

FINITE ELEMENT CFD ANALYSIS
OF SUPER-MANEUVERING AND
SPINNING STRUCTURES

By

TIMOTHY JOHN COWAN

Bachelor of Science
Oklahoma State University
Stillwater, Oklahoma
1996

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1998

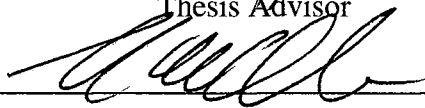
Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
August, 2003

FINITE ELEMENT CFD ANALYSIS
OF SUPER-MANEUVERING AND
SPINNING STRUCTURES

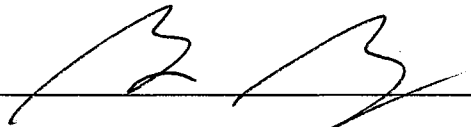
Thesis Approved:



Thesis Advisor



David E. Lillay



Timothy J. Petterson

Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to thank Dr. A. S. Arena, Jr. for inspiring and encouraging me in my pursuit of enlightenment. This work never would have been completed without his understanding and patience as I moved across the country and lived in four different states during my work on this project over the past four years. I also want to thank the other members of my committee: Dr. B. T. Binegar, Dr. F. W. Chambers and Dr. D. G. Lilley.

But, no one deserves more thanks than my wife Leslie whose support, patience, and encouragement ultimately made this all possible. Her support and companionship were critical ingredients in providing me the strength to complete this work.

TABLE OF CONTENTS

Chapter	Page
1. INTRODUCTION.....	1
1.1 Background.....	1
1.2 Research Objective	4
1.3 Overview.....	5
2. PROBLEM DEFINITION	7
2.1 Choice of Governing Equations.....	7
2.2 The Compressible Euler Equations	12
2.3 Non-Inertial Formulation.....	13
2.4 Dimensionless Forms.....	18
3. METHODOLOGY	21
3.1 Overview of Finite Element Methods.....	22
3.2 Space-Time Formulation	26
3.3 Finite Element Discretization	28
3.3.1 One-Dimensional Elements.....	32
3.3.2 Two-Dimensional Elements	35
3.3.3 Three-Dimensional Elements	39
3.3.4 Element Edge Data.....	45
3.4 Boundary Conditions and the Boundary Integral	46
3.4.1 Symmetry Boundary Condition.....	47
3.4.2 Far-Field Boundary Condition	48
3.4.3 Solid Wall Boundary Condition	51
3.5 Predictor Multi-Corrector Algorithms.....	57
3.5.1 Steady Solutions	60
3.5.2 1 st Order Unsteady Solutions.....	62
3.5.3 2 nd Order Unsteady Solutions.....	65
3.6 Stabilization	67
3.6.1 SUPG and GLS	68
3.6.2 Artificial Dissipation	71
4. COMPUTER IMPLEMENTATION	74
4.1 Data Structures.....	75
4.2 Basic Algorithm.....	77
4.3 Available Codes.....	80

4.4	Algorithm Control	82
4.5	Aerodynamic Forces	84
4.6	Structural Dynamics	86
	4.6.1 Core Dynamics Algorithm	88
	4.6.2 Sampling Sensitivity.....	94
	4.6.3 Higher Order Algorithms	98
4.7	Summary.....	104
	4.7.1 Memory Requirements	105
	4.7.2 Computational Performance.....	107
5.	RESULTS.....	113
5.1	Verification	114
	5.1.1 Oblique Shock	116
	5.1.2 Prandtl-Meyer Expansion.....	127
	5.1.3 Converging-Diverging Nozzle	135
	5.1.4 Subsonic NACA-0012 Airfoil.....	141
	5.1.5 Conical Shock on a 10 degree Cone.....	154
	5.1.6 Unsteady Shock Tube.....	160
	5.1.7 Impulsively Accelerated Airfoil.....	169
	5.1.8 Pitching and Plunging Airfoil.....	177
	5.1.9 Spinning Centrifuge	185
5.2	Validation	188
	5.2.1 RAE 2822 Transonic Airfoil	190
	5.2.2 AGARD 445.6 Aeroelastic Test Wing.....	194
	5.2.3 Benchmark Active Controls Technology (BACT) Wing.....	204
	5.2.4 Vortex Generation Behind a Sharp-Edged Body	213
	5.2.5 80 Degree Delta Wing.....	220
	5.2.6 Hovering Rotor.....	230
6.	CONCLUSIONS.....	240
6.1	Discussion of Results.....	241
6.2	Future Challenges	242
	BIBLIOGRAPHY	248
	APPENDIX A: Summary of 2-D File Formats.....	254
	Geometry Input File (case . g2d).....	255
	Solver Control Input File (case . con)	257
	Solution Unknowns Input/Output File (case . un*).....	259
	Dynamic Mesh Input File (case . dyn)	260
	Elastic Vectors Input File (case . vec).....	263
	Solution Residuals Output File (case . rsd)	266
	Aerodynamic Loads Output File (case . lds).....	268
	Dynamic Output File (xd . dat)	270
	Elastic Output File (xn . dat)	272
	APPENDIX B: Summary of 3-D File Formats.....	274

Geometry Input File (case . g3d).....	275
Solver Control File (case . con).....	277
Solution Unknowns Input/Output File (case . un*).....	280
Dynamic Mesh Input File (case . dyn)	281
Elastic Vectors Input File (case . vec).....	284
External Force Input File (case . frc)	287
Solution Residuals Output File (case . rsd)	289
Aerodynamic Loads Output File (case . lds).....	291
Dynamic Output File (xd . dat)	293
Elastic Output File (xn . dat)	295
APPENDIX C: Time History Data For AGARD Wing.....	297
Time History Data for Mach 0.96, $dt = 16.0$	298
Time History Data for Mach 0.96, $dt = 4.0$	300
Time History Data for Mach 0.96, $dt = 1.0$	302
Time History Data for Mach 0.96, $dt = 0.25$	304
APPENDIX D: Steady Validation Data For BACT Wing.....	306
C_p Data for Mach 0.51	307
C_p Data for Mach 0.67	309
C_p Data for Mach 0.71	311
C_p Data for Mach 0.77	313
C_p Data for Mach 0.80	315
C_p Data for Mach 0.82	317
APPENDIX E: Steady Validation Data For 80 Degree Delta Wing.....	319
C_p Data for Mach 0.30, Roll Angles of 0 and 10 Degrees.....	320
C_p Data for Mach 0.30, Roll Angles of 20 and 30 degrees.....	321
C_p Data for Mach 0.30, Roll Angles of 40 and 50 degrees.....	322
C_p Data for Mach 0.30, Roll Angles of 60 and 70 degrees.....	323
C_p Data for Mach 1.20, Roll Angles of 0 and 10 Degrees.....	324
C_p Data for Mach 1.20, Roll Angles of 20 and 30 Degrees.....	325
C_p Data for Mach 1.20, Roll Angles of 40 and 50 Degrees.....	326
C_p Data for Mach 1.20, Roll Angles of 60 and 70 Degrees.....	327
VITA	328

LIST OF TABLES

Table	Page
Table 3.1: Gauss points and weights for one-dimensional elements.	34
Table 3.2: Gauss points and weights for two-dimensional elements.	38
Table 3.3: Gauss points and weights for three-dimensional elements.	44
Table 4.1: Summary of solver parameters for unsteady solution performance tests.	110
Table 5.1: Summary of grid parameters for oblique shock problem.	118
Table 5.2: Summary of average error downstream of the shock for all solutions to the oblique shock at Mach 2.5.	121
Table 5.3: Summary of one-point, low-order dissipation solution to the oblique shock at Mach 6.0.	125
Table 5.4: Summary of grid parameters for Prandtl-Meyer expansion.	128
Table 5.5: Summary of average error downstream of the shock for all solutions to the Prandtl-Meyer expansion at Mach 2.5.	132
Table 5.6: Summary of grid parameters for converging-diverging nozzle.	136
Table 5.7: Summary of Mach number at inlet and exit of converging-diverging nozzle.	140
Table 5.8: Summary of grid parameters for NACA 0012 airfoil.	142
Table 5.9: Summary of sectional lift and drag coefficients for the NACA 0012 airfoil at Mach 0.30 and zero angle of attack.	146
Table 5.10: Summary of sectional lift coefficients for the NACA 0012 airfoil at Mach 0.30 and 5.0 degree angle of attack.	153
Table 5.11: Summary of average pressure and Mach number on the surface of the 10 degree cone at Mach 2.35.	160

Table 5.12: Summary of solver control parameters used for time step refinement with unsteady shock tube.....	165
Table 5.13: Summary of solver control parameters used for time step refinement with the impulsively accelerated airfoil.	172
Table 5.14: Summary of solver control parameters used for time step refinement with AGARD wing at Mach 0.96.....	202
Table 5.15: Summary of computed flutter velocity for the AGARD wing at Mach 0.96.	204
Table 5.16: Measured frequency, damping and stiffness reported for the BACT wing.	208
Table 5.17: Summary of experimental and computed flutter characteristics for the BACT wing at Mach 0.82.	212
Table 5.18: Summary of computed Strouhal numbers for the wedge solution.....	219
Table 5.19: Summary of solver angle parameters for various roll angles with a constant 30° angle of attack.....	223

LIST OF FIGURES

Figure	Page
Figure 2.1: Vector relationship between inertial and non-inertial reference frames.....	13
Figure 3.1: Variation in time of unknowns using a constant-in-time approximation.	29
Figure 3.2: Typical one-dimensional finite element in natural coordinates.....	32
Figure 3.3: Typical two-dimensional finite element in natural coordinates.	35
Figure 3.4: Typical three-dimensional finite element in natural coordinates.	40
Figure 3.5: Illustration of an average surface normal for the trailing edge of an airfoil..	52
Figure 3.6: Illustration of transpiration concept.....	53
Figure 3.7: Model problem for testing transpiration with non-inertial rotation.....	54
Figure 3.8: Time history for the pitch angle of a rotating airfoil.	56
Figure 3.9: Comparison of predicted pressure coefficient for a pitching airfoil.....	57
Figure 4.1: Pseudo-code summary of core CFD algorithm.	78
Figure 4.2: Unsteady Solution Flow Chart.....	87
Figure 4.3: Block diagram representing a discrete-time aeroelastic system.	95
Figure 4.4: Comparison between continuous and discrete models for a one-dimensional aeroelastic model problem.....	97
Figure 4.5: Error comparison between several discrete-time models for the one- dimensional aeroelastic model problem.	99
Figure 4.6: Error comparison between several discrete-time models for the two- dimensional aeroelastic model problem.	101
Figure 4.7: Error comparison between several discrete-time models for the two- dimensional aeroelastic CFD problem.	102

Figure 4.8: Plot of absolute difference between the zero-order and second-order integrators for each mode of the aeroelastic CFD problem.....	103
Figure 4.9: Comparison of computational performance of old solver and new solver for various operating system/processor combinations.	108
Figure 4.10: Comparison of residual convergence histories of old and new CFD solvers for two different free stream mach numbers.	109
Figure 4.11: Comparison of computational performance for different types of unsteady solutions with new CFD solver.	111
Figure 4.12: Comparison of CPU time for different geometries.	112
Figure 5.1: Layout of computational domain for oblique shock problem.	117
Figure 5.2: Representative surface triangulation for oblique shock.....	118
Figure 5.3: Summary of solver control parameters for oblique shock at Mach 2.5.....	119
Figure 5.4: Plot of residual histories for oblique shock at Mach 2.5.	120
Figure 5.5: Pressure and Mach distributions for one-point, low-order dissipation solution to the oblique shock at Mach 2.5.....	122
Figure 5.6: Pressure and Mach distributions for one-point, high-order dissipation solution to the oblique shock at Mach 2.5.....	123
Figure 5.7: Summary of solver control parameters for oblique shock at Mach 6.0.....	124
Figure 5.8: Plot of residual histories for oblique shock at Mach 6.0.	125
Figure 5.9: Pressure and Mach distributions for one-point, low-order dissipation solution to the oblique shock at Mach 6.0.....	126
Figure 5.10: Layout of computational domain for Prandtl-Meyer expansion.....	128
Figure 5.11: Representative surface triangulation for Prandtl-Meyer expansion.	129
Figure 5.12: Summary of solver control parameters for oblique shock at Mach 2.5.....	130
Figure 5.13: Plot of residual histories for Prandtl-Meyer expansion at Mach 2.5.	131
Figure 5.14: Pressure and Mach distributions for one-point, low-order dissipation solution to the Prandtl-Meyer at Mach 2.5.....	133
Figure 5.15: Pressure and Mach distributions for one-point, high-order dissipation solution to the Prandtl-Meyer at Mach 2.5.....	134

Figure 5.16: Layout of computational domain for converging-diverging nozzle.....	136
Figure 5.17: Representative surface triangulation for converging-diverging nozzle.....	136
Figure 5.18: Summary of solver control parameters for converging-diverging nozzle..	137
Figure 5.19: Plot of residual histories for converging-diverging nozzle.	137
Figure 5.20: Colored Mach profile for converging-diverging nozzle.....	138
Figure 5.21: Mach distributions for one-point, low-order and high-order dissipation solutions to the converging-diverging nozzle.....	139
Figure 5.22: Layout of computational domain for NACA 0012 airfoil.....	141
Figure 5.23: Representative grid for the NACA 0012 airfoil.	142
Figure 5.24: Close-up of coarse and medium surface grids near the NACA 0012 airfoil.	143
Figure 5.25: Summary of solver control parameters for NACA 0012 airfoil.....	143
Figure 5.26: Plot of residual histories for the NACA 0012 airfoil at Mach 0.3 and zero angle of attack.....	144
Figure 5.27: Comparison of pressure coefficient distributions for NACA 0012 airfoil at Mach 0.3 and zero angle of attack.....	145
Figure 5.28: Summary of solver control parameters for transpiration solutions with the NACA 0012 airfoil.....	149
Figure 5.29: Head of elastic vectors file for the NACA 0012 airfoil with a specified mode one generalized velocity.....	150
Figure 5.30: Plot of residual histories for the NACA 0012 airfoil at Mach 0.3 and 5.0 degree angle of attack.....	151
Figure 5.31: Comparison of pressure coefficient distributions for NACA 0012 airfoil at Mach 0.3 and 5.0 degree angle of attack.....	152
Figure 5.32: Layout of computational domain for the 10 degree cone.....	156
Figure 5.33: Surface triangulation for 10 degree cone.....	157
Figure 5.34: Close-up of surface triangulation at the apex of 10 degree cone.....	157
Figure 5.35: Summary of solver control parameters for 10 degree cone.....	158
Figure 5.36: Plot of residual histories for 10 degree cone.....	158

Figure 5.37: Pressure and Mach distributions on the surface of the 10 degree cone at Mach 2.35.....	159
Figure 5.38: Layout of computational domain for unsteady shock tube.....	161
Figure 5.39: Surface triangulation for unsteady shock tube.	161
Figure 5.40: Summary of solver control parameters for unsteady shock tube.....	162
Figure 5.41: Initial conditions for unsteady shock tube.....	162
Figure 5.42: Plot of residual histories for 1 st and 2 nd order solution to the unsteady shock tube using a dimensionless time step of 0.001.....	163
Figure 5.43: Density, pressure, and Mach distributions for 1 st and 2 nd order solution to the unsteady shock tube at $t^* = 0.2$ using a dimensionless time step of 0.001.....	164
Figure 5.44: Plot of residual histories for 1 st and 2 nd order solution to the unsteady shock tube using various dimensionless time steps.....	166
Figure 5.45: Density, pressure, and Mach distributions for 1 st and 2 nd order solution to the unsteady shock tube at $t^* = 0.2$ using various dimensionless time steps.....	167
Figure 5.46: Surface triangulation with wake refinement for impulsively accelerated airfoil.	170
Figure 5.47: Close-up of surface triangulation for impulsively accelerated airfoil.	170
Figure 5.48: Summary of solver control parameters for impulsively accelerated airfoil.	171
Figure 5.49: Plot of residual histories for impulsively accelerated airfoil using various dimensionless time steps.	172
Figure 5.50: Plot of lift coefficient variation for the impulsively accelerated airfoil using various dimensionless time steps.....	173
Figure 5.51: Colored Mach profile showing unsteady wake development for the impulsively accelerated airfoil.	174
Figure 5.52: Summary of solver control parameters for impulsively accelerated airfoil in a non-inertial frame.	175
Figure 5.53: Dynamics file for the impulsively accelerated airfoil in a non-inertial frame.	176
Figure 5.54: Comparison of transient lift coefficient for the impulsively accelerated airfoil solved using inertial and non-inertial formulations.....	177

Figure 5.55: Summary of solver control parameters for the plunging airfoil.	178
Figure 5.56: Dynamics input file for the plunging airfoil.	179
Figure 5.57: Response time history for the plunging airfoil.	180
Figure 5.58: Plot of residual time histories for the plunging airfoil solution.	180
Figure 5.59: Comparison of lift coefficient time histories for the plunging airfoil.	181
Figure 5.60: Dynamics input file for the pitching airfoil.	182
Figure 5.61: Plot of residual time histories for the pitching airfoil solution.	183
Figure 5.62: Comparison of lift coefficient time histories for the pitching airfoil.	184
Figure 5.63: Summary of solver control parameters for spinning centrifuge.	186
Figure 5.64: Dynamics file for the spinning centrifuge.	187
Figure 5.65: Comparison of computed and theoretical pressure and density distributions along the centerline of the spinning centrifuge after four revolutions.	188
Figure 5.66: Layout of computational domain for the RAE 2822 transonic airfoil.	191
Figure 5.67: Close-up of surface grid near the RAE-2822 airfoil.	192
Figure 5.68: Summary of solver control parameters for transonic airfoil solution.	192
Figure 5.69: Plot of residual histories for transonic airfoil solution.	193
Figure 5.70: Plot of pressure coefficient distribution on the surface of the transonic airfoil at Mach 0.729 and angle of attack 2.31 degrees.	194
Figure 5.71: Close-up of surface grid for AGARD 445.6 aeroelastic test wing.	195
Figure 5.72: Summary of solver control parameters for the AGARD steady solution. ..	196
Figure 5.73: Plot of steady pressure coefficient distribution on the surface of the AGARD wing at Mach 0.96 for various spanwise locations, η	197
Figure 5.74: Summary of solver control parameters for the AGARD aeroelastic solution.	198
Figure 5.75: Header of elastic vectors input file for the AGARD aeroelastic solution. .	198
Figure 5.76: External force input file for the AGARD aeroelastic solution.	199

Figure 5.77: Comparison between time history data computed by STARS and euler3d solvers for the AGARD wing at Mach 0.96.	200
Figure 5.78: Comparison of time history data for AGARD wing at Mach 0.96.	201
Figure 5.79: Plot of flutter velocity versus time step for the AGARD wing at Mach 0.96.	203
Figure 5.80: Layout of computational domain for BACT wing.	205
Figure 5.81: Close-up of surface grid for BACT wing.	206
Figure 5.82: Summary of solver control parameters for the steady BACT wing solutions.	207
Figure 5.83: Comparison of steady surface pressure distribution at the 60% span location for the BACT wing at Mach 0.82 and zero angle of attack.	207
Figure 5.84: Summary of solver control parameters for the unsteady BACT wing solutions.	209
Figure 5.85: Dynamics input file for the unsteady BACT wing solutions.	209
Figure 5.86: Aeroelastic response of BACT wing at Mach 0.82 for a free-stream dynamic pressure of 144 psf.	210
Figure 5.87: Pitch response of BACT wing at Mach 0.82 for various free-stream dynamic pressures using $idsol = 0$	211
Figure 5.88: Surface triangulation around wedge-shaped body.	214
Figure 5.89: Close-up of surface triangulation near wedge-shaped body.	214
Figure 5.90: Summary of solver control parameters for the wedge solution.	215
Figure 5.91: Contour plot showing vortex generation behind a wedge-shaped cylinder.	216
Figure 5.92: Plot of lift coefficient time history data for the wedge solution.	217
Figure 5.93: Comparison of lift coefficient time history data for the wedge solution with various outer radius dimensions.	218
Figure 5.94: Plot of Strouhal number versus outer radius of the computational domain for the wedge solution.	219
Figure 5.95: Layout of computational domain for 80 degree delta wing.	221
Figure 5.96: Surface triangulation for 80 degree delta wing.	221

Figure 5.97: Summary of solver control parameters for the steady delta wing solutions.	222
Figure 5.98: Plot of velocity vectors along a normal cut plane at $x/c = 0.60$ for the Mach 0.30 delta wing.	223
Figure 5.99: Plot of velocity vectors along a normal cut plane at $x/c = 0.60$ for the Mach 1.20 delta wing at a 30° angle of attack.....	225
Figure 5.100: Summary of solver control parameters for the unsteady free to roll delta wing solutions.....	226
Figure 5.101: Dynamics input file for the unsteady free to roll delta wing solution.	226
Figure 5.102: Roll time history data for the Mach 1.20 delta wing at a 30° angle of attack.	227
Figure 5.103: Plot of roll moment coefficient versus roll angle for the Mach 1.20 delta wing at a 30° angle of attack.	228
Figure 5.104: Roll time history data for the Mach 1.20 delta wing at a 10° angle of attack.	229
Figure 5.105: Plot of roll moment coefficient versus roll angle for the Mach 1.20 delta wing at a 10° angle of attack.	229
Figure 5.106: Layout of computational domain for hovering rotor.	231
Figure 5.107: Close-up of surface grid for hovering rotor.....	231
Figure 5.108: Summary of solver control parameters for the hovering rotor with tip Mach 0.520.	232
Figure 5.109: Dynamics input file for the hovering rotor with tip Mach 0.520.....	233
Figure 5.110: Plots of z -force time history for the hovering rotor with a 2° collective pitch angle and a tip Mach number of 0.520.	234
Figure 5.111: Comparison of surface pressure distributions after various revolutions for a hovering rotor with a 2° collective pitch and tip Mach number of 0.520.	234
Figure 5.112: Plot of surface pressure distribution for a hovering rotor with a 2° collective pitch and tip Mach number of 0.520 after eight revolutions.....	235
Figure 5.113: Plots of z -force time history for the hovering rotor with an 8° collective pitch angle and a tip Mach number of 0.439.	236
Figure 5.114: Comparison of surface pressure distributions after various revolutions for a hovering rotor with an 8° collective pitch and tip Mach number of 0.439.	237

Figure 5.115: Plot of surface pressure distribution for a hovering rotor with an 8° collective pitch and tip Mach number of 0.439 after five revolutions. 237

Figure 5.116: Comparison of computed surface pressure distribution for a hovering rotor using an actual and simulated 2° collective pitch angle for a tip Mach number of 0.520. 238

Figure 5.117: Comparison of computed surface pressure distribution for a hovering rotor using an actual and simulated 8° collective pitch angle for a tip Mach number of 0.439. 239

CHAPTER 1

1. INTRODUCTION

This dissertation presents a space-time finite element method for solving the compressible Euler equations in a non-inertial reference frame. The methodology developed here has been formulated with an emphasis on solving problems that are found in aerospace applications. In particular, we are interested in modeling aeroservoelastic interactions for complicated three-dimensional problems such as fighter aircraft.

