :
nstp (RSD(i), i = 1,5) 1
:                   :
:                   :
```

Definition of Terms

<code>istp</code> :	<code>int</code>	current solution step
<code>icyc</code> :	<code>int</code>	iteration of current residual
<code>nstp</code> :	<code>int</code>	total or last solution step
<code>RSD(1)</code> :	<code>real</code>	density solution residual
<code>RSD(2)</code> :	<code>real</code>	x-momentum solution residual
<code>RSD(3)</code> :	<code>real</code>	y-momentum solution residual
<code>RSD(4)</code> :	<code>real</code>	z-momentum solution residual
<code>RSD(5)</code> :	<code>real</code>	energy solution residual

Comments

- This is a plain text (ASCII) file.
- This file is output when `irsds = .true.` in the controls `case.con` file. The residuals shown in this file represent the RMS changes at all nodes in the domain for this inner cycle. The convergence of residuals within any iteration can be seen in the trend in the residuals through the cycles used.

Sample File

1	0.38320E-05	0.10743E-04	0.69854E-05	0.69854E-05	0.69854E-05	0.69854E-05	0.10598E-03	1
1	0.20317E-05	0.50694E-05	0.40436E-05	0.40436E-05	0.40436E-05	0.40436E-05	0.56307E-04	2
1	0.12024E-05	0.35187E-05	0.26241E-05	0.26241E-05	0.26241E-05	0.26241E-05	0.32195E-04	3
1	0.91334E-06	0.25166E-05	0.23637E-05	0.23637E-05	0.23637E-05	0.23637E-05	0.24240E-04	4
1	0.73183E-06	0.19442E-05	0.22228E-05	0.22228E-05	0.22228E-05	0.22228E-05	0.19376E-04	5
1	0.59870E-06	0.16179E-05	0.20889E-05	0.20889E-05	0.20889E-05	0.20889E-05	0.15963E-04	6
1	0.51663E-06	0.14311E-05	0.19719E-05	0.19719E-05	0.19719E-05	0.19719E-05	0.13946E-04	7
1	0.44924E-06	0.12989E-05	0.18536E-05	0.18536E-05	0.18536E-05	0.18536E-05	0.12398E-04	8
2	0.39510E-06	0.12095E-05	0.17283E-05	0.17283E-05	0.17283E-05	0.17283E-05	0.11156E-04	1
2	0.34726E-06	0.11478E-05	0.15878E-05	0.15878E-05	0.15878E-05	0.15878E-05	0.99450E-05	2
2	0.30775E-06	0.10746E-05	0.14329E-05	0.14329E-05	0.14329E-05	0.14329E-05	0.88159E-05	3
2	0.26207E-06	0.98700E-06	0.12833E-05	0.12833E-05	0.12833E-05	0.12833E-05	0.76280E-05	4
2	0.22418E-06	0.87924E-06	0.11245E-05	0.11245E-05	0.11245E-05	0.11245E-05	0.65113E-05	5
2	0.18904E-06	0.77764E-06	0.98148E-06	0.98148E-06	0.98148E-06	0.98148E-06	0.54617E-05	6
2	0.15809E-06	0.69345E-06	0.84471E-06	0.84471E-06	0.84471E-06	0.84471E-06	0.44739E-05	7
2	0.13411E-06	0.62203E-06	0.72991E-06	0.72991E-06	0.72991E-06	0.72991E-06	0.37422E-05	8
3	0.11564E-06	0.55717E-06	0.64350E-06	0.64350E-06	0.64350E-06	0.64350E-06	0.32661E-05	1
3	0.10516E-06	0.50502E-06	0.57520E-06	0.57520E-06	0.57520E-06	0.57520E-06	0.30152E-05	2
3	0.10101E-06	0.46193E-06	0.53100E-06	0.53100E-06	0.53100E-06	0.53100E-06	0.29279E-05	3
3	0.98711E-07	0.43618E-06	0.49934E-06	0.49934E-06	0.49934E-06	0.49934E-06	0.28901E-05	4

Aerodynamic Loads Output File (case.lds)

Basic File Format

```

0      0.0      (FD(i), i = 1,6)
:      :      :
:      :      :
istp   t_istp   (FD(i), i = 1,6)
:      :      :
:      :      :
nstp   t_nstp   (FD(i), i = 1,6)

```

Definition of Terms

```

istp:  int    current solution step
nstp:  int    total or last solution step
t:     real   dimensionless time at step i

FD(1): real   x-force coefficient
FD(2): real   y-force coefficient
FD(3): real   z-force coefficient
FD(4): real   x-moment coefficient
FD(5): real   y-moment coefficient
FD(6): real   z-moment coefficient

```

Comments

- This is a plain text (ASCII) file.
- The force coefficients FD in this output file are dimensionless values based on the reference conditions specified in the solver control file `case.con`:

$$FD(1) = \frac{F_x}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^2}$$

$$FD(2) = \frac{F_y}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^2}$$

$$FD(3) = \frac{F_z}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^2}$$

$$FD(4) = \frac{M_x}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^3}$$

$$FD(5) = \frac{M_y}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^3}$$

$$FD(6) = \frac{M_z}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^3}$$

where M is the free-stream Mach number and L is the reference dimension, both appearing in `case.con`.

- The moment coefficient is calculated in reference to the origin of the mesh.
- For dynamic (non-inertial) problems, the force coefficients stored in this file are referenced to the body-fixed coordinate system.

Sample File

0	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
1	0.10000E+00	0.29186E+01	0.40893E-04	0.20386E-04	0.20386E-04	0.00000E+00	0.00000E+00
2	0.20000E+00	0.53878E+01	0.74036E-04	0.36920E-04	0.36920E-04	0.00000E+00	0.00000E+00
3	0.30000E+00	0.73426E+01	0.10798E-03	0.53867E-04	0.53867E-04	0.00000E+00	0.00000E+00
4	0.40000E+00	0.87754E+01	0.14611E-03	0.72920E-04	0.72920E-04	0.00000E+00	0.00000E+00
5	0.50000E+00	0.97221E+01	0.18835E-03	0.94033E-04	0.94033E-04	0.00000E+00	0.00000E+00
6	0.60000E+00	0.10250E+02	0.23574E-03	0.11772E-03	0.11772E-03	0.00000E+00	0.00000E+00
7	0.70000E+00	0.10444E+02	0.26257E-03	0.13112E-03	0.13112E-03	0.00000E+00	0.00000E+00
8	0.80000E+00	0.10391E+02	0.25348E-03	0.12657E-03	0.12657E-03	0.00000E+00	0.00000E+00
9	0.90000E+00	0.10173E+02	0.25302E-03	0.12632E-03	0.12632E-03	0.00000E+00	0.00000E+00
10	0.10000E+01	0.98630E+01	0.23377E-03	0.11668E-03	0.11668E-03	0.00000E+00	0.00000E+00
11	0.11000E+01	0.95136E+01	0.20482E-03	0.10219E-03	0.10219E-03	0.00000E+00	0.00000E+00
12	0.12000E+01	0.91543E+01	0.19370E-03	0.96640E-04	0.96640E-04	0.00000E+00	0.00000E+00
13	0.13000E+01	0.88118E+01	0.23365E-03	0.11662E-03	0.11662E-03	0.00000E+00	0.00000E+00
14	0.14000E+01	0.85116E+01	0.28553E-03	0.14256E-03	0.14256E-03	0.00000E+00	0.00000E+00
15	0.15000E+01	0.82553E+01	0.37539E-03	0.18747E-03	0.18747E-03	0.00000E+00	0.00000E+00
16	0.16000E+01	0.80367E+01	0.55544E-03	0.27749E-03	0.27749E-03	0.00000E+00	0.00000E+00
17	0.17000E+01	0.78461E+01	0.76662E-03	0.38306E-03	0.38306E-03	0.00000E+00	0.00000E+00
18	0.18000E+01	0.76747E+01	0.10095E-02	0.50449E-03	0.50449E-03	0.00000E+00	0.00000E+00
19	0.19000E+01	0.75147E+01	0.12664E-02	0.63292E-03	0.63292E-03	0.00000E+00	0.00000E+00
20	0.20000E+01	0.73607E+01	0.15058E-02	0.75262E-03	0.75262E-03	0.00000E+00	0.00000E+00
∴	∴	∴	∴	∴	∴	∴	∴

Dynamic Output File (xd.dat)

Basic File Format

```

0    0.0    (XD(i), i = 1,3)
          roll pitch yaw
          (XD(i), i = 4,9)
          (FD(i), i = 1,6)
          (XD(i), i = 10,13)
:      :      : :
istp  t_istp (XD(i), i = 1,3)
          roll pitch yaw
          (XD(i), i = 4,9)
          (FD(i), i = 1,6)
          (XD(i), i = 10,13)
:      :      : :
nstp  t_nstp (XD(i), i = 1,3)
          roll pitch yaw
          (XD(i), i = 4,9)
          (FD(i), i = 1,6)
          (XD(i), i = 10,13)

```

Definition of Terms

```

istp:  int    current solution step
nstp:  int    total or last solution step
t:     real   dimensionless time at step i

XD(1): real   x- position
XD(2): real   y- position
XD(3): real   z- position
XD(4): real   x-velocity
XD(5): real   y-velocity
XD(6): real   z-velocity
XD(7): real   roll rate (rad/s)
XD(8): real   pitch rate (rad/s)
XD(9): real   yaw rate (rad/s)
XD(10): real  first quaternion q0
XD(11): real  second quaternion q1
XD(12): real  third quaternion q2
XD(13): real  fourth quaternion q3

roll:  real   roll angle (rad)
pitch: real   pitch angle (rad)
yaw:   real   yaw angle (rad)

FD(1): real   x- force
FD(2): real   y- force
FD(3): real   z- force
FD(4): real   roll moment
FD(5): real   pitch moment
FD(6): real   yaw moment

```

Comments

- This is a plain text (ASCII) file.
- The dynamic data in this output file is dimensionless based on the reference conditions specified in the solver control file `case.con`:

$$\begin{aligned}
 XD(1) &= \frac{x_0}{L} & XD(2) &= \frac{y_0}{L} & XD(3) &= \frac{z_0}{L} \\
 roll &= \phi_0 \text{ (rad)} & pitch &= \theta_0 \text{ (rad)} & yaw &= \psi_0 \text{ (rad)} \\
 XD(4) &= \frac{V_x}{M a_\infty} & XD(5) &= \frac{V_y}{M a_\infty} & XD(6) &= \frac{V_z}{M a_\infty} \\
 XD(7) &= \frac{L}{M a_\infty} p_0 \text{ (rad/s)} & XD(8) &= \frac{L}{M a_\infty} q_0 \text{ (rad/s)}
 \end{aligned}$$

$$XD(9) = \frac{L}{M a_\infty} r_0 \text{ (rad / s)}$$

$$FD(1) = \frac{F_x}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^2}$$

$$FD(2) = \frac{F_y}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^2}$$

$$FD(3) = \frac{F_z}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^2}$$

$$FD(4) = \frac{M_x}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^3}$$

$$FD(5) = \frac{M_y}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^3}$$

$$FD(6) = \frac{M_z}{\frac{1}{2} \rho_\infty M^2 a_\infty^2 L^3}$$

where M is the Mach number and L is the reference dimension in `case.con`. The dimensionless forces are expressed in the same form as the `case.lids` file.

- The position, velocity, and acceleration vectors in this file are defined relative to the global coordinate system, while the rotational quantities are defined as rotations about the local or body-fixed coordinate system.

Sample File

```

1 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00
2 0.203E-05 0.506E-05 0.404E-05 0.563E-04 0.120E-05 0.351E-05 0.262E-05 0.321E-04 0.262E-05 0.351E-05
3 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06
4 0.598E-06 0.141E-05 0.141E-05 0.141E-05 0.598E-06 0.141E-05 0.141E-05 0.141E-05 0.598E-06 0.141E-05
5 0.516E-06 0.197E-05 0.197E-05 0.197E-05 0.516E-06 0.197E-05 0.197E-05 0.197E-05 0.516E-06 0.197E-05
6 0.449E-06 0.122E-05 0.122E-05 0.122E-05 0.449E-06 0.122E-05 0.122E-05 0.122E-05 0.449E-06 0.122E-05
7 0.395E-06 0.172E-05 0.172E-05 0.172E-05 0.395E-06 0.172E-05 0.172E-05 0.172E-05 0.395E-06 0.172E-05
8 0.307E-06 0.107E-05 0.107E-05 0.107E-05 0.307E-06 0.107E-05 0.107E-05 0.107E-05 0.307E-06 0.107E-05
9 0.224E-06 0.879E-06 0.879E-06 0.112E-05 0.224E-06 0.879E-06 0.879E-06 0.112E-05 0.224E-06 0.879E-06
10 0.189E-06 0.779E-06 0.779E-06 0.981E-06 0.189E-06 0.779E-06 0.779E-06 0.981E-06 0.189E-06 0.779E-06
11 0.134E-06 0.623E-06 0.623E-06 0.749E-06 0.134E-06 0.623E-06 0.623E-06 0.749E-06 0.134E-06 0.623E-06
12 0.115E-06 0.557E-06 0.557E-06 0.643E-06 0.115E-06 0.557E-06 0.557E-06 0.643E-06 0.115E-06 0.557E-06
13 0.105E-06 0.505E-06 0.505E-06 0.301E-05 0.105E-06 0.505E-06 0.505E-06 0.301E-05 0.105E-06 0.505E-06
14 0.101E-06 0.461E-06 0.461E-06 0.436E-06 0.101E-06 0.461E-06 0.461E-06 0.436E-06 0.101E-06 0.461E-06
15 0.987E-07 0.436E-06 0.436E-06 0.289E-05 0.987E-07 0.436E-06 0.436E-06 0.289E-05 0.987E-07 0.436E-06
16 0.203E-05 0.120E-05 0.120E-05 0.203E-05 0.203E-05 0.120E-05 0.120E-05 0.203E-05 0.203E-05 0.120E-05
17 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06
18 0.598E-06 0.141E-05 0.141E-05 0.598E-06 0.598E-06 0.141E-05 0.141E-05 0.598E-06 0.598E-06 0.141E-05
19 0.516E-06 0.197E-05 0.197E-05 0.516E-06 0.516E-06 0.197E-05 0.197E-05 0.516E-06 0.516E-06 0.197E-05
20 0.449E-06 0.122E-05 0.122E-05 0.449E-06 0.449E-06 0.122E-05 0.122E-05 0.449E-06 0.449E-06 0.122E-05

0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00
0.203E-05 0.120E-05 0.120E-05 0.203E-05 0.203E-05 0.120E-05 0.120E-05 0.203E-05 0.203E-05 0.120E-05
0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06 0.731E-06
0.598E-06 0.598E-06 0.598E-06 0.598E-06 0.598E-06 0.141E-05 0.141E-05 0.598E-06 0.598E-06 0.141E-05
0.449E-06 0.449E-06 0.449E-06 0.449E-06 0.449E-06 0.197E-05 0.197E-05 0.449E-06 0.449E-06 0.197E-05
0.395E-06 0.395E-06 0.395E-06 0.395E-06 0.395E-06 0.172E-05 0.172E-05 0.395E-06 0.395E-06 0.172E-05
0.307E-06 0.307E-06 0.307E-06 0.307E-06 0.307E-06 0.107E-05 0.107E-05 0.307E-06 0.307E-06 0.107E-05
0.224E-06 0.224E-06 0.224E-06 0.224E-06 0.224E-06 0.879E-06 0.879E-06 0.224E-06 0.224E-06 0.879E-06
0.189E-06 0.189E-06 0.189E-06 0.189E-06 0.189E-06 0.779E-06 0.779E-06 0.189E-06 0.189E-06 0.779E-06
0.134E-06 0.134E-06 0.134E-06 0.134E-06 0.134E-06 0.623E-06 0.623E-06 0.134E-06 0.134E-06 0.623E-06
0.115E-06 0.115E-06 0.115E-06 0.115E-06 0.115E-06 0.557E-06 0.557E-06 0.115E-06 0.115E-06 0.557E-06
0.105E-06 0.105E-06 0.105E-06 0.105E-06 0.105E-06 0.505E-06 0.505E-06 0.105E-06 0.105E-06 0.505E-06
0.101E-06 0.101E-06 0.101E-06 0.101E-06 0.101E-06 0.461E-06 0.461E-06 0.101E-06 0.101E-06 0.461E-06
0.987E-07 0.987E-07 0.987E-07 0.987E-07 0.987E-07 0.436E-06 0.436E-06 0.987E-07 0.987E-07 0.436E-06

```

Dynamic Output File (xn.dat)

Basic File Format

```

0      0.0      (XN(i), i = 1, nr)
              (VN(i), i = 1, nr)
              (FA(i), i = 1, nr)

  ⋮      ⋮      ⋮      ⋮

istp   t_istp   (XN(i), i = 1, nr)
              (VN(i), i = 1, nr)
              (FA(i), i = 1, nr)