In this Chapter we discuss the background behind our current interest in the modeling of aeroservoelastic problems. This includes a discussion on the computational tools we seek to enhance through the development of a non-inertial finite element methodology. We then proceed by defining the objectives of this research effort, and finish the introduction with a brief overview for the remaining Chapters of this document.

1.1 Background

Modern high-performance aerospace vehicles are highly maneuverable, operate over a wide range of speeds, in some cases hypersonic speeds, and are designed to have lightweight, sometimes flexible structures. Examples of such vehicles currently being developed include the X-33 single stage to orbit vehicle, the X-43 or Hyper-X, and the X-34 reusable launch vehicle technology demonstrator. Unfortunately, this type of vehicle

often encounters aeroservoelastic (ASE) instabilities during part of its flight profile as a result of complicated aerodynamic, elastic, inertial, and control interactions. Hence, the accurate prediction of such interactions prior to flight testing is a necessary part of the design process.

In terms of accuracy, the most attractive model for the aerodynamic interaction in an ASE analysis is a computational fluid dynamics (CFD) model. Unsteady CFD solutions provide an accurate physical model of the flow field for all flight regimes with the ability to account for nonlinear generation and unsteady movement of shock waves when dealing with compressible flows. Traditionally, the use of such models in an ASE analysis has been limited to academic research due to long computational times, especially for the three-dimensional models of complicated aerospace vehicles. However, this limitation is rapidly diminishing as technological advances provide affordable, faster computers each year. Additionally, research into synergistic combinations of aerodynamic modeling techniques and CFD have demonstrated a significant reduction in computational times for aeroelastic simulations.^{1,2,3} Thanks to these developments, CFD is now more likely to be applied to ASE analysis not just by academic researchers, but also by aircraft designers in an operational environment.

The capability for completely coupled, nonlinear aeroservoelastic analysis has recently been integrated into STARS (S^TStructural Analysis RoutineS) developed by Gupta at the NASA Dryden Flight Research Center. STARS is an highly integrated, finite element based code for multidisciplinary analysis of flight vehicles including static and dynamic structural analysis, computational fluid dynamics, heat transfer, and aeroservoelastic capabilities.⁴ The core of the nonlinear ASE analysis routine in STARS

is a finite element CFD algorithm based on a time-marched solution to the unsteady, compressible Euler equations. This routine is capable of simulating the time accurate fluid-structure interactions for arbitrary three-dimensional geometries interacting with inviscid, compressible flows. The accuracy of this routine has been demonstrated in the literature for both aeroelastic and aeroservoelastic analysis of several flight vehicle configurations.^{5,6}

One of the interesting features of the STARS ASE analysis routine is the methodology used to apply the unsteady CFD boundary conditions for an elastically deforming structure. Rather than utilizing a computationally expensive moving grid algorithm or attempting to re-grid the problem at each time step, STARS uses transpiration to simulate the correct boundary conditions for the deformed structure by updating only the wall surface normals at each time step. Obviously, this methodology is more efficient than attempting to update all of the nodes in the computational domain by some other method, such as a moving mesh algorithm. At the same time, research has demonstrated that STARS CFD results employing the transpiration boundary condition are in excellent agreement with those obtained using actually deformed grids for the small deformations typically seen in ASE simulations.⁷

Unfortunately, the transpiration boundary condition will actually limit the type of applications the STARS ASE routines are capable of analyzing. Simulations involving large amplitude motion cannot be modeled accurately using this method. Common examples of these applications include spinning structures such as propellers and turbines, rigid body translation and rotation of fighter aircraft under combat and maneuvering conditions, and free-to-roll oscillations of delta wings or “wing-rock.” For

such applications, it would be convenient to solve the governing CFD equations in a non-inertial reference frame, thus allowing for uniform motion of the entire computational grid. This formulation should still be substantially faster than updating each individual node in an arbitrarily deforming grid, and may also be combined with the transpiration boundary condition to include elastic deformation of the local surface geometry.

1.2 Research Objective

The objective of this research effort will be to extend the capabilities of the STARS ASE analysis routine by incorporating the capability for unsteady CFD analysis in a non-inertial reference frame. Hence, we will focus on the development of an efficient and accurate finite element methodology for solving the compressible Euler equations expressed in non-inertial coordinates. Although this is non-trivial and arguably a complicated task, it also is not a unique or novel topic of research as it has been accomplished in the past. Several finite volume based codes already exist which employ a non-inertial reference frame, and at least one finite element based code allows for a rotating (but non-accelerating) frame of reference.

The unique aspects of this project involve the more general goal of developing the capacity for an aeroservoelastic analysis of a super-maneuvering or spinning structure. This type of multidisciplinary analysis will rely on the integration of the aforementioned finite element CFD algorithm into the existing STARS framework and require that the algorithm developed be as general as possible. To solve the most general aeroservoelastic problem, the finite element CFD algorithm will be required to account not only for velocities and accelerations produced by rigid body motion of the non-

inertial reference frame, but also simulate elastic deformations and control surface deflections through an appropriately modified transpiration boundary condition that accounts for an arbitrarily rotating coordinate system. The resulting code is potentially a general purpose flight simulator capable of accurately modeling an aircraft configuration performing any sort of arbitrary dynamic flight maneuver. However, the simulation will obviously not be performed in “real-time” due to the immense computational requirements of a large-scale CFD analysis.

1.3 Overview

Our discussion begins in Chapter 2 with an examination of the equations we seek to solve. This includes a derivation of the non-inertial Euler equations in conservation form. Chapter 3 then discusses the details of discretizing and numerically solving the governing equations using advanced finite element methods. Although many aspects of the current STARS unsteady CFD module will be retained, we do not simply start by modifying the existing algorithm to accommodate a non-inertial frame. Rather, we seek to develop a new algorithm by examining state of the art finite element methods currently in use for computational fluid dynamics research. In fact, we will find that the current STARS unsteady CFD module is unsuitable for our purposes due to the limitations of its edge-based data structure.

After evaluating a variety of methods, we develop a space-time finite element algorithm which re-uses some of the best aspects of the STARS methodology, but is modified appropriately for a non-inertial formulation. In order to facilitate future enhancements to this algorithm, Chapter 3 is intended to provide the reader with all of the

relevant theory and equations needed to reconstruct the methodology presented in this document. Furthermore, Chapter 4 presents many of the details of how the finite element methodology is implemented as a working computer algorithm, including details on the solution of the structural dynamics equations not presented in Chapter 3. In Chapter 5, we attempt a rigorous verification and validation of the CFD algorithm for a wide range of flow problems. Finally, Chapter 6 presents conclusions and makes recommendations for future research.

CHAPTER 2

2. PROBLEM DEFINITION

In this Chapter we review the governing equations that define our problem. We begin by discussing the choice of governing equations used in this study, namely the compressible Euler equations. This includes a brief literature review investigating the trade-offs associated with using the Euler equations rather than the Navier-Stokes equations for aeroelastic simulations. In the next section we present the classic form of the compressible Euler equations which will serve as the basis for deriving our numerical algorithm. This is followed by a discussion of the transformations necessary to re-cast the equations for use in a non-inertial formulation, and the appropriate dimensionless form of the governing equations needed for development of a numerical algorithm.

2.1 Choice of Governing Equations

The emphasis of the present work is to extend the capabilities of the STARS nonlinear ASE module to include CFD analysis in a non-inertial reference frame. As mentioned previously, the current unsteady CFD module uses a time-marched approach to solving the unsteady, compressible Euler equations. Although the Navier-Stokes equations represent the most precise mathematical model for fluid flow, the Euler equations adequately model most of the physical characteristics of transonic and

supersonic flows with the obvious exception of substantial viscous effects. Flow features of particular interest for compressible flows are the generation and motion of shock waves, entropy increases across shocks, and vorticity production and convection behind shocks, all of which can be successfully modeled using the Euler equations.^{8,9}

One of the primary motivations for employing the Euler equations rather than the full Navier-Stokes equations is computational efficiency. We expect that any numerical solution to the Euler equations will be faster than a similar solution to the Navier-Stokes equations since there are fewer terms to compute. In fact, Roache¹⁰ has suggested that, “one could swallow up one order of magnitude (increase in computing power) with Reynolds Averaged Navier-Stokes equations and a two-equation model of turbulence.” This problem is compounded by the fact that Navier-Stokes solutions typically require greater grid resolution, and even small increases in grid resolution swallow up enormous amounts of computing power.

If one were ambitious enough to be interested in direct numerical simulation (DNS) of aerospace flows using the Navier-Stokes equations and brute-force calculation of turbulence, multiple references have suggested that the required number of grid points is approximately proportional to the Reynolds number raised to the $9/4$ power. With this relationship in mind, Moin¹¹ estimates that about 10^{16} grid points would be required to model a typical transport aircraft cruising at 250 meters per second. Furthermore, he estimates that even with teraflop supercomputers it would take several thousand years to compute the flow for one second of flight time.¹¹

To date, only simple, low Reynolds number flows such as plate, channel, and step flows have been successfully analyzed using DNS, and only at an extreme computational

expense using parallel supercomputers.^{12,13} Because of this, any practical viscous solution employs the Reynolds Averaged Navier-Stokes (RANS) equations plus some form of turbulence model. Although this averaged form of the Navier-Stokes equations does relax the grid spacing requirements when compared to DNS, it still requires significantly more grid points over comparable Euler solutions to accurately resolve the boundary layer. Considering that many of the existing STARS CFD models consist of several hundred thousand elements and take on the order of weeks or even months to analyze using an Euler solution, increasing the computational time by incorporating a Navier-Stokes model does not appear to be practical at this time.

It is important, however, to consider what physical aspects of the problem may be lost when the viscous terms in the Navier-Stokes equations are neglected. Some of the more interesting features that the full Navier-Stokes equations are capable of predicting include boundary layer development and interaction with shocks, as well as turbulence generation and vorticity shedding. In general, these flow features cannot be modeled by the Euler equations since they are inherently viscous effects. This is a significant problem if the expressed purpose for modeling a particular aircraft is to search for these specific flow features. However, the primary use of the STARS unsteady CFD module is in coupled aeroelastic and aeroservoelastic analysis. Hence it may be more relevant to question what accuracy might be lost in a typical ASE analysis as a result of neglecting the viscous terms from the Navier-Stokes equations.

In a typical CFD-based ASE analysis, the dynamic response of a structure is computed by solving Equation (2.1), the matrix equation of motion for an arbitrary structure in generalized coordinates.

$$(2.1) \quad \mathbf{M}\ddot{\mathbf{x}}(t) + \mathbf{C}\dot{\mathbf{x}}(t) + \mathbf{K}\mathbf{x}(t) = \mathbf{f}_a(t)$$

In the above relation, $\mathbf{x} = \{x_1, x_2, \dots, x_n\}^T$ is a vector of n generalized displacements, \mathbf{f}_a is a vector of n generalized aerodynamic forces, and \mathbf{M} , \mathbf{C} and \mathbf{K} are $n \times n$ matrices of generalized mass, damping, and stiffness coefficients respectively.

Notice in Equation (2.1) that the dynamics of an aeroelastic problem is driven by an unsteady aerodynamic force vector, the calculation of which involves an integration of pressure over the surface of the structure. With this in mind, it is often argued that small or localized pressure fluctuations due to viscous effects should not significantly impact aeroelastic stability since their integrated effect would be negligible.

Obviously this will not be true for all classes of problems, but for many aerospace applications, excluding high angle of attack problems, this does seem to be the case. One example of such an application is the AGARD 445.6 aeroelastic test wing. Gupta⁵ modeled this wing configuration in order to demonstrate the capabilities of the STARS ASE module. Although the results were not quantitatively precise, the qualitative trends accurately matched those observed during the original wind tunnel testing at the Langley Transonic Dynamics tunnel, including the transonic dip in the flutter boundary near Mach 1.0. These results seem to indicate that the Euler solution is capable of modeling the dominant flow physics which drive this aeroelastic problem, while the small numerical discrepancy is most likely the result of structural modeling uncertainty along with some viscous effects. In addition to the AGARD, a variety of other aeroelastic simulations have been performed using STARS, including the BACT wing.⁶ Recent efforts at NASA are focused on modeling the Hyper-X supersonic vehicle using STARS.

Another interesting numerical study was completed by Steger and Bailey.¹⁴ It involved the prediction of a single degree-of-freedom aeroelastic phenomenon known as buzz. One of the more interesting results from their simulation was a comparison between the buzz boundary predicted by a viscous and inviscid flow solver. The viscous solution predicted buzz at both Mach 0.82 and 0.83, while the inviscid solution predicted buzz slightly later at Mach 0.84. Furthermore, a limit cycle oscillation was predicted by the viscous flow model while the inviscid model predicted exponential divergence. On the difference between the observed limit cycle oscillations and exponential divergence, Steger and Bailey concluded that: "...while inviscid unsteady shock wave motion is the driving force of transonic aileron buzz, the viscosity is nevertheless crucial and can both sustain and moderate the flap motion."

Again, the conclusion is that an inviscid solution is entirely capable of predicting the dominate flow physics which initially drive an aeroelastic problem, or at least enough to obtain a reasonable approximation for a stability boundary. The viscosity, on the other hand, seems to have a nonlinear effect on the rate of divergence for an unstable time history, making limit cycles and other nonlinear behavior possible. That is not to say that an inviscid solution cannot predict nonlinear behavior as well though. As discussed at the beginning of this section, an Euler solution is entirely capable of predicting nonlinear generation and motion of shock waves.

For the remainder of this research, it is assumed that an Euler solution is appropriate for modeling many aeroelastic problems, and it is left for individual researchers to decide whether viscosity will play a dominate role in their particular problem.

2.2 The Compressible Euler Equations

The compressible Euler equations are fundamental to the subject of inviscid flow. As such, their derivation from the three conservation laws; conservation of mass, momentum and energy; can be found in most introductory texts on fluid dynamics. Here, the compressible Euler equations are presented in their most commonly used form, the so-called conservative variables formulation. Using indicial notation, a single vector equation representing the compressible Euler equations is written as follows:

$$(2.2) \quad \frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x_i} = 0$$

where, in three dimensions,

$$(2.3) \quad \mathbf{U} = \begin{Bmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ \rho e \end{Bmatrix}$$

$$(2.4) \quad \mathbf{F}_i = \begin{Bmatrix} \rho u_i \\ \rho u_i u_1 \\ \rho u_i u_2 \\ \rho u_i u_3 \\ \rho u_i e \end{Bmatrix} + \begin{Bmatrix} 0 \\ p \delta_{1i} \\ p \delta_{2i} \\ p \delta_{3i} \\ p u_i \end{Bmatrix}$$

In the above equations, \mathbf{U} is the Euler unknowns vector, \mathbf{F}_i is the i^{th} component of the Euler flux vector, ρ is the fluid density, $\mathbf{u} = \{u_1, u_2, u_3\}^T$ is the fluid velocity vector, e is the total energy per unit mass, p is the thermodynamic pressure, and δ_{ij} is the Kronecker delta (i.e. $\delta_{ij} = 1$ for $i = j$, and $\delta_{ij} = 0$ for $i \neq j$).

Along with Equation (2.2), we will also require the following constitutive equations relating the thermodynamic pressure and total enthalpy per unit mass, h , to the Euler unknowns:

$$(2.5) \quad p = \rho(\gamma - 1) \left(e - \frac{1}{2} |\mathbf{u}|^2 \right)$$

$$(2.6) \quad h = e + p/\rho = \frac{\mathcal{H}}{\rho(\gamma - 1)} + \frac{1}{2} |\mathbf{u}|^2$$

In the above equations, γ is the ratio of specific heats, which is assumed to be constant for a given problem.

2.3 Non-Inertial Formulation

The Euler equations presented in the previous section were formulated in terms of an inertial or stationary reference frame. In this section we will extend these equations to account for the arbitrary motion of a non-inertial reference frame. Consider the layout presented in Figure 2.1, where XYZ represents an inertial reference frame and xyz represents a non-inertial reference frame which is free to translate and rotate in all three dimensions.

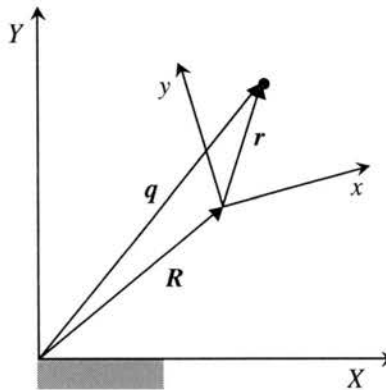


Figure 2.1: Vector relationship between inertial and non-inertial reference frames.

The position vector, \mathbf{q} , for a particle located in the non-inertial reference frame can be expressed in terms of the inertial reference frame by Equation (2.7).

$$(2.7) \quad \mathbf{q}_{XYZ} = \mathbf{R}_{XYZ} + \mathbf{B}\mathbf{r}_{xyz}$$

In the above relation, \mathbf{B} is a coordinate transformation matrix that maps a particular coordinate location in the non-inertial frame, xyz , to the inertial frame, XYZ . In three-dimensions, the matrix \mathbf{B} may be defined as follows:

$$(2.8) \quad \mathbf{B} = \begin{bmatrix} C_\theta C_\psi & S_\phi S_\theta C_\psi - C_\phi S_\psi & C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\theta S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi \\ -S_\theta & S_\phi C_\theta & C_\phi C_\theta \end{bmatrix}$$

where the angles θ , ψ and ϕ are the classic Euler angles used in flight dynamics to describe the orientation of an aircraft.¹⁵

Next, consider the time rate of change of Equation (2.7).

$$(2.9) \quad \frac{d\mathbf{q}_{XYZ}}{dt} = \frac{d\mathbf{R}_{XYZ}}{dt} + \frac{d\mathbf{B}}{dt}\mathbf{r}_{xyz} + \mathbf{B}\frac{d\mathbf{r}_{xyz}}{dt} = \mathbf{V}_0 + \dot{\mathbf{B}}\mathbf{r}_{xyz} + \mathbf{B}\mathbf{V}_r$$

In the above equation, \mathbf{V}_0 is the velocity vector for the origin of the non-inertial frame expressed in inertial coordinates, and \mathbf{V}_r is the relative velocity vector for the fluid particle expressed in non-inertial coordinates. For convenience, a matrix which has the property given by Equation (2.10) is introduced.

$$(2.10) \quad \mathbf{\Omega} = \mathbf{B}^{-1}\dot{\mathbf{B}}$$

where

$$(2.11) \quad \dot{\mathbf{B}} = \mathbf{B}\mathbf{\Omega}$$

For a rotating system, the angular velocity matrix, $\mathbf{\Omega}$, is defined by Equation (2.12).

$$(2.12) \quad \mathbf{\Omega r}_{xyz} = \mathbf{\omega} \times \mathbf{r}_{xyz} = \underbrace{\begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}}_{\mathbf{\Omega}} \begin{Bmatrix} r_x \\ r_y \\ r_z \end{Bmatrix}$$

where the angular rates, ω_x , ω_y and ω_z , are the angular velocities of the non-inertial frame expressed in non-inertial coordinates. These angular rates are equivalent to the body angular rates, p , q and r , typically used to describe the motion of an aircraft in flight dynamics.¹⁵

Now consider the time rate of change of Equation (2.9).

$$(2.13) \quad \frac{d}{dt} (\mathbf{V}_0 + \mathbf{B}\mathbf{\Omega r}_{xyz} + \mathbf{B}\mathbf{V}_r) = \frac{d\mathbf{V}_0}{dt} + \frac{d\mathbf{B}}{dt} \mathbf{\Omega r}_{xyz} + \mathbf{B} \frac{d\mathbf{\Omega}}{dt} \mathbf{r}_{xyz} + \mathbf{B}\mathbf{\Omega} \frac{d\mathbf{r}_{xyz}}{dt} + \frac{d\mathbf{B}}{dt} \mathbf{V}_r + \mathbf{B} \frac{d\mathbf{V}_r}{dt}$$

Simplifying Equation (2.13) yields the equation needed to define a velocity derivative in terms of its non-inertial components.

$$(2.14) \quad \frac{d\mathbf{V}}{dt} = \mathbf{a}_0 + \mathbf{B}\mathbf{\Omega}^2 \mathbf{r}_{xyz} + \mathbf{B}\dot{\mathbf{\Omega}} \mathbf{r}_{xyz} + 2\mathbf{B}\mathbf{\Omega} \mathbf{V}_r + \mathbf{B} \frac{d\mathbf{V}_r}{dt}$$

Notice that the derivation of Equation (2.14) was based on a Lagrangian description of motion and defines the total acceleration vector for a particle moving in a non-inertial reference frame. This expression is also valid in an Eulerian description of motion if the total time derivative is replaced by the particle, or substantial derivative.

$$(2.15) \quad \frac{D\mathbf{V}}{Dt} = \mathbf{a}_0 + \mathbf{B}\mathbf{\Omega}^2 \mathbf{r}_{xyz} + \mathbf{B}\dot{\mathbf{\Omega}} \mathbf{r}_{xyz} + 2\mathbf{B}\mathbf{\Omega} \mathbf{V}_r + \mathbf{B} \frac{D\mathbf{V}_r}{Dt}$$

For convenience, a transformation velocity vector, \mathbf{V}_t , and transformation acceleration vector, \mathbf{a}_t , both expressed in the inertial frame, are defined as follows:

$$(2.16) \quad \mathbf{V}_t = \mathbf{V}_0 + \mathbf{B}\mathbf{\Omega r}_{xyz}$$

$$(2.17) \quad \mathbf{a}_t = \frac{d\mathbf{V}_t}{dt} = \mathbf{a}_0 + \mathbf{B}\mathbf{\Omega}^2 \mathbf{r}_{xyz} + \mathbf{B}\dot{\mathbf{\Omega}} \mathbf{r}_{xyz} + \mathbf{B}\mathbf{\Omega} \mathbf{V}_r$$

The above equations allow us to rewrite the definition for the total velocity and acceleration vectors, Equations (2.9) and (2.15), in a more compact form as follows:

$$(2.18) \quad \mathbf{V} = \mathbf{V}_t + \mathbf{B}\mathbf{V}_r$$

$$(2.19) \quad \frac{D\mathbf{V}}{Dt} = \mathbf{a}_t + \mathbf{B}\boldsymbol{\Omega}\mathbf{V}_r + \mathbf{B}\frac{D\mathbf{V}_r}{Dt}$$

These equations will help us to cast the compressible Euler equations in terms of relative quantities expressed in non-inertial coordinates. To do so, we must first define a relative energy, e_r , by substituting the definition for the total velocity into the perfect gas law, Equation (2.5), as follows:

$$(2.20) \quad \rho e = \frac{P}{\gamma-1} + \frac{1}{2}\rho(\mathbf{V}_r \cdot \mathbf{V}_r) - \frac{1}{2}\rho(\mathbf{V}_t \cdot \mathbf{V}_t) + \rho \mathbf{V} \cdot \mathbf{V}_t = \rho e_r + \rho \mathbf{V} \cdot \mathbf{V}_t$$

where

$$(2.21) \quad e_r = e - \mathbf{V} \cdot \mathbf{V}_t$$

Using this definition for the relative energy, we now write our constitutive equations in terms of relative and transformation velocities as follows:

$$(2.22) \quad p = \rho(\gamma-1)\left(e_r - \frac{1}{2}|\mathbf{V}_r|^2 + \frac{1}{2}|\mathbf{V}_t|^2\right)$$

$$(2.23) \quad h_r = e_r + p/\rho = \frac{\gamma P}{\rho(\gamma-1)} + \frac{1}{2}|\mathbf{V}_r|^2 - \frac{1}{2}|\mathbf{V}_t|^2$$

where h_r is a relative enthalpy per unit mass.

Through careful substitution of the above non-inertial definitions into the compressible Euler equations for flow relative to an inertial frame, we are able to derive the following vector equation for the compressible Euler equations for flow relative to a non-inertial frame:

$$(2.24) \quad \frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x_i} = \mathbf{S}$$

where, in three dimensions,

$$(2.25) \quad \mathbf{U} = \begin{Bmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ \rho \mathbf{e}_r \end{Bmatrix}$$

$$(2.26) \quad \mathbf{F}_i = \begin{Bmatrix} \rho u_i \\ \rho u_i u_1 \\ \rho u_i u_2 \\ \rho u_i u_3 \\ \rho u_i \mathbf{e}_r \end{Bmatrix} + \begin{Bmatrix} 0 \\ p \delta_{1i} \\ p \delta_{2i} \\ p \delta_{3i} \\ p u_i \end{Bmatrix}$$

$$(2.27) \quad \mathbf{S} = -\rho \begin{Bmatrix} 0 \\ (\mathbf{B}^{-1} \mathbf{a}_i + \boldsymbol{\Omega} \mathbf{V}_r) \\ \mathbf{a}_i \cdot (\mathbf{V}_i + \mathbf{B} \mathbf{V}_r) \end{Bmatrix}$$

In the above relations, \mathbf{S} is the non-inertial source vector, and $\mathbf{u} = \{u_1, u_2, u_3\}^T$ now represents the relative fluid velocity vector expressed in non-inertial coordinates, which is equivalent to the relative velocity vector, \mathbf{V}_r , appearing in the non-inertial source vector and used in the derivation of the non-inertial quantities.

Notice that the transformation matrix, \mathbf{B} , and its inverse, \mathbf{B}^{-1} , are required in order to evaluate Equation (2.24). This is not a desirable definition to form the basis of an efficient numerical algorithm. Instead, let us redefine the transformation velocity and acceleration, \mathbf{V}_i and \mathbf{a}_i , respectively, so that they are vectors expressed in non-inertial coordinates. From Equations (2.16) and (2.17) the non-inertial transformation velocity and acceleration, \mathbf{V}'_i and \mathbf{a}'_i respectively, may be expressed in non-inertial coordinates as follows:

$$(2.28) \quad \mathbf{V}'_i = \mathbf{B}^{-1} \mathbf{V}_0 + \boldsymbol{\Omega} \mathbf{r}_{xyz}$$

$$(2.29) \quad \mathbf{a}'_i = \mathbf{B}^{-1} \mathbf{a}_0 + \boldsymbol{\Omega}^2 \mathbf{r}_{xyz} + \dot{\boldsymbol{\Omega}} \mathbf{r}_{xyz} + \boldsymbol{\Omega} \mathbf{V}_r$$

Taking advantage of the above definitions, we also define the following definition for the non-inertial source vector such that all vector quantities are expressed in terms of non-inertial coordinates:

$$(2.30) \quad \mathbf{S} = -\rho \begin{Bmatrix} 0 \\ \mathbf{a}'_i + \boldsymbol{\Omega} \mathbf{V}_r \\ \mathbf{a}'_i \cdot (\mathbf{V}'_i + \mathbf{V}_r) \end{Bmatrix}$$

In later sections, we will drop the prime notation and assume that all vector quantities are appropriately expressed in terms of non-inertial coordinates.

2.4 Dimensionless Forms

The successful application of a numerical algorithm to solve the compressible Euler equations requires that they first be cast in dimensionless form. This is accomplished by defining a dimensionless density, velocity, pressure, energy, time, and coordinate location as follows:

$$(2.31) \quad \rho^* = \rho / \rho_o$$

$$(2.32) \quad u_i^* = u_i / U_o$$

$$(2.33) \quad p^* = p / \rho_o U_o^2$$

$$(2.34) \quad e^* = e / U_o^2$$

$$(2.35) \quad t^* = t U_o / L_o$$

$$(2.36) \quad x^* = x / L_o$$

In the above expressions, ρ_o is the reference density, U_o is the reference velocity, L_o is a reference length, and the asterisk is used to denote dimensionless forms of the usual quantities.

Next, the above definitions are used to derive dimensionless forms for the unknowns vector and inviscid flux vectors.

$$(2.37) \quad \mathbf{U} = \begin{bmatrix} \rho_o & 0 & 0 & 0 & 0 \\ 0 & \rho_o U_o & 0 & 0 & 0 \\ 0 & 0 & \rho_o U_o & 0 & 0 \\ 0 & 0 & 0 & \rho_o U_o & 0 \\ 0 & 0 & 0 & 0 & \rho_o U_o^2 \end{bmatrix} \begin{bmatrix} \rho^* \\ \rho^* u^* \\ \rho^* v^* \\ \rho^* w^* \\ \rho^* e^* \end{bmatrix} = [\mathbf{U}_o] \mathbf{U}^*$$

$$(2.38) \quad \mathbf{F}_i = \begin{bmatrix} \rho_o U_o & 0 & 0 & 0 & 0 \\ 0 & \rho_o U_o^2 & 0 & 0 & 0 \\ 0 & 0 & \rho_o U_o^2 & 0 & 0 \\ 0 & 0 & 0 & \rho_o U_o^2 & 0 \\ 0 & 0 & 0 & 0 & \rho_o U_o^3 \end{bmatrix} \begin{bmatrix} \rho^* u_i^* \\ \rho^* u_i^* u_1^* + p^* \delta_{1i} \\ \rho^* u_i^* u_2^* + p^* \delta_{2i} \\ \rho^* u_i^* u_3^* + p^* \delta_{3i} \\ u_i^* (\rho^* e^* + p^*) \end{bmatrix} = [\mathbf{F}_o] \mathbf{F}_i^*$$

In the above expressions, \mathbf{U}^* is the dimensionless Euler unknowns vector, \mathbf{F}_i^* is the i^{th} component of the dimensionless Euler flux vector, and $[\mathbf{U}_o]$ and $[\mathbf{F}_o]$ are matrices defined such that they convert these dimensionless vectors back into their dimensional form.

Using Equations (2.37) and (2.38), the compressible Euler equations are now written as follows:

$$(2.39) \quad [\mathbf{U}_o] \frac{U_o}{L_o} \frac{\partial \mathbf{U}^*}{\partial t^*} + [\mathbf{F}_o] \frac{1}{L_o} \frac{\partial \mathbf{F}_i^*}{\partial x_i^*} = \mathbf{S}$$

The dimensionless form of the compressible Euler equations in non-inertial coordinates may then be expressed as follows:

$$(2.40) \quad \frac{\partial \mathbf{U}^*}{\partial t^*} + \frac{\partial \mathbf{F}_i^*}{\partial x_i^*} = \mathbf{S}^*$$

where

$$(2.41) \quad \mathbf{S} = \frac{U_o}{L_o} [\mathbf{U}_o] \mathbf{S}^*$$

For the remainder of this study, this dimensionless form will be used exclusively as the basis of our numerical algorithm. Therefore, we will drop the asterisks when referring to the compressible Euler equations and assume that all variables are in their appropriate dimensionless form.

To conclude this section, we present three additional dimensionless forms that are useful in the development of a solution for the compressible Euler equations. First, the local acoustic speed, a , is written as follows:

$$(2.42) \quad a^2 = \gamma \frac{p}{\rho}$$

Utilizing the above definition, the local Mach number, M , may be written as follows:

$$(2.43) \quad M^2 = \frac{|\mathbf{u}|^2}{a^2} = \frac{\rho |\mathbf{u}|^2}{\gamma p}$$

Finally, the perfect gas equation of state in inertial coordinates, may be written using the local Mach number as follows:

$$(2.44) \quad \rho e = p \left(\frac{1}{\gamma - 1} + \frac{\gamma \cdot M^2}{2} \right)$$

CHAPTER 3

3. METHODOLOGY

In this chapter we define the finite element methodology that will be employed in our numerical solution for the unsteady, compressible Euler equations. We begin our discussion with an overview of finite element methods along with an introduction to some of the basic terminology for describing such methods. This includes an introduction to the integral formulation that results when we apply the Galerkin method to the governing equations. In the next section, we present the fully discrete space-time formulation that will serve as the basis of our finite element methodology. This is followed by a discussion on the discretization by finite elements of the space-time integrals found in our formulation and numerical evaluation of those integrals using one, two and three-dimensional finite elements.

Also included in this chapter is a discussion of relevant boundary conditions and their practical implementation through the boundary integrals. This includes a derivation of the transpiration boundary condition for elastic problems in both inertial and non-inertial formulations. The next section discusses several predictor multi-corrector algorithms for advancing the finite element solutions in time. The chapter concludes with a review of techniques for stabilizing our finite element solution scheme, including the rationale behind our choice of stabilization operator to adopt.

3.1 Overview of Finite Element Methods

The partial differential equations presented in the previous chapter provide the basis for describing a general, three-dimensional flow field. Since there is no general analytical solution to these equations for three-dimensional problems, numerical methods for approximating their solution have been the topic of research for decades. Traditionally, numerical methods for fluid dynamics were based on finite difference approximations for derivatives in the governing equations. This type of formulation required the use of carefully structured, body-conforming grids; the generation of which is extremely laborious and often a practical impossibility for geometrically and topologically complicated domains. Hence, finite difference methods have quickly become obsolete as new methods without this geometric limitation have been developed.

Most state of the art computational fluid dynamics algorithms are based on either finite volume or finite element formulations. Both of these formulations have the necessary geometric flexibility to allow for the use of unstructured grids when analyzing complicated domains. Finite volume methods currently dominate the field, but finite element methods for fluid flow have matured rapidly over the past several years and are beginning to gain more widespread acceptance by the CFD community, despite their perception as strictly a structural modeling technique based on their origins in that field. In reality, there is very little difference between the two formulations. It has even been demonstrated that a finite element formulation based on the Galerkin method is entirely equivalent to a finite volume formulation when a diagonal form of the finite element mass matrix is used.¹⁶

For our purposes, we seek an efficient and accurate finite element formulation in order to maintain a consistent framework for analysis within the finite element based STARS routines. A standard finite element solution begins by dividing the spatial domain, Ω , into non-overlapping sub-domains called elements. The elements are either line segments in one dimension, triangles/quadrilaterals in two dimensions, or tetrahedrons/hexahedrons in three dimensions. Each element consists of nodes that are typically located at the element's vertices and are shared by neighboring elements. The basic idea of the finite element method is to then construct in a piecewise manner the solution for the fluid unknowns everywhere within Ω using shape functions for each element. These shape functions represent the variation of the solution over an element's sub-domain through an interpolation of discrete values at the element's nodes.

With the geometry suitably defined, the next step is to write an integral formulation for the governing equations, usually by applying some type of Galerkin weighted residual procedure. Applying the basic Galerkin formulation to the dimensionless form of the compressible Euler equations over a closed spatial domain Ω results in the following integral equation:

$$(3.1) \quad \int_{\Omega} \Phi^T \left(\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x_i} - \mathbf{S} \right) d\Omega = 0$$

In the above expression, Φ is a vector of element weighting functions, which are the same as the finite element shape functions assumed in our discretization of the spatial domain.

Next, we apply the Gauss divergence theorem, or integration by parts, and re-write the integral expression in Equation (3.1) as follows:

$$(3.2) \quad \iint_{\Omega} \left(\Phi^T \frac{\partial \mathbf{U}}{\partial t} - \frac{\partial \Phi^T}{\partial x_i} \mathbf{F}_i - \Phi^T \mathbf{S} \right) d\Omega + \int_{\Gamma} \Phi^T \mathbf{F}_n d\Gamma = 0$$

where

$$(3.3) \quad \mathbf{F}_n = \mathbf{F}_i n_i = \begin{Bmatrix} \rho u_n \\ \rho u_1 u_n + p n_1 \\ \rho u_2 u_n + p n_2 \\ \rho u_3 u_n + p n_3 \\ (\rho e + p) u_n \end{Bmatrix}$$

In the above expression, Γ is the boundary of the spatial domain, $\mathbf{n} = \{n_1, n_2, n_3\}^T$ is the outward facing normal vector at the boundary, $u_n = \mathbf{u} \cdot \mathbf{n}$ is the normal fluid velocity at the boundary, and \mathbf{F}_n is the normal flux vector.

The integrated by parts formulation in Equation (3.2) results in the conservation of fluxes under inexact quadrature rules.¹⁷ This form also eliminates the flux derivatives from Equation (3.1), which allows for a discontinuous representation of the Euler fluxes.¹⁶ Furthermore, the introduction of a boundary integral in Equation (3.2) provides us with a natural means of enforcing the boundary conditions on the spatial domain.^{16,17}

To complete our discrete formulation for the compressible Euler equations, it is necessary to approximate the remaining time derivative in Equation (3.2). In most of the existing finite element methods for fluid flow, this is accomplished through some sort of finite difference approximation. The current literature abounds with methods that utilize either forward, backward or central-difference formulas with varying orders of accuracy, all of which lead to either explicit or implicit formulations depending on the choice of differencing formula. In fact, the current STARS unsteady CFD module utilizes an implicit, second-order accurate, backwards-difference operator for its time discretization.¹⁸

While there is nothing fundamentally wrong with employing finite differences as described above, this type of formulation is sometimes labeled as a semi-discrete formulation due to the mixing of finite element and finite difference approximations within one algorithm. In contrast, a fully discrete finite element formulation would utilize element shape functions to represent both the spatial and temporal variation of unknowns within the computational domain. This approach leads to an implicit formulation where the time discretization is derived directly from the Galerkin integrals. The development of these so-called space-time finite element methods has been a major contribution to the field and “should forever dispel the prevalent myth in finite difference circles that somehow finite element methods are not appropriate for hyperbolic problems.”¹⁸ Space-time finite element formulations represent the current state of the art within the field and have been successfully employed in some of the more recently developed algorithms.^{23,24} As we will see later, the differences between the semi-discrete and fully discrete formulations is, in most cases, a question of semantics. However, we will pursue a fully discrete finite element formulation for this study due to the general flexibility it offers when considering the discretization of temporal derivatives.

It is well documented through out the finite element literature that the Galerkin method lacks stability. A solution scheme based on the methodology presented so far will manifest spurious oscillations that are both non-physical and undesirable. To combat this problem, the numerical scheme needs to be stabilized by adding some form of stabilization operator. This is, in fact, the focus of most finite element CFD research, and there are almost as many stabilization operators as there are finite element based CFD codes.

It is important to note that this instability is not a problem that is unique to finite element methods. Stabilization techniques are also necessary for both finite difference and finite volume methods as the instability is inherent to all discretization methods. In fact, many of the stabilization techniques used for finite element methods are simply extrapolations of similar techniques originally derived for finite difference solutions. The remainder of this chapter will focus on developing a working numerical algorithm out of the equations presented so far, including a discussion on the relevant stabilization methods for such an algorithm.

3.2 Space-Time Formulation

The basis of our formulation is the time-discontinuous Galerkin method, which employs finite elements that are piecewise continuous in space and discontinuous in time. To begin our discretization of the problem, consider the space-time domain S , where the time interval $I = [0, T]$ is divided into N intervals $I_n = [t_n, t_{n+1}]$, where $n = 0, 1, \dots, N - 1$. For each time interval, we then define a space-time sub-domain as $S_n = \Omega \times I_n$ and its boundary $B_n = \Gamma \times I_n$, where Ω is the spatial domain and Γ is its boundary.

The introduction of a space-time sub-domain in this discussion can very easily lead to conceptual difficulties when deriving the Galerkin integral formulation. To avoid this, it is useful to consider time as simply an additional dimension for the problem. This works particularly well for a two-dimensional geometry where time can then be thought as a third dimension for each element. The elements then encompass a space-time volume which could be computed as the spatial area of the element times the size of the time increment, or the height of the space-time sub-domain. In three-dimensions, this

idea does not have a solid physical representation, but the mathematical implementation is extended in the same manner.

It is now necessary to modify the previously defined Galerkin formulation, Equation (3.1), to include integration over the entire space-time sub-domain rather than just the spatial domain. Hence, we re-apply the basic Galerkin formulation to the dimensionless form of the compressible Euler equations over a closed space-time sub-domain S_n to derive the following integral equation:

$$(3.4) \quad \int_{S_n} \Phi^T \left(\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_i}{\partial x_i} - \mathbf{S} \right) dS = 0$$

In the above expression, Φ is now a vector of weighting functions that includes both the spatial and temporal variation of the unknowns within the space-time sub-domain. The weighting function is assumed to be continuous within S_n , but discontinuous across the boundary of the space-time sub-domains.

As before, we apply the Gauss divergence theorem, or integration by parts, and re-write the integral expression in Equation (3.4) as follows:

$$(3.5) \quad \int_{S_n} \left(-\frac{\partial \Phi^T}{\partial t} \mathbf{U} - \frac{\partial \Phi^T}{\partial x_i} \mathbf{F}_i - \Phi^T \mathbf{S} \right) dS + \int_{\Omega_n} (\Phi_{n+1}^T \mathbf{U}_{n+1} - \Phi_n^T \mathbf{U}_{n-}) d\Omega + \int_{B_n} \Phi^T \mathbf{F}_n dB = 0$$

In the above expression, Φ_{n+1} and Φ_n are the weight functions evaluated at the t_{n+1} and t_n boundaries of S_n respectively, \mathbf{U}_{n+1} is the value of the unknowns at the t_{n+1} boundary of S_n , and \mathbf{U}_{n-} is the value of the unknowns at the t_n boundary from the previous space-time sub-domain, S_{n-1} .

The second integral in Equation (3.5) represents a time boundary integral, which results from the integration by parts of the time flux term. Since our formulation is

discontinuous in time across the space-time sub-domains, this integral serves as a mechanism for information to jump from one sub-domain to the next. As such, it will be referred to as the jump condition and has been formulated such that it imposes a weakly enforced initial condition for the space-time sub-domains.¹⁷

3.3 Finite Element Discretization

We now decompose our space-time sub-domain S_n into finite elements S_n^e , where $e = 1, 2, \dots, n_{el}$. The result is a fully discrete finite element formulation where the integrals of Equation (3.5) are assembled by summing the contributions of each individual element within S_n as follows:

$$(3.6) \quad \sum_{e=1}^{n_{el}} \int_{S_n^e} \left(-\frac{\partial \Phi^T}{\partial t} \mathbf{U} - \frac{\partial \Phi^T}{\partial x_i} \mathbf{F}_i - \Phi^T \mathbf{S} \right) dS + \sum_{e=1}^{n_{el}} \int_{\Omega_n^e} (\Phi_{n+1}^T \mathbf{U}_{n+1} - \Phi_n^T \mathbf{U}_{n-}) d\Omega$$

$$+ \sum_{e=1}^{n_{be}} \int_{B_n^e} \Phi^T \mathbf{F}_n dB = 0$$

In the above expression, notice that the boundary integral is only computed for the elements along the boundary of the computational domain, where n_{be} is the number of boundary elements. This is because the boundary integral will identically cancel on interior elements that have shared boundaries with other elements.

To accomplish the integration on an individual element, the solution for the vector of fluid unknowns within an element is constructed through an interpolation of discrete values at that element's nodes. For our purposes, we will consider the spatial and temporal interpolation within an element separately by defining different shape functions for each. Furthermore, we will assume that the spatial sub-domain defining an element,

Ω_n^e , does not change with time, i.e. we have a non-deforming grid. For our application, deforming grids will be simulated through a suitable modification of the boundary conditions as is done with the current STARS unsteady CFD module. This approach is more computationally efficient than employing a fully deforming grid. However, it should be noted that a space-time formulation has been demonstrated for arbitrarily deforming grids if this capability is desired later.^{24,25}

For our derivation in this section, it will be assumed that the unknowns vector for each element varies linearly in space and is piecewise constant in time. This constant-in-time discretization can be modified later to account for higher-order temporal variation without the need to modify the geometric transformations that will be derived here. For a constant-in-time discretization, the unknowns are considered constant along each individual space-time sub-domain, but they are discontinuous across different sub-domains. Figure 3.1 presents the notation used for describing the temporal variation of the unknowns from one space-time sub-domain to the next when viewed with respect to the current sub-domain S_n .

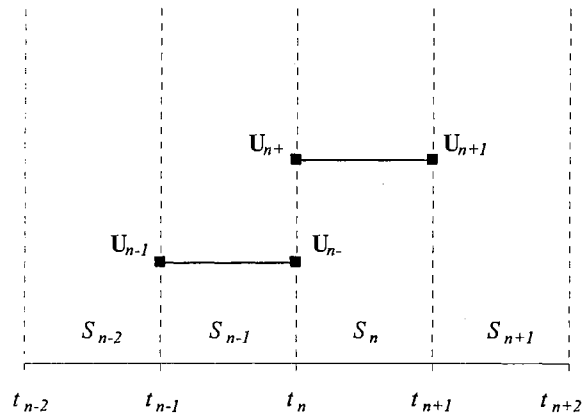


Figure 3.1: Variation in time of unknowns using a constant-in-time approximation.