  ⋮      ⋮      ⋮      ⋮

nstp   t_nstp   (XN(i), i = 1, nr)
              (VN(i), i = 1, nr)
              (FA(i), i = 1, nr)

```

Definition of Terms

istp:	int	current solution step
nstp:	int	total or last solution step
t:	real	dimensionless time at step i
XN(i):	real	generalized displ. on mode i
VN(i):	real	generalized velocity on mode i
FA(i):	real	generalized force on mode i

Comments

- This is a plain text (ASCII) file.
- The number of modes nr is given in the control file `case.con`.
- The elastic data in this output file is dimensionless based on the reference conditions specified in the solver control file `case.con`:

$$XN(i) = x_{n,i} \qquad VN(i) = \frac{L}{M a_\infty} \dot{x}_{n,i} \qquad FA(i) = \frac{F_{n,i}}{\rho_\infty M^2 a_\infty^2 L^3}$$

where M is the Mach number and L is the reference dimension in `case.con`.

- To reassemble the elastic displacements at all points along the surface of the model, use the modal deflections (mode shapes) `PHIA` in the elastics file (`case.vec`) and $x_{n,i}$. The displacement vector at the k^{th} node is:

$$\vec{\delta}(k) = \sum_{i=1}^N x_{n,i} \begin{Bmatrix} PHIA(2k-1, i) \\ PHIA(2k, i) \end{Bmatrix}$$

where N is the number of mode shapes in the elastic system. The boundary velocity at the k^{th} node is:

$$\vec{V}_b(k) = \sum_{i=1}^N \dot{x}_{n,i} \begin{Bmatrix} PHIA(2k-1, i) \\ PHIA(2k, i) \end{Bmatrix}$$

Sample File (*nr* = 3)

1	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
2	0.20317E-05	0.50694E-05	0.40436E-05	0.56307E-04	0.56307E-04	0.56307E-04	0.56307E-04	0.56307E-04	0.56307E-04
3	0.12024E-05	0.35187E-05	0.26241E-05	0.24240E-04	0.24240E-04	0.24240E-04	0.24240E-04	0.24240E-04	0.24240E-04
4	0.91334E-06	0.25166E-05	0.23637E-05	0.19376E-04	0.19376E-04	0.19376E-04	0.19376E-04	0.19376E-04	0.19376E-04
5	0.73183E-06	0.19442E-05	0.22228E-05	0.15963E-04	0.15963E-04	0.15963E-04	0.15963E-04	0.15963E-04	0.15963E-04
6	0.59870E-06	0.16179E-05	0.20889E-05	0.13946E-04	0.13946E-04	0.13946E-04	0.13946E-04	0.13946E-04	0.13946E-04
7	0.51663E-06	0.14311E-05	0.19719E-05	0.12398E-04	0.12398E-04	0.12398E-04	0.12398E-04	0.12398E-04	0.12398E-04
8	0.44924E-06	0.12989E-05	0.18536E-05	0.11566E-04	0.11566E-04	0.11566E-04	0.11566E-04	0.11566E-04	0.11566E-04
9	0.39510E-06	0.12095E-05	0.17283E-05	0.99450E-05	0.99450E-05	0.99450E-05	0.99450E-05	0.99450E-05	0.99450E-05
10	0.34726E-06	0.11478E-05	0.15878E-05	0.88159E-05	0.88159E-05	0.88159E-05	0.88159E-05	0.88159E-05	0.88159E-05
11	0.30775E-06	0.10746E-05	0.14329E-05	0.76280E-05	0.76280E-05	0.76280E-05	0.76280E-05	0.76280E-05	0.76280E-05
12	0.26207E-06	0.98700E-06	0.12833E-05	0.65113E-05	0.65113E-05	0.65113E-05	0.65113E-05	0.65113E-05	0.65113E-05
13	0.22418E-06	0.87924E-06	0.11245E-05	0.54617E-05	0.54617E-05	0.54617E-05	0.54617E-05	0.54617E-05	0.54617E-05
14	0.18904E-06	0.77764E-06	0.98148E-06	0.44739E-05	0.44739E-05	0.44739E-05	0.44739E-05	0.44739E-05	0.44739E-05
15	0.15809E-06	0.69345E-06	0.84471E-06	0.37422E-05	0.37422E-05	0.37422E-05	0.37422E-05	0.37422E-05	0.37422E-05
16	0.13411E-06	0.62203E-06	0.72991E-06	0.32661E-05	0.32661E-05	0.32661E-05	0.32661E-05	0.32661E-05	0.32661E-05
17	0.11564E-06	0.55717E-06	0.64350E-06	0.30152E-05	0.30152E-05	0.30152E-05	0.30152E-05	0.30152E-05	0.30152E-05
18	0.10516E-06	0.50502E-06	0.57520E-06	0.29279E-05	0.29279E-05	0.29279E-05	0.29279E-05	0.29279E-05	0.29279E-05
19	0.10101E-06	0.46193E-06	0.53100E-06	0.29279E-05	0.29279E-05	0.29279E-05	0.29279E-05	0.29279E-05	0.29279E-05

Restart Files (case.rst and case.rs2)

Basic File Format

```
istp nnd gam xmi ref dt
((UN(i,j), i = 1,nnd), j = 1,6)
((UNO(i,j), i = 1,nnd), j = 1,6)
(XN(i,j), i = 1,2*nr),
      (XN1(i), i = 1,2*nr)
(FA(i,j), i = 1,3*nr),
      (FA2(i), i = 1,nr)
(XD(i,j), i = 1,13),
      (XD1(i), i = 1,13)
(FD(i,j), i = 1,18),
      (FD2(i), i = 1,6)
```

Definition of Terms

istp:	int	step in global solution
nnd:	int	number of nodes
gam:	real	ratio of specific heats
xmi:	real	free-stream Mach number
ref:	real	reference dimension
dt:	real	global time step
UN:	real	unknowns at previous step
UNO:	real	unknowns at two steps prior
XN:	real	elastic deflect / velocity
XN1:	real	prev. elastic deflect / velocity
FA:	real	gen. aero. forces at 3 steps
FA2:	real	external forcing on modes
XD:	real	rigid body position / velocity
XD1:	real	prev. rigid body pos. / vel.
FD:	real	aero. forces at 3 steps
FD2:	real	external forcing on vehicle

Comments

- This is an unformatted (binary) file.
- The solution unknowns, deflections, and forces stored in this file are dimensionless quantities.
- Unknowns properties vector UNO is only written for 2nd order unsteady solutions (isol = 2).
- Elastic properties XN, XN1, FA, and FA2 is only written when the elastics flag ielast is set to .true.
- Non-inertial properties XD, XD1, FD, and FD2 is only written when the non-inertial flag idynm is set to .true.
- A two-file system is used so that one file is written while the other file is untouched. If the program crashes, one file is always untouched so that one of the two files is always recoverable.

Acoustic Pressure Output File (case.pac)

Basic File Format

Controls:

```
dt = "dt"
Lref = "refdim"
Uinf = "uinf"
ainf = "ainf"
mach = "mach"
gam = "gam"
```

Data Layout:

```
nacp = "nacp"
nacl = "nacl"
-- "NN" intersections with Line #"N"
  :           :           :
-- "NN" intersections with Line #"N"

x-coord ==> "x1" "x2" || "x3" . . . .
y-coord ==> "y1" "y2" || "y3" . . . .
z-coord ==> "z1" "z2" || "z3" . . . .
--- Time --- ---- ---- ++ ---- . . . .
  "t"      "Cp1" "Cp2" || "Cp3" . . . .
  :           :   :   ||   :
  :           :   :   ||   :
```

Definition of Terms

dt:	real	current solution step
refdim:	real	total or last solution step
uinf:	real	total or last solution step
ainf:	real	total or last solution step
mach:	real	total or last solution step
gam:	real	total or last solution step
nacp:	int	number of acoustic points
nacl:	int	number of acoustic lines
NN:	int	number of intersections along a particular acoustic line
N:	int	index of this acoustic lines
x#:	real	x-coordinate at node
y#:	real	y-coordinate at node
z#:	real	z-coordinate at node
t:	real	solution time
Cp#:	real	coefficient of pressure at node

Comments

- This is a plain text (ASCII) file.
- Text shown in the file format above without quotes (“ ”) is used directly in the output file. Text shown above in quotes represents a variable, or number written to the file. These variables are defined above on the right.
- Descriptions may be included in the output file to the right of line of header data.
- The first section represents solution controls used to generate the data file.
- The second section describes the data in the third section. If any of the acoustic points are not found within the solution domain, a warning is written before the number of acoustic lines is written. The number of intersections is listed for each acoustic line in the input file.
- The data is written out so that each column represents a node or intersection within the domain. The first three rows give the (x,y,z) coordinate of the node or intersection. The rows under the dashed divider line are the coefficient of pressure at the node or intersection point at the solution time designated in the first column.
- A double vertical line breaks the acoustic point and acoustic line data. Subsequent single vertical lines break data between acoustic lines.

- The intersections along a particular acoustic line are ordered parametrically along the length of the acoustic line, from starting to end points, as listed in the input file.

Sample File

Controls:

```

dt = 0.0010000 Dimensionless time step
Lref = 1.0000000 Reference dimension
Uinf = 781.2000000 Freestream velocity
ainf = 1116.0000000 Freestream acoustic speed
mach = 0.7000000 Freestream Mach number
gam = 1.4000000 Ratio of specific heats
  
```

Data Layout:

```

nacr = 2 Number of acoustic points
nacl = 2 Number of acoustic lines
  
```

```

-- 2 intersections with Line # 1
-- 1 intersections with Line # 2
  
```

x-coord ==>	0.0000000	0.2000000	0.4000000	0.6000000	0.8000000
Y-coord ==>	0.0000000	0.7000000	0.4000000	0.3000000	0.2000000
Z-coord ==>	0.0000000	0.1000000	0.2000000	0.3000000	0.4000000
Time	0.0010000	0.0000000	0.0000000	0.0000000	0.0000000
	0.0020000	0.0000000	0.0000000	0.0000000	0.0000000
	0.0030000	0.0000000	0.0000000	0.0000000	0.0000000
	0.0040000	0.0000000	0.0000000	0.0000000	0.0000000
:	:	:	:	:	:
:	:	:	:	:	:

Acoustic Density Output File (case.rac)

Basic File Format

Controls:

```
dt = "dt"
Lref = "refdim"
Uinf = "uinf"
ainf = "ainf"
mach = "mach"
gam = "gam"
```

Data Layout:

```
nacp = "nacp"
nacl = "nacl"
-- "NN" intersections with Line #"N"
  :           :           :
-- "NN" intersections with Line #"N"

x-coord ==> "x1" "x2" || "x3" . . . .
y-coord ==> "y1" "y2" || "y3" . . . .
z-coord ==> "z1" "z2" || "z3" . . . .
--- Time --- ---- ---- ++ ---- . . . .
  "t"      "rh1" "rh2" || "rh3" . . . .
  :           :           : ||           :
```

Definition of Terms

dt:	real	current solution step
refdim:	real	total or last solution step
uinf:	real	total or last solution step
ainf:	real	total or last solution step
mach:	real	total or last solution step
gam:	real	total or last solution step
nacp:	int	number of acoustic points
nacl:	int	number of acoustic lines
NN:	int	number of intersections along a particular acoustic line
N:	int	index of this acoustic lines
x#:	real	x-coordinate at node
y#:	real	y-coordinate at node
z#:	real	z-coordinate at node
t:	real	solution time
rh#:	real	dimensionless density at node

Comments

- This is a plain text (ASCII) file.
- Text shown in the file format above without quotes (“ ”) is used directly in the output file. Text shown above in quotes represents a variable, or number written to the file. These variables are defined above on the right.
- Descriptions may be included in the output file to the right of line of header data.
- The first section represents solution controls used to generate the data file.
- The second section describes the data in the third section. If any of the acoustic points are not found within the solution domain, a warning is written before the number of acoustic lines is written. The number of intersections is listed for each acoustic line in the input file.
- The data is written out so that each column represents a node or intersection within the domain. The first three rows give the (x,y,z) coordinate of the node or intersection. The rows under the dashed divider line are the dimensionless density at the node or intersection point at the solution time designated in the first column.
- Density data is presented in dimensionless form:

$$rh\# = \frac{\rho}{\rho_{\infty}}$$

- A double vertical line breaks the acoustic point and acoustic line data. Subsequent single vertical lines break data from acoustic lines.
- The intersections along a particular acoustic line are ordered parametrically along the length of the acoustic line, from starting to end points, as listed in the input file.

Sample File

Controls:

```

dt = 0.00100000 Dimensionless time step
Lref = 1.00000000 Reference dimension
Uinf = 781.20000000 Freestream velocity
ainf = 1116.00000000 Freestream acoustic speed
mach = 0.70000000 Freestream Mach number
gam = 1.40000000 Ratio of specific heats

```

Data Layout:

```

nacp = 2 Number of acoustic points
nacl = 2 Number of acoustic lines

```

```

-- 2 intersections with Line # 1
-- 1 intersections with Line # 2

```

	x-coord ==>	0.00000000	0.20000000	0.40000000	0.60000000	0.80000000	1.00000000
Y-coord ==>	0.00000000	0.70000000	0.40000000	0.30000000	0.20000000	0.10000000	0.00000000
Z-coord ==>	0.00000000	0.10000000	0.20000000	0.30000000	0.40000000	0.50000000	0.60000000
Time	0.00100000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
	0.00200000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
	0.00300000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
	0.00400000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
	:	:	:	:	:	:	:
	:	:	:	:	:	:	:

Acoustic u-Velocity Output File (case.uac)

Basic File Format

Controls:

```
dt = "dt"
Lref = "refdim"
Uinf = "uinf"
ainf = "ainf"
mach = "mach"
gam = "gam"
```

Data Layout:

```
nacp = "nacp"
nacl = "nacl"
-- "NN" intersections with Line #"N"
  :           :           :
-- "NN" intersections with Line #"N"

x-coord ==> "x1" "x2" || "x3" . . . .
y-coord ==> "y1" "y2" || "y3" . . . .
z-coord ==> "z1" "z2" || "z3" . . . .
--- Time --- ---- ---- ++ ---- . . . .
  "t"      "uu1" "uu2" || "uu3" . . . .
  :           :     : || :
  :           :     : || :
```

Definition of Terms

dt:	real	current solution step
refdim:	real	total or last solution step
uinf:	real	total or last solution step
ainf:	real	total or last solution step
mach:	real	total or last solution step
gam:	real	total or last solution step
nacp:	int	number of acoustic points
nacl:	int	number of acoustic lines
NN:	int	number of intersections along a particular acoustic line
N:	int	index of this acoustic lines
x#:	real	x-coordinate at node
y#:	real	y-coordinate at node
z#:	real	z-coordinate at node
t:	real	solution time
uu#:	real	dimensionless u-velocity at node

Comments

- This is a plain text (ASCII) file.
- Text shown in the file format above without quotes (“ ”) is used directly in the output file. Text shown above in quotes represents a variable, or number written to the file. These variables are defined above on the right.
- Descriptions may be included in the output file to the right of line of header data.
- The first section represents solution controls used to generate the data file.
- The second section describes the data in the third section. If any of the acoustic points are not found within the solution domain, a warning is written before the number of acoustic lines is written. The number of intersections is listed for each acoustic line in the input file.
- The data is written out so that each column represents a node or intersection within the domain. The first three rows give the (x,y,z) coordinate of the node or intersection. The rows under the dashed divider line are the dimensionless u-velocity at the node or intersection point at the solution time designated in the first column.
- u-Velocity data is presented in dimensionless form:

$$uu\# = \frac{u}{U_\infty}$$

- A double vertical line breaks the acoustic point and acoustic line data. Subsequent single vertical lines break data from acoustic lines.
- The intersections along a particular acoustic line are ordered parametrically along the length of the acoustic line, from starting to end points, as listed in the input file.

Sample File

Controls:

```

dt = 0.00100000 Dimensionless time step
Lref = 1.00000000 Reference dimension
Uinf = 781.20000000 Freestream velocity
ainf = 1116.00000000 Freestream acoustic speed
mach = 0.70000000 Freestream Mach number
gam = 1.40000000 Ratio of specific heats
  
```

Data Layout:

```

nacp = 2 Number of acoustic points
nacl = 2 Number of acoustic lines
  
```

```

-- 2 intersections with Line # 1
-- 1 intersections with Line # 2
  
```

x-coord ==>	0.00000000	0.20000000	0.40000000	0.60000000	0.80000000	1.00000000
Y-coord ==>	0.00000000	0.70000000	0.40000000	0.30000000	0.20000000	0.10000000
Z-coord ==>	0.00000000	0.10000000	0.20000000	0.30000000	0.40000000	0.50000000
Time	0.00100000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
	0.00200000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
	0.00300000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
	0.00400000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
:	:	:	:	:	:	:
:	:	:	:	:	:	:

Acoustic v-Velocity Output File (case.vac)

Basic File Format

Controls:

```
dt = "dt"
Lref = "refdim"
Uinf = "uinf"
ainf = "ainf"
mach = "mach"
gam = "gam"
```

Data Layout:

```
nacp = "nacp"
nacl = "nacl"
-- "NN" intersections with Line #"N"
  :           :           :
-- "NN" intersections with Line #"N"

x-coord ==> "x1" "x2" || "x3" . . . .
y-coord ==> "y1" "y2" || "y3" . . . .
z-coord ==> "z1" "z2" || "z3" . . . .
--- Time --- ---- ---- ++ ---- . . . .
  "t"      "vv1" "vv2" || "vv3" . . . .
  :           :           : ||           :
  :           :           : ||           :
```

Definition of Terms

dt:	real	current solution step
refdim:	real	total or last solution step
uinf:	real	total or last solution step
ainf:	real	total or last solution step
mach:	real	total or last solution step
gam:	real	total or last solution step
nacp:	int	number of acoustic points
nacl:	int	number of acoustic lines
NN:	int	number of intersections along a particular acoustic line
N:	int	index of this acoustic lines
x#:	real	x-coordinate at node
y#:	real	y-coordinate at node
z#:	real	z-coordinate at node
t:	real	solution time
vv#:	real	dimensionless v-velocity at node

Comments

- This is a plain text (ASCII) file.
- Text shown in the file format above without quotes (“ ”) is used directly in the output file. Text shown above in quotes represents a variable, or number written to the file. These variables are defined above on the right.
- Descriptions may be included in the output file to the right of line of header data.
- The first section represents solution controls used to generate the data file.
- The second section describes the data in the third section. If any of the acoustic points are not found within the solution domain, a warning is written before the number of acoustic lines is written. The number of intersections is listed for each acoustic line in the input file.
- The data is written out so that each column represents a node or intersection within the domain. The first three rows give the (x,y/z) coordinate of the node or intersection. The rows under the dashed divider line are the dimensionless v-velocity at the node or intersection point at the solution time designated in the first column.
- v-Velocity data is presented in dimensionless form:

$$vv\# = \frac{v}{U_\infty}$$

- A double vertical line breaks the acoustic point and acoustic line data. Subsequent single vertical lines break data from acoustic lines.
- The intersections along a particular acoustic line are ordered parametrically along the length of the acoustic line, from starting to end points, as listed in the input file.

Sample File

Controls:

```

dt = 0.0010000 Dimensionless time step
Lref = 1.0000000 Reference dimension
Uinf = 781.2000000 Freestream velocity
ainf = 1116.0000000 Freestream acoustic speed
mach = 0.7000000 Freestream Mach number
gam = 1.4000000 Ratio of specific heats

```

Data Layout:

```

nacp = 2 Number of acoustic points
nacl = 2 Number of acoustic lines

```

```

-- 2 intersections with Line # 1
-- 1 intersections with Line # 2

```

	x-coord ==>	0.0000000	0.2000000	0.0000000	0.0000000	1.0000000	1.0000000	0.7000000
Y-coord ==>	0.0000000	0.7000000	0.4000000	0.4000000	1.3000000	0.6000000	0.6000000	0.6000000
Z-coord ==>	0.0000000	0.1000000	0.2000000	0.2000000	0.2000000	0.5000000	0.5000000	0.5000000
Time ----	0.0010000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
	0.0020000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
	0.0030000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000

Acoustic w-Velocity Output File (case.wac)

Basic File Format

Controls:

```
dt = "dt"
Lref = "refdim"
Uinf = "uinf"
ainf = "ainf"
mach = "mach"
gam = "gam"
```

Data Layout:

```
nacp = "nacp"
nacl = "nacl"
-- "NN" intersections with Line #"N"
  :           :           :
-- "NN" intersections with Line #"N"

x-coord ==> "x1" "x2" || "x3" . . . .
y-coord ==> "y1" "y2" || "y3" . . . .
z-coord ==> "z1" "z2" || "z3" . . . .
--- Time --- ---- ---- ++ ---- . . . .
  "t"      "ww1" "ww2" || "ww3" . . . .
  :           :           : ||           :
```

Definition of Terms

dt:	real	current solution step
refdim:	real	total or last solution step
uinf:	real	total or last solution step
ainf:	real	total or last solution step
mach:	real	total or last solution step
gam:	real	total or last solution step
nacp:	int	number of acoustic points
nacl:	int	number of acoustic lines
NN:	int	number of intersections along a particular acoustic line
N:	int	index of this acoustic lines
x#:	real	x-coordinate at node
y#:	real	y-coordinate at node
z#:	real	z-coordinate at node
t:	real	solution time
ww#:	real	dimensionless w-velocity at node

Comments

- This is a plain text (ASCII) file.
- Text shown in the file format above without quotes (“ ”) is used directly in the output file. Text shown above in quotes represents a variable, or number written to the file. These variables are defined above on the right.
- Descriptions may be included in the output file to the right of line of header data.
- The first section represents solution controls used to generate the data file.
- The second section describes the data in the third section. If any of the acoustic points are not found within the solution domain, a warning is written before the number of acoustic lines is written. The number of intersections is listed for each acoustic line in the input file.
- The data is written out so that each column represents a node or intersection within the domain. The first three rows give the (x,y,z) coordinate of the node or intersection. The rows under the dashed divider line are the dimensionless w-velocity at the node or intersection point at the solution time designated in the first column.
- w-Velocity data is presented in dimensionless form:

$$ww\# = \frac{w}{U_\infty}$$

- A double vertical line breaks the acoustic point and acoustic line data. Subsequent single vertical lines break data from acoustic lines.
- The intersections along a particular acoustic line are ordered parametrically along the length of the acoustic line, from starting to end points, as listed in the input file.

Sample File

Controls:

```

dt = 0.00100000 Dimensionless time step
Lref = 1.00000000 Reference dimension
Uinf = 781.20000000 Freestream velocity
ainf = 1116.00000000 Freestream acoustic speed
mach = 0.70000000 Freestream Mach number
gam = 1.40000000 Ratio of specific heats
  
```

Data Layout:

```

nacp = 2 Number of acoustic points
nacl = 2 Number of acoustic lines
  
```

```

-- 2 intersections with Line # 1
-- 1 intersections with Line # 2
  
```

	x-coord ==>	0.00000000	0.20000000	0.00000000	0.00000000	1.00000000	1.00000000	0.70000000
Y-coord ==>	0.00000000	0.70000000	0.40000000	0.40000000	1.30000000	1.30000000	0.60000000	0.60000000
Z-coord ==>	0.00000000	0.10000000	0.20000000	0.20000000	0.20000000	0.20000000	0.50000000	0.50000000
Time	0.00100000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
	0.00200000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
	0.00300000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000

APPENDIX F

NS3D FILE FORMATS

The file formats included in this section are used to create the input files required to operate NS3D and interpret the files that are written by NS3D. In order to make this appendix more compact, file formats that are already outlined in Appendices C, D, and E are not repeated here. If additional terms or options are added to model viscous or turbulent effects, the file formats are called out in full detail here to show the differences.

NS3D

Summary of File Formats

Input Files:

- `case.g3d` (required, if `inetcdf = .false.`) contains geometry data structures representing the computational mesh as required by the flow solver (binary)
- `case.nc3d` (required, if `inetcdf = .true.`) contains geometry data structures representing the computational mesh as required by the flow solver (netCDF)
- `case.con` (required) contains values for the solver control parameters and flow conditions (ASCII)
- `case.unk` (optional) contains the nodal values of the primitive flow variables (density, velocity, pressure, and total enthalpy) for each node of the computational mesh to be used as the initial conditions for the flow solution (binary or netCDF)
- `case.dyn` (optional) contains the non-inertial matrices and initial conditions as required for a non-inertial solution (ASCII) [see Appendix E]
- `case.vec` (optional) contains the elastic mode matrices, initial conditions, and mode shape vectors for the solid wall surfaces as required for an aeroelastic solution (ASCII) [see Appendix E]
- `case.frc` (optional) contains external forces to be applied to each solution step in a dynamic or aeroelastic solution (ASCII) [see Appendix E]
- `case.cmb` (optional) contains mass and enthalpy generation data used by the quasi-combustion model (ASCII) [see Appendix C]
- `case.eng` (optional) contains rocket and turbojet engine (boundary condition) data (ASCII) [see Appendix C]
- `case.acst` (optional) contains the acoustic output data (ASCII) [see Appendix E]
- `case.tbc` (optional) contains temperature boundary condition data (ASCII) [see Appendix D]

Output Files:

- `case.un#` contains the values of the primitive flow variables (density, velocity, pressure, and total enthalpy) and the turbulent flow variables (turbulent viscosity, turbulent kinetic energy, turbulent dissipation, and model variables) for each node of the computational mesh; # is iterated as more files are produced so the progress of the solution can be followed (Binary)
- `caset.un#` contains the values of the turbulent flow variables (turbulent viscosity, turbulent kinetic energy, turbulent dissipation, and model variables) for each node of the computational mesh; # is iterated as more files are produced so the progress of the solution can be followed (Binary)
- `case.rsd` contains a history of the solution residuals for the conservation variables (density, momentum, and total energy) (ASCII)
- `case.rsd2` contains a history of the solution residuals for the conservation variables for each inner cycle (ASCII)
- `case.cyc` contains a history of the number of inner cycles used to converge each iteration (ASCII) [see Appendix C]
- `case.time` contains a history of the local time step ratios (ASCII)
- `case.lfs` contains a history of the dimensionless aerodynamic forces acting on the solid walls of the geometry (ASCII) [see Appendix E]
- `xd.dat` contains a history of the non-inertial displacements, velocities, and accelerations for a dynamic solution (ASCII) [see Appendix E]
- `xn.dat` contains a history of the generalized displacements, velocities, and forces for an unsteady, aeroelastic solution (ASCII) [see Appendix E]
- `case.rst` and `case.rs2` contain information on up to two sets of unknowns data, elastic system data, and dynamic motion data (binary)
- `case.pac` contains a history of pressure data at prescribed nodes (ASCII) [see Appendix E]
- `case.rac` contains a history of density data at prescribed nodes (ASCII) [see Appendix E]
- `case.uac` contains a history of u-velocity data at prescribed nodes (ASCII) [see Appendix E]
- `case.vac` contains a history of v-velocity data at prescribed nodes (ASCII) [see Appendix E]

- `case.wac` contains a history of w-velocity data at prescribed nodes (ASCII) [see Appendix E]

Geometry Input File (case.g3d)

Basic File Format

```

nnd nel nsg nbe nbp nwl nsd nsf
  nwlv nsdv
(LBE(i), i = 1, 8)
(COOR(i,j), j = 1,3) (i = 1,nnd)
(IELM(i,j), j = 1,4) (i = 1,nel)
(ISEG(i,j), j = 1,2) (i = 1,nsg)
(IBEL(i,j), j = 1,5) (i = 1,nbe)

```

Definition of Terms

nnd: int number of nodes
nel: int number of elements
nsg: int number of segments
nbe: int number of boundary elements
nbp: int number of boundary points
nwl: int number of wall nodes
nsd: int number of singular nodes
nsf: int number of boundary surfaces
nwlv: int number of viscous wall nodes
nsdv: int number of viscous singular nodes

LBE(i): int start/ stop index for 4 BC types

COOR(i,1): real x-coordinate for node i
COOR(i,2): real y-coordinate for node i
COOR(i,3): real z-coordinate for node i

IELM(i,1): int node 1 for element i
IELM(i,2): int node 2 for element i
IELM(i,3): int node 3 for element i
IELM(i,4): int node 4 for element i

ISEG(i,1): int node 1 for segment i
ISEG(i,2): int node 2 for segment i

IBEL(i,1): int node 1 for boundary elem. i
IBEL(i,2): int node 2 for boundary elem. i
IBEL(i,3): int node 3 for boundary elem. i
IBEL(i,4): int surface index in case.sur
IBEL(i,5): int domain elem. associated with
boundary elem. i

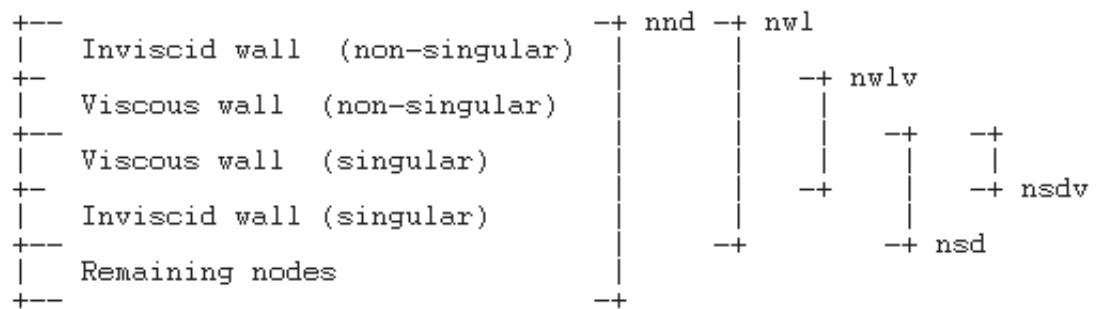
Comments

- This is an unformatted (binary) file.
- The nodal coordinates in this file are treated as dimensional values and are non-dimensionalized using the reference dimension `refdim` specified in the control file.
- The element connectivity data `IELM` and `IBEL` define positive element volumes \mathcal{V}_e and boundary normal vectors \hat{n} pointed into the solution domain:

$$6\mathcal{V}_e = \begin{vmatrix} x_{14} & y_{14} & z_{14} \\ x_{24} & y_{24} & z_{24} \\ x_{34} & y_{34} & z_{34} \end{vmatrix} > 0 \quad \hat{n} = \frac{\vec{x}_{21} \times \vec{x}_{31}}{|\vec{x}_{21} \times \vec{x}_{31}|} \quad \text{where} \quad \vec{x}_{ij} = \vec{x}_i - \vec{x}_j$$

- The program `makeg3d` is used to convert a standard STARS surface triangulation file `case.fro`, mesh file `case.gri`, and modified boundary conditions file `case.bco` into an appropriately sorted three-dimensional geometry file.
- Nodal data `COOR` is sorted such that the first `nwl` nodes are defined as solid wall nodes. Out of the first `nwl` nodes, the last `nsd` nodes are defined as singular nodes. The viscous nodes are placed in the middle (`nwlv` and `nsdv`), according to the following diagram:

Nodes:



- Boundary element data is sorted based on the starting/stopping indexes for the three BC types, i.e. boundary elements `LBE(1)` through `LBE(2)` are solid wall elements, `LBE(3)` through `LBE(4)` are symmetry elements, `LBE(5)` through `LBE(6)` are far-field elements, and `LBE(7)` through `LBE(8)` are viscous solid wall elements, where the two solid wall elements are restricted to `LBE(1) ≤ LBE(7) < LBE(8) ≤ LBE(2)`. (In other words, the viscous solid walls must exist within the limits of the viscous walls.)

Boundary elements:

+-		+- nbe	+-	
	Inviscid walls			LBE(1)
+-				
	Viscous walls			LBE(2)
+-			+-	LBE(7)
	Symmetry planes			LBE(8)
+-			+-	
	Far fields			LBE(3)
+-				LBE(4)
	Engine inlets		+-	LBE(5)
+-				LBE(6)
	Engine/rocket outlets		+-	
+-				
	Internal "boundaries"			
+-		+-		

Geometry Input File, netCDF Format (case.nc3d)

Basic File Format

File Attributes:

```
"title" :: "NS3D Geometry File"
"TimeDay" :: TimeDay
"name" :: filen
"Version" :: VerYMD
```

Dimensions:

```
"nnd" :: nnd (IDnnd)
"nel" :: nel (IDnel)
"nsg" :: nsg (IDnsg)
"nbe" :: nbe (IDnbe)
"nbp" :: nbp (IDnbp)
"nsf" :: nsf (IDnsf)
"ncv" :: ncv (IDncv)
"mxs" :: mxs (IDmxs)
"nLBE" :: 8 (nLBE)
"Dim2" :: 2 (ID2)
"Dim3" :: 3 (ID3)
"Dim4" :: 4 (ID4)
"Dim5" :: 5 (ID5)
```

Variables:

```
"nwl" :: nwl (IDnwl) [ ]
"nsd" :: nsd (IDnsd) [ ]
"nwlv" :: nwlv (IDnwlv) [ ]
"nsdv" :: nsdv (IDnsdv) [ ]
"LBE" :: LBE (IDLBE) [nLBE]
"COOR" :: COOR (IDcoor) [IDnnd, ID3]
"IELM" :: IELM (IDelm) [IDnel, ID4]
"ISEG" :: ISEG (IDseg) [IDnsg, ID2]
"IBEL" :: IBEL (IDbel) [IDnbe, ID5]
```

Definition of Terms

```
TimeDay: int time and date generated
          filen: char case name
VerYMD: int MakeNC3D version, written
          using YYYYMMDD notation
```

```
nnd: int number of nodes
nel: int number of elements
nsg: int number of segments
nbe: int number of boundary elements
nbp: int number of boundary points
nsf: int number of surfaces in front file
ncv: int number of curves in front file
mxs: int maximum dimension (= 2 nel)
```

```
nLBE: int = 8, used to dimension LBE
Dim2: int = 2, used to dimension arrays
Dim3: int = 3, used to dimension arrays
Dim4: int = 4, used to dimension arrays
Dim5: int = 5, used to dimension arrays
```

```
nwl: int number of wall nodes
nsd: int number of singular nodes
nwlv: int number of viscous wall nodes
nsdv: int number of viscous singular nodes
```

```
LBE(i): int start/ stop index for 4 BC types
```

```
COOR(i,1): real x-coordinate for node i
COOR(i,2): real y-coordinate for node i
COOR(i,3): real z-coordinate for node i
```

```
IELM(i,1): int node 1 for element i
IELM(i,2): int node 2 for element i
IELM(i,3): int node 3 for element i
IELM(i,4): int node 4 for element i
```

```
ISEG(i,1): int node 1 for segment i
ISEG(i,2): int node 2 for segment i
```

```
IBEL(i,1): int node 1 for boundary elem. i
IBEL(i,2): int node 2 for boundary elem. i
IBEL(i,3): int node 3 for boundary elem. i
IBEL(i,4): int surface index in case.sur
IBEL(i,5): int domain elem. associated with
          boundary elem. i
```

Comments

- This file is created using the netCDF library. Formatting is handled using the netCDF library file `netCDF.dll`. The `case.nc3d` file contains the same base information in the `case.g3d` file.
- The netCDF formatting has been represented here using four designations:
 - Names in quotes (" ") represent the human-name of the variable or array
 - The value following the double-colon (::) is the variable stored under this name.
 - The name in parentheses () is the handle used to recall information from netCDF.
 - The values in brackets [] are the array dimensions. The empty brackets are shown for scalar variables. Single-dimension arrays (vectors) are shown as a single value between the brackets. Multi-dimension arrays (matrices, etc.) are shown by values separated by commas.
- Data in `case.nc3d` can be written or read in any particular order, but for simplicity, the file has been represented here in three sections:
 - File Attributes: Values describing the file (name, date, version, etc.) are not read by Euler3D.
 - Dimensions: Values used to size the arrays (variables) that follow are read using `nf_inq_dimid` and `nf_inq_dimlen`. The first function inquires of the netCDF ID to identify the dimension, and the second is used to read the value. The file structure has been established so that all of the dimensions are positive-definite (no zeros or negatives). For example, the number of nodes (`nnd`) are read:

```
call check(nf_inq_dimid(ncid, "nnd", IDnnd))
call check(nf_inq_dimlen(ncid, IDnnd, nnd))
```
 - Variables: Scalars and arrays are read using `nf_inq_varid`, `nf_get_var_int`, `nf_put_var_double`, and `nf_get_var1_int`. The first function inquires of the netCDF ID to identify the variable. The second and third read an entire integer and double-precision real array, respectively. The final function is used to read an array one part at a time, used for LBE and IBEL. For example, the number of wall nodes (`nwl`, scalar), nodal coordinates (`COOR`, double, array), and element connectivity (`IELM`, integer, array) are read:

```
call check(nf_inq_varid(ncid, "nwl", IDnwl))
call check(nf_get_var_int(ncid, IDnwl, nwl))

call check(nf_inq_varid(ncid, "COOR", IDcoor))
call check(nf_get_var_double(ncid, IDcoor, COOR))

call check(nf_inq_varid(ncid, "IELM", IDelm))
call check(nf_get_var_int(ncid, IDelm, IELM))
```
 - The functions used to read and write using the netCDF format can be found in the netCDF manuals (www.unidata.ucar.edu/software/netcdf/).
- Nodal data `COOR` is sorted such that the first `nwl` nodes are defined as solid wall nodes. Out of the first `nwl` nodes, the last `nsd` nodes are defined as singular nodes.
- The nodal coordinates in this file are treated as dimensional values and are non-dimensionalized using the reference dimension `refdim` specified in the control file.

- The element connectivity data IELM and IBEL define positive element volumes V_e and boundary normal vectors \hat{n} pointed into the solution domain:

$$6V_e = \begin{vmatrix} x_{14} & y_{14} & z_{14} \\ x_{24} & y_{24} & z_{24} \\ x_{34} & y_{34} & z_{34} \end{vmatrix} > 0 \quad \hat{n} = \frac{\vec{x}_{21} \times \vec{x}_{31}}{|\vec{x}_{21} \times \vec{x}_{31}|} \quad \text{where} \quad \vec{x}_{ij} = \vec{x}_i - \vec{x}_j$$

- Boundary element data is sorted based on the starting/stopping indexes for the three BC types, i.e. boundary elements $\text{LBE}(1)$ through $\text{LBE}(2)$ are solid wall elements, $\text{LBE}(3)$ through $\text{LBE}(4)$ are symmetry elements, $\text{LBE}(5)$ through $\text{LBE}(6)$ are far-field elements, and $\text{LBE}(7)$ through $\text{LBE}(8)$ are viscous wall elements and overlap the solid wall elements so that $\text{LBE}(1) \leq \text{LBE}(7) < \text{LBE}(8) \leq \text{LBE}(2)$.
- The program `makenc3d` is used to convert a standard STARS surface triangulation file `case.fro`, mesh file `case.gri`, and modified boundary conditions file `case.bco` into an appropriately sorted three-dimensional geometry file in netCDF format.

Solver Control Input File (case . con)

Basic File Format

```

&control

      dt          = 0.1d0,
      gamma       = 1.4d0,
      diss        = 1.0d0,
      cfl         = 0.5d0,
      lamb        = -0.666d0,
      Smod        = 0.0d0,
      dislen      = 1.0d-20,

      mach        = 0.6d0,
      Re          = 2.0d5,
      Pr          = 0.7d0,
      alpha       = 0.0d0,
      beta        = 0.0d0,
      refdim      = 1.0d0,

      iturb       = 0,
      turbI       = 0.01d0,
      muTinf      = 0.01d0,
      rnuinf      = 0.0d0,
      rhoKinf     = 0.0d0,
      rhoWinf     = 0.0d0,
      PrT         = 0.9d0,
      disst       = 1.0d0,

      nstp        = 100,
      nout        = 50,
      ncyc        = 4,

      rsdtol      = 1.0d-20,
      rsdmax      = 10.0d0,

      isol        = 0,
      idiss       = 0,
      ndiss       = 1,
      idsol       = 2,
      icomb       = 0,
      itime       = 0,
      iquasi2D    = 0,

      isize       = 5,
      displ       = 0.1d0,
      omega       = 0.2d0,
      ratio       = 600.0d0,
  
```

Definition of Terms

dt:	real	dimensionless global time step
gamma:	real	ratio of specific heats
diss:	real	dissipation factor
cfl:	real	local time step stability factor
lamb:	real	ratio of 2 nd over 1 st viscosity (λ/μ)
Smod:	real	modified Sutherland's constant
dislen:	real	distance from wall where no artificial dissipation is used
mach:	real	free-stream Mach number
Re:	real	free-stream Reynolds number
Pr:	real	Prandtl number
alpha:	real	free-stream angle of attack (deg)
beta:	real	side slip angle (deg)
refdim:	real	reference length (dim'l)
iturb:	int	turbulence model flag
turbI:	real	freestream turbulence intensity
muTinf:	real	freestream eddy viscosity
rnuinf:	real	turbulence IC and FF BC for $\rho\nu$
rhoKinf:	real	turbulence IC and FF BC for ρK
rhoWinf:	real	turbulence IC and FF BC for $\rho\omega$
PrT:	real	turbulent Prandtl number
disst:	real	turb dissipation factor
nstp:	int	total solution steps
nout:	int	output frequency, steps/output
ncyc:	int	iterative cycles per solution step
rsdtol:	real	energy residual converg tolerance
rsdmax:	real	energy residual divergence criteria
isol:	int	CFD solution type
idiss:	int	dissipation type
ndiss:	int	# of inner cycles / diss. calc.
idsol:	int	order of elastic forces integration
icomb:	int	combustion model type
itime:	int	viscous local time stepping type
iquasi2D:	int	directionality
isize:	int	width of multisteps
displ:	real	max displacement of forced mode
omega:	real	frequency scalar
Ratio:	real	chirp envelop length

<pre> istrt = .false., iaero = .true., idynm = .false., ielast = .false., iprop = .false., ifree = .true., iforce = .false., isafe = .false., irsds = .false., inetcdf = .false., iacoust = .false., itempbc = .false., nr = 0, ainf = 1.0d0, rhoINF = 1.0d0, gravity = 0.0d0, / </pre>	<pre> istrt: log restart flag iaero: log aerodynamic forces flag idynm: log dynamic/non-inertial flag ielast: log elastic flag iprop: log propulsion flag ifree: log free-stream velocity flag iforce: log external forces flag isafe: log safe-mode flag irsds: log residual watching flag inetcdf: log NetCDF input / output flag iacoust: log acoustics output flag itempbc: log temperature BC flag nr: int number of elastic modes ainf: real free-stream sonic speed (dim'l) rhoINF: real free-stream density (dim'l) gravity: real gravity (dim'l) </pre>
---	--

Comments

- This is a plain text (ASCII) file formatted as a Fortran namelist.
 - The namelist must begin with the line “&control” and end with the line “/”.
 - The remaining lines can be listed in any order or omitted, if desired.
 - The intermediate lines work like variable assignments with the loose format: *variable_name = value*, followed by a comma.
 - Integers (*int*) are listed as whole numbers.
 - Real numbers (*real*) are listed in double precision, scientific notation: *###d±###*.
 - Logical variables (*log*) are listed as either “.true.” or “.false.”.
 - Lines can be commented out by inserting an exclamation point “!” prior to any other information on the line.
 - The default values, shown above, are used for variables omitted or commented out of the namelist.
 - Any information listed after the last line of the namelist “/” are not read by the program and can be used to store notes and other calculations.
- The global time step Δt is only used for unsteady solutions. Δt is a dimensionless value calculated: $\Delta t = \Delta t U / L$, where Δt is the dimensional time step, U is the free-stream velocity ($= \text{mach } a_{\text{INF}}$), and L is the reference length refdim .
- Appropriate values for the dissipation factor are in the range $0.0 < \text{diss} \leq 2.0$. Some dissipation is required to stabilize the solution, but too much dissipation will corrupt the solution and possibly be a destabilizing influence.
- The local time step stability factor is a safety factor used to compute local time steps for each solution step. For steady solutions, a stability factor of 0.8 is typical for most problems. For unsteady solutions, the stability factor is typically $0.3 \leq \text{cfl} \leq 0.8$.
- The coefficient λ is the ratio of second to first viscosity μ/m , which is considered constant across the domain. λ is restricted to be greater than or equal to $-2/3$ to maintain positive viscous dissipation in the energy equation (Stoke’s hypothesis).

- The coefficient s_{mod} is the modified Sutherland's coefficient: $s_{\text{mod}} = c_p S / U_{\text{inf}}^2$, where c_p is the specific heat at constant pressure, S is Sutherland's constant, and U_{inf} is the freestream velocity ($= \text{mach} \ a_{\text{inf}}$). If s_{mod} is set to zero, the viscosity is constant, equal to the freestream viscosity throughout the domain. If $s_{\text{mod}} > 0$, Sutherland's equation is used to vary viscosity with temperature (enthalpy).
- The values of refdim , mach , ainf , and rhoinf are used to non-dimensionalize all values read into the flow solver.
- Prandtl number Pr is used to relate viscosity and thermal conductivity k . The freestream Reynolds number Re is used to relate the density and velocity in the freestream to the viscosity in the freestream, using L as refdim :

$$\text{Pr} = \frac{c_p \mu}{k} \quad \text{Re} = \frac{\rho_{\infty} U_{\infty} L}{\mu_{\infty}} \quad U_{\infty} = M_{\infty} a_{\infty}$$

- The free-stream orientation angles are ignored for dynamic (non-inertial) problems.
- Turbulence is represented by the addition of differential equations. The order and accuracy differs by turbulence model:
 - $\text{iturb} = 0$, no turbulence model (N-S only)
 - $\text{iturb} = 1$, Spalart-Allmarus model (one-equation; good for streamlined bodies)
 - $\text{iturb} = 2$, Menter's SST model (two-equation; good for more complex flows)
- Initial conditions, far field (freestream), and lowest allowable values for turbulence are set through rnuinf , rhoKinf , rhoWinf , turbI , and muTinf . The first three variables are used to set turbulent quantities directly, while the other two variables can be used to back-calculate the turbulent quantities from more user-friendly values:
 - For the SA model ($\text{iturb} = 1$), rnuinf represents the SA variable $\rho\nu$, which is used to directly calculate the local eddy viscosity. The amount of turbulence in the freestream affects the laminar performance and growth rate of turbulence along the wall. $\text{rnuinf} = 10^{-4}$ is suggested to create "natural transition", and $\text{rnuinf} = 3$ is suggested to create a wholly turbulent solution. If $\text{rnuinf} \leq 0$, then $\rho\nu$ is calculated from the eddy viscosity muTinf , using:

$$\mu_{T,\infty}^* = f_{\nu 1} \rho \hat{\nu}_{\infty}^* \quad \text{or} \quad f_{\nu 1} = \frac{(\rho \hat{\nu}_{\infty}^*)^3}{(\rho \hat{\nu}_{\infty}^*)^3 + 7.1^3}$$

- For the SST model ($\text{iturb} = 2$), rhoKinf represents the turbulent kinetic energy ρK , and rhoWinf represents the rate of dissipation of turbulent kinetic energy $\rho \omega$. The amount of turbulence in the freestream affects the laminar performance and growth rate of turbulence along the wall. If $\text{rhoKinf} \leq 0$, then ρK is calculated from turbulent intensity turbI , shown as T' :

$$T'_{\infty} = \sqrt{\frac{2}{3} \frac{K_{\infty}}{U_{\infty}^2}} \quad \text{or} \quad \rho K_{\infty}^* = 1.5 \rho_{\infty}^* (T'_{\infty})^2$$

If $\text{rhoWinf} \leq 0$, then $\rho \omega$ is calculated from eddy viscosity muTinf , shown as μ_T :

$$\mu_{T,\infty}^* = \rho_\infty^* \frac{\rho K_\infty^*}{\rho \omega_\infty^*} Re_L \quad \text{or} \quad \rho \omega_\infty^* = \rho_\infty^* \frac{\rho K_\infty^*}{\mu_{T,\infty}^*} Re_L$$

`muTinf` and `turbI` are more user-friendly and therefore the suggested means of specifying ICs for the SST model. Turbulent intensity has a physical meaning:

- For internal flows, `turbI` is 1% to 5%.
- For external (quiescent) flows and high-quality wind tunnel, `turbI` < 1%.

Freestream eddy viscosity `muTinf` represents additional diffusion throughout the solution. To minimize additional diffusion with a non-zero value:

$$\text{muTinf} = \mu_{T,\infty}^* = \frac{\mu_T}{\mu_\infty} \approx 1\%$$

- The turbulent Prandtl number `PrT` is used to model the transport of heat through turbulence, using Reynolds analogy. The turbulent Prandtl number for air is 0.9.
- Artificial dissipation is added to each turbulence model using the same algorithms applied to the N.S. equations. The N.S. dissipation factor has been combined with another scalar to allow the user to adapt the dissipation used in the turbulence models. The artificial dissipation in the turbulence models is scaled by (`disst*dis`).
- The number of iterative cycles should be set to 4 for steady solutions. For unsteady solutions, use a sufficient number of cycles to allow for an appropriate level of convergence at each step. A sufficient number can be estimated as $N = dt / \Delta t_{loc,min}$.
 - The following is a good practice for finding a sufficient number of iterations for unsteady solutions:
 1. Select initial values for `dt`, `ncyc`, and `rsdtol`.
 2. Step the solution forward 20-50 iterations.
 3. Check the `*.cyc` file for the number of cycles required per iteration. The number of cycles should level off after 10 iterations. If not, run enough iterations for the number of required cycles to level off.
 4. If the last 10 iterations require more than 20 cycles, lower the time step.
 5. If the last 10 iterations require less than 8 iterations, increase the time step.
 6. The sweet spot is 10-15 iterations.
 - The residual tolerance can be used to exit the iterative cycles if the energy residual meets a specified criteria `rsdtol`. (The energy residual is used because the other residuals normally converge faster than energy.) This feature can be used to set the number of iterations to a very large number with a residual tolerance `rsdtol`. When the residual drops below the tolerance, the solution will progress to the next time step. Lower `rsdtol` values require more iterations.
 - The divergence tolerance `rsdmax` creates an upper tolerance on the energy residual. If the solution is diverging, the energy residual will grow larger than `rsdmax` and terminate the run. The solution also terminates if the residuals become NAN or INFINITY. Larger `rsdmax` values will allow the solution to diverge further and ensure that the solution is in fact diverging.
- There are four available CFD solution types defined as follows:
 - `isol = 0` is a steady solution (not time accurate)
 - `isol = 1` is a first-order unsteady solution
 - `isol = 2` is a second-order unsteady solution

- `isol = 3` is a supersonic piston perturbation solution
- There are three available options for viscous local time stepping:
 - `itime = -1` uses the minimum distance across each element (algebraic)
 - `itime = 0` uses diagonals of the stiffness matrix (heat transfer only)
 - `itime = 1` uses diagonals of the stiffness matrices (momentum & heat transfer)
 - `itime = 2` uses diagonals of the stiffness matrices (mom, heat trans & turb model)
- Euler3D uses various orders of numerical integration, specified by `ipnt`. NS3D only uses single point (or first order) Gauss quadrature (`ipnt = 1`), which is hard-coded into the controls. If a file from Euler3D is used to create the NS3D file `case.con`, then the integration order `ipnt` must be removed or commented out (“!”).
- There are three available dissipation types defined as follows:
 - `idiss = -1` is no artificial dissipation (only viscous dissipation)
 - `idiss = 0` is a lower order dissipation
 - `idiss = 1` is a high order dissipation with gradient limiters
- The lower order dissipation is typically overly diffuse and should be used in conjunction with low values of the dissipation factor. Low-order dissipation works best for problems without strong vortices and for supersonic/hypersonic flows.
- The higher order dissipation is more CPU intensive than the low-order dissipation and less stable. Larger values for the dissipation factor are typically required for stabilization. The high-order dissipation works best for subsonic to transonic flows with strong gradients or vortices. Rotating domains will typically require high-order dissipation to resolve the circulating pattern of the relative flow velocities.
- `dislen` is the distance, near walls, where no artificial dissipation is used. The artificial dissipation model is scaled by $f(d)$, where d is the distance to the nearest wall:

$$f(d) = \begin{cases} 0 & \text{if } d \leq \text{dislen} \\ 1 & \text{if } d \geq 2 \text{dislen} \\ \frac{1}{2} \left(1 - \cos\left(\pi \left(\frac{d}{\text{dislen}} - 1 \right)\right) \right) & \text{otherwise} \end{cases}$$

- Combustion properties are specified in the `case.cmb` file. The mass and heat generation are distributed throughout the domain using the following distributions:
 - `icomb = 0`, no combustion (`case.cmb` not read)
 - `icomb = 1`, combustion properties are piece-wise linear (specified at the nodes)
 - `icomb = 2`, combustion properties are constant (specified) on the elements
- The values of `ndiss` controls the number of iterations between dissipation calculations. For example, if `ndiss = 1`, the dissipation is recalculated at every inner cycle.; if `ndiss = 2`, the dissipation is calculated ever other inner cycle, and stored in between; and, etc. `ndiss` can only be used to control the higher-order dissipation model (`idiss = 1`).
- There are three available elastic solution types defined as follows:
 - `idsol = 0` is a zeroth-order integration for applied forces
 - `idsol = 1` is a first-order integration for applied forces
 - `idsol = 2` is a second-order integration for applied forces
- There are four available directionality conditions as follows:
 - `iquasi2D = 0` is fully three-dimensional

- `iquasi2D = 1` is quasi two-dimensional (yz only)
- `iquasi2D = 2` is quasi two-dimensional (xz only)
- `iquasi2D = 3` is quasi two-dimensional (xy only)
- Four IBXN controls are listed in the `case.con` file: `isize`, `displ`, `omega`, and `ratio`. These controls are only necessary if `ielast` is turned on and `IBXN = 3` to `9`.
- When the restart flag `istrt` is set to `.true.`, the solver will read one set of solution unknowns from the `case.unk` file and apply this set of unknowns as the initial conditions for the new iterative solution.
- A restarted solution assumes that the time gradient of the initial state is zero, i.e. the solution stored in the `case.unk` file is a converged, steady state solution. This has a significant impact on the second-order unsteady solution since it relies on two sets of solution unknowns for advancement to the next time step, i.e. a second-order unsteady solution should not be restarted from the last time step of a similar unsteady solution that was stopped because both sets of unsteady data from the last solution step are not available for accurate evaluation of the time gradients in the flow.
- If the aerodynamics flag `iaero` is set to `.true.`, the aerodynamic forces are applied to the dynamic and elastic motion. If the flag is set to `.false.`, the dynamic and elastic motion must be forced externally or occur as free-response vibrations.
- The non-inertial dynamics routine is turned on when `idynm` is set to `.true.`. NS3D will then read in the `case.dyn` file for dynamic inputs and write out dynamic motion to the `xd.dat`.
 - If the free-stream velocity flag `ifree` is set to `.false.`, the free-stream velocity is set to zero, and relative flow velocities must be generated through dynamic rotation or translation of the non-inertial coordinate system.
 - If `ifree = .true.`, the freestream starts aligned with the global fixed x-direction (i.e., $\alpha = \beta = 0$) but can be rotated using the initial orientation of the body in `case.dyn`.
- The elastic deflection routine is turned on when `ielast` is set to `.true.`. NS3D will then read in the `case.vec` file for modal elastic inputs and write out modal deformations and forces to the `xn.dat`.
- For steady solutions (`isol = 0`), the dynamics flags for each degree of freedom in the `case.dyn` and `case.vec` should be set to 1 (clamped condition).
- The propulsion boundary conditions are turned on when `iprop` is set to `.true.`. NS3D will read in the `case.eng` file for rocket and engine inputs.
- If the external forces flag is set to `.true.`, the solver will read the user defined external force vector for each global time step from the input file `case.frc`. If the solver reaches the end of the input file before completing the solution, the last force vector in the file carries over to each of the remaining time steps if it was non-zero.
- If the safe-mode flag is set to `.true.`, NS3D writes two files per step that are used to restart the solution: `case.rst` and `case.rs2`. Two files are used so while one file is being over-written, the other file is still preserved. Each file stores the previous two values of all unknowns, elastic mode shapes, and generalized elastic forces. Safe-mode can be used for fast restarts for very long runs that are not time sensitive.
- When the safe-mode flag is set to `.true.`, NS3D checks for both restart files. If the `case.rst` exists, but has an error, the error is reported to the user. The `case.rst` can be moved, renamed, or deleted. The solution will then be restarted from the

`case.rs2` file. (NS3D does not skip between files to avoid overwriting files that contain correctable errors.)

- If the residual watching flag is set to `.true.`, residuals are written out at each inner iteration to the `case.rs2` file. This option can be used to check the residual convergence within steps. The number of inner cycles used at each iteration is written to the `case.cyc` file for plotting and comparison of convergence.
- If the acoustic output flag is set to `.true.`, the acoustic input file `case.acst` is read for controls, and one or more of the acoustic output files (pressure – `case.pac`; density – `case.rac`; u-velocity – `case.uac`; v-velocity – `case.vac`; w-velocity – `case.wac`) are written.
- If the NetCDF flag is set to `.true.`, then the geometry and unknowns information is passed through NetCDF formats instead of the traditional binary files. The geometry is read in through the `case.nc3d` instead of the `case.g3d`. The unknowns files (`case.unk` and `case.un#`) retain the same name, but the format is the NetCDF format instead of the traditional binary format.
- If the temperature boundary conditions flag is set to `.true.`, then the temperature boundary conditions are read in through the `case.tbc` file. *The temperature boundary conditions have not yet been verified.*
- Gravity is assumed to act on the vehicle along the inertial z -axis. In the non-inertial reference frame, the body force vector rotates so that gravity is always pointed down in the positive z -direction. The value gravity is non-dimensionalized using `refdim` (L), `mach` (M), and `ainf`, so the dimensions of these variables should be consistent:

$$g^* = \frac{g L}{M^2 a_\infty^2}$$

Unknowns (Initial Conditions) Input File (`case.unk`)

Basic File Format