For our finite element discretization, we define the vector of unknowns \mathbf{U} within an element e as follows:

$$(3.7) \quad \mathbf{U} = [\Phi_1 \quad \dots \quad \Phi_{nd}] \begin{bmatrix} \mathbf{U}_1 \\ \vdots \\ \mathbf{U}_{nd} \end{bmatrix} = \Phi_e \mathbf{U}_e$$

In the above expression, nd is the number of nodes for the element, $\Phi_e = [\Phi_1, \dots, \Phi_{nd}]$ is a $1 \times nd$ vector of spatial shape functions for each node of the element, and $\mathbf{U}_e = [\mathbf{U}_1, \dots, \mathbf{U}_{nd}]^T$ is a $nd \times 1$ vector of unknowns for each node of the element.

Substituting Equation (3.7) into Equation (3.6) and reducing the space-time integrals to spatial integrals only using the transformations $dS = \Delta t d\Omega$ and $dB = \Delta t d\Gamma$ produces the following expression:

$$(3.8) \quad -\Delta t \sum_{e=1}^{n_{el}} \int_{\Omega^e} \left(\frac{\partial \Phi_e^T}{\partial x_i} \mathbf{F}_i + \Phi_e^T \mathbf{S} \right) d\Omega + \sum_{e=1}^{n_{el}} \int_{\Omega^e} \left(\Phi_e^T \Phi_e (\mathbf{U}_e)_{n+1} - \Phi_e^T \Phi_e (\mathbf{U}_e)_{n-} \right) d\Omega \\ + \Delta t \sum_{e=1}^{n_{he}} \int_{\Gamma^e} \Phi_e^T \mathbf{F}_n d\Gamma = 0$$

In the above expression, we have omitted the temporal subscripts wherever possible since the shape functions and the geometry are now assumed to be constant in time.

It is possible to further simplify Equation (3.8) if the flux vectors are also assumed to vary linearly within an element. This assumption leads to what is sometimes called a mixed finite element method. One advantage of a mixed finite element formulation is that the element flux integrals may be re-constructed by integrating over only the edges of the elements. The current STARS unsteady CFD module takes advantage of the edge-based data structure that results from this type of formulation to improve its computational efficiency. In fact, an edge-based data structure is reportedly

up to 30% more efficient than a similar element-based data structure for three-dimensional problems.¹⁸ However, the improved computational efficiency is lost for two-dimensional problems where we have observed that edge-based algorithms are slower than similar element-based algorithms.

For our derivation, element integrals will not be evaluated using an edge-based data structure for several reasons. First, an element-based data structure is required for a consistent evaluation of the jump condition and non-inertial source integrals. The current STARS unsteady CFD module actually uses an approximation for its time flux integrals in order to remain within the context of its edge-based data structure. This approximation, known as mass lumping, is required for edge-based data structures since there is insufficient nodal connectivity information to re-construct the consistent finite element mass matrix that will be derive in the next sections. Furthermore, an edge-based data structure is only useful for approximating the Euler flux integrals for a mixed finite element formulation. This type of formulation does not appear to be extendable to viscous flows where an additional viscous flux integral must be assembled. Hence, the generality and consistency of our formulation will be lost if an edge-based data structure is utilized exclusively.

In order to maintain a consistent finite element formulation, the flux integrals on element interiors will be approximated using Gauss quadrature rather than by making an assumption about the variation of fluxes on element interiors. As discussed previously, this is a significant deviation from the edge-based STARS unsteady CFD module. However, we will follow the current STARS formulation and approximate the boundary flux integrals of Equation (3.8) using a mixed finite element formulation. This is a

reasonable approximation that has been proven to be sufficient for our applications, and it eliminates the complexity of interpolating nodal boundary conditions to Gauss points on the boundary elements. This will be particularly important later when we attempt to implement the transpiration method to simulate deforming boundaries.

3.3.1 One-Dimensional Elements

In this section we derive the geometric transformations necessary to compute the integrals in Equation (3.8) for a one-dimensional element. Although a one-dimensional finite element solver is not particularly useful for aerospace applications, the derivations presented here will be necessary for evaluating boundary integrals in two-dimensions. The non-inertial source and boundary flux integrals will be omitted in this section since they are trivial for a one-dimensional problem and will not be needed. Figure 3.2 presents the general layout for a typical one-dimensional element in element natural coordinates and the variation of the element shape functions over the length of the element.

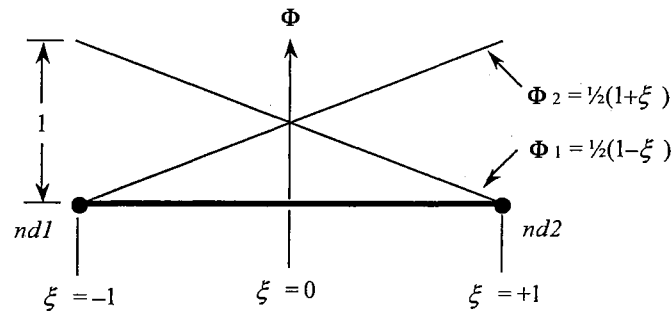


Figure 3.2: Typical one-dimensional finite element in natural coordinates.

The shape function for a one-dimensional linear element is defined in terms of element natural coordinates as follows:

$$(3.9) \quad \Phi_e = \frac{1}{2}[1 - \xi \quad 1 + \xi]$$

In the above expression, ξ is the element's natural coordinate, which ranges from -1 to $+1$ across the length of the element.

In order to convert from element natural coordinates to homogenous Cartesian coordinates, we define the following scalar transformation:

$$(3.10) \quad x = \frac{1}{2}(x_{nd1} + x_{nd2}) + \frac{1}{2}x_{21}\xi$$

where

$$(3.11) \quad x_{21} = x_{nd2} - x_{nd1} = \Delta x_e$$

In the above expressions, x_{nd1} and x_{nd2} are constants that are equal to the x -location for the first and second node of the element respectively.

Using the chain rule for partial derivatives, the Jacobean, \mathbf{J} , of the coordinate transformation may be derived from Equation (3.10).

$$(3.12) \quad \frac{\partial}{\partial \xi} = \underbrace{\frac{\partial x}{\partial \xi}}_{\mathbf{J}} \cdot \frac{\partial}{\partial x}$$

where

$$(3.13) \quad \mathbf{J} = \frac{1}{2}x_{21} = \frac{1}{2}\Delta x_e$$

From Equation (3.12), the inverse relationship follows:

$$(3.14) \quad \frac{\partial}{\partial x} = \frac{2}{\Delta x_e} \frac{\partial}{\partial \xi}$$

From Equation (3.14), spatial derivatives of the element shape function may be converted from Cartesian coordinates into element natural coordinates as follows:

$$(3.15) \quad \frac{\partial \Phi_e}{\partial x} = \frac{2}{\Delta x_e} \frac{\partial \Phi_e}{\partial \xi} = \frac{1}{\Delta x_e} \{-1 \quad 1\}$$

With the shape functions and geometric transformations suitably defined, we now evaluate the integral expressions in Equation (3.8). First, the jump condition integral is evaluated as follows:

$$(3.16) \quad \int_{\Omega^e} (\Phi_e^T \Phi_e (\mathbf{U}_e)_{n+1} - \Phi_e^T \Phi_e (\mathbf{U}_e)_{n-}) d\Omega = \mathbf{M}_e \Delta \mathbf{U}_e$$

where

$$(3.17) \quad \mathbf{M}_e = \frac{1}{2} \Delta x_e \left(\int_{-1}^1 \Phi_e^T \Phi_e d\xi \right) = \frac{1}{6} \Delta x_e \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

and

$$(3.18) \quad \Delta \mathbf{U}_e = (\mathbf{U}_e)_{n+1} - (\mathbf{U}_e)_{n-}$$

In the above expressions, \mathbf{M}_e will be referred to as the finite element mass matrix. Next, the flux integral is evaluated using Gauss quadrature as follows:

$$(3.19) \quad \int_{\Omega_e} \left(\frac{\partial \Phi_e^T}{\partial x} \mathbf{F} \right) d\Omega = \int_{-1}^1 \frac{\partial \Phi_e^T}{\partial \xi} \mathbf{F}(\xi) d\xi \approx \begin{bmatrix} -1 \\ 1 \end{bmatrix} \sum_{i=1}^{np} w_i \mathbf{F}(\xi_i)$$

where np is the number of Gauss points, w_i are the Gauss weights, and ξ_i are the Gauss points. Table 3.1 gives the appropriate values of the Gauss points and weights for one-point and two-point approximations on a one-dimensional element.²⁶

Table 3.1: Gauss points and weights for one-dimensional elements.

Number of points, np	Points, ξ_i	Weights, w_i
1	0.0	2.0
2	-0.5773502692	1.0
	+0.5773502692	1.0

3.3.2 Two-Dimensional Elements

In this section we derive the geometric transformations necessary to compute the integrals in Equation (3.8) for a two-dimensional element. Since the boundary of a two-dimensional element is one-dimensional, this section will also utilize the transformations from the previous section when deriving the boundary flux integrals for two-dimensional elements. Figure 3.3 presents the general layout for a typical two-dimensional element in element natural coordinates.

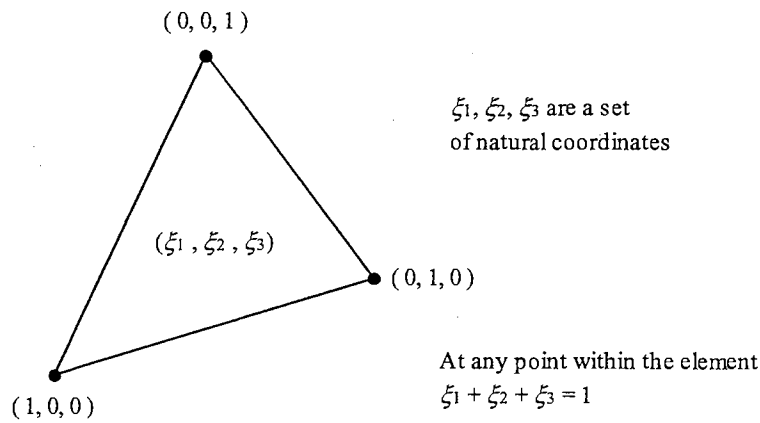


Figure 3.3: Typical two-dimensional finite element in natural coordinates.

The shape function for a two-dimensional linear element is defined in terms of element natural coordinates as follows:

$$(3.20) \quad \Phi_e = [\xi_1 \quad \xi_2 \quad \xi_3]$$

where

$$(3.21) \quad \xi_3 = 1 - \xi_1 - \xi_2$$

In the above expression, ξ_i are the element's natural coordinates, which range from 0 to 1 across the element.

In order to convert from element natural coordinates to homogenous Cartesian coordinates, we define the following transformation:

$$(3.22) \quad \begin{Bmatrix} x \\ y \\ h \end{Bmatrix} = \begin{bmatrix} x_{nd1} & x_{nd2} & x_{nd3} \\ y_{nd1} & y_{nd2} & y_{nd3} \\ 1 & 1 & 1 \end{bmatrix} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix}$$

where x_{nd1} , x_{nd2} and x_{nd3} are constants that are equal to the x -location for the first, second and third node of the element respectively. Alternatively, Equation (3.22) may be reduced to a 2×2 transformation by substituting in the expression for ξ_3 .

$$(3.23) \quad \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} x_{nd3} \\ y_{nd3} \end{Bmatrix} + \begin{bmatrix} x_{13} & x_{23} \\ y_{13} & y_{23} \end{bmatrix} \begin{Bmatrix} \xi_1 \\ \xi_2 \end{Bmatrix}$$

where

$$(3.24) \quad \begin{aligned} x_{ij} &= x_{ndi} - x_{ndj} \\ y_{ij} &= y_{ndi} - y_{ndj} \end{aligned}$$

Using the chain rule for partial derivatives, the Jacobean, \mathbf{J} , of the coordinate transformation may be derived from Equation (3.23).

$$(3.25) \quad \begin{Bmatrix} \partial/\partial\xi_1 \\ \partial/\partial\xi_2 \end{Bmatrix} = \underbrace{\begin{bmatrix} \partial x/\partial\xi_1 & \partial y/\partial\xi_1 \\ \partial x/\partial\xi_2 & \partial y/\partial\xi_2 \end{bmatrix}}_{\mathbf{J}} \begin{Bmatrix} \partial/\partial x \\ \partial/\partial y \end{Bmatrix}$$

where

$$(3.26) \quad \mathbf{J} = \begin{bmatrix} x_{13} & y_{13} \\ x_{23} & y_{23} \end{bmatrix}$$

From Equation (3.25), the inverse relationship follows:

$$(3.27) \quad \begin{Bmatrix} \partial/\partial x \\ \partial/\partial y \end{Bmatrix} = \frac{1}{2A_e} \underbrace{\begin{bmatrix} y_{23} & -y_{13} \\ -x_{23} & x_{13} \end{bmatrix}}_{\mathbf{A}} \begin{Bmatrix} \partial/\partial\xi_1 \\ \partial/\partial\xi_2 \end{Bmatrix}$$

where the area of an element A_e is computed as follows:

$$(3.28) \quad 2A_e = |\det \mathbf{J}| = |x_{13}y_{23} - x_{23}y_{13}|$$

From Equation (3.27), spatial derivatives of the shape function may be converted from Cartesian coordinates into element natural coordinates as follows:

$$(3.29) \quad \frac{\partial \Phi}{\partial x} = \frac{1}{2A_e} \left(\mathbf{A}_{11} \frac{\partial \Phi}{\partial \xi_1} + \mathbf{A}_{12} \frac{\partial \Phi}{\partial \xi_2} \right) = \frac{1}{2A_e} \begin{bmatrix} \mathbf{A}_{11} \\ \mathbf{A}_{12} \\ \mathbf{A}_{13} \end{bmatrix}$$

where

$$(3.30) \quad \mathbf{A}_{13} = -\mathbf{A}_{11} - \mathbf{A}_{12}$$

and

$$(3.31) \quad \frac{\partial \Phi}{\partial y} = \frac{1}{2A_e} \left(\mathbf{A}_{21} \frac{\partial \Phi}{\partial \xi_1} + \mathbf{A}_{22} \frac{\partial \Phi}{\partial \xi_2} \right) = \frac{1}{2A_e} \begin{bmatrix} \mathbf{A}_{21} \\ \mathbf{A}_{22} \\ \mathbf{A}_{23} \end{bmatrix}$$

where

$$(3.32) \quad \mathbf{A}_{23} = -\mathbf{A}_{21} - \mathbf{A}_{22}$$

With the shape functions and geometric transformations suitably defined, we now evaluate the integral expressions in Equation (3.8). First, the jump condition integral is evaluated as follows:

$$(3.33) \quad \int_{\Omega^e} (\Phi_e^T \Phi_e (\mathbf{U}_e)_{n+1} - \Phi_e^T \Phi_e (\mathbf{U}_e)_{n-}) d\Omega = \mathbf{M}_e \Delta \mathbf{U}_e$$

where

$$(3.34) \quad \mathbf{M}_e = 2A_e \left(\int_0^1 \int_0^{1-\xi_2} \Phi_e^T \Phi_e d\xi_1 d\xi_2 \right) = \frac{A_e}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

In the above expressions, \mathbf{M}_e is referred to as the finite element mass matrix, and $\Delta\mathbf{U}_e$ is as defined previously by Equation (3.18). Next, the flux integrals are evaluated using Gauss quadrature as follows:

$$(3.35) \quad \int_{\Omega_e} \left(\frac{\partial \Phi_e^T}{\partial x} \mathbf{F}_1 \right) d\Omega = \int_0^1 \int_0^{1-\xi_2} \left(\mathbf{A}_{11} \frac{\partial \Phi_e^T}{\partial \xi_1} + \mathbf{A}_{12} \frac{\partial \Phi_e^T}{\partial \xi_2} \right) \mathbf{F}_1(\xi_1, \xi_2) d\xi_1 d\xi_2$$

$$\approx \begin{Bmatrix} \mathbf{A}_{11} \\ \mathbf{A}_{12} \\ \mathbf{A}_{13} \end{Bmatrix} \sum_{i=1}^{np} w_i \mathbf{F}_1(\xi_{1,i}, \xi_{2,i})$$

$$(3.36) \quad \int_{\Omega_e} \left(\frac{\partial \Phi_e^T}{\partial y} \mathbf{F}_2 \right) d\Omega = \int_0^1 \int_0^{1-\xi_2} \left(\mathbf{A}_{21} \frac{\partial \Phi_e^T}{\partial \xi_1} + \mathbf{A}_{22} \frac{\partial \Phi_e^T}{\partial \xi_2} \right) \mathbf{F}_2(\xi_1, \xi_2) d\xi_1 d\xi_2$$

$$\approx \begin{Bmatrix} \mathbf{A}_{21} \\ \mathbf{A}_{22} \\ \mathbf{A}_{23} \end{Bmatrix} \sum_{i=1}^{np} w_i \mathbf{F}_2(\xi_{1,i}, \xi_{2,i})$$

where np is the number of Gauss points, w_i are the Gauss weights, and ξ_i are the Gauss points. Table 3.2 gives the appropriate values of the Gauss points and weights for one-point and three-point approximations on a two-dimensional element.²⁶

Table 3.2: Gauss points and weights for two-dimensional elements.

Number of points, np	Points, ξ_i	Weights, w_i
1	1/3	0.5
3	2/3	1/6
	1/6	1/6
	1/6	1/6

The boundary integrals for a two-dimensional element are computed by assuming the normal flux varies linearly along the one-dimensional edge of the element as follows:

$$(3.37) \quad \mathbf{F}_n = \frac{1}{2} [1 - \xi \quad 1 + \xi] \begin{bmatrix} (\mathbf{F}_n)_{nd1} \\ (\mathbf{F}_n)_{nd2} \end{bmatrix} = \Phi_{be} (\mathbf{F}_n)_{be}$$

Substitution of Equation (3.37) into the boundary edge integral results in an integral identical to that used when defining the one-dimensional finite element mass matrix.

$$(3.38) \quad \int_{\Gamma_e} (\Phi_{be}^T \mathbf{F}_n) d\Gamma \approx \frac{1}{2} \Delta x_{be} \left(\int_{-1}^1 \Phi_{be}^T \Phi_{be} d\xi \right) (\mathbf{F}_n)_{be} = \frac{1}{6} \Delta x_{be} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} (\mathbf{F}_n)_{be}$$

Finally, the non-inertial source integral is evaluated using Gauss quadrature as follows:

$$(3.39) \quad \int_{\Omega_e} (\Phi_e^T \mathbf{S}) d\Omega = 2A_e \int_0^1 \int_0^{1-\xi_2} \Phi_e^T \mathbf{S}(\xi_{1,i}, \xi_{2,i}) d\xi_1 d\xi_2 \approx 2A_e \sum_{i=1}^{np} w_i \begin{bmatrix} \xi_{1,i} \\ \xi_{2,i} \\ \xi_{3,i} \end{bmatrix} \mathbf{S}(\xi_{1,i}, \xi_{2,i})$$

3.3.3 Three-Dimensional Elements

In this section we derive the geometric transformations necessary to compute the integrals in Equation (3.8) for a three-dimensional element. Since the boundary of a three dimensional element is two-dimensional, this section will also utilize the transformations from the previous section when deriving the boundary flux integrals for three-dimensional elements. Figure 3.4 presents the general layout for a typical three-dimensional element in element natural coordinates.

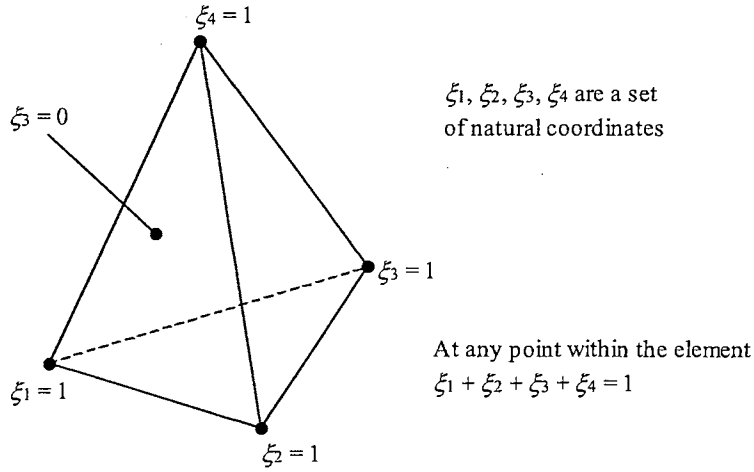


Figure 3.4: Typical three-dimensional finite element in natural coordinates.

The shape function for a three-dimensional linear element is defined in terms of element natural coordinates as follows:

$$(3.40) \quad \Phi_e = [\xi_1 \quad \xi_2 \quad \xi_3 \quad \xi_4]$$

where

$$(3.41) \quad \xi_4 = 1 - \xi_1 - \xi_2 - \xi_3$$

In the above expression, ξ_i are the element's natural coordinates, which range from 0 to 1 linearly across the element.

In order to convert from element natural coordinates to homogenous Cartesian coordinates, we define the following transformation:

$$(3.42) \quad \begin{Bmatrix} x \\ y \\ z \\ h \end{Bmatrix} = \begin{bmatrix} x_{nd1} & x_{nd2} & x_{nd3} & x_{nd4} \\ y_{nd1} & y_{nd2} & y_{nd3} & y_{nd4} \\ z_{nd1} & z_{nd2} & z_{nd3} & z_{nd4} \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_4 \end{Bmatrix}$$

where x_{nd1} through x_{nd4} are constants that are equal to the x -location for the four nodes of the element. Alternatively, Equation (3.22) may be reduced to a 3×3 transformation by substituting in the expression for ξ_4 .

$$(3.43) \quad \begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{Bmatrix} x_{nd4} \\ y_{nd4} \\ z_{nd4} \end{Bmatrix} + \begin{bmatrix} x_{14} & x_{24} & x_{34} \\ y_{14} & y_{24} & y_{34} \\ z_{14} & z_{24} & z_{34} \end{bmatrix} \begin{Bmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{Bmatrix}$$

where

$$(3.44) \quad \begin{aligned} x_{ij} &= x_{ndi} - x_{ndj} \\ y_{ij} &= y_{ndi} - y_{ndj} \\ z_{ij} &= z_{ndi} - z_{ndj} \end{aligned}$$

Using the chain rule for partial derivatives, the Jacobean, \mathbf{J} , of the coordinate transformation may be derived from Equation (3.43).

$$(3.45) \quad \begin{Bmatrix} \partial/\partial\xi_1 \\ \partial/\partial\xi_2 \\ \partial/\partial\xi_3 \end{Bmatrix} = \underbrace{\begin{bmatrix} \partial x/\partial\xi_1 & \partial y/\partial\xi_1 & \partial z/\partial\xi_1 \\ \partial x/\partial\xi_2 & \partial y/\partial\xi_2 & \partial z/\partial\xi_2 \\ \partial x/\partial\xi_3 & \partial y/\partial\xi_3 & \partial z/\partial\xi_3 \end{bmatrix}}_{\mathbf{J}} \begin{Bmatrix} \partial/\partial x \\ \partial/\partial y \\ \partial/\partial z \end{Bmatrix}$$

where

$$(3.46) \quad \mathbf{J} = \begin{bmatrix} x_{14} & y_{14} & z_{14} \\ x_{24} & y_{24} & z_{24} \\ x_{34} & y_{34} & z_{34} \end{bmatrix}$$

From Equation (3.45), the inverse relationship follows:

$$(3.47) \quad \begin{Bmatrix} \partial/\partial x \\ \partial/\partial y \\ \partial/\partial z \end{Bmatrix} = \frac{1}{6V_e} \underbrace{\begin{bmatrix} y_{24}z_{34} - y_{34}z_{24} & y_{34}z_{14} - y_{14}z_{34} & y_{14}z_{24} - y_{24}z_{14} \\ z_{24}x_{34} - z_{34}x_{24} & z_{34}x_{14} - z_{14}x_{34} & z_{14}x_{24} - z_{24}x_{14} \\ x_{24}y_{34} - x_{34}y_{24} & x_{34}y_{14} - x_{14}y_{34} & x_{14}y_{24} - x_{24}y_{14} \end{bmatrix}}_{\Lambda} \begin{Bmatrix} \partial/\partial\xi_1 \\ \partial/\partial\xi_2 \\ \partial/\partial\xi_3 \end{Bmatrix}$$

where the volume of an element V_e is computed as follows:

$$(3.48) \quad 6V_e = |\det \mathbf{J}|$$

From Equation (3.47), the spatial derivatives of the shape function may be converted from Cartesian coordinates into element natural coordinates as follows:

$$(3.49) \quad \frac{\partial \Phi}{\partial x} = \mathbf{A}_{11} \frac{\partial \Phi}{\partial \xi_1} + \mathbf{A}_{12} \frac{\partial \Phi}{\partial \xi_2} + \mathbf{A}_{13} \frac{\partial \Phi}{\partial \xi_3} = \frac{1}{6V_e} \begin{bmatrix} \mathbf{A}_{11} \\ \mathbf{A}_{12} \\ \mathbf{A}_{13} \\ \mathbf{A}_{14} \end{bmatrix}$$

where

$$(3.50) \quad \mathbf{A}_{14} = -\mathbf{A}_{11} - \mathbf{A}_{12} - \mathbf{A}_{13}$$

and

$$(3.51) \quad \frac{\partial \Phi}{\partial y} = \mathbf{A}_{21} \frac{\partial \Phi}{\partial \xi_1} + \mathbf{A}_{22} \frac{\partial \Phi}{\partial \xi_2} + \mathbf{A}_{23} \frac{\partial \Phi}{\partial \xi_3} = \frac{1}{6V_e} \begin{bmatrix} \mathbf{A}_{21} \\ \mathbf{A}_{22} \\ \mathbf{A}_{23} \\ \mathbf{A}_{24} \end{bmatrix}$$

where

$$(3.52) \quad \mathbf{A}_{24} = -\mathbf{A}_{21} - \mathbf{A}_{22} - \mathbf{A}_{23}$$

and

$$(3.53) \quad \frac{\partial \Phi}{\partial z} = \mathbf{A}_{31} \frac{\partial \Phi}{\partial \xi_1} + \mathbf{A}_{32} \frac{\partial \Phi}{\partial \xi_2} + \mathbf{A}_{33} \frac{\partial \Phi}{\partial \xi_3} = \frac{1}{6V_e} \begin{bmatrix} \mathbf{A}_{31} \\ \mathbf{A}_{32} \\ \mathbf{A}_{33} \\ \mathbf{A}_{34} \end{bmatrix}$$

where

$$(3.54) \quad \mathbf{A}_{34} = -\mathbf{A}_{31} - \mathbf{A}_{32} - \mathbf{A}_{33}$$

With the shape functions and geometric transformations suitably defined, we now evaluate the integral expressions in Equation (3.8). First, the jump condition integral is evaluated as follows:

$$(3.55) \quad \int_{\Omega^e} (\Phi_e^T \Phi_e (\mathbf{U}_e)_{n+1} - \Phi_e^T \Phi_e (\mathbf{U}_e)_{n-}) d\Omega = \mathbf{M}_e \Delta \mathbf{U}_e$$

where

$$(3.56) \quad \mathbf{M}_e = 6V_e \left(\int_0^1 \int_0^{1-\xi_3} \int_0^{1-\xi_2-\xi_3} \Phi_e^T \Phi_e d\xi_1 d\xi_2 d\xi_3 \right) = \frac{V_e}{20} \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$$

and

$$(3.57) \quad \Delta \mathbf{U}_e = (\mathbf{U}_e)_{n+1} - (\mathbf{U}_e)_{n-1}$$

In the above expressions, \mathbf{M}_e is referred to as the finite element mass matrix and $\Delta \mathbf{U}_e$ is as defined previously by Equation (3.18). Next, the flux integrals are evaluated using Gauss quadrature as follows:

$$(3.58) \quad \int_{\Omega^e} \left(\frac{\partial \Phi_e^T}{\partial x} \mathbf{F}_1 \right) d\Omega \approx \begin{Bmatrix} \mathbf{A}_{11} \\ \mathbf{A}_{12} \\ \mathbf{A}_{13} \\ \mathbf{A}_{14} \end{Bmatrix} \left\{ \sum_{i=1}^{np} w_i \mathbf{F}_1(\xi_{1,i}, \xi_{2,i}, \xi_{3,i}) \right\}$$

$$(3.59) \quad \int_{\Omega^e} \left(\frac{\partial \Phi_e^T}{\partial y} \mathbf{F}_2 \right) d\Omega \approx \begin{Bmatrix} \mathbf{A}_{21} \\ \mathbf{A}_{22} \\ \mathbf{A}_{23} \\ \mathbf{A}_{24} \end{Bmatrix} \left\{ \sum_{i=1}^{np} w_i \mathbf{F}_2(\xi_{1,i}, \xi_{2,i}, \xi_{3,i}) \right\}$$

$$(3.60) \quad \int_{\Omega^e} \left(\frac{\partial \Phi_e^T}{\partial z} \mathbf{F}_3 \right) d\Omega \approx \begin{Bmatrix} \mathbf{A}_{31} \\ \mathbf{A}_{32} \\ \mathbf{A}_{33} \\ \mathbf{A}_{34} \end{Bmatrix} \left\{ \sum_{i=1}^{np} w_i \mathbf{F}_3(\xi_{1,i}, \xi_{2,i}, \xi_{3,i}) \right\}$$

where np is the number of Gauss points, w_i are the Gauss weights, and ξ_i are the Gauss points. Table 3.3 gives the appropriate values of the Gauss points and weights for one-point and four-point approximations on a three-dimensional element.^{27,28}

Table 3.3: Gauss points and weights for three-dimensional elements.

Number of points, np	Points, ξ_i	Weights, w_i
1	0.25	1/6
4	0.58541 01966 249685	1/24
	0.13819 66011 250105	1/24
	0.13819 66011 250105	1/24
	0.13819 66011 250105	1/24

The boundary integral for a three-dimensional element is computed by assuming the normal flux varies linearly along the two-dimensional boundary face of the element as follows:

$$(3.61) \quad \mathbf{F}_n = \begin{bmatrix} \xi_1 & \xi_2 & 1 - \xi_1 - \xi_2 \end{bmatrix} \begin{bmatrix} (\mathbf{F}_n)_{nd1} \\ (\mathbf{F}_n)_{nd2} \\ (\mathbf{F}_n)_{nd3} \end{bmatrix} = \mathbf{\Phi}_{be} (\mathbf{F}_n)_{be}$$

Substitution of Equation (3.61) into the boundary integral results in an integral identical to that used when defining the two-dimensional finite element mass matrix.

$$(3.62) \quad \int_{\Gamma_e} (\mathbf{\Phi}_{be}^T \mathbf{F}_n) d\Gamma \approx 2A_{be} \left(\int_0^1 \int_0^{1-\xi_2} \mathbf{\Phi}_{be}^T \mathbf{\Phi}_{be} d\xi_1 d\xi_2 \right) (\mathbf{F}_n)_{be} = \frac{A_{be}}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} (\mathbf{F}_n)_{be}$$

Finally, the non-inertial source integral is evaluated using Gauss quadrature as follows:

$$(3.63) \quad \int_{\Omega_e} (\mathbf{\Phi}_e^T \mathbf{S}) d\Omega \approx 6V_e \sum_{i=1}^{np} w_i \begin{bmatrix} \xi_{1,i} \\ \xi_{2,i} \\ \xi_{3,i} \\ \xi_{4,i} \end{bmatrix} \mathbf{S}(\xi_{1,i}, \xi_{2,i}, \xi_{3,i})$$

3.3.4 Element Edge Data

While the edge-based method employed by STARS will not be used for evaluating element integrals, element edge data will prove useful later for evaluating other quantities such as local time steps and artificial dissipation. For this reason, it is also prudent to discuss the nomenclature used to describe an edge-based data structure. For the specific theory and justification behind an edge-based data structure, we refer to the original FELISA manual.¹⁹ The basic idea is to lump the element geometry data to the edges or segments associated with each element. This will result in a vector of geometric weights for each unique segment in the computational domain. As such we define the weight vector for a segment s that connects node i to node j as follows:

$$(3.64) \quad \mathbf{W}_{ij,s} = \{W_1 \quad W_2 \quad W_3\}^T$$

The segment weight vectors are built-up by finding each element containing the segment and adding in one sixth of the components of the element jacobian associated with node i of the segment. For segments which lie on boundary elements, it will also be necessary to add one sixth of the area weighted boundary element normal vector to the segment weights. The result is a vector of weights that is useful for computing edge gradients along the segments of an element as follows:

$$(3.65) \quad \nabla q_{ij,s} = \begin{Bmatrix} q_j \mathbf{W}_{ij,s} \\ -q_i \mathbf{W}_{ij,s} \end{Bmatrix}$$

where q is some flow quantity and the subscripts i and j denote the values of q at the first and second node of the segment respectively.

3.4 Boundary Conditions and the Boundary Integral

As suggested in Section 3.1, the finite element boundary integral provides us with a natural means for enforcing boundary conditions within our finite element framework. Specifically, boundary conditions should be weakly enforced by modifying the boundary flux during the integration over the boundary rather than by explicitly modifying the nodal unknowns along the boundary. This is the most common method of enforcing boundary conditions within a finite element algorithm, and typically yields the most accuracy. Unfortunately, our experience has shown that weakly enforced boundary conditions exhibit instabilities for solid wall boundary conditions in three dimensional flows, especially when combined with the transpiration boundary condition used to simulate elastic deformations of the grid. This fact will require us to explicitly impose the solid wall boundary conditions, which will include the transpiration boundary condition.

Regardless of our method for imposing boundary conditions, it is important to note that the boundary flux integral should be computed on the entire boundary. Neglecting the boundary integral will lead to a loss of flux conservation in the finite element formulation.²³ Also, the boundary integral must always be computed using the original normal vector for the boundary element regardless of whether an elastic deformation is being simulated using transpiration. This is again necessary to maintain flux conservation within the solution methodology.

In the sub-sections that follow, we address the practical implementation of three different boundary condition types: symmetry, far-field and solid wall. These are the three types of boundary conditions currently supported by the STARS unsteady CFD

module. Therefore, each boundary element, which is either a line segment in two-dimensions or a triangular face in three dimensions, will have an associated boundary condition flag that indicates which of the three boundary conditions should be applied on that element.

3.4.1 Symmetry Boundary Condition

The symmetry boundary condition is the simplest to implement. For inviscid flow, a symmetry condition will behave similar to a wall in that it requires the normal component of the fluid velocity be zero. This will be the only condition imposed on elements that are flagged with the symmetry boundary condition. Substitution of this condition, $u_n = 0$, into the definition of the boundary flux, Equation (3.3), results in the following modified boundary flux:

$$(3.66) \quad \hat{\mathbf{F}}_n = \begin{Bmatrix} 0 \\ pn_1 \\ pn_2 \\ pn_3 \\ 0 \end{Bmatrix}$$

If the modified boundary flux given above is utilized in the calculation of the boundary flux integral the symmetry condition will be weakly enforced on that boundary element.

Since the symmetry boundary condition is essentially the same as a rigid, inviscid wall boundary condition, it would seem to make sense that the symmetry and wall boundary conditions could be used interchangeably for rigid walls. However, when we refer to a symmetry boundary condition, we are typically referring to a symmetry plane, or a flat wall with a constant surface normal. In fact, the symmetry boundary condition

has proven useful for flat walls that do not include any elastic effects. However, the weakly enforced wall boundary condition used for defining a symmetry boundary condition has proven to be mildly unstable for three-dimensional flows on curved walls where the surface normal is not constant. Therefore the symmetry boundary condition should only be used on boundary surfaces that represent flat planes.

3.4.2 Far-Field Boundary Condition

As with the symmetry boundary condition, the far field boundary condition will be weakly enforced by modifying the boundary integral. Multiple references, for both finite volume and finite element methods, suggest imposing far field boundary conditions by computing the one-dimensional Riemann invariants for flow normal to the boundary.^{8,19,20} This methodology has proven to be quite robust and reliable as it is implemented within the current STARS unsteady CFD module. Therefore, we will simply review the current STARS far field boundary condition and modify it as necessary to account for a non-inertial reference frame.

For a far field boundary element, the boundary conditions are enforced by computing a modified boundary flux vector for each node and using the modified boundary flux vectors to compute the boundary integral for that element. The modified boundary flux vector is computed by comparing the specified free stream conditions with the computed unknowns at each node. We begin by defining a density ratio for a far field node as follows:

$$(3.67) \quad \rho_i = \sqrt{\frac{\rho_\infty}{\rho_c}}$$

where ρ is the fluid density and the subscripts ∞ and c refer to the free stream conditions and the computed values, respectively.

The above density ratio is now used to compute appropriately scaled ratios for the velocity components and enthalpy using the following relations:

$$(3.68) \quad \mathbf{u}_i = \frac{\rho_i \mathbf{u}_\infty + \mathbf{u}_c}{\rho_i + 1}$$

$$(3.69) \quad h_i = \frac{\rho_i h_\infty + h_c}{\rho_i + 1}$$

$$(3.70) \quad a_i = \sqrt{(\gamma - 1) \left(h_i - \frac{1}{2} |\mathbf{u}_i|^2 \right)}$$

where $\mathbf{u} = \{u_1, u_2, u_3\}^T$ is the fluid velocity vector, h is the total enthalpy per unit mass, and a is the local speed of sound. However, Equations (3.68) and (3.69) represent relative velocity and enthalpy for non-inertial problems. Based on the definition for relative enthalpy given by Equation (2.23), we see that Equation (3.70) must be modified to include the transformation velocity \mathbf{V}_i as follows:

$$(3.71) \quad a_i = \sqrt{(\gamma - 1) \left(h_i - \frac{1}{2} |\mathbf{u}_i|^2 + \frac{1}{2} |\mathbf{V}_i|^2 \right)}$$

Next, we compute three Riemann invariants and an Euler unknowns difference vector for the boundary node as follows:

$$(3.72) \quad \lambda_1 = |\mathbf{u}_i \cdot \mathbf{n} + a_i|$$

$$(3.73) \quad \lambda_2 = |\mathbf{u}_i \cdot \mathbf{n} - a_i|$$

$$(3.74) \quad \lambda_3 = |\mathbf{u}_i \cdot \mathbf{n}|$$

$$(3.75) \quad \Delta \mathbf{U} = \begin{Bmatrix} \rho_\infty - \rho_c \\ (\rho u_1)_\infty - (\rho u_1)_c \\ (\rho u_2)_\infty - (\rho u_2)_c \\ (\rho u_3)_\infty - (\rho u_3)_c \\ (\rho e)_\infty - (\rho e)_c \end{Bmatrix}$$

Using the above definitions, we next define the following ‘‘correction’’ factors:

$$(3.76) \quad a_1 = (\gamma - 1) \left(\frac{1}{2} |\mathbf{u}_i|^2 \Delta U_1 - u_{1,i} \Delta U_2 - u_{2,i} \Delta U_3 - u_{3,i} \Delta U_4 + \Delta U_5 \right)$$

$$(3.77) \quad a_2 = n_1 \Delta U_2 + n_2 \Delta U_3 + n_3 \Delta U_4 - (\mathbf{u}_i \cdot \mathbf{n}) \Delta U_1$$

$$(3.78) \quad c_1 = \frac{\frac{1}{2}(\lambda_1 + \lambda_2)a_1 - \lambda_3 a_1}{a_i^2} + \frac{\frac{1}{2}(\lambda_1 - \lambda_2)a_2}{a_i}$$

$$(3.79) \quad c_2 = \frac{\frac{1}{2}(\lambda_1 - \lambda_2)a_1}{a_i} + \frac{1}{2}(\lambda_1 + \lambda_2)a_2 - \lambda_3 a_2$$

Notice that Equation (3.76) has the form of a scaled pressure difference. Hence, this equation must be modified for non-inertial problems by including the transformation velocity as follows:

$$(3.80) \quad a_1 = (\gamma - 1) \left(\frac{1}{2} |\mathbf{u}_i|^2 \Delta U_1 + \frac{1}{2} |\mathbf{V}_i|^2 \Delta U_1 - u_{1,i} \Delta U_2 - u_{2,i} \Delta U_3 - u_{3,i} \Delta U_4 + \Delta U_5 \right)$$

Finally, we define the corrected normal flux vector for a node on a far field element as follows:

$$(3.81) \quad \hat{\mathbf{F}}_n = \frac{1}{2} (\mathbf{F}_{n,\infty} + \mathbf{F}_{n,c} - \Delta \mathbf{F})$$

where

$$(3.82) \quad \Delta \mathbf{F} = \begin{Bmatrix} \lambda_3 \Delta U_1 + c_1 \\ \lambda_3 \Delta U_2 + c_1 u_{1,i} + c_2 n_1 \\ \lambda_3 \Delta U_3 + c_1 u_{2,i} + c_2 n_2 \\ \lambda_3 \Delta U_4 + c_1 u_{3,i} + c_2 n_3 \\ \lambda_3 \Delta U_5 + c_1 h_i + c_2 (\mathbf{u}_i \cdot \mathbf{n}) \end{Bmatrix}$$

If the modified boundary flux given above is utilized in the calculation of the boundary flux integral the far field boundary condition will be weakly enforced on that boundary element. This leads to a stable and accurate representation of inflow-outflow boundary conditions for flow conditions ranging from subsonic to supersonic speeds.

3.4.3 Solid Wall Boundary Condition

Unlike the previous two boundary conditions, the solid wall boundary condition will be strongly enforced by directly modifying the unknowns at the end of each iteration. This is the methodology currently implemented within the STARS unsteady CFD module, and seems to be necessary to prevent mild instabilities which arise along the solid wall boundaries if this boundary condition were weakly enforced for three-dimensional flows. Strongly enforced wall boundary conditions have also been utilized by Shapiro,²⁰ and Shakib¹⁷ reported that a strongly enforced pressure leads to a more stable solution with a faster convergence rate.

The basic wall boundary condition for a rigid wall will require that the relative velocity of the flow be tangent to the boundary surface, or alternatively that the normal component of the flow velocity is zero as seen by an observer moving with the body. This is implemented by subtracting off the normal component of the fluid velocity at a node as follows:

$$(3.83) \quad \mathbf{u} = \mathbf{u} - (\mathbf{u} \cdot \mathbf{n})\mathbf{n}$$

Equation (3.83) is explicitly applied to every node which lies on a solid wall boundary element using an appropriately averaged normal from the surrounding boundary elements. However, nodes where the average surface normal is not well defined will

present a problem. This includes wall nodes at the intersection of two surfaces where the direction of the surface normal changes substantially, such as the trailing edge of an airfoil as shown in Figure 3.5.

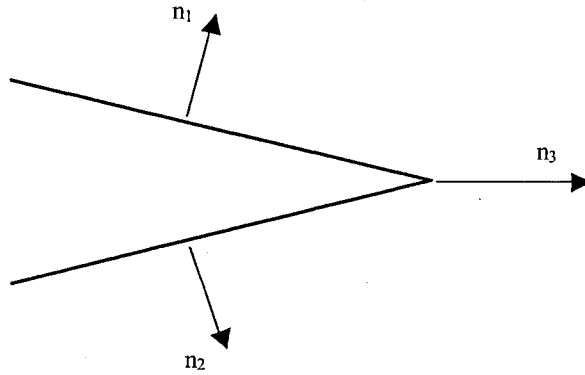


Figure 3.5: Illustration of an average surface normal for the trailing edge of an airfoil.

Obviously, the flow tangency condition would not be accurate for the node at the trailing edge of this airfoil since it would require that the flow move in the vertical direction. This is contrary to the requirement of the Kutta condition,²¹ which states that the flow must leave the upper and lower surface of the trailing edge smoothly and with a finite velocity. The simplest way of handling this problem is to skip the enforcement of flow tangency on these nodes. For a surface which is sufficiently discretized, this will still result in flow which remains tangent to the body thanks to the boundary conditions being applied at the neighboring nodes.

The rigid wall boundary condition discussed so far is sufficient for both inertial and non-inertial problems. However, it is also necessary to account for elastic walls when simulating aeroelastic problems. As suggested previously, this can be accomplished by using transpiration to simulate a deformed surface using a modified surface normal. For an elastic problem, the deformed state of the structure is defined by a

set of structural mode shapes. The mode shapes define a deformation vector for each solid wall node in the computational domain. Consider the deformed state given by Figure 3.6, where the two nodes of an edge have been displaced from their initial position by a deformation vector $\Delta \mathbf{r}$.

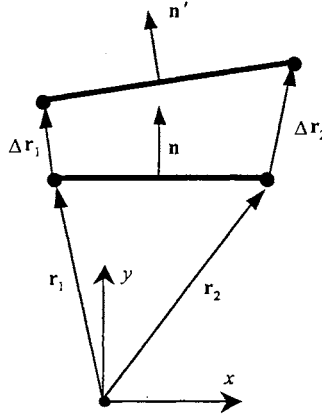


Figure 3.6: Illustration of transpiration concept.

This statically deformed shape can be simulated without actually deforming the surface grid by using the deformed normal vector, \mathbf{n}' , when enforcing flow tangency. The original normal vector is still used to evaluate the boundary integral and maintain flux conservation, but the averaged normal vector used to explicitly enforce flow tangency at each wall node is now based on the deformed shape of the structure. This results in a situation where the normal component of the fluid velocity on the wall surfaces is no longer zero since the flow is forced to follow the deformed shape of the surface.