```
np gam xmi alp bet ref t Re
((UN(i,j), i = 1,nnd ), j = 1,6)
```

Definition of Terms

np:	int	number of nodes
gam:	real	ratio of specific heats
xmi:	real	free-stream Mach number
alp:	real	free-stream angle of attack
bet:	real	side slip angle
ref:	real	reference dimension
t:	real	dimensionless time
Re:	real	freestream Reynolds number
UN(i,1):	real	density for node <i>i</i>
UN(i,2):	real	x-velocity for node <i>i</i>
UN(i,3):	real	y-velocity for node <i>i</i>
UN(i,4):	real	z-velocity for node <i>i</i>
UN(i,5):	real	pressure for node <i>i</i>
UN(i,6):	real	total enthalpy for node <i>i</i>

Comments

- This is an unformatted (binary) file.
- The solution unknowns stored in this file are dimensionless quantities.
- For dynamic (non-inertial) problems, the solution unknowns stored in the file are relative quantities referenced to the body-fixed coordinate system. The fluid velocities (in the inertial frame) can be calculated by subtracting out the translational and rotational components of the body-fixed coordinate system.
- The quantities `nnd` must match the values in the geometry file `case.g3d` as `nnd`.
- The quantities `gam`, and `xmi`, and `Re` must match the values in the control file `case.con` as `gamma` and `mach`.
- When restarting a solution, the most recent unknowns output file `case.un#` can be renamed as the initial conditions file `case.unk`.

Geometry Input File, netCDF Format (case.unk)

Basic File Format

File Attributes:

```
"title"    :: "NS3D Unk File"
"time"     :: TimeDay
"name"     :: filen
"Version"  :: VerYMD
"gam"      :: gam
"xmi"      :: xmi
"alp"      :: nbe
"bet"      :: nbp
"refdim"   :: ref
"time"     :: t
"Re"       :: Re
"iturb"    :: iturb
```

Dimensions:

```
"nnd"      :: nnd   (IDnnd)
"Dim1"     :: 1     (ID1)
```

Variables:

```
[nsz] = [IDnnd, ID1]
"Density"  :: UN(:,1) (IDrho) [nsz]
"Xvelocity" :: UN(:,2) (IDxvel) [nsz]
"Yvelocity" :: UN(:,3) (IDyvel) [nsz]
"Zvelocity" :: UN(:,4) (IDzvel) [nsz]
"Pressure" :: UN(:,5) (IDp) [nsz]
"Eddy_Visc"  :: UNT(:,1) (IDmuT)
                                     [nsz]
"Turb_KE"    :: UNT(:,2) (IDtKE)
                                     [nsz]
"Diss_Turb_KE" :: UNT(:,3) (IDeps)
                                     [nsz]
"4th_Turb_Var" :: UNT(:,4) (IDvar4)
                                     [nsz]
```

Definition of Terms

```
TimeDay:  int   time and date generated
          filen: char case name
VerYMD:   int   NS3D version, written using
              YYYYMMDD notation
```

```
gam:  real  ratio of specific heats
xmi:  real  free-stream Mach number
alp:  real  free-stream angle of attack
bet:  real  side slip angle
ref:  real  reference dimension
t:    real  dimensionless time
Re:   real  freestream Reynolds number
iturb: int  turbulence model flag
```

```
nnd:  int   number of nodes
Dim1: int   = 1, used to dimension arrays
```

```
UN(i,1): real  density for node i
UN(i,2): real  x-velocity for node i
UN(i,3): real  y-velocity for node i
UN(i,4): real  z-velocity for node i
UN(i,5): real  pressure for node i
```

```
UNT(i,1): real  eddy viscosity for node i
UNT(i,2): real  turb kinetic energy for node i
UNT(i,3): real  dissipation of turbulent
                kinetic energy for node i
UNT(i,4): real  4th turb variable for node i
```

Comments

- This file is created using the netCDF library. Formatting is handled using the netCDF library file `netCDF.dll`.
- The netCDF formatting has been represented here using four designations:
 - Names in quotes (" ") represent the human-name of the variable or array
 - The value following the double-colon (::) is the variable stored under this name.
 - The name in parentheses () is the handle used to recall information from netCDF.
 - The values in brackets [] are the array dimensions. Multi-dimension arrays (matrices, etc.) are shown by values separated by commas.
- Data in `case.unk` can be written or read in any particular order, but for simplicity, the file has been represented here in three sections:

- File Attributes: Values describing the file (name, date, version, etc.) are read using `nf_get_att_double`. For example, the ratio of specific heats (`gam`) and mach number (`xmi`) are read:

```
call check(nf_get_att_double(ncid, nf_global, "gam", gam_unk))
call check(nf_get_att_double(ncid, nf_global, "xmi", xmi_unk))
```

- Dimensions: Values used to size the arrays (variables) that follow are read using `nf_inq_dimid` and `nf_inq_dimlen`. The first function inquires of the netCDF ID to identify the dimension, and the second is used to read the value. The file structure has been established so that all of the dimensions are positive-definite (no zeros or negatives). For example, the number of nodes (`nnd`) are read:

```
call check(nf_inq_dimid(ncid, "nnd", IDnnd))
call check(nf_inq_dimlen(ncid, IDnnd, nnd))
```

- Variables: The IDs for arrays are identified using `nf_inq_varid`, and then the arrays are read using `nf_get_var_double` (double-precision reals). For example, the nodal density (`UN(:,1)`, double, array) are read:

```
call check(nf_inq_varid(ncid, "Density", IDrho))
call check(nf_get_var_double(ncid, IDrho, UN(:,1)))
```

- The functions used to read and write using the netCDF format can be found in the netCDF manuals (www.unidata.ucar.edu/software/netcdf/).
- The solution unknowns stored in this file are dimensionless quantities.
- For dynamic (non-inertial) problems, the solution unknowns stored in the file are relative quantities referenced to the body-fixed coordinate system. The fluid velocities (in the inertial frame) can be calculated by subtracting out the translational and rotational components of the body-fixed coordinate system.
- The quantities `nnd` must match the values in the geometry file `case.nc3d` as `nnd`.
- The quantities `gam`, `xmi`, `Re` and `iturb` must match the values in the control file `case.con` as `gamma` and `mach`.
- When restarting a solution, the most recent unknowns output file `case.un#` can be renamed as the initial conditions file `case.unk`.

Unknowns Output File (case.un#)

Basic File Format

```
np gam xmi alp bet ref t Re
((UN(i,j), i = 1,nnd ), j = 1,6)
((UNT(i,j), i = 1,nnd ), j = 1,4)
```

Definition of Terms

np:	int	number of nodes
gam:	real	ratio of specific heats
xmi:	real	free-stream Mach number
alp:	real	free-stream angle of attack
bet:	real	side slip angle
ref:	real	reference dimension
t:	real	dimensionless time
Re:	real	freestream Reynolds number
UN(i,1):	real	density for node <i>i</i>
UN(i,2):	real	x-velocity for node <i>i</i>
UN(i,3):	real	y-velocity for node <i>i</i>
UN(i,4):	real	z-velocity for node <i>i</i>
UN(i,5):	real	pressure for node <i>i</i>
UN(I,6):	real	total enthalpy for node <i>i</i>
UNT(i,1):	real	eddy viscosity for node <i>i</i>
UNT(i,2):	real	turb kinetic energy for node <i>i</i>
UNT(i,3):	real	dissipation of turbulent kinetic energy for node <i>i</i>
UNT(i,4):	real	4 th turb variable for node <i>i</i>

Comments

- This is an unformatted (binary) file.
- The solution unknowns stored in this file are dimensionless quantities.
- For dynamic (non-inertial) problems, the solution unknowns stored in the file are relative quantities referenced to the body-fixed coordinate system. The fluid velocities (in the inertial frame) can be calculated by subtracting out the translational and rotational components of the body-fixed coordinate system.

Unknowns File, netCDF Format (case.un#)

Basic File Format

File Attributes:

```
"title"      :: "NS3D Un# File"
"time"       :: TimeDay
"name"       :: filen
"Version"    :: VerYMD
"gam"        :: gam
"xmi"        :: xmi
"alp"        :: alp
"bet"        :: bet
"refdim"     :: ref
"time"       :: t
"Re"         :: Re
"iturb"      :: iturb
```

Dimensions:

```
"nnd"        :: nnd   (IDnnd)
"Dim1"       :: 1     (ID1)
"Dim4"       :: 1     (ID4)
"Dim6"       :: 1     (ID6)
```

Variables:

```
[nsz] = [IDnnd, ID1]
"Density"    :: UN(:,1) (IDrho) [nsz]
"Xvelocity"  :: UN(:,2) (IDxvel) [nsz]
"Yvelocity"  :: UN(:,3) (IDyvel) [nsz]
"Zvelocity"  :: UN(:,4) (IDzvel) [nsz]
"Pressure"   :: UN(:,5) (IDp)    [nsz]
"Eddy_Visc"  :: UNT(:,1) (IDmuT)
                                     [nsz]
"Turb_KE"    :: UNT(:,2) (IDtKE)
                                     [nsz]
"Diss_Turb_KE" :: UNT(:,3) (IDeps)
                                     [nsz]
"4th_Turb_Var" :: UNT(:,4) (IDvar4)
                                     [nsz]
```

Definition of Terms

```
TimeDay:  int   time and date generated
          filen: char case name
VerYMD:   int   NS3D version, written using
              YYYYMMDD notation
```

```
gam:  real  ratio of specific heats
xmi:  real  free-stream Mach number
alp:  real  free-stream angle of attack
bet:  real  free-stream side slip angle
ref:  real  reference dimension
t:    real  dimensionless time
Re:   real  freestream Reynolds number
iturb: int  turbulence model flag
```

```
nnd:  int   number of nodes
Dim1:  int   = 1, used to dimension arrays
Dim4:  int   = 4, used to dimension arrays
Dim6:  int   = 6, used to dimension arrays
```

```
UN(i,1): real  density for node i
UN(i,2): real  x-velocity for node i
UN(i,3): real  y-velocity for node i
UN(i,4): real  z-velocity for node i
UN(i,5): real  pressure for node i
```

```
UNT(i,1): real  eddy viscosity for node i
UNT(i,2): real  turb kinetic energyfor node i
UNT(i,3): real  dissipation of turbulent
                kinetic energy for node i
UNT(i,4): real  4th turb variable for node i
```

Comments

- This file is created using the netCDF library. Formatting is handled using the netCDF library file `netCDF.dll`.
- The netCDF formatting has been represented here using four designations:
 - Names in quotes (" ") represent the human-name of the variable or array
 - The value following the double-colon (::) is the variable stored under this name.
 - The name in parentheses () is the handle used to recall information from netCDF.
 - The values in brackets [] are the array dimensions. Multi-dimension arrays (matrices, etc.) are shown by values separated by commas.
- Data in case.un# can be written or read in any particular order, but for simplicity, the file has been represented here in three sections:

- File Attributes: Values describing the file (name, date, version, etc.) are written using `nf_put_att_text`, `nf_put_att_int`, or `nf_put_att_double` if the value is a string, integer, or double-precision real, respectively. For example, the date (int), case name (char), and Mach number (xmi, double) are written:

```
call check(nf_put_att_int(ncid, nf_global, "time", nf_int, 8, Day ))
call check(nf_put_att_text(ncid, nf_global, "name",
                          len(trim(filen)), trim(filen) ))
call check(nf_put_att_double(ncid, nf_global, "xmi", nf_double,
                             1, xmi ))
```

- Dimensions: Values used to size the arrays (variables) that follow are written using `nf_def_dim`. The file structure has been established so that all of the dimensions are positive-definite (no zeros or negative numbers). For example, the number of nodes (nnd) is written:

```
call check(nf_def_dim(ncid, "nnd", nnd, IDnnd))
```

- Variables: Arrays are defined using `nf_def_var` and written using `nf_put_var_double` (double-precision reals). For example, the nodal density (UN(:,1), double, array) are written:

```
RankTwo = (/ IDnnd, ID1 /)
call check(nf_def_var(ncid, "Density", nf_double, 2, RankTwo,
                    IDrho))
call check(nf_put_var_double(ncid, IDrho, UN(:,1)))
```

- The functions used to read and write using the netCDF format can be found in the netCDF manuals (www.unidata.ucar.edu/software/netcdf/).
- The solution unknowns stored in this file are dimensionless quantities.
- For dynamic (non-inertial) problems, the solution unknowns stored in the file are relative quantities referenced to the body-fixed coordinate system. The fluid velocities (in the inertial frame) can be calculated by subtracting out the translational and rotational components of the body-fixed coordinate system.

Turbulent Unknowns Output File (caset.un#)

Basic File Format

```
np gam xmi alp bet ref t Re iturb  
( (UT(i,j), i = 1,nnd ), j = 1,6)
```

Definition of Terms

np:	int	number of nodes
gam:	real	ratio of specific heats
xmi:	real	free-stream Mach number
alp:	real	free-stream angle of attack
bet:	real	side-slip angle
ref:	real	reference dimension
t:	real	dimensionless time
Re:	real	free-stream Reynolds number
iturb:	int	turbulence model type
UT(i,1):	int	eddy viscosity for node <i>i</i>
UT(i,2):	real	turb. KE for node <i>i</i>
UT(i,3):	real	= 0.0
UT(i,4):	real	= 0.0
UT(i,5):	real	model variable for node <i>i</i> (in press. coeff. form, see below)
UT(i,6):	real	turb. diss. for node <i>i</i> (in total energy form, see below)

Comments

- This is an unformatted (binary) file.
- The solution unknowns stored in this file are dimensionless quantities.
- The turbulent variables UNT are converted to the UT variables shown in this file so that this file can be coupled with a caset.g3d file and plotted using GIPlot2D. (A caset.g3d can be created by copying the case.g3d file and adding a “t” to its name.) In this way, the four turbulence variables in UNT can be plotted:
 - Eddy viscosity $UNT(:, 1)$ can be seen in density plots.
 - Turbulent kinetic energy $UNT(:, 2)$ can be seen in velocity magnitude plots.
 - Turbulent dissipation $UNT(:, 3)$ can be seen in total energy plots.
 - Model variable $UNT(:, 4)$ can be seen in pressure coefficient plots.
- Total energy is calculated in GIPlot3D by subtracting $UT(:, 5)$ from $UT(:, 6)$.
- The pressure coefficient is calculated in GIPlot3D by subtracting the dimensionless freestream pressure ($= 1.0 / \text{gam} / \text{xmi}^2$) from $UT(:, 5)$ and scaling the result by 2.

Residuals Output File (`case.rsd`)

Basic File Format

```
1      (RSD(i), i = 1,7)
:
:
istp   (RSD(i), i = 1,7)
:
:
nstp   (RSD(i), i = 1,7)
```

Definition of Terms

istp:	int	current solution step
nstp:	int	total or last solution step
RSD(1):	real	density solution residual
RSD(2):	real	x-momentum solution residual
RSD(3):	real	y-momentum solution residual
RSD(4):	real	z-momentum solution residual
RSD(5):	real	energy solution residual
RSD(6):	real	1st turb. model eq. residual
RSD(7):	real	2nd turb. model eq. residual

Comments

- This is a plain text (ASCII) file.
- For steady problems, the solution residuals indicate the degree of convergence to the final steady state solution. All five solution residuals should converge to approximately the same order of magnitude.
- For unsteady problems, the solution residuals indicate the degree of convergence for each global step of the solution, or the degree of convergence for the steady solution that is solved at each step.
- RSD(5) and RSD(6) represent residuals for the equation(s) of the turbulence model:
 - If `iturb = 0`, RSD(5) and RSD(6) are omitted.
 - If `iturb = 1`, RSD(5) is written, representing the SA differential equation; RSD(6) is omitted.
 - If `iturb = 1`, RSD(5) and RSD(6) are both written, representing the SST differential equations.

Sample File

1	0.38320E-05	0.10743E-04	0.69854E-05	0.69854E-05	0.10598E-03
2	0.20317E-05	0.50694E-05	0.40436E-05	0.40436E-05	0.56307E-04
3	0.12024E-05	0.35187E-05	0.26241E-05	0.26241E-05	0.32195E-04
4	0.91334E-06	0.25166E-05	0.23637E-05	0.23637E-05	0.24240E-04
5	0.73183E-06	0.19442E-05	0.22228E-05	0.22228E-05	0.19376E-04
6	0.59870E-06	0.16179E-05	0.20889E-05	0.20889E-05	0.15963E-04
7	0.51663E-06	0.14311E-05	0.19719E-05	0.19719E-05	0.13946E-04
8	0.44924E-06	0.12989E-05	0.18536E-05	0.18536E-05	0.12398E-04
9	0.39510E-06	0.12095E-05	0.17283E-05	0.17283E-05	0.11156E-04
10	0.34726E-06	0.11478E-05	0.15878E-05	0.15878E-05	0.99450E-05
11	0.30775E-06	0.10746E-05	0.14329E-05	0.14329E-05	0.88159E-05
12	0.26207E-06	0.98700E-06	0.12833E-05	0.12833E-05	0.76280E-05
13	0.22418E-06	0.87924E-06	0.11245E-05	0.11245E-05	0.65113E-05
14	0.18904E-06	0.77764E-06	0.98148E-06	0.98148E-06	0.54617E-05
15	0.15809E-06	0.69345E-06	0.84471E-06	0.84471E-06	0.44739E-05
16	0.13411E-06	0.62203E-06	0.72991E-06	0.72991E-06	0.37422E-05
17	0.11564E-06	0.55717E-06	0.64350E-06	0.64350E-06	0.32661E-05
18	0.10516E-06	0.50502E-06	0.57520E-06	0.57520E-06	0.30152E-05
19	0.10101E-06	0.46193E-06	0.53100E-06	0.53100E-06	0.29279E-05

Residuals Output File (*case.rsd2*)

Basic File Format

```

1      (RSD(i), i = 1,7)      1
1      (RSD(i), i = 1,7)      2
:      :                      :
1      (RSD(i), i = 1,7)      icyc
:      :                      :
istp   (RSD(i), i = 1,7)      1
:      :                      :
nstp   (RSD(i), i = 1,7)      1
:      :                      :

```

Definition of Terms

istp:	int	current solution step
icyc:	int	iteration of current residual
nstp:	int	total or last solution step
RSD(1):	real	density solution residual
RSD(2):	real	x-momentum solution residual
RSD(3):	real	y-momentum solution residual
RSD(4):	real	z-momentum solution residual
RSD(5):	real	energy solution residual
RSD(6):	real	1st turb. model eq. residual
RSD(7):	real	2nd turb. model eq. residual

Comments

- This is a plain text (ASCII) file.
- This file is output when `irsds = .true.` in the controls `case.con` file. The residuals shown in this file represent the RMS changes at all nodes in the domain for this inner cycle. The convergence of residuals within any iteration can be seen in the trend in the residuals through the cycles used.
- RSD(6) and RSD(7) represent residuals for the equation(s) of the turbulence model:
 - If `iturb = 0`, RSD(6) and RSD(7) are omitted.
 - If `iturb = 1`, RSD(6) is written, representing the SA differential equation; RSD(7) is omitted.
 - If `iturb = 2`, RSD(6) and RSD(7) are both written, representing the SST differential equations.

Sample File

1	0.38320E-05	0.10743E-04	0.69854E-05	0.69854E-05	0.69854E-05	0.69854E-05	0.10598E-03	1
1	0.20317E-05	0.50694E-05	0.40436E-05	0.40436E-05	0.40436E-05	0.40436E-05	0.56307E-04	2
1	0.12024E-05	0.35187E-05	0.26241E-05	0.26241E-05	0.26241E-05	0.26241E-05	0.32195E-04	3
1	0.91334E-06	0.25166E-05	0.23637E-05	0.23637E-05	0.23637E-05	0.23637E-05	0.24240E-04	4
1	0.73183E-06	0.19442E-05	0.22228E-05	0.22228E-05	0.22228E-05	0.22228E-05	0.19376E-04	5
1	0.59870E-06	0.16179E-05	0.20889E-05	0.20889E-05	0.20889E-05	0.20889E-05	0.15963E-04	6
1	0.51663E-06	0.14311E-05	0.19719E-05	0.19719E-05	0.19719E-05	0.19719E-05	0.13946E-04	7
1	0.44924E-06	0.12989E-05	0.18536E-05	0.18536E-05	0.18536E-05	0.18536E-05	0.12398E-04	8
2	0.39510E-06	0.12095E-05	0.17283E-05	0.17283E-05	0.17283E-05	0.17283E-05	0.11156E-04	1
2	0.34726E-06	0.11478E-05	0.15878E-05	0.15878E-05	0.15878E-05	0.15878E-05	0.99450E-05	2
2	0.30775E-06	0.10746E-05	0.14329E-05	0.14329E-05	0.14329E-05	0.14329E-05	0.88159E-05	3
2	0.26207E-06	0.98700E-06	0.12833E-05	0.12833E-05	0.12833E-05	0.12833E-05	0.76280E-05	4
2	0.22418E-06	0.87924E-06	0.11245E-05	0.11245E-05	0.11245E-05	0.11245E-05	0.65113E-05	5
2	0.18904E-06	0.77764E-06	0.98148E-06	0.98148E-06	0.98148E-06	0.98148E-06	0.54617E-05	6
2	0.15809E-06	0.69345E-06	0.84471E-06	0.84471E-06	0.84471E-06	0.84471E-06	0.44739E-05	7
2	0.13411E-06	0.62203E-06	0.72991E-06	0.72991E-06	0.72991E-06	0.72991E-06	0.37422E-05	8
3	0.11564E-06	0.55717E-06	0.64350E-06	0.64350E-06	0.64350E-06	0.64350E-06	0.32661E-05	1
3	0.10516E-06	0.50502E-06	0.57520E-06	0.57520E-06	0.57520E-06	0.57520E-06	0.30152E-05	2
3	0.10101E-06	0.46193E-06	0.53100E-06	0.53100E-06	0.53100E-06	0.53100E-06	0.29279E-05	3

Time Step Output File (case.time)

Basic File Format

```

utime  htime  vtime  stime
  :      :      ::

```

Definition of Terms

```

utime:  real  unsteady time step ratio
htime:  real  heat transfer time step ratio
vtime:  real  momentum time step ratio
stime:  real  turbulence time step ratio

```

Comments

- This is a plain text (ASCII) file.
- The time step file `case.time` is written out when the residuals watching flag is turned on (`irsds = .true.`). The third column, containing `vtime`, is only written when `itime > 1`; and the fourth column, containing `stime`, is only written when `itime = 2`.
- The four ratios are calculated using the following equations:

$$htime = MIN\left(\left(\frac{\Delta t_{HT}}{\Delta t_{inv}}\right)_n, 1\right) \quad (\Delta t_{mom})_n = MIN\left((\Delta t_{mom,x})_n, (\Delta t_{mom,y})_n, (\Delta t_{mom,z})_n\right)$$

$$vtime = MIN\left(\left(\frac{\Delta t_{mom}}{\Delta t_{min}^{HT}}\right)_n, 1\right) \quad (\Delta t_{min}^{HT})_n = MIN\left((\Delta t_{inv})_n, (\Delta t_{HT})_n\right)$$

$$stime = MIN\left(\left(\frac{(\Delta t_{turb})_n}{(\Delta t_{min}^{mom})_n}\right), 1\right) \quad (\Delta t_{min}^{mom})_n = MIN\left((\Delta t_{min}^{HT})_n, (\Delta t_{mom})_n\right)$$

$$utime = MIN\left(\left(\frac{(\Delta t_{min}^{turb})_n}{dt}\right), 1\right) \quad (\Delta t_{min}^{turb})_n = MIN\left((\Delta t_{min}^{mom})_n, (\Delta t_{turb})_n\right)$$

$$\text{For SST,} \quad (\Delta t_{turb})_n = MIN\left((\Delta t_K)_n, (\Delta t_\omega)_n\right)$$

where Δt_{inv} , Δt_{HT} , Δt_{mom} , and Δt_{turb} are the inviscid, heat transfer, momentum, and turbulence local time steps and dt is the global time step.

- The heat transfer time step is calculated using its stiffness matrix for $itime \geq 0$. For $itime < 0$, the minimum element length is used to calculate the local time step.
- For momentum, the diagonal of both the x - and y -stiffness matrices are tested for their local time steps and then combined into a single momentum time step.
- For the SA model, the SA diffusion terms are used to calculate its local time step. For the SST model, both the k - and ω -diffusion terms are used to calculate the local time step, as shown above.

- A conservative rule of thumb is to apply $itime = 2$ to use the minimum of all local time steps. If a column of the file shows all ones, like the example below, then the corresponding equation is more stable than those tested before it, and $itime$ can be reduced. In the example below shows the turbulence model is more stable than the other equations at all time steps. The example also shows that the momentum stability is necessary at the beginning of the run but can be removed after several iterations. For this case, after 20 iterations, $itime$ can be decreased to 0 to limit testing to the heat transfer stiffness matrix.
- Maximum viscous stability is obtained when all columns except for the first take a value of unity (1).
- The first column should be less than unity (1). The ratio of local to global time step acts like a relaxation factor in the solver. The solver limits this ratio to a maximum of unity, so unity in the first column shows that the global time step is smaller than all of the local time steps on the domain. The global time step should be increased until a value less than unity is reached somewhere on the domain.
- The number of cycles $ncyc$ can also be gauged using the first column. The number of cycles should be greater than $dt / \Delta t_{min}$ to maintain a reasonable assumption of time accuracy. Column represents the minimum of $\Delta t_{min} / dt$ on the domain, or the maximum cycles needed on the domain.

Sample File

0.85620E+00	0.75632E+00	0.56382E+00	1.00000E+00
0.87930E+00	0.75555E+00	0.62853E+00	1.00000E+00
0.78200E+00	0.75334E+00	0.75932E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.88937E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.93048E+00	1.00000E+00
0.79381E+00	0.75475E+00	0.95043E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.97490E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.98403E+00	1.00000E+00
0.79379E+00	0.75475E+00	0.98738E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.99018E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.99205E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.99405E+00	1.00000E+00
0.79378E+00	0.75475E+00	0.99739E+00	1.00000E+00
0.79380E+00	0.75475E+00	0.99993E+00	1.00000E+00
0.79380E+00	0.75475E+00	1.00000E+00	1.00000E+00
0.79381E+00	0.75475E+00	1.00000E+00	1.00000E+00
0.79380E+00	0.75475E+00	1.00000E+00	1.00000E+00
0.79380E+00	0.75475E+00	1.00000E+00	1.00000E+00
0.79380E+00	0.75475E+00	1.00000E+00	1.00000E+00
0.79382E+00	0.75475E+00	1.00000E+00	1.00000E+00
0.79382E+00	0.75475E+00	1.00000E+00	1.00000E+00
⋮	⋮	⋮	⋮

Restart Files (case.rst and case.rs2)

Basic File Format