In addition to statically deformed structures, it is also necessary to simulate dynamically deforming structures for an aeroelastic simulation. In this case, the elastic surface is continuously changing its deformed shape and has an associated deformation

velocity, \mathbf{V}_b , for each node on the boundary. The transpiration boundary condition for a node on a solid wall that is dynamically deforming is given as follows:

$$(3.84) \quad \mathbf{u} = \mathbf{u} - (\mathbf{u} \cdot \mathbf{n}' - \mathbf{V}_b \cdot \mathbf{n}') \mathbf{n}'$$

Equation (3.84) simply ensures that the normal component of the fluid velocity is equal to the normal component of the boundary velocity at each node using the deformed normal vector.

The solid wall boundary condition described so far defines the transpiration method currently implemented in STARS to simulate dynamically deforming structures in an inertial frame of reference. Research has demonstrated that STARS CFD results employing this transpiration boundary condition are in excellent agreement with those obtained using actually deformed grids for the small deformations typically seen in aeroelastic simulations.⁷ However, it is not expected that this methodology will accurately represent a deformed body in a non-inertial frame. Consider the two cases illustrated in Figure 3.7.

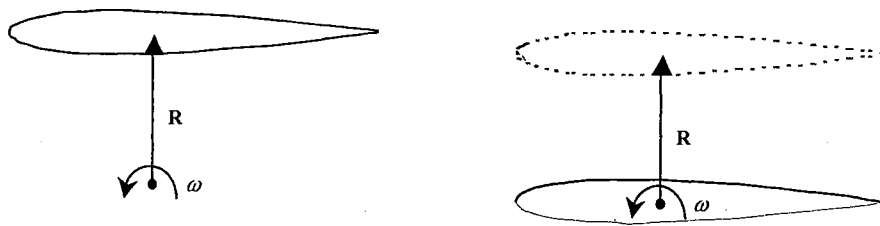


Figure 3.7: Model problem for testing transpiration with non-inertial rotation.

The graphic on the left represents an airfoil in a rotating frame of reference with the origin of rotation some vector distance \mathbf{R} away from the midpoint of the airfoil, while the graphic on the right is attempting to simulate the same rotating airfoil using

transpiration to shift an airfoil rotating about its midpoint to the vector location \mathbf{R} . Obviously, these two problems are very different unless the transpiration method accurately accounts for the non-inertial rotation for the shifted body. However, the transpiration procedure presented so far would effectively do nothing because the body has been shifted uniformly, with no change in surface normals, to a new static position.

Consider what would happen if we rotated the coordinate system illustrated in Figure 3.6. In this case, the mesh transformation velocity for the two nodes of the boundary element would be given by the following equation:

$$(3.85) \quad \mathbf{V}_{i,j} = \boldsymbol{\Omega}(\mathbf{r}_i + \Delta\mathbf{r}_i) = \boldsymbol{\Omega}\mathbf{r}_i + \boldsymbol{\Omega}\Delta\mathbf{r}_i$$

Since we are simulating the elastic deformation using transpiration, it will be necessary to account for the $\boldsymbol{\Omega}\Delta\mathbf{r}$ term in the transpiration boundary condition. This term will simply be treated as an additional boundary velocity for each node, giving us the following equation for our solid wall boundary condition:

$$(3.86) \quad \mathbf{u} = \mathbf{u} - [\mathbf{u} \cdot \mathbf{n}' - (\mathbf{V}_b + \boldsymbol{\Omega}\Delta\mathbf{r}) \cdot \mathbf{n}'] \mathbf{n}'$$

Equation (3.86) now simulates the elastic deformations for a solid wall boundary in either an inertial or non-inertial frame of reference.

As a brief verification for this non-inertial transpiration methodology, consider the airfoil illustrated in Figure 3.7 where the displacement vector \mathbf{R} is oriented in the negative x -direction with a magnitude equal to half the chord of the airfoil, or $\mathbf{R} = \{ -\frac{1}{2}c, 0 \}$. This puts the origin of rotation at the trailing edge of the airfoil. The airfoil is then forced to pitch sinusoidal with a dimensionless frequency of $\omega^* = 2.0$, where the dimensionless frequency is defined as follows:

$$(3.87) \quad \omega^* = \frac{\omega c}{2U_\infty}$$

Figure 3.8 shows the time history of the pitch angle for this airfoil.

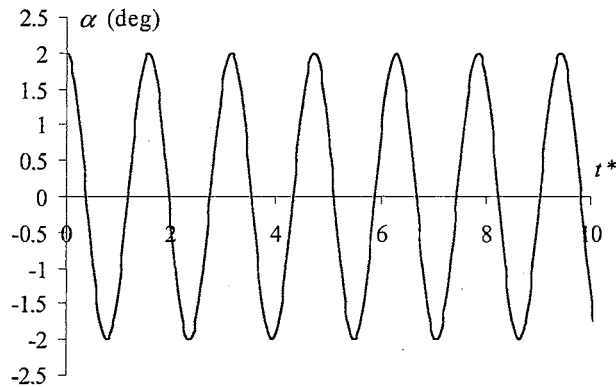


Figure 3.8: Time history for the pitch angle of a rotating airfoil.

Now, three different simulations are performed for this airfoil. First, a baseline solution is run, which consists of a non-inertial solution for the airfoil pitching at its midpoint. Secondly, a “shifted” solution is run, which consists of a non-inertial solution where the pitch location has been shifted a distance \mathbf{R} from the midpoint. Finally, a transpiration solution is run, which consists of a non-inertial solution for the airfoil pitching about its midpoint with a simulated shift to the location \mathbf{R} using the transpiration method corrected for a rotating non-inertial frame. Figure 3.9 presents a comparison of the surface pressure distribution predicted by these three solutions at four different instants in time.

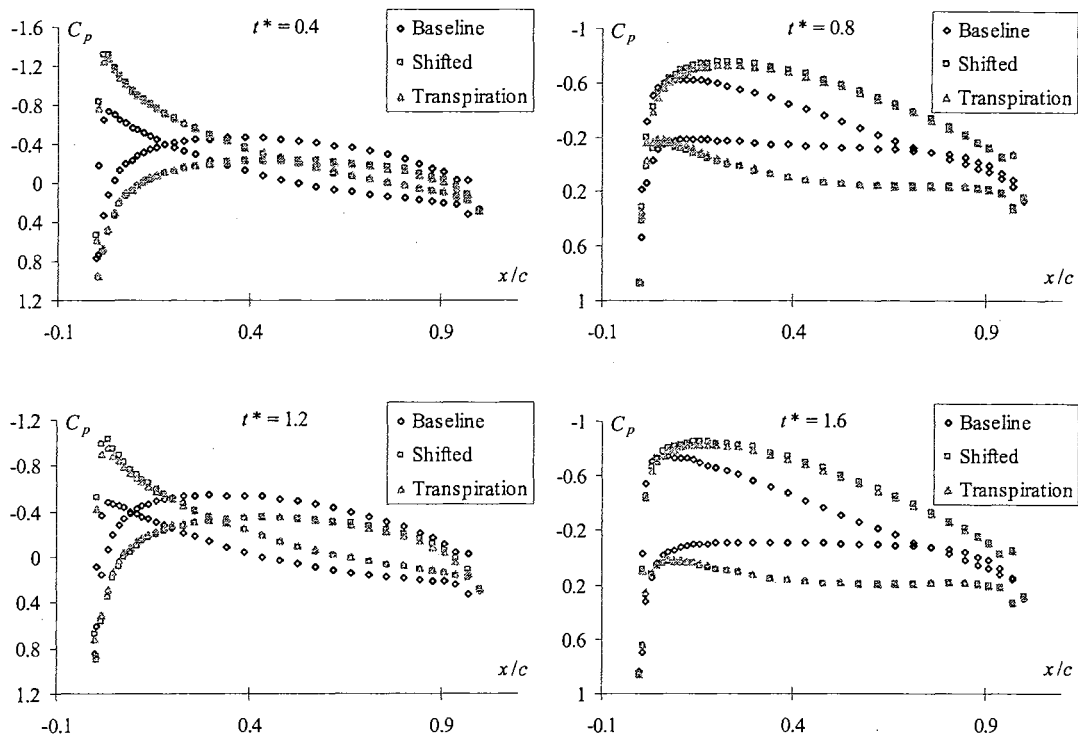


Figure 3.9: Comparison of predicted pressure coefficient for a pitching airfoil.

Notice that both the shifted and transpiration solutions are in excellent agreement even for this relatively large displacement. The baseline solution is presented only to show that the shifted axis has a significant effect in the resulting pressure distributions. Thus, we see that without the non-inertial correction to the transpiration method we would have a significantly inaccurate representation for the shifted non-inertial solution. Further verification of the transpiration method will be pursued in more detail later.

3.5 Predictor Multi-Corrector Algorithms

The basic finite element formulation derived so far may be summarized as follows:

$$(3.88) \quad \mathbf{M}(\mathbf{U}_{n+1} - \mathbf{U}_n) - \Delta t \cdot \mathbf{FS}(\mathbf{U}_{n+1}) + \Delta t \cdot \mathbf{B}(\mathbf{U}_{n+1}) = 0$$

The above equation is typically referred to as the global form of the finite element formulation and is assembled by summing the nodal contributions from each element. In the above expression, \mathbf{M} is the assembled finite element mass matrix, \mathbf{U}_{n+1} and \mathbf{U}_n are vectors of nodal unknowns for the current and previous time intervals respectively, $\mathbf{FS}(\mathbf{U}_{n+1})$ is the assembled flux and source integrals for the current time interval, and $\mathbf{B}(\mathbf{U}_{n+1})$ is the assembled boundary integrals for the current time interval. Notice that this formulation is implicit in time since the values of the flux, source and boundary integrals all depend on the unknowns for the current time interval. Furthermore, there is an implicit coupling between the unknowns since the finite element mass matrix derived in Section 3.3 is not diagonal.

There are a variety of numerical schemes documented in the current literature for solving the nonlinear system of equations represented by Equation (3.88). Although these equations have been formulated with an implicit coupling in time, it is still possible to develop an explicit numerical scheme for iteratively advancing the solution. An explicit method is definitely the most attractive option in terms of memory usage and computational speed. However, the maximum allowable time step for explicit algorithms is known to be limited by a stability criterion that is based on the element dimensions and local fluid velocities.^{16,23} For our application, it will be necessary to accept the stability limitations of explicit algorithms and work to extend their stability limit since we require the speed and economy of such methods.

In developing an explicit solution, we first notice that any iterative algorithm we develop should drive Equation (3.88) to zero for each time interval. As such, we call

Equation (3.88) the solution residual and denote it as $\mathbf{R}(\mathbf{U}_{n+1}, \mathbf{U}_n)$. Now, the solution residual is driven to zero using a predictor multi-corrector algorithm written as follows:

$$(3.89) \quad \mathbf{R}(\mathbf{U}_{n+1}^{i+1}, \mathbf{U}_n) = \mathbf{R}(\mathbf{U}_{n+1}^i, \mathbf{U}_n) + \frac{\partial \mathbf{R}(\mathbf{U}_{n+1}^i, \mathbf{U}_n)}{\partial \mathbf{U}} \Delta \mathbf{U}_{n+1}^i$$

where

$$(3.90) \quad \Delta \mathbf{U}_{n+1}^i = \mathbf{U}_{n+1}^{i+1} - \mathbf{U}_{n+1}^i$$

Equation (3.89) represents what is known as a residual driven algorithm. The residual $\mathbf{R}(\mathbf{U}_{n+1}^{i+1}, \mathbf{U}_n)$ on the left-hand side should be driven to zero by equating the right-hand side of the equation to zero and solving for \mathbf{U}_{n+1}^{i+1} . As long as this iterative solution converges, it will converge toward a solution of our original system of nonlinear equations, and the order of accuracy for the final converged solution is preserved.²³

As it is written, Equation (3.89) defines an implicit iterative algorithm due to coupling in the tangent vector $\partial \mathbf{R} / \partial \mathbf{U}$. This coupling may be eliminated by approximating the tangent vector with a lumped form of the finite element mass matrix. The “lumped mass” matrix is obtained by summing each row of the previously derived finite element mass matrix and lumping that sum on the diagonal of the matrix. In three-dimensions, the lumped mass matrix is given by Equation (3.91).

$$(3.91) \quad \int_{\Omega_e} (\Phi_e^T \Phi_e - \Phi_e^T \Phi_e) d\Omega \approx \frac{V_e}{4} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{M}_L$$

Substitution of the lumped mass matrix into Equation (3.89) produces the following explicit equation for advancing each iteration:

$$(3.92) \quad \mathbf{R}(\mathbf{U}_{n+1}^i, \mathbf{U}_n) + \mathbf{M}_L \Delta \mathbf{U}_{n+1}^i = 0$$

The iterative algorithm defined by this equation is explicit now because the lumped mass matrix is diagonal and may be inverted directly to solve for the new nodal unknowns vector. Also notice that this method retains all of the implicit terms in the solution residual, so it is still considered an implicit formulation. As suggested above, the final order of accuracy for the converged solution is not affected by our use of the lumped mass matrix since it is only necessary to drive the solution residual to zero. This approximation only affects the stability and convergence rate of the algorithm.

3.5.1 Steady Solutions

When computing a steady solution, it is possible to further simplify the predictor multi-corrector algorithm given by Equation (3.92). At a converged steady state condition, the jump condition in the solution residual should vanish since the nodal unknowns vector becomes constant. Because we are only interested in computing the final converged state, it is common to approximate the finite element mass matrix, which is multiplied by the jump condition in the solution residual, with the lumped form of the mass matrix. This approximation eliminates some of the implicit coupling in the solution residual at the expense of time accuracy, but does not affect the final converged state since the jump condition is identically zero at steady state.

Substitution of the lumped mass matrix into Equation (3.92) allows us to write the following iterative time advancement scheme:

$$(3.93) \quad \mathbf{U}_{n+1}^{i+1} = \mathbf{U}_n - \Delta t \cdot \mathbf{M}_L^{-1} \cdot \mathbf{R}_s(\mathbf{U}_{n+1}^i)$$

where

$$(3.94) \quad \mathbf{R}_s = -\mathbf{FS}(\mathbf{U}_{n+1}^i) + \mathbf{B}(\mathbf{U}_{n+1}^i)$$

In the above expression, \mathbf{R}_s is the “steady” solution residual. Notice that the steady solution residual now represents the difference between the nodal unknowns vectors from the previous and current time intervals rather than the difference between two explicit iterations on the current time interval.

When implementing the predictor multi-corrector algorithm give by Equation (3.94), it is necessary to chose a suitable time increment for advancing the solution. It is well known that explicit iterative methods are subject to a Courant type stability criteria where the maximum allowable time increment is proportional to the time it takes for a fluid particle to traverse the length of an element. Unfortunately, this means that the time increment will be limited by the smallest elements in the spatial domain if we use one global time increment for the entire domain. Obviously this is not an optimal situation since flow information will propagate at different rates throughout the domain based on the local element size and fluid velocity. Therefore, it is common to compute local time increments for each node in the spatial domain in order to advance the solution at the most optimal rate. Obviously, this will further degrade the time accuracy of the multi-corrector algorithm, but it does not affect the final converged state since the solution residual vanishes at steady state.

There are a variety of techniques documented in the literature for computing the local time increment for a node. For our application, we employ the methodology currently implemented in the STARS CFD modules. The methodology is derived according to an energy stability analysis, which relates the local time increment to the eigenvalues for each segment connected to a node, when using the edge-based data structure previously described.¹⁸

$$(3.95) \quad \Delta t_i = 2cfl[\mathbf{M}_L]_i \left[\sum_{s=1}^{ns} |\lambda_{ij,s}| \right]^{-1}$$

where

$$(3.96) \quad \lambda_{ij,s} = |\mathbf{u}_{ij,s} \cdot \mathbf{W}_{ij,s}| + a_{ij,s} |\mathbf{W}_{ij,s}|$$

In the above expressions, cfl is the Courant stability factor, $[\mathbf{M}_L]_i$ is the value of the global lumped mass matrix at node i , ns is the number of segments connected to node i , $\lambda_{ij,s}$ is the eigenvalues for the segment s connecting nodes i and j , and $\mathbf{u}_{ij,s}$, $\mathbf{W}_{ij,s}$ and $a_{ij,s}$ are the fluid velocity vector, geometric weight vector and speed of sound for the same segment.

3.5.2 1st Order Unsteady Solutions

If we are interested in a time-accurate unsteady solution, it will be necessary to use the unmodified, implicit solution residual as the basis for our predictor multi-corrector algorithm. Such a formulation will allow us to compute true transient flows where the jump condition is nonzero. However, we will modify the original predictor multi-corrector algorithm given by Equation (3.92) so that it resembles the steady algorithm derived in the previous section with the time increment factored out of the solution residual.

$$(3.97) \quad \mathbf{U}_{n+1}^{i+1} = \mathbf{U}_{n+1}^i - \Delta t \cdot \mathbf{M}_L^{-1} \cdot \mathbf{R}_1(\mathbf{U}_{n+1}^i, \mathbf{U}_n)$$

where

$$(3.98) \quad \mathbf{R}_1 = \frac{1}{\Delta t} \mathbf{M}(\mathbf{U}_{n+1}^i - \mathbf{U}_n) - \mathbf{FS}(\mathbf{U}_{n+1}^i) + \mathbf{B}(\mathbf{U}_{n+1}^i)$$

In the above expression, \mathbf{R}_1 is the unsteady solution residual. This predictor multi-corrector algorithm, which is based on a constant-in-time approximation, has been demonstrated to be first-order accurate in time.²⁹

Since we are now computing time-accurate unsteady solutions, it is typically necessary to choose a global time increment for advancing the solution for the entire computational domain. However, we could instead view Equation (3.97) as a modified steady problem since we have factored the time increment out of the solution residual. In this case, the solution residual represents the difference between two iterations for one time increment and we utilize local time stepping to accelerate the convergence of this residual to zero. This idea, where an unsteady problem is solved as a modified steady problem using local time stepping, was first proposed by Jameson³⁰ and is implemented in the current STARS unsteady CFD module.¹⁸

To implement this method, we select a global time increment for computing the solution residual given by Equation (3.98), and use a local time increment to update the unknowns for each iteration in the predictor multi-corrector algorithm given by Equation (3.97). The final converged state for each global time step will satisfy the condition that the unsteady solution residual is zero. Thus we maintain the time accuracy of the algorithm despite using local time stepping to drive the solution to convergence. Furthermore, the stability of this algorithm is maintained even for large global time increments since the algorithm is driven to convergence using a locally stable time increment for each node. However, we do expect that the number of iterations necessary to attain convergence will increase as the size of the global time increment increases.

In the original work on this method, Jameson³⁰ proposed the use of a multistage advancement scheme for each iteration in order to maximize the stability of the method for large time increments. The same multistage advancement scheme was also adopted within the STARS unsteady CFD module. The idea is to use up to five stages to advance the unknowns for each iteration. For example, one might use ten iterations for each global time increment and five stages for each iteration, which means the algorithm must re-compute the unknowns a total of fifty times for each global time increment. Our experience has shown that the same effect can be obtained by decreasing the Courant stability factor and simply using more iterations with the basic one stage scheme already presented. Furthermore, excessively large global time increments will exceed the time resolution of the algorithm, which is only first-order accurate, and should not be used. Therefore, we do not advocate the use of a multistage scheme here.

While the focus of this method is to increase the size of the maximum allowable global time increment, an interesting numerical problem manifests itself for small global time increments. Surprisingly, the current STARS unsteady CFD module becomes unstable during a simple time increment convergence test where the global time increment is successively decreased in size. This is contrary to how a well formulated numerical algorithm should behave, but it is easily explainable and corrected. Notice that the first term of the unsteady solution residual defined by Equation (3.98) will be scaled by a ratio of local to global time increments when it is substituted into the multi-corrector algorithm defined by Equation (3.97). This ratio is analogous to a numerical relaxation factor for the iterative algorithm and explains the numerical instability we observe for small global time increments.

As long as the global time increment is larger than the computed local time increments, the algorithm is using an under-relaxation factor. Conversely, an over-relaxation factor results whenever the computed local time increment exceeds the size of the global time increment. From a physical standpoint, over-relaxation is not expected to be desirable since the algorithm would actually over-shoot the correct answer. This insight concurs with our numerical tests, which showed that the STARS unsteady CFD module has an obvious numerical instability when the global time increment is relatively small. To our knowledge, this instability has not been documented in the current literature, but it is now obvious how to remedy the problem.

Any time the computed local time increment exceeds the value of the global time increment, the local time increment defaults to the value of the global time increment. This guarantees that the ratio of local to global time increments does not exceed one, and the multi-corrector algorithm always uses under-relaxation to advance the solution.

3.5.3 2nd Order Unsteady Solutions

The unsteady algorithm developed in the previous section is known to be only first-order accurate in time.²⁹ Our tests have shown that this algorithm is sufficient for a wide variety of unsteady problems as long as a sufficiently small global time increment is used. However, an algorithm with a higher order of accuracy is needed if a large global time increment is desired. The obvious procedure for increasing this order of accuracy is to modify the finite element interpolation function used to represent time in our derivation. Thus far, it has been assumed that the unknowns are piecewise constant in time. Alternatively, a third-order accurate method can be developed if the unknowns are

assumed to be piecewise linear in time.²⁹ This assumption leads to a significantly more complicated algorithm with two sets of unknowns that must be updated for each space-time sub-domain.¹⁷

In order to avoid this increase in complexity, we will attempt to increase the order of accuracy by modifying our algorithm in a different manner. Notice in the previous derivation that the jump condition is identical to a backwards difference operator when Equation (3.88) is re-written as follows:

$$(3.99) \quad \mathbf{M} \left(\frac{\mathbf{U}_{n+1} - \mathbf{U}_n}{\Delta t} \right) - \mathbf{FS}(\mathbf{U}_{n+1}) + \mathbf{B}(\mathbf{U}_{n+1}) = 0$$

With this in mind, we can consider increasing the order of accuracy for the algorithm by substituting a higher-order difference operator for the jump condition. This would be analogous to changing the initial conditions for the space-time sub-domain using a higher-order interpolation of unknowns from the previous sub-domain, but without actually changing the interpolation of unknowns within the sub-domain.