```
istp nnd gam xmi ref dt
((UN(i,j), i = 1,nnd), j = 1,6)
((UNO(i,j), i = 1,nnd), j = 1,6)
((UNT(i,j), i = 1,nnd), j = 1,4)
((UNTO(i,j), i = 1,nnd), j = 1,4)
(XN(i,j), i = 1,2*nr),
      (XN1(i), i = 1,2*nr)
(FA(i,j), i = 1,3*nr),
      (FA2(i), i = 1,nr)
(XD(i,j), i = 1,13),
      (XD1(i), i = 1,13)
(FD(i,j), i = 1,18),
      (FD2(i), i = 1,6)
```

Definition of Terms

istp:	int	step in global solution
nnd:	int	number of nodes
gam:	real	ratio of specific heats
xmi:	real	free-stream Mach number
ref:	real	reference dimension
dt:	real	global time step
UN:	real	unknowns at previous step
UNO:	real	unknowns at two steps prior
UNT:	real	turb unknowns at prev step
UNTO:	real	turb unknowns back 2 steps
XN:	real	elastic deflect / velocity
XN1:	real	prev. elastic deflect / velocity
FA:	real	gen. aero. forces at 3 steps
FA2:	real	external forcing on modes
XD:	real	rigid body position / velocity
XD1:	real	prev. rigid body pos. / vel.
FD:	real	aero. forces at 3 steps
FD2:	real	external forcing on vehicle

Comments

- This is an unformatted (binary) file.
- The solution unknowns, deflections, and forces stored in this file are dimensionless quantities.
- Turbulent unknowns properties vector UNT and UNTO are only written when turbulence models are used ($iturb > 0$).
- Unknowns properties vector UNO and UNTO are only written for 2nd order unsteady solutions ($isol = 2$).
- Elastic properties XN, XN1, FA, and FA2 is only written when the elastics flag `ielast` is set to `.true`.
- Non-inertial properties XD, XD1, FD, and FD2 is only written when the non-inertial flag `idynm` is set to `.true`.
- A two-file system is used so that one file is written while the other file is untouched. If the program crashes, one file is always untouched so that one of the two files is always recoverable.

APPENDIX G

2D ACOUSTIC OUTPUTS

The acoustic output files report density, velocity, and pressure data at a series of points defined by points and lines in space. The points exist in an element in the domain and are therefore interpolated from its three points. The lines cross one or more elements in the domain and are interpolated where the line enters and leaves each element, along the segments between its nodes.

Acoustic Points. The point (x_p, y_p) is the desired location within the flow at which properties are to be recorded into an acoustic history. This point exists within at least one element. (The point can exist within more than one element if the point exists along the intersection of two or more elements.) Locations within an element can be tracked within the global (x, y) and local (ξ_1, ξ_2, ξ_3) frames. The two frames are related:

$$\begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} x_1 \\ y_1 \end{Bmatrix} \xi_1 + \begin{Bmatrix} x_2 \\ y_2 \end{Bmatrix} \xi_2 + \begin{Bmatrix} x_3 \\ y_3 \end{Bmatrix} \xi_3 \quad (\text{G.1})$$

$$\bar{x} = \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{bmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{bmatrix} \begin{Bmatrix} \xi_1 \\ \xi_2 \end{Bmatrix} + \begin{Bmatrix} x_3 \\ y_3 \end{Bmatrix} = [J_e] \bar{\xi} + \bar{x}_3 \quad (\text{G.2})$$

$$\bar{\xi} = [J_e]^{-1}(\bar{x} - \bar{x}_3) = \frac{1}{|J_e|} [\mathbf{A}](\bar{x} - \bar{x}_3) \quad (\text{G.3})$$

\mathbf{A} is the same inverse Jacobian matrix used to construct gradients within each element. If $0 \leq \xi_1, \xi_2 \leq 1$, then the desired point lies within that element, and the properties at the point can be reconstructed:

$$\mathbf{U}_p = \mathbf{U}_1 \xi_1 + \mathbf{U}_2 \xi_2 + \mathbf{U}_3 (1 - \xi_1 - \xi_2) \quad (\text{G.4})$$

Acoustic Lines. The acoustic line is defined by two end points (x_{1L}, y_{1L}) and (x_{2L}, y_{2L}) so that:

$$\begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} x_{1L} \\ y_{1L} \end{Bmatrix} \eta + \begin{Bmatrix} x_{2L} \\ y_{2L} \end{Bmatrix} (1 - \eta) = \begin{Bmatrix} x_{1L} - x_{2L} \\ y_{1L} - y_{2L} \end{Bmatrix} \eta + \begin{Bmatrix} x_{2L} \\ y_{2L} \end{Bmatrix} \quad (\text{G.5})$$

The line intersects each element within the domain at its edges. The edges are represented by the segments within the domain:

$$\begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} x_1 \\ y_1 \end{Bmatrix} \xi_1 + \begin{Bmatrix} x_2 \\ y_2 \end{Bmatrix} \xi_2 = \begin{Bmatrix} x_1 - x_2 \\ y_1 - y_2 \end{Bmatrix} \xi_1 + \begin{Bmatrix} x_2 \\ y_2 \end{Bmatrix} \quad (\text{G.6})$$

The intersection of the acoustic line and segment is represented by:

$$\begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} x_{1L} - x_{2L} \\ y_{1L} - y_{2L} \end{Bmatrix} \eta + \begin{Bmatrix} x_{2L} \\ y_{2L} \end{Bmatrix} = \begin{Bmatrix} x_1 - x_2 \\ y_1 - y_2 \end{Bmatrix} \xi_1 + \begin{Bmatrix} x_2 \\ y_2 \end{Bmatrix} \quad (\text{G.7})$$

$$[\mathbf{C}] \begin{Bmatrix} \eta \\ \xi_1 \end{Bmatrix} = \begin{bmatrix} x_{1L} - x_{2L} & x_2 - x_1 \\ y_{1L} - y_{2L} & y_2 - y_1 \end{bmatrix} \begin{Bmatrix} \eta \\ \xi_1 \end{Bmatrix} = \begin{Bmatrix} x_2 - x_{2L} \\ y_2 - y_{2L} \end{Bmatrix} = \bar{b} \quad (\text{G.8})$$

$$\begin{Bmatrix} \eta \\ \xi_1 \end{Bmatrix} = [\mathbf{C}]^{-1} \bar{b} \quad (\text{G.9})$$

If $0 \leq \eta, \xi_1 \leq 1$, then the point lies on both the segment and the acoustic line (within the bounds of each). The properties at that point can be reconstructed from the properties at the endpoints of the segment:

$$\mathbf{U}_p = \mathbf{U}_1 \xi_1 + \mathbf{U}_2 (1 - \xi_1) \quad (\text{G.10})$$

If the \mathbf{C} -matrix is singular, then the segment is parallel with the acoustic line. If one or both of the end points of the segment lies on the acoustic line, then an infinite number of points along the segment also lie on the acoustic line. For simplicity, the two end points of the segment can be checked. If either or both end points lie on the acoustic line, then these points should be treated as separate intersections with the acoustic line. The first end point is:

$$\begin{Bmatrix} x_1 \\ y_1 \end{Bmatrix} = \begin{Bmatrix} x_{1L} - x_{2L} \\ y_{1L} - y_{2L} \end{Bmatrix} \eta + \begin{Bmatrix} x_{2L} \\ y_{2L} \end{Bmatrix} \quad (\text{G.11})$$

$$0 \leq \eta = \frac{x_1 - x_{2L}}{x_{1L} - x_{2L}} = \frac{y_1 - y_{2L}}{y_{1L} - y_{2L}} \leq 1 \quad (\text{G.12})$$

If the acoustic line is constant in x , then $x_{1L} = x_{2L}$ and the denominator of the x -test will be singular. In this case, the segment must also lie along the same x ($x_1 = x_{2L}$). If the acoustic line is constant in y , then $y_{1L} = y_{2L}$ and the denominator of the y -test will be singular. In this case, the segment must also lie along the same y ($y_1 = y_{2L}$). The properties are equal to the properties at the first node.

The second end point is checked:

$$0 \leq \eta = \frac{x_2 - x_{2L}}{x_{1L} - x_{2L}} = \frac{y_2 - y_{2L}}{y_{1L} - y_{2L}} \leq 1 \quad (\text{G.13})$$

If the acoustic line is constant in x , then $x_{1L} = x_{2L}$ and the denominator of the x -test will be singular. In this case, the segment must also lie along the same x ($x_2 = x_{2L}$). If the acoustic line is constant in y , then $y_{1L} = y_{2L}$ and the denominator of the y -test will be singular. In this case, the segment must also lie along the same y ($y_2 = y_{2L}$). The properties are equal to the properties at the second node.

APPENDIX H

3D ACOUSTIC OUTPUTS

The acoustic output files report density, velocity, and pressure data at a series of points defined by points and lines in space. The points exist in an element in the domain and are therefore interpolated from its four points. The lines cross one or more elements in the domain and are interpolated where the line enters and leaves each element, along the segments between its nodes.

Acoustic Points. The point (x_p, y_p, z_p) is the desired location within the flow at which properties are to be recorded into an acoustic history. This point exists within at least one element. (The point can exist within more than one element if the point exists along the intersection of two or more elements.) Locations within an element can be tracked within the global (x, y, z) and local $(\xi_1, \xi_2, \xi_3, \xi_4)$ frames. The two frames are related:

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{Bmatrix} x_1 \\ y_1 \\ z_1 \end{Bmatrix} \xi_1 + \begin{Bmatrix} x_2 \\ y_2 \\ z_2 \end{Bmatrix} \xi_2 + \begin{Bmatrix} x_3 \\ y_3 \\ z_3 \end{Bmatrix} \xi_3 + \begin{Bmatrix} x_4 \\ y_4 \\ z_4 \end{Bmatrix} \xi_4 \quad (\text{H.1})$$

$$\bar{\mathbf{x}} = \begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{bmatrix} x_1 - x_4 & x_2 - x_4 & x_3 - x_4 \\ y_1 - y_4 & y_2 - y_4 & y_3 - y_4 \\ z_1 - z_4 & z_2 - z_4 & z_3 - z_4 \end{bmatrix} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix} + \begin{Bmatrix} x_4 \\ y_4 \\ z_4 \end{Bmatrix} = [J_e] \bar{\xi} + \bar{x}_4 \quad (\text{H.2})$$

$$\bar{\xi} = [J_e]^{-1}(\bar{\mathbf{x}} - \bar{x}_4) = \frac{1}{|J_e|} [\mathbf{A}_e](\bar{\mathbf{x}} - \bar{x}_4) \quad (\text{H.3})$$

\mathbf{A} is the same inverse Jacobian matrix used to construct gradients within elements. If $0 \leq \xi_1, \xi_2, \xi_3 \leq 1$, then the desired point lies within that element, and the properties at the point can be reconstructed:

$$\mathbf{U}_p = \mathbf{U}_1 \xi_1 + \mathbf{U}_2 \xi_2 + \mathbf{U}_3 \xi_3 + \mathbf{U}_4 (1 - \xi_1 - \xi_2 - \xi_3) \quad (\text{H.4})$$

Acoustic Lines. The acoustic line is defined by two end points (x_{1L}, y_{1L}, z_{1L}) and (x_{2L}, y_{2L}, z_{2L}) so that:

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{Bmatrix} x_{1L} \\ y_{1L} \\ z_{1L} \end{Bmatrix} \eta + \begin{Bmatrix} x_{2L} \\ y_{2L} \\ z_{2L} \end{Bmatrix} (1 - \eta) = \begin{Bmatrix} x_{1L} - x_{2L} \\ y_{1L} - y_{2L} \\ z_{1L} - z_{2L} \end{Bmatrix} \eta + \begin{Bmatrix} x_{2L} \\ y_{2L} \\ z_{2L} \end{Bmatrix} \quad (\text{H.5})$$

The line intersects each element within the domain at its faces. The faces are represented by ζ_1, ζ_2 , and ζ_3 represent the face-local coordinates between the three nodes of a face. The face-local coordinates ζ_i correspond to three of the element-local coordinates ξ_i . The fourth element-local coordinate is zero. For example, if the face corresponds to nodes 1, 3, and 4, then the face-local and element-local coordinates are related:

$$\xi_1 = \zeta_1 \quad \xi_2 = 0 \quad \xi_3 = \zeta_2 \quad \xi_4 = \zeta_3 \quad (\text{H.6})$$

Each face is represented in face-local coordinates by:

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{Bmatrix} x_1 \\ y_1 \\ z_1 \end{Bmatrix} \zeta_1 + \begin{Bmatrix} x_2 \\ y_2 \\ z_2 \end{Bmatrix} \zeta_2 + \begin{Bmatrix} x_3 \\ y_3 \\ z_3 \end{Bmatrix} \zeta_3 \quad (\text{H.7})$$

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{Bmatrix} x_1 - x_3 \\ y_1 - y_3 \\ z_1 - z_3 \end{Bmatrix} \zeta_1 + \begin{Bmatrix} x_2 - x_3 \\ y_2 - y_3 \\ z_2 - z_3 \end{Bmatrix} \zeta_2 + \begin{Bmatrix} x_3 \\ y_3 \\ z_3 \end{Bmatrix} \quad (\text{H.8})$$

The intersection of the acoustic line and element face is represented by:

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{Bmatrix} x_{1L} - x_{2L} \\ y_{1L} - y_{2L} \\ z_{1L} - z_{2L} \end{Bmatrix} \eta + \begin{Bmatrix} x_{2L} \\ y_{2L} \\ z_{2L} \end{Bmatrix} = \begin{Bmatrix} x_1 - x_3 \\ y_1 - y_3 \\ z_1 - z_3 \end{Bmatrix} \zeta_1 + \begin{Bmatrix} x_2 - x_3 \\ y_2 - y_3 \\ z_2 - z_3 \end{Bmatrix} \zeta_2 + \begin{Bmatrix} x_3 \\ y_3 \\ z_3 \end{Bmatrix} \quad (\text{H.9})$$

$$[\mathbf{C}] \begin{Bmatrix} \eta \\ \zeta_1 \\ \zeta_2 \end{Bmatrix} = \begin{bmatrix} x_{1L} - x_{2L} & x_3 - x_1 & x_3 - x_2 \\ y_{1L} - y_{2L} & y_3 - y_1 & y_3 - y_2 \\ z_{1L} - z_{2L} & z_3 - z_1 & z_3 - z_2 \end{bmatrix} \begin{Bmatrix} \eta \\ \zeta_1 \\ \zeta_2 \end{Bmatrix} = \begin{Bmatrix} x_3 - x_{2L} \\ y_3 - y_{2L} \\ z_3 - z_{2L} \end{Bmatrix} = \bar{\mathbf{b}} \quad (\text{H.10})$$

$$\begin{Bmatrix} \eta \\ \zeta_1 \\ \zeta_2 \end{Bmatrix} = [\mathbf{C}]^{-1} \bar{\mathbf{b}} \quad (\text{H.11})$$

If $0 \leq \eta, \zeta_1, \zeta_2 \leq 1$, then the point lies on both the face and the acoustic line (within the bounds of each). The properties at that point can be reconstructed from the properties at the vertices of the face:

$$\mathbf{U}_p = \mathbf{U}_1 \zeta_1 + \mathbf{U}_2 \zeta_2 + \mathbf{U}_3 (1 - \zeta_1 - \zeta_2) \quad (\text{H.12})$$

The properties can also be written in terms of the four vertices of the element, where one of the four element-local coordinates is identically zero:

$$\mathbf{U}_p = \mathbf{U}_1 \xi_1 + \mathbf{U}_2 \xi_2 + \mathbf{U}_3 \xi_3 + \mathbf{U}_4 (1 - \xi_1 - \xi_2 - \xi_3) \quad (\text{H.13})$$

If the \mathbf{C} -matrix is singular, then the face is parallel with the acoustic line. If one or two points along an edge of this face lie on the acoustic line, then an infinite number of points

along the face also lie on the acoustic line. For simplicity, intersections with each of the three edges are checked against the acoustic line. Each edge is treated just like the segments in the 2D case and represented by two nodes, in edge-local coordinates χ_1 and χ_2 , which relate to two of the four element-local coordinates. For example, if nodes 1 and 3 are used by the edge, then edge-local and element-local coordinates are related:

$$\xi_1 = \chi_1 \quad \xi_2 = 0 \quad \xi_3 = \chi_2 \quad \xi_4 = 0 \quad (\text{H.14})$$

The edges are represented by:

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{Bmatrix} x_1 \\ y_1 \\ z_1 \end{Bmatrix} \chi_1 + \begin{Bmatrix} x_2 \\ y_2 \\ z_2 \end{Bmatrix} \chi_2 = \begin{Bmatrix} x_1 - x_2 \\ y_1 - y_2 \\ z_1 - z_2 \end{Bmatrix} \chi_1 + \begin{Bmatrix} x_2 \\ y_2 \\ z_2 \end{Bmatrix} \quad (\text{H.15})$$

The intersection of the acoustic line and segment is represented by:

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{Bmatrix} x_{1L} - x_{2L} \\ y_{1L} - y_{2L} \\ z_{1L} - z_{2L} \end{Bmatrix} \eta + \begin{Bmatrix} x_{2L} \\ y_{2L} \\ z_{2L} \end{Bmatrix} = \begin{Bmatrix} x_1 - x_2 \\ y_1 - y_2 \\ z_1 - z_2 \end{Bmatrix} \chi_1 + \begin{Bmatrix} x_2 \\ y_2 \\ z_2 \end{Bmatrix} \quad (\text{H.16})$$

$$\begin{bmatrix} x_{1L} - x_{2L} & x_2 - x_1 \\ y_{1L} - y_{2L} & y_2 - y_1 \\ z_{1L} - z_{2L} & z_2 - z_1 \end{bmatrix} \begin{Bmatrix} \eta \\ \chi_1 \end{Bmatrix} = \begin{Bmatrix} x_2 - x_{2L} \\ y_2 - y_{2L} \\ z_2 - z_{2L} \end{Bmatrix} \quad (\text{H.17})$$

Only two of the equations need to be solved at a time to determine the two unknowns:

$$[\mathbf{C}] \begin{Bmatrix} \eta \\ \chi_1 \end{Bmatrix} = \bar{\mathbf{b}} \quad \text{so that} \quad \begin{Bmatrix} \eta \\ \chi_1 \end{Bmatrix} = [\mathbf{C}]^{-1} \bar{\mathbf{b}} \quad (\text{H.18})$$

where the three \mathbf{C} -matrices and \mathbf{b} -vectors are based in the xy -, xz -, and yz -planes:

$$[\mathbf{C}_{xy}] = \begin{bmatrix} x_{1L} - x_{2L} & x_2 - x_1 \\ y_{1L} - y_{2L} & y_2 - y_1 \end{bmatrix} \quad \bar{\mathbf{b}}_{xy} = \begin{Bmatrix} x_2 - x_{2L} \\ y_2 - y_{2L} \end{Bmatrix} \quad (\text{H.19})$$

$$[\mathbf{C}_{xz}] = \begin{bmatrix} x_{1L} - x_{2L} & x_2 - x_1 \\ z_{1L} - z_{2L} & z_2 - z_1 \end{bmatrix} \quad \bar{\mathbf{b}}_{xz} = \begin{Bmatrix} x_2 - x_{2L} \\ z_2 - z_{2L} \end{Bmatrix} \quad (\text{H.20})$$

$$[\mathbf{C}_{yz}] = \begin{bmatrix} y_{1L} - y_{2L} & y_2 - y_1 \\ z_{1L} - z_{2L} & z_2 - z_1 \end{bmatrix} \quad \bar{\mathbf{b}}_{yz} = \begin{Bmatrix} y_2 - y_{2L} \\ z_2 - z_{2L} \end{Bmatrix} \quad (\text{H.21})$$

As long as the line and edge are non-constant in the x -, y -, or z -directions, then any of the three systems (\mathbf{C} and \mathbf{b}) can be solved to find the same two unknowns. If the line and edge are constant in the x -direction, for example, then the xy - and xz -systems will be singular. For this reason, the solution is checked in each plane, and then the solutions are compared. In the case of a pair of singular planes, the common dimension is checked for intersection (e.g., if xy - and xz -systems are singular, then x is common and $x_1 = x_2 = x_{1L} = x_{2L}$ must exist for intersection to occur). The properties at that point can be reconstructed from the properties at the endpoints of the segment:

$$\mathbf{U}_p = \mathbf{U}_1 \chi_1 + \mathbf{U}_2 (1 - \chi_1) \quad (\text{H.22})$$

The properties can also be written in terms of the four vertices of the element, where two of the four element-local coordinates are identically zero:

$$\mathbf{U}_p = \mathbf{U}_1 \xi_1 + \mathbf{U}_2 \xi_2 + \mathbf{U}_3 \xi_3 + \mathbf{U}_4 (1 - \xi_1 - \xi_2 - \xi_3) \quad (\text{H.23})$$

If all three systems are singular, then the edge is parallel with the acoustic line. If one or both of the end points of the segment lies on the acoustic line, then an infinite number of points along the segment also lie on the acoustic line. For simplicity, the two end points of the segment can be checked. If either or both end points lie on the acoustic line, then these points should be treated as separate intersections with the acoustic line. The first end point is checked using:

$$\begin{Bmatrix} x_1 \\ y_1 \\ z_1 \end{Bmatrix} = \begin{Bmatrix} x_{1L} - x_{2L} \\ y_{1L} - y_{2L} \\ z_{1L} - z_{2L} \end{Bmatrix} \boldsymbol{\eta} + \begin{Bmatrix} x_{2L} \\ y_{2L} \\ z_{2L} \end{Bmatrix} \quad (\text{H.24})$$

$$\eta_x = \frac{x_1 - x_{2L}}{x_{1L} - x_{2L}} \quad \eta_y = \frac{y_1 - y_{2L}}{y_{1L} - y_{2L}} \quad \eta_z = \frac{z_1 - z_{2L}}{z_{1L} - z_{2L}} \quad (\text{H.25})$$

$$0 \leq \eta = \eta_x = \eta_y = \eta_z \leq 1 \quad (\text{H.26})$$

If the acoustic line is constant in x , then $x_{1L} = x_{2L}$ and the denominator of the x -test will be singular. In this case, the segment must also lie along the same x ($x_1 = x_{2L}$). If the acoustic line is constant in y , then $y_{1L} = y_{2L}$ and the denominator of the y -test will be singular. In this case, the segment must also lie along the same y ($y_1 = y_{2L}$). If the acoustic line is constant in z , then $z_{1L} = z_{2L}$ and the denominator of the z -test will be singular. In this case, the segment must also lie along the same z ($z_1 = z_{2L}$). The properties can be reconstructed using:

$$\mathbf{U}_p = \mathbf{U}_1 \xi_1 + \mathbf{U}_2 (1 - \xi_1) \quad \text{where} \quad \xi_1 = 1 \quad (\text{H.27})$$

This can also be written in terms of the four properties at the vertices of the element, where three of the element-local coordinates are identically zero.

The second end point is checked using:

$$\begin{Bmatrix} x_2 \\ y_2 \\ z_2 \end{Bmatrix} = \begin{Bmatrix} x_{1L} - x_{2L} \\ y_{1L} - y_{2L} \\ z_{1L} - z_{2L} \end{Bmatrix} \boldsymbol{\eta} + \begin{Bmatrix} x_{2L} \\ y_{2L} \\ z_{2L} \end{Bmatrix} \quad (\text{H.28})$$

$$\eta_x = \frac{x_2 - x_{2L}}{x_{1L} - x_{2L}} \quad \eta_y = \frac{y_2 - y_{2L}}{y_{1L} - y_{2L}} \quad \eta_z = \frac{z_2 - z_{2L}}{z_{1L} - z_{2L}} \quad (\text{H.29})$$

$$0 \leq \eta = \eta_x = \eta_y = \eta_z \leq 1 \quad (\text{H.30})$$

If the acoustic line is constant in x , then $x_{1L} = x_{2L}$ and the denominator of the x -test will be singular. In this case, the segment must also lie along the same x ($x_I = x_{2L}$). If the acoustic line is constant in y , then $y_{1L} = y_{2L}$ and the denominator of the y -test will be singular. In this case, the segment must also lie along the same y ($y_I = y_{2L}$). If the acoustic line is constant in z , then $z_{1L} = z_{2L}$ and the denominator of the z -test will be singular. In this case, the segment must also lie along the same z ($z_I = z_{2L}$). The properties can be reconstructed using:

$$\mathbf{U}_p = \mathbf{U}_1 \xi_1 + \mathbf{U}_2 (1 - \xi_1) \quad \text{where} \quad \xi_1 = 0 \quad (\text{H.31})$$

This can also be written in terms of the four properties at the vertices of the element, where three of the element-local coordinates are identically zero.

APPENDIX I

ENTROPY-BASED ARTIFICIAL DISSIPATION

A better artificial dissipation model needs to be found. The current dissipation model in the OSU codes is sufficient for inviscid solutions but overly diffuse in the near-wall region. The result is a stable solution with a thickened boundary layer and reduced skin friction. Neither of these influences is acceptable. A zero dissipation length has been applied to NS2D/3D to keep the current model until a new model is found. CFDsol applies a shock-capturing model that is overly diffuse everywhere within the flow, and the zero dissipation length cannot be applied to CFDsol because artificial dissipation is required to stabilize the solution, even in the near-wall region.

Both artificial dissipation models are *ad hoc* attempts at creating positive entropy growth everywhere in the solution. A more appropriate model would incorporate the Second Law of Thermodynamics to ensure a positive entropy growth, using entropy (Guermond, 2011; Olson, 2012). The “first T ds equation” (Moran and Shapiro, 2000) can be used to create both parts of the model:

$$Tds = de + p dv = de + p d\left(\frac{1}{\rho}\right) = de - \frac{p}{\rho^2} d\rho \quad (I.1)$$

Eq. I.1 can be used to relate the time rate of change in entropy s to the rates of change in internal energy e and density ρ and similar relationships for the spatial derivatives:

$$T \frac{ds}{dt} = \frac{de}{dt} - \frac{p}{\rho^2} \frac{d\rho}{dt} \quad T \frac{ds}{dx_i} = \frac{de}{dx_i} - \frac{p}{\rho^2} \frac{d\rho}{dx_i} \quad (I.2)$$

$$\rho T \frac{Ds}{Dt} = T \left(\rho \frac{ds}{dt} + \rho u_i \frac{ds}{dx_i} \right) = \left(\rho \frac{de}{dt} + \rho u_i \frac{de}{dx_i} \right) - \frac{p}{\rho^2} \left(\rho \frac{d\rho}{dt} + \rho u_i \frac{d\rho}{dx_i} \right) \quad (I.3)$$

Eq. I.3 is the substantial derivative of entropy s or rate of change of entropy traveling with the flow (Lagrangian frame). We can pose Eq. I.3 in terms of the Navier-Stokes equations (Euler frame) that we are already using in the CFD solution. We address the first term on the right side of Eq. I.3, starting from the energy equation (Eqs. 3.12 and 3.13 with Eq. 3.6):

$$\frac{d\rho E}{dt} + \frac{d}{dx_i} (u_i (\rho E + p)) = \frac{d}{dx_i} (\tau_{ij} u_j - q_i^{\prime\prime}) \quad (I.4)$$

$$\frac{d}{dt} (\rho e + \frac{1}{2} \rho u_j u_j) + \frac{d}{dx_i} (u_i (\rho e + \frac{1}{2} \rho u_j u_j + p)) = \frac{d}{dx_i} (\tau_{ij} u_j - q_i^{\prime\prime}) \quad (I.5)$$

$$\begin{aligned} \rho \frac{de}{dt} + \rho u_i \frac{de}{dx_i} + \left(e - \frac{1}{2} u_j u_j \right) \left[\frac{d\rho}{dt} + \frac{d\rho u_i}{dx_i} \right] \\ + u_j \left\{ \frac{d\rho u_j}{dt} + \frac{d}{dx_i} (\rho u_i u_j + p \delta_{ij}) \right\} + p \frac{du_i}{dx_i} = \frac{d}{dx_i} (\tau_{ij} u_j - q_i^{\prime\prime}) \end{aligned} \quad (I.6)$$

The terms in brackets [] is the continuity equation (Eqs. 3.12 and 3.13), which is identically zero. The terms in curly brackets { } are the left side of the momentum equation (Eqs. 3.12 and 3.13), which can be replaced with the right side of the momentum equation:

$$\rho \frac{de}{dt} + \rho u_i \frac{de}{dx_i} = \frac{d}{dx_i} (\tau_{ij} u_j - q_i^{\prime\prime}) - u_j \frac{d\tau_{ij}}{dx_i} - p \frac{du_i}{dx_i} \quad (I.7)$$

$$\rho \frac{de}{dt} + \rho u_i \frac{de}{dx_i} = \tau_{ij} \frac{du_j}{dx_i} - p \frac{du_i}{dx_i} - \frac{dq_i''}{dx_i} \quad (\text{I.8})$$

Eq. I.8 is substituted back into Eq. I.3:

$$\rho T \frac{Ds}{Dt} = \tau_{ij} \frac{du_j}{dx_i} - p \frac{du_i}{dx_i} - \frac{dq_i''}{dx_i} - \frac{p}{\rho^2} \left(\rho \frac{d\rho}{dt} + \rho u_i \frac{d\rho}{dx_i} \right) \quad (\text{I.9})$$

$$\rho T \frac{Ds}{Dt} = \tau_{ij} \frac{du_j}{dx_i} - p \frac{du_i}{dx_i} - \frac{dq_i''}{dx_i} - \frac{p}{\rho^2} \left(\rho \frac{d\rho}{dt} + \rho \frac{d\rho u_i}{dx_i} - \rho^2 \frac{du_i}{dx_i} \right) \quad (\text{I.10})$$

$$\rho T \frac{Ds}{Dt} = \tau_{ij} \frac{du_j}{dx_i} - \frac{dq_i''}{dx_i} - \frac{p}{\rho} \left[\frac{d\rho}{dt} + \frac{d\rho u_i}{dx_i} \right] \quad (\text{I.11})$$

The term in parentheses in Eq. I.10 is rearranged to cancel the pressure term and creating a continuity term, shown in brackets in Eq. I.11. The continuity equation is identically zero so the substantial derivative of entropy can be written using Eq. I.12:

$$\rho T \frac{Ds}{Dt} = \tau_{ij} \frac{du_j}{dx_i} - \frac{dq_i''}{dx_i} \quad (\text{I.12})$$

Returning to Eqs. I.4 and I.6, the left side of the energy equation can be written in terms that are seen in Eq. I.8, so that Eq. I.8 can be written in terms of the left hand sides (*LHS*) of the governing equations (Eqs. 3.12 and 3.13):

$$\begin{aligned} \rho \frac{de}{dt} + \rho u_i \frac{de}{dx_i} = & \left\langle \frac{d\rho E}{dt} + \frac{d}{dx_i} (u_i (\rho E + p)) \right\rangle - u_j \left\{ \frac{d\rho u_j}{dt} + \frac{d}{dx_i} (\rho u_i u_j + p \delta_{ij}) \right\} \\ & - \left(e - \frac{1}{2} u_j u_j \right) \left[\frac{d\rho}{dt} + \frac{d\rho u_i}{dx_i} \right] - p \frac{du_i}{dx_i} \end{aligned} \quad (\text{I.13})$$

$$\rho \frac{de}{dt} + \rho u_i \frac{de}{dx_i} = LHS_{\rho E} - u_j LHS_{\rho u_j} - \left(e - \frac{1}{2} u_j u_j \right) LHS_{\rho} - p \frac{du_i}{dx_i} \quad (\text{I.14})$$

$$\rho T \frac{Ds}{Dt} = LHS_{\rho E} - u_j LHS_{\rho u_j} - \left(\bar{h} - \frac{1}{2} \bar{u}_j \bar{u}_j \right) LHS_{\rho} \quad (\text{I.15})$$

Eqs. I.15 can now be integrated using Galerkin's method:

$$\int_{\Omega} \Phi_e^T \left(\rho T \frac{Ds}{Dt} \right) d\Omega = \int_{\Omega} \Phi_e^T \left(LHS_{\rho E} - \bar{u}_j LHS_{\rho u_j} - \left(\bar{h} - \frac{1}{2} \bar{u}_j \bar{u}_j \right) LHS_{\rho} \right) d\Omega \quad (\text{I.16})$$

$$\begin{aligned} \int_{\Omega} \Phi_e^T \left(\rho T \frac{Ds}{Dt} \right) d\Omega = \sum_e \left(\mathbf{M}_{c,e} \frac{\partial \rho E_e}{\partial t} - \bar{u}_j \mathbf{M}_{c,e} \frac{\partial \rho u_{j,e}}{\partial t} - \left(\bar{h} - \frac{1}{2} \bar{u}_j \bar{u}_j \right) \mathbf{M}_{c,e} \frac{\partial \rho_e}{\partial t} \right) \\ - \sum_e \frac{1}{d!} A_i^T \left(F_{i,\rho E} - \bar{u}_j F_{i,\rho u_j} - \left(\bar{h} - \frac{1}{2} \bar{u}_j \bar{u}_j \right) F_{i,\rho} \right) \\ + \sum_{be} \mathbf{M}_{c,be} \left(F_{n,\rho E} - \bar{u}_j F_{n,\rho u_j} - \left(\bar{h} - \frac{1}{2} \bar{u}_j \bar{u}_j \right) F_{n,\rho} \right) \end{aligned} \quad (\text{I.17})$$

$$\int_{\Omega} \Phi_e^T \left(\rho T \frac{Ds}{Dt} \right) d\Omega = \mathbf{R}_{inv,\rho E} - \bar{u}_j \mathbf{R}_{inv,\rho u_j} - \left(\bar{h} - \frac{1}{2} \bar{u}_j \bar{u}_j \right) \mathbf{R}_{inv,\rho} \quad (\text{I.18})$$

The integral in Eq. I.12 can be greatly simplified by averaging the velocity and enthalpy terms over each element. These terms can then be removed from the element integrals. Eq. 8.19 represents the simplified integration. All of the term in Eq. 8.19 have already been implemented in Euler2D/3D and NS2D/3D as the unsteady and inviscid flux contributions to the residual (Eq. I.18). The momentum (inviscid) residual is dotted with the velocity vector and the continuity residual is scaled by enthalpy less kinetic energy. Eq. I.18 gives a node-wise comparison of the entropy production at the nodes with little additional cost over the current implementation.

Eq. I.18 can be integrated using Galerkin's method (Eq. I.19). The stress tensor and heat flux have been expanded in Eq. I.20 to illustrate how the viscosity has been factored out in Eq.

I.21. The viscosity was then split into real viscosity μ and artificial viscosity μ_{AD} :

$$\int_{\Omega} \Phi_e^T \left(\rho T \frac{Ds}{Dt} \right) d\Omega = \int_{\Omega} \Phi_e^T \left(\tau_{ij} \frac{du_j}{dx_i} - \frac{dq_i''}{dx_i} \right) d\Omega = \sum_e \frac{\Omega_e}{d+1} \{1\} \left(\tau_{ij} \frac{du_j}{dx_i} - \frac{dq_i''}{dx_i} \right) \quad (\text{I.19})$$

$$\int_{\Omega} \Phi_e^T \left(\rho T \frac{Ds}{Dt} \right) d\Omega = \sum_e \frac{\Omega_e}{d+1} \{1\} \left(\frac{\mu}{\text{Re}} \left(\frac{du_j}{dx_i} + \frac{du_i}{dx_j} + \frac{\lambda}{\mu} \frac{du_k}{dx_k} \delta_{ij} \right) \frac{du_j}{dx_i} + \frac{d}{dx_i} \left(\frac{\mu}{\text{Re Pr}} \frac{dh}{dx_j} \right) \right) \quad (\text{I.20})$$

$$\int_{\Omega} \Phi_e^T \left(\rho T \frac{Ds}{Dt} \right) d\Omega = \mathbf{K}_{AD} (\mu_e + \mu_{AD,e}) \quad (\text{I.21})$$

Eq. I.21 represents the entropy production as a stiffness matrix times the effective viscosity (real plus artificial). The diagonal of the stiffness matrix \mathbf{K}'_{AD} can be used to back solve for the effective viscosity. The viscosity ratio μ (from Sutherland's equation) can be removed from the effective viscosity, leaving the additional viscosity required to stabilize the solution:

$$\mu_{AD,e} \approx \text{diss} \left(\mathbf{K}'_{AD} \int_{\Omega} \Phi_e^T \left(\rho T \frac{Ds}{Dt} \right) d\Omega - \mu_e \right) \quad (\text{I.22})$$

$$K_{AD,ii} = \sum_e \frac{\Omega_e}{d+1} \{1\} \left(\frac{1}{\text{Re}} \left(\frac{du_j}{dx_i} + \frac{du_i}{dx_j} + \frac{\lambda}{\mu} \frac{du_k}{dx_k} \delta_{ij} \right) \frac{du_j}{dx_i} + \frac{d}{dx_i} \left(\frac{1}{\text{Re Pr}} \frac{dh}{dx_j} \right) \right) \quad (\text{I.23})$$

Eq. I.18 can be assembled alongside the inviscid contributions to the residual while the diagonal of the stiffness matrix (Eq. I.23) can be assembled alongside the viscous contributions to the residual. Any positive terms assembled using Eq. I.18 represent entropy production, which satisfies the Second Law. These terms are discarded. The magnitude of the entropy destruction (negative terms) are used in Eq. I.23 to calculate the artificial viscosity necessary at each node in the domain, similar to the method presented by Olson (2012). The artificial viscosity can be used to reassemble viscous fluxes with μ_{AD} instead of μ . The stiffness matrix can be used for quick reassembly. Notice that the artificial viscosity has been scaled by *diss* to artificial dissipation scalable.

This model can be implemented in Euler2D/3D, where the real viscosity ratio μ is zero. The magnitude of entropy destruction from Eq. I.21 can be scaled by *diss* and applied directly back into the residual. Viscous contributions are necessary in NS2D/3D so that the entropy production from real viscosity can be accounted for before applying artificial dissipation.

VITA

Nicholas J Moffitt

Candidate for the Degree of

Doctor of Philosophy

Thesis: GALERKIN CFD SOLVERS FOR USE IN A MULTI-DISCIPLINARY
SUITE FOR MODELING ADVANCED FLIGHT VEHICLES

Major Field: Mechanical and Aerospace Engineering

Biographical:

Education:

Completed the requirements for the Doctor of Philosophy in Mechanical and Aerospace Engineering at Oklahoma State University, Stillwater, Oklahoma in May 2013.

Completed the requirements for the Master of Science in Mechanical Engineering at Oklahoma State University, Stillwater, Oklahoma in December 2004.

Completed the requirements for the Bachelor of Science in Aerospace Engineering at Oklahoma State University, Stillwater, Oklahoma in December 2002.

Experience:

Teaching Assistant, Oklahoma State University, Aug 2002-Dec 2007.

Research Assistant, CASE Lab, Oklahoma State University, Mar 2003-May 2013; worked on three NASA contracts in finite element CFD development.

Engineer Intern, Boeing Research & Technology, Boeing, June 2012-Aug 2012; worked on finite volume CFD development.

Engineer II, Boeing Research & Technology, Boeing, Jan 2013-present; worked on finite volume CFD development.

Professional Memberships:

American Institute of Aeronautics and Astronautics (AIAA), since 1998.