The current STARS unsteady CFD module uses the second-order accurate backwards difference operator suggested by Jameson.³⁰ Substitution of this difference operator for the jump condition produces the following predictor multi-corrector algorithm for unsteady problems:

$$(3.100) \quad \mathbf{U}_{n+1}^{i+1} = \mathbf{U}_{n+1}^i - \Delta t \cdot \mathbf{M}_L^{-1} \cdot \mathbf{R}_2(\mathbf{U}_{n+1}^i, \mathbf{U}_n)$$

where

$$(3.101) \quad \mathbf{R}_2 = \frac{1}{\Delta t} \mathbf{M} \left(\frac{3}{2} \mathbf{U}_{n+1}^i - 2\mathbf{U}_n + \frac{1}{2} \mathbf{U}_{n-1} \right) - \mathbf{FS}(\mathbf{U}_{n+1}^i) + \mathbf{B}(\mathbf{U}_{n+1}^i)$$

The unsteady algorithm defined above is expected to be second-order accurate in time, and the computational difference between the two unsteady algorithms is negligible.

Notice that the only difference between the two algorithms is the form of the jump condition. One of the side effects of modifying the jump condition is that the stability characteristics of the two algorithms will be different. One of the roles the jump condition plays is to add a consistent and higher-order numerical dissipation to the algorithm.¹⁷ Our tests have shown that the second-order method is less stable and will typically require more artificial dissipation than the first-order method.

3.6 Stabilization

It is well documented that any discrete CFD solution for the compressible Euler equations will exhibit numerical instabilities in the form of spurious oscillations, despite the fact that the analytical solution remains smooth, monotone and bounded. These oscillations emanate from regions with sharp gradients and grow rapidly until the solution is globally corrupted. As stated previously, this situation is not unique to our finite element discretization. In fact, most of the stabilization techniques we will discuss in this section were derived from similar techniques used to stabilize finite difference schemes.

There are a variety of techniques for improving the poor stability properties of our discrete CFD solution, but all of them are based on adding some sort of dissipative mechanism to the solution. Therefore it is necessary to modify the solution residuals derived in the previous section to include an extra dissipative operator \mathbf{D} . For example, the first-order unsteady solution residual would now be expressed as follows:

$$(3.102) \quad \mathbf{R}_1 = \frac{1}{\Delta t} \mathbf{M}(\mathbf{U}_{n+1}^i - \mathbf{U}_n) - \mathbf{F}\mathbf{S}(\mathbf{U}_{n+1}^i) + \mathbf{B}(\mathbf{U}_{n+1}^i) + \mathbf{D}(\mathbf{U}_{n+1}^i)$$

This dissipative mechanism may come from upwind differencing, explicit addition of artificial diffusion, or a residual based dissipation operator. Within these broad classifications, literally hundreds of algorithms exist, and we will review several of the most advanced finite element methods here.

The current STARS CFD modules use a form of explicit artificial dissipation to stabilize the solution scheme. While this method has proven reliable for a variety of applications, it has some limitations that make it less than the optimal choice. In particular, artificial dissipation methods violate the consistency of our weighted residual formulation because the exact solution no longer satisfies the algorithmic residual.²² This degrades the overall order of accuracy for the algorithm.

3.6.1 SUPG and GLS

The Streamline-Upwind/Petrov-Galerkin (SUPG) method and the Galerkin Least Squares (GLS) method have been the topic of much research lately, especially with regards to space-time finite element formulations. These methods are similar in that they both rely on residual based dissipation mechanisms. One of the key advantages of such a method is that the stabilization control is introduced directly within the weighted residual expression. This maintains the consistency of the formulation since the dissipation operator vanishes as the residual approaches zero.²² The following equations present the SUPG and GLS stabilization operators for a conservation variables formulation in an inertial frame using constant-in-time elements:

$$(3.103) \quad \mathbf{D}_{SUPG} = \sum_{e=1}^{n_{el}} \int_{\Omega_e} \boldsymbol{\tau}_s \left(\mathbf{A}_i^T \frac{\partial \Phi}{\partial x_i} \right) \cdot \left(\mathbf{A}_i \frac{\partial \Phi}{\partial x_i} \mathbf{U} \right) d\Omega$$

$$(3.104) \quad \mathbf{D}_{GLS} = \sum_{e=1}^{n_{el}} \int_{\Omega_e} \left(\mathbf{A}_i^T \frac{\partial \Phi}{\partial x_i} \right) \cdot \boldsymbol{\tau}_g \left(\mathbf{A}_i \frac{\partial \Phi}{\partial x_i} \mathbf{U} \right) d\Omega$$

where

$$(3.105) \quad \mathbf{A}_i = \frac{\partial \mathbf{F}_i}{\partial \mathbf{U}}$$

Notice in the above expressions that the two operators are very similar. The fundamental difference between the two lies in the structure of the $\boldsymbol{\tau}$ matrix. The SUPG methodology has been the most widely implemented of the two, in part because the formulation of the $\boldsymbol{\tau}$ matrix is fairly simple and straightforward. In fact, there are a variety of recommendations for the structure of $\boldsymbol{\tau}_{SUPG}$ for the compressible Euler equations. For example, Hughes suggests that $\boldsymbol{\tau}_{SUPG}$ should be a diagonal matrix whose diagonal elements are given by the following equation:

$$(3.106) \quad \tau_{SUPG} = \max \left[0, \frac{h}{2(a + |\mathbf{u}|)} \right]$$

where h is the minimum element dimension and a is the local speed of sound.

In contrast, the construction of $\boldsymbol{\tau}_{GLS}$ is neither simple nor straightforward for the three-dimensional Euler or Navier-Stokes equations. In fact, the formulation for $\boldsymbol{\tau}_{GLS}$ involves the inverse square root of the flux jacobian matrices, \mathbf{A}_i , for each element.¹⁷ This is a very expensive computational operation as it requires the solution of an eigenvalues problem for each element. Specifically, our tests showed that this increase in complexity leads to a factor of ten increase in CPU time over stabilized artificial dissipation methods for simple one-dimensional problems. The GLS methodology is arguably the more accurate of the two methods and leads to a system of equations whose stability and convergence characteristics can be rigorously analyzed.²⁹ However, it is

unlikely that this methodology will be widely accepted for complicated aerospace applications unless a more direct formulation for τ_{GLS} is developed.

Although the GLS method appears to be too numerically expensive for our applications, the SUPG method still appears to be an efficient option that has the capacity to stabilize the solution scheme while maintaining the order of accuracy of the algorithm. However, both GLS and SUPG are linear methods that cannot produce monotone solutions for discontinuities.³¹ In practice, this means that overshoots and undershoots still develop around shocks, but are controlled so that they no longer globally pollute the solution.²² Therefore, it becomes necessary to introduce non-linear operators to further control these oscillations and produce smooth solutions for discontinuities.

It is also possible to formulate a residual based discontinuity capturing operator to accomplish this. This type of non-linear operator maintains the consistency of the finite element formulation as with the SUPG and GLS stabilization operators already presented. Unfortunately, residual based discontinuity operators are formulated using the same least squares operator from the GLS formulation, which involved the inverse square-root of a matrix.^{17,32} Thus, the computational performance of the SUPG method will be similar to that of the GLS method already discussed. It is unfortunate that both of these methods suffer from such poor computational efficiency when compared with similar artificial dissipation method since they would seem to offer a significant improvement in accuracy over such methods. However, it is unlikely that the improved accuracy would be welcomed if it accompanied an order of magnitude increase in computational time since the current STARS artificial dissipation model has proven to be reliable for many applications.

3.6.2 Artificial Dissipation

The computational fluid dynamics literature abounds with artificial dissipation models for compressible flows. In comparing several of these models for this research effort, the STARS dissipation model performs comparably to other models in terms of its ability to capture shocks over relatively small intervals while still sufficiently stabilizing the solution. The STARS dissipation model is identical to the model employed by the FELISA codes, and is an edge-based scheme that relies on the computation of flow gradients along the segments of the computational domain.¹⁹ Interested readers should refer to the FELISA research literature for the mathematical equations describing this dissipation model. We will only describe its basic function and make some general comments on its application to non-inertial solutions in this work since the dissipation model is not our intellectual achievement.

STARS employs two different versions of its dissipation model, the so-called low-order and high-order dissipation. Both versions are based on the same fundamental set of equations, with the difference being that the high-order dissipation utilizes gradient limiters to reduce the amount of dissipation that is added to the solution in regions where there are real flow discontinuities. This is accomplished using a pre-processing step, where first the spatial gradients for every node are computed using the segment weights as discussed in Section 3.3.4. Next, a set of modified unknowns are computed using the nodal gradient information. Using this set of modified unknowns, the amount of dissipation for each node is then computed in the same way as the low-order dissipation.

Since the dissipation operates directly on relative flow quantities, the fundamental dissipation equations used by STARS required essentially no modification for use with

our non-inertial solutions. However, one significant change was required for the high-order dissipation in the pre-processing stage where it computes the nodal gradients. The original high-order dissipation methodology applied boundary conditions to the gradient vectors it computed. This translated into the assumption of zero gradient across the far-field boundaries. Therefore, the nodal gradients for every node on the far-field boundary were set equal to zero. For a non-inertial problem, this is no longer a good or accurate assumption. Consider the case of a non-translating, rotating domain where the relative flow velocity is given by Equation (3.107).

$$(3.107) \quad \mathbf{u} = \mathbf{\Omega} \mathbf{r}$$

In the above expression, $\mathbf{u} = \{u_1, u_2, u_3\}^T$ is the fluid velocity vector, $\mathbf{\Omega}$ is the angular rotation matrix defined in Section 2.3, and \mathbf{r} is a vector location within the computational domain.

For this type of problem, the velocity gradient will be non-zero everywhere in the computational domain, including at the far-field boundary. The simplest way to correct for this is to skip the enforcement of any boundary conditions on the nodal gradients and use whatever relative gradient the solver computes for the far field nodes of the domain. Our tests have shown that this has no adverse effect on inertial or stationary domains, and it corrects an observed problem with rotating non-inertial domains where too much dissipation is added at the far-field boundaries due to a lack of accurate gradient information. In fact, the modified high-order dissipation is a practical requirement for non-inertial solutions because the low-order dissipation (with-out the gradient limiters) will effectively smooth out all strong gradients produced by a rotating domain to the point of converging on a solution where the relative velocity is zero everywhere.

This is a rather unfortunate result because the high-order dissipation requires more than twice as much computational time as the low-order dissipation. Furthermore, solutions that use the high-order dissipation typically converge at a slower rate than the same solution using the low-order dissipation. Based on its computational performance, the low-order dissipation would be the ideal choice for efficient numerical solutions. However, it is typically too dissipative, resulting in solutions with gradients that are excessively smeared.

That is not to say the low-order dissipation is completely useless. For supersonic problems, solutions using the high-order dissipation tend to be unstable for high Mach numbers, while the low-order dissipation is actually quite stable and produces accurate solutions. For subsonic and transonic flows, accurate solutions can sometimes be obtained using the low-order dissipation by decreasing the amount of dissipation added to the solution. This is accomplished by decreasing the dissipation scaling factor, which is a constant scaling factor applied in the calculation of the dissipation for each node. However, this is a uniform decrease of the dissipation everywhere in the domain and typically leads to a numerical instability before reaching the point of producing an accurate solution. In contrast, the high-order dissipation uses a non-linear mechanism for locally decreasing the amount of dissipation.

CHAPTER 4

4. COMPUTER IMPLEMENTATION

This Chapter discusses the practical implementation of our finite element methodology as a working computer algorithm. The first two sections discuss the details of how the source code for the algorithm is structured. Specifically, we define the fundamental data structures that are required by the algorithm and provide an overview of the basic algorithm's structure and timing. The next two sections provide some details on the operation and control of the computer algorithm, including a definition of the algorithmic control parameters and all of the available computer executables developed for this research. This information is supplemented by Appendix A and Appendix B, which define the format of all input and output data files used by our codes.

The next section discusses the structural dynamics solver that is currently available for unsteady flow solutions. This discussion includes some key issues on consistent non-dimensionalization of the structural parameters for a coupled unsteady CFD analysis of aeroelastic or flight dynamics problems. The final section summarizes the differences between the new algorithm that has been developed here and the old STARS CFD modules. This includes details on memory usage and computational performance for the two algorithms.

4.1 Data Structures

Perhaps the most complicated aspect of any two or three-dimensional CFD algorithm is the organization and interpretation of its data structures. As such, the computational performance of our numerical algorithm will depend on how efficiently these structures are organized. In order to develop an algorithm based on the methodology presented so far, we require the following fundamental sets of data:

- COOR \Rightarrow real vectors of nodal coordinate data for nodes 1 through nnd
- IELM \Rightarrow integer vectors of element connectivity data for elements 1 through nel
- ISEG \Rightarrow integer vectors of segment connectivity data for segment 1 through nsg
- IBEL \Rightarrow integer vectors of boundary element connectivity data for boundary elements 1 through nbe
- PHIA \Rightarrow real vectors of elastic deformations for each wall node
- G2D/G3D \Rightarrow real vectors of element geometry data for each element
- DM \Rightarrow real vector of inverse lumped mass values for each node
- WSG \Rightarrow real vectors of segment weights for each segment
- RBE \Rightarrow real vectors of boundary element geometry data for each boundary element
- ANOR \Rightarrow real vectors of averaged surface normals for wall nodes 1 to nwl
- BVEL \Rightarrow real vectors of elastic boundary velocities for each wall node
- UN \Rightarrow real vectors of algorithm unknowns at $t = n$ for each node
- UN1 \Rightarrow real vectors of algorithm unknowns at $t = n + 1$ for each node
- UNO \Rightarrow real vectors of algorithm unknowns at $t = n - 1$ for each node
- RHS \Rightarrow real vectors of assembled solution residuals for each node

Only the first five sets of data listed above will actually be read in by the solver. This is to allow for maximum compatibility with a wide range of unstructured grid generation packages. However, some pre-processing of this geometry data is necessary so that it is sorted appropriately. The algorithm developed here requires that the boundary elements be sorted based on boundary condition type so that the boundary integrals can be evaluated efficiently, i.e. without complicated switches or if blocks. Furthermore, nodes which lie on solid wall surfaces must be sorted to the front of the nodal arrays, i.e. nodes 1 through nwl are solid wall nodes and nodes $nwl + 1$ through nnd are the rest of the nodes in the domain. This facilitates an efficient implementation of the solid wall boundary condition, which is applied explicitly to each solid wall node. As discussed in Section 3.4.3, some solid wall nodes are omitted from the enforcement of boundary conditions due to ambiguity in the average surface normal for that node. These nodes are flagged as singular and are sorted to be last nsd nodes of the solid wall nodes, i.e. nodes 1 through $nwl - nsd$ are the solid wall nodes on which the boundary conditions will be enforced.

The next five sets of geometry data, G2D/G3D through ANOR, are assembled once by the algorithm when it is initialized. The total computational time required for this operation is approximately equal to the time required for one or two iterations of the actual flow solver. The element geometry vectors, G2D/G3D, contain the element area or volume for two or three-dimensional elements respectively and the coordinate transformation matrix needed to convert gradients from Cartesian coordinates to element natural coordinates. The lumped mass vector, DM, contains the inverse of the diagonal elements of the lumped mass matrix for each node scaled such that it is proportional to

the area or volume of the two or three-dimensional elements surrounding that node. This is equivalent to multiplying the entire algorithmic equation by three for two-dimensional problems or four for three-dimensional problems, and is purely a choice of convenience. The segment weight vector, WSG, contains the appropriately scaled segment weight vector for each segment as discussed in Section 3.3.4. The boundary element data vector, RBE, contains the length or area for two or three-dimensional boundary elements respectively, and the outward pointing, unit normal vector for each boundary element. Finally, the averaged normal vector, ANOR, contains the length or area weighted average normal vector for each node that lies on a two or three-dimensional solid wall boundary respectively. These averaged wall normals are also unit vectors pointing outward from the computational domain.

The last five sets of fundamental data represent vectors that are computed by the algorithm during the iterative solution process presented thus far. BVEL is a vector of computed velocities for each solid wall node that is used for the transpiration boundary condition when solving elastic problems. UN, UN1, and UNO are vectors of algorithm unknowns from the current, next, and previous time steps respectively, and RHS is the computed solution residual for whichever predictor multi-corrector algorithm is being used.

4.2 Basic Algorithm

With all of the fundamental data structures defined, the core finite element methodology can be summarized in pseudo-code as follows:

```

read solver control parameters
read geometry data: COOR, IELM, ISEG, IBEL
compute additional geometry data: G2D/G3D, DM, RBE, WSG, ANOR
read any elastic/dynamic data
set/read initial conditions for UN for  $t = 0$ 
compute initial aerodynamic loads for  $t = 0$ 
compute initial structural dynamics state for  $t = 0$ 
output initial conditions for  $t = 0$ 
UN0 = UN
UN1 = UN
do istp = 1, nstp
    advance structural dynamics from  $t = n$  to  $t = n + 1$ 
    update ANOR and compute BVEL (transpiration) for  $t = n + 1$ 
    compute local time step, DELT, for each node
    do icyc = 1, ncyc
        initialize RHS
        enforce flow tangency on UN1
        add element integrals to RHS
        add boundary integrals to RHS
        add dissipation to RHS
        enforce flow tangency on RHS
        UN1 = UN1 - DELT·DM·RHS
    enddo
    output solution residuals
    UN0 = UN
    UN = UN1
    compute new aerodynamic loads for  $t = n + 1$ 
    output forces and dynamics for  $t = n + 1$ 
    if MOD(istp, nout) = 0, output solution unknowns
enddo

```

Figure 4.1: Pseudo-code summary of core CFD algorithm.

The above pseudo-code shows the overall timing of the code in terms of the defined data structures. It also defines three solver control parameters: `nstp`, `ncyc`, and `nout`. The total number of solution steps is controlled by the parameter `nstp`, the number of iterative convergence cycles is controlled by the parameter `ncyc`, and the output frequency is controlled by the parameter `nout`. Solver control parameters will be discussed in more detail in the Section 4.4. Additional information on all solver control

parameters and other input/output data files for both the two-dimensional and three-dimensional solvers is also present in Appendix A and Appendix B.

One final modification will be made to this algorithm in order to help maintain the stability of the iterative procedure. It is possible during the iterative process that the update equation in the above algorithm, $UN1 = UN1 - \Delta t \cdot DM \cdot RHS$, will predict a negative value for the density or pressure due to a local instability in the algorithm. Such a value is obviously a nonphysical representation for either of these quantities and will cause the program to crash when it computes other quantities that require the square-root of these variables, such as the local speed of sound. The current STARS CFD modules use a method originally developed for the FELISA CFD solver to force these quantities to remain positive.¹⁹ This helps maintain the stability of the algorithm in the face of mild instabilities that might develop locally and extends the range of the maximum allowable global time step and Courant stability factor, $cf1$.

In order to efficiently implement this method, it is necessary to use a set of algorithmic unknowns that includes the pressure. As such, the algorithmic unknowns vector will be defined as follows:

$$(4.1) \quad \mathbf{Q} = \begin{Bmatrix} \rho \\ u_1 \\ u_2 \\ u_3 \\ p \end{Bmatrix}$$

Notice that the algorithmic unknowns vector differs from the Euler unknowns vector previously derived. To some extent, this will complicate the evaluation of element integrals, but it streamlines the calculation of quantities that involve the pressure since the Euler unknowns did not include pressure as an unknown. Furthermore, this modification

will make the output from our new algorithm more compatible with the post-processors developed to analyze data output by the STARS CFD solvers.

The equation for updating the density and pressure for each iteration is then as follows:

$$(4.2) \quad q_{i+1} = \begin{cases} q_i + \Delta q & \Delta q/q_i \geq -0.1 \\ q_i + \frac{\Delta q}{1 - 4(0.1 + \Delta q/q_i)} & \Delta q/q_i < -0.1 \end{cases}$$

In the above expression, q is the unknown quantity being updated and Δq is the predicted change in that quantity computed from the solution residual. Use of this equation for updating the density and pressure will ensure that these quantities remain positive from one iteration to the next. However, it will not universally guarantee the stability of the algorithm. This process only prevents mild instabilities from globally polluting the solution.

4.3 Available Codes

A variety of computer codes have been developed for this research effort. Although the names may be changed in the near future, a summary of all the available codes and a description of their function is provided here. As mentioned in the previous section, refer to Appendix A and Appendix B for details on the input and output file formats for most of these codes.

The following executables are available for two-dimensional CFD analysis:

- `makeg2d.exe` is a pre-processor used to convert a standard STARS surface triangulation file and modified boundary conditions file into an appropriately sorted two-dimensional geometry file. The surface

triangulation data must be restricted to one of the following coordinate planes: *xy*-plane, *xz*-plane, or *yz*-plane.

- `euler2d.exe` is the finite element CFD algorithm used to perform a two-dimensional steady or unsteady CFD analysis for the specified two-dimensional computational domain.
- `makecut2d.exe` is a post-processor used to extract relevant flow quantities along arbitrary cut-lines through the computational domain or along the individual boundary curves of the computational domain.
- `particle2d.exe` is a post-processor used to generate steady particle traces or stream lines through-out the computational domain for a given set of solution unknowns.
- `glplot2d.exe` is a graphical post-processor used for visualization of the two-dimensional geometry, flow solution, and particle traces.

The following executables are available for three-dimensional CFD analysis:

- `makeg3d.exe` is a pre-processor used to convert a standard STARS surface triangulation file, tetrahedral volume file, and boundary conditions file into an appropriately sorted three-dimensional geometry file.
- `euler3d.exe` is the finite element CFD algorithm used to perform a three-dimensional steady or unsteady CFD analysis for the specified three-dimensional computational domain.
- `makecut3d.exe` is a post-processor used to extract relevant flow quantities along arbitrary cut-lines through the computational domain or

along the intersection of cut-planes with the individual boundary surfaces of the computational domain.

- `particle3d.exe` is a post-processor used to generate steady particle traces or stream lines through-out the computational domain for a given set of solution unknowns.
- `glplot3d.exe` is a graphical post-processor used for visualization of the three -dimensional geometry, flow solution, and particle traces.

4.4 Algorithm Control

The behavior of the primary CFD algorithms, `euler3d.exe` and `euler2d.exe`, is controlled through a set of control parameters that are read from a Fortran namelist file. The following parameters and flags are available with their default setting given in parentheses:

- `dt` \Rightarrow dimensionless global time step for unsteady solutions (0 . 1)
- `gamma` \Rightarrow ratio of specific heats (1 . 4)
- `diss` \Rightarrow artificial dissipation scaling factor (1 . 0)
- `cfl` \Rightarrow courant stability factor for local time steps (0 . 5)
- `mach` \Rightarrow free stream Mach number (0 . 6)
- `alpha` \Rightarrow 1st free stream orientation angle (0 . 0)
- `beta` \Rightarrow 2nd free stream orientation angle (0 . 0)
- `refdim` \Rightarrow reference dimension for non-dimensionalization of problem (1 . 0)
- `ainf` \Rightarrow dimensional free stream speed of sound (1 . 0)
- `rhoinf` \Rightarrow dimensional free stream density (1 . 0)

- `nstp` \Rightarrow total number of solution time steps (100)
- `nout` \Rightarrow output frequency, number of steps per output (50)
- `ncyc` \Rightarrow number of iterative cycles per solution step (3)
- `istrt` \Rightarrow restart flag (.false.)
- `isol` \Rightarrow solution type (0)
 - steady solution: `isol = 0`
 - 1st order unsteady solution: `isol = 1`
 - 2nd order unsteady solution: `isol = 2`
 - piston perturbation solution: `isol = 3`
- `idsol` \Rightarrow structural dynamics solution type (2)
 - zero-order integration: `idsol = 0`
 - 1st order integration: `idsol = 1`
 - 2nd order integration: `idsol = 2`
- `idiss` \Rightarrow dissipation type (0)
 - low-order dissipation: `idiss = 0`
 - high-order dissipation: `idiss = 1`
- `ipnt` \Rightarrow number of points for numerical integration of flux/source vectors(1)
 - one-point gauss quadrature: `ipnt = 1`
 - three-point symmetric gauss quadrature (2-D only): `ipnt = 3`
 - four-point symmetric gauss quadrature (3-D only): `ipnt = 4`
- `iaero` \Rightarrow aerodynamic forces flag (.false.)
- `idynm` \Rightarrow dynamic/non-inertial solution flag (.false.)
- `ielast` \Rightarrow elastic solution flag (.false.)
- `ifree` \Rightarrow free-stream velocity flag (.true.)

- `nr` \Rightarrow number of elastic modes (0)

Many of the solver control parameters listed above are, to some extent, self-explanatory based on the structure of the algorithm presented thus far. However, additional information about these parameters is provided in Appendix A and Appendix B in the section that details the format of the control input files for each solver. Furthermore, Chapter 5 will investigate the application of three-dimensional solver to flow problems of interest. For each of these problems, the appropriate choice of control parameters will be specified.

4.5 Aerodynamic Forces

For `iaero = .true.`, fundamental aerodynamic forces and moments are computed following each solution step. A vector of aerodynamic forces and moments is computed by summing the force/moment contributions for every solid wall boundary element as follows:

$$(4.3) \quad \mathbf{f}_a^* = \sum_{i=\text{LBE}(1)}^{\text{LBE}(2)} 2A_i (p_i - p_{\text{inf}}) \mathbf{n}_i$$

$$(4.4) \quad \mathbf{M}_a^* = \sum_{i=\text{LBE}(1)}^{\text{LBE}(2)} [2A_i (p_i - p_{\text{inf}}) \mathbf{n}_i] \times \mathbf{r}_i$$

In the above expression, `LBE(1)` and `LBE(2)` are the starting and stopping indexes for solid wall boundary elements, A_i is the area for boundary element i , p_i is the average pressure acting on boundary element i , p_{inf} is the free-stream pressure, \mathbf{n}_i is the unit normal vector for boundary element i , and \mathbf{r}_i is a vector from the origin of rotation to the

center of boundary element i . For two-dimensional problems, the element area in Equations (4.3) and (4.4) is replaced with the element length.

Since the CFD algorithm solves for a dimensionless pressure as defined in Section 2.4, the aerodynamic forces and moments computed using Equations (4.3) and (4.4) will be dimensionless force and moment coefficients respectively and are marked by an asterisk. Both of these equations are multiplied by two so that they are non-dimensionalized with respect to a dynamic pressure, since that is the aerodynamic standard for non-dimensionalization of forces and moments. These dimensionless quantities can be converted to dimensional forces and moments as follows:

$$(4.5) \quad \mathbf{f}_a = \left(\frac{1}{2} \rho_{\text{inf}} u_{\text{inf}}^2 L^2\right) \mathbf{f}_a^*$$

$$(4.6) \quad \mathbf{M}_a = \left(\frac{1}{2} \rho_{\text{inf}} u_{\text{inf}}^2 L^3\right) \mathbf{M}_a^*$$

In the above expressions, ρ_{inf} is the free-stream density, u_{inf} is the free stream velocity, and L is the reference length or dimension.

In addition to the fundamental aerodynamic forces and moments, generalized aerodynamic forces are also computed for elastic problems, `ielast = .true.` A generalized aerodynamic force is computed for each elastic mode by summing nodal contribution for every solid wall node as follows:

$$(4.7) \quad f_a^* = \sum_{i=1}^{\text{nw1}} [2A_i (p_i - p_{\text{inf}}) \mathbf{n}_i] \cdot \Phi_i$$

In the above expression, `nw1` is the number of solid wall nodes and Φ_i is the elastic deformation vector for node i . For two-dimensional problems, the element area in Equation (4.7) is again replaced with the element length.

Although Equation (4.7) is defined as a summation over solid wall nodes, the generalized aerodynamic forces are actually assembled in the same manner as the fundamental aerodynamic forces, i.e. by looping over solid wall boundary elements and calculating their contribution to each node of that element. The dimensionless form of a generalized aerodynamic force is similar to that of an aerodynamic moment since the elastic deformation vector has units of length. Therefore, the generalize aerodynamic force given by Equation (4.7) can be converted to a dimensional force as follows:

$$(4.8) \quad f_a = \left(\frac{1}{2} \rho_{\text{inf}} u_{\text{inf}}^2 L^3 \right) f_a^*$$

Both sets of aerodynamic forces described here will be required by the structural dynamics solver described in the next section. The fundamental aerodynamic forces and moments are used to compute the non-inertial or rigid-body dynamics of the system and are stored in a six-element array (three-element for two-dimensional problems) called FD by the algorithm, while the generalized aerodynamic forces are used to compute the elastic deformation of the structure and are stored in a nr element array designated as FA by the algorithm.

4.6 Structural Dynamics

Through-out most of this research effort, the structural dynamics algorithm required for a couple unsteady CFD solution was considered to be a low risk area of development. The existing STARS structural dynamics algorithm was thought to be more than suitable for our application. It would simply be necessary to add the non-inertial degrees of freedom to the existing algorithm. This assumption has proven to be inaccurate as the structural dynamics solver exhibits an unfortunate sensitivity to time

step, which will be investigated further in the following sections. Nevertheless, the focus of this project remains the development of an efficient and stable unsteady CFD algorithm that can be coupled to an existing structural dynamics solver.

It has been a fundamental design goal during development that the structural dynamics solver and unsteady CFD solver should be maintained as two separate entities so that either could be enhanced, upgraded, or replaced without impacting the other. Figure 4.2 presents a conceptual flow chart of how the structural dynamics solver and unsteady CFD solver are coupled together to advance the solution through time. This flow chart illustrates what is referred to as a time-marched aeroelastic solution.

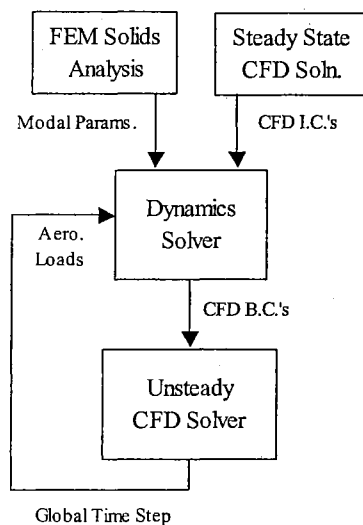


Figure 4.2: Unsteady Solution Flow Chart

Notice that the dynamics solver updates the boundary conditions for the unsteady CFD solver, while the unsteady CFD solver provides a set of aerodynamic loads to the dynamics solver. In such a solution scheme, the unsteady CFD solver is typically the more complicated module which requires the overwhelming majority of CPU cycles. In the following sections, we will develop the core algorithm for a discrete dynamics solver,

investigate the sensitivity of the algorithm to the time scale of the simulation, and propose enhancements that will improve the accuracy of the algorithm.

4.6.1 Core Dynamics Algorithm

The core dynamics algorithm remains essentially unchanged from the methodology used with the original STARS unsteady CFD module. The methodology relies on converting the continuous-time equation of motion to a discrete state-space representation that can be easily integrated to advance the solution one discrete time step. We start by converting the basic aeroelastic equation of motion presented previously in Equation (2.1) to state-space form as follows:

$$(4.9) \quad \begin{Bmatrix} \dot{\mathbf{x}}(t) \\ \ddot{\mathbf{x}}(t) \end{Bmatrix} = \mathbf{A} \begin{Bmatrix} \mathbf{x}(t) \\ \dot{\mathbf{x}}(t) \end{Bmatrix} + \mathbf{B} \mathbf{f}_a(t)$$

where the continuous-time state matrix, \mathbf{A} , and input matrix, \mathbf{B} , are defined as follows:

$$(4.10) \quad \mathbf{A} = \begin{bmatrix} 0 & \mathbf{I} \\ -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{C} \end{bmatrix}$$

$$(4.11) \quad \mathbf{B} = \begin{bmatrix} 0 \\ \mathbf{M}^{-1} \end{bmatrix}$$

Next, Equation (4.9) is converted to the zero-order hold equivalent system, which has a discrete-time state equation with the following form:

$$(4.12) \quad \begin{Bmatrix} \mathbf{x}(k+1) \\ \dot{\mathbf{x}}(k+1) \end{Bmatrix} = \mathbf{G} \begin{Bmatrix} \mathbf{x}(k) \\ \dot{\mathbf{x}}(k) \end{Bmatrix} + \mathbf{H} \mathbf{f}_a(k)$$

where the discrete-time state matrix, \mathbf{G} , and input matrix, \mathbf{H} , are defined as follows:

$$(4.13) \quad \mathbf{G} = e^{\mathbf{A}\Delta t}$$

$$(4.14) \quad \mathbf{H} = \left(\int_0^{\Delta t} e^{\mathbf{A}\lambda} d\lambda \right) \mathbf{B}$$

If the continuous-time state matrix is not singular, the discrete-time input matrix can be computed using the inverse of the state matrix as follows:

$$(4.15) \quad \mathbf{H} = (\mathbf{G} - \mathbf{I})\mathbf{A}^{-1}\mathbf{B}$$

If the continuous-time state matrix is singular, the discrete-time input matrix must be approximated numerically by first expanding the discrete-time state matrix into an infinite series and simplifying equation (4.15) as follows:

$$(4.16) \quad \mathbf{G} = \sum_{i=0}^n \mathbf{A}^i \frac{\Delta t^i}{i!} = \mathbf{I} + \mathbf{A}\Delta t + \mathbf{A}^2 \frac{\Delta t^2}{2} + \mathbf{A}^3 \frac{\Delta t^3}{6} + \dots + \mathbf{A}^n \frac{\Delta t^n}{n!}$$

$$(4.17) \quad \mathbf{H} = \left(\mathbf{I} + \mathbf{A}\Delta t + \mathbf{A}^2 \frac{\Delta t^2}{2} + \mathbf{A}^3 \frac{\Delta t^3}{6} + \dots + \mathbf{A}^n \frac{\Delta t^n}{n!} - \mathbf{I} \right) \mathbf{A}^{-1} \mathbf{B}$$

$$(4.18) \quad \mathbf{H} = \left(\Delta t + \mathbf{A} \frac{\Delta t^2}{2} + \mathbf{A}^2 \frac{\Delta t^3}{6} + \dots + \mathbf{A}^{n-1} \frac{\Delta t^n}{n!} \right) \mathbf{B} = \sum_{i=0}^n \mathbf{A}^i \frac{\Delta t^{i+1}}{(i+1)!}$$

Equation (4.18) represent a modification to the original STARS matrix assembler that will allow us to analyze systems with rigid-body modes. Such modes have mass but zero stiffness, which will result in a singular continuous-time state matrix.

In order to effectively couple the structural dynamics solver with the unsteady CFD solver, the two modules must utilize the same dimensionless system of units. The structural dynamics solver must be able to accept dimensionless aerodynamic forces from the CFD solver and return dimensionless boundary conditions to the CFD solver. Rather than maintaining two systems of measurement and converting back and forth, the matrices derived for the structural dynamics algorithm will be converted to dimensionless units that are consistent with the CFD solver.

First, we define the dimensionless form of the generalized mass, damping and stiffness matrices. In terms of units, a generalized degree of freedom resembles a rotational degree of freedom since the finite element mass matrix is multiplied by the elastic deformation vectors twice in its derivation.⁴ Hence, the generalized mass matrix will have units of inertia and its dimensionless form is defined as follows:

$$(4.19) \quad \mathbf{M} = \left(\frac{1}{2} \rho_{\text{inf}} L^5\right) \mathbf{M}^*$$

Similarly, dimensionless forms of the generalized damping and stiffness matrices are defined as follows:

$$(4.20) \quad \mathbf{C} = \left(\frac{1}{2} \rho_{\text{inf}} u_{\text{inf}} L^4\right) \mathbf{C}^*$$

$$(4.21) \quad \mathbf{K} = \left(\frac{1}{2} \rho_{\text{inf}} u_{\text{inf}}^2 L^3\right) \mathbf{K}^*$$

In the above expressions, ρ_{inf} is the free-stream density, u_{inf} is the free stream velocity, L is the reference length or dimension, and an asterisk indicates the dimensionless form of each matrix.

Substituting the above definitions along with the definition of a dimensionless aerodynamic force, Equation (4.8), into the aeroelastic equation of motion, Equation (2.1), yields the following:

$$(4.22) \quad \mathbf{M}^* \ddot{\mathbf{x}}^*(t^*) + \mathbf{C}^* \dot{\mathbf{x}}^*(t^*) + \mathbf{K}^* \mathbf{x}^*(t^*) = \mathbf{f}_o^*(t^*)$$

where the dimensionless form of the generalized displacement, velocity, and acceleration are defined as follows:

$$(4.23) \quad \mathbf{x}^*(t^*) = \mathbf{x}(t)$$

$$(4.24) \quad \dot{\mathbf{x}}^*(t^*) = \frac{L}{u_{\text{inf}}} \dot{\mathbf{x}}(t)$$

$$(4.25) \quad \ddot{\mathbf{x}}^*(t^*) = \frac{L^2}{u_{\text{inf}}^2} \ddot{\mathbf{x}}(t)$$

Notice that the dimensionless form of the generalized displacement is identical to its dimensional form. This is because the elastic deformation vector itself is dimensionless and the generalized displacement is simply a dimensionless scaling factor for the elastic deformation vector.

All of the above dimensionless forms are used exclusively when assembling the state space matrices and for advancing the structural dynamics through time. The dimensionless elastic state vector is stored in an array with $\text{nr} * 2$ elements. Two copies of this array are needed for advancing the structural dynamics. The state vectors at time n and $n + 1$ are stored in the arrays `XN` and `XN1` respectively.

Following a similar process, it is possible to define dimensionless forms for the non-inertial coefficient matrices. The matrix solution developed previously for the aeroelastic equation of motion is not directly applicable to the non-inertial equation of motion due to the non-linear coupling between the three rotational degrees of freedom. However, a matrix of structural coefficients is a convenient way of defining the non-inertial structural system. As such, we begin by defining a three-dimensional rigid-body state vector as follows:

$$(4.26) \quad \mathbf{x} \equiv \{x_r \quad y_r \quad z_r \quad \phi_r \quad \theta_r \quad \psi_r\}^T$$

In the above expression, x_r is the x -displacement for the non-inertial frame, y_r is the y -displacement for the non-inertial frame, z_r is the z -displacement for the non-inertial frame, ϕ_r is the roll angle for the non-inertial frame in radians, θ_r is the pitch angle for the

non-inertial frame in radians, and ψ_r is the yaw angle for the non-inertial frame in radians.

Dimensionless forms for the rigid-body state vector and structural coefficient matrices are defined as follows:

$$(4.27) \quad \mathbf{x}(t) = L \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{L} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{L} \end{bmatrix} \mathbf{x}^*(t^*)$$

$$(4.28) \quad \dot{\mathbf{x}}(t) = u_{\text{inf}} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{L} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{L} \end{bmatrix} \dot{\mathbf{x}}^*(t^*)$$

$$(4.29) \quad \ddot{\mathbf{x}}(t) = \frac{u_{\text{inf}}^2}{L} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{L} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{L} \end{bmatrix} \ddot{\mathbf{x}}^*(t^*)$$

$$(4.30) \quad \mathbf{M} = \frac{1}{2} \rho_{\text{inf}} L^3 \begin{bmatrix} 1 & 1 & 1 & L & L & L \\ 1 & 1 & 1 & L & L & L \\ 1 & 1 & 1 & L & L & L \\ L & L & L & L^2 & L^2 & L^2 \\ L & L & L & L^2 & L^2 & L^2 \\ L & L & L & L^2 & L^2 & L^2 \end{bmatrix} \mathbf{M}^*$$

$$(4.31) \quad \mathbf{C} = \frac{1}{2} \rho_{\text{inf}} u_{\text{inf}} L^2 \begin{bmatrix} 1 & 1 & 1 & L & L & L \\ 1 & 1 & 1 & L & L & L \\ 1 & 1 & 1 & L & L & L \\ L & L & L & L^2 & L^2 & L^2 \\ L & L & L & L^2 & L^2 & L^2 \\ L & L & L & L^2 & L^2 & L^2 \end{bmatrix} \mathbf{C}^*$$

$$(4.32) \quad \mathbf{K} = \frac{1}{2} \rho_{\text{inf}} u_{\text{inf}}^2 L \begin{bmatrix} 1 & 1 & 1 & L & L & L \\ 1 & 1 & 1 & L & L & L \\ 1 & 1 & 1 & L & L & L \\ L & L & L & L^2 & L^2 & L^2 \\ L & L & L & L^2 & L^2 & L^2 \\ L & L & L & L^2 & L^2 & L^2 \end{bmatrix} \mathbf{K}^*$$

The dimensionless rigid-body state vector is stored in an array with twelve elements. Two copies of this array are needed for advancing the structural dynamics. The state vectors at time n and $n+1$ are stored in the arrays XD and XD1 respectively. As a matter of convenience, the initial conditions for the three rotational degrees of freedom are input by the user using degrees rather than radians, and the solver performs the conversion to radians when deriving the dimensionless rigid-body state vector.

The implementation of a fully coupled dynamics solution where both the non-inertial and elastic degrees of freedom are solved simultaneously is left for a topic of future research. For the verification and validation of the non-inertial algorithm, we will restrict ourselves to uncoupled non-inertial problems without elasticity. In this case, the structural equation of motion for each non-inertial degree of freedom can be solved separately. The non-inertial dynamics is solved with respect to the inertial frame and the inertial vectors are transformed to the non-inertial coordinate system before they are passed back to the CFD module. By convention, the solver outputs the rigid-body state vector in the inertial frame following each step of the solution.

4.6.2 Sampling Sensitivity

The algorithm presented in Section 4.6.1 was derived using the classic methodology for converting a continuous-time system to its discrete-time equivalent, which can be found in any textbook on discrete-time control systems. This methodology assumes that the input force vector is held constant between any two consecutive sampling instants. Provided that this assumption is true, Equation (4.12) is an exact representation for the original continuous-time equation of motion. Such a discrete-time model is typically sufficient for representing digital control systems where the input is provided by the user for the purpose of controlling or stabilizing the system. In which case, the control input provided to the system almost certainly is held constant across consecutive sampling intervals and Equation (4.12) exactly matches the physical reality of the real continuous-time system.

Unfortunately, the aeroelastic system that we are trying to simulate does not perfectly fit the assumptions of this derivation. The input that we have defined for the discrete-time structural equation certainly is not constant across consecutive sampling intervals. As shown in Figure 4.3, the model we are using to represent an aeroelastic system is made up of a structural dynamics block and an aerodynamics block connected in a feedback loop.

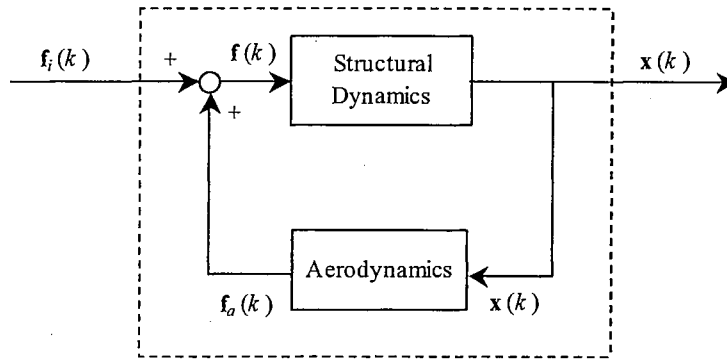


Figure 4.3: Block diagram representing a discrete-time aeroelastic system.

When representing an aeroelastic system in this fashion, it is natural to think of it as simply a control system with a feedback loop. However, the reality of this problem is that it consists of two complicated continuous-time systems where the output of each system, which is the corresponding input to the complementary system, is not constant across the sampling interval. Regardless, it is tempting to try and apply this solution scheme to an aeroelastic problem. Certainly the error incurred by assuming a constant input to the system may be offset by choosing a sufficiently small sampling interval. The only question then is how small must the sampling interval be?

In order to evaluate the accuracy of the discrete-time solution scheme, let us consider the one-dimensional structural system given by equation (4.33).

$$(4.33) \quad \ddot{x} + 0.05\dot{x} + 2x = f(t)$$

The state-space form of equation (4.33) is as follows:

$$(4.34) \quad \begin{Bmatrix} \dot{x} \\ \ddot{x} \end{Bmatrix} = \begin{bmatrix} 0 & 1 \\ -2 & -0.05 \end{bmatrix} \begin{Bmatrix} x \\ \dot{x} \end{Bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} f(t)$$

The natural frequency of this continuous-time system is 0.225 Hz with a damping ratio of 0.0177.

The simplified aerodynamic force model that will be used to complete the feedback loop for this one-dimensional model problem is given by equation (4.35).

$$(4.35) \quad f(t) = -x$$

The resulting coupled system has a modified natural frequency of 0.277 Hz and a damping ratio of 0.144. These are the exact values for the natural frequency and damping of the coupled system computed from the eigenvalues of the continuous-time state matrix.

Following the methodology of Section 4.6.1, the zero-order hold equivalent of the one-dimensional model problem is given by equation (4.36).

$$(4.36) \quad \begin{Bmatrix} x(k+1) \\ \dot{x}(k+1) \end{Bmatrix} = e^{\begin{bmatrix} 0 & 1 \\ -2 & -0.05 \end{bmatrix} \Delta t} \begin{Bmatrix} x(k) \\ \dot{x}(k) \end{Bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} f(k)$$

where,

$$(4.37) \quad f(k) = -x(k)$$

Notice that the state matrix for the discrete-time system has a dependency on the sampling time, Δt . As with the continuous-time system, the natural frequency and damping of the system can be computed from the eigenvalues of the state matrix. Figure 4.4 presents plots of damping ratio, ζ , and percent error, ε , versus sampling frequency, T , in samples per period of the structural system.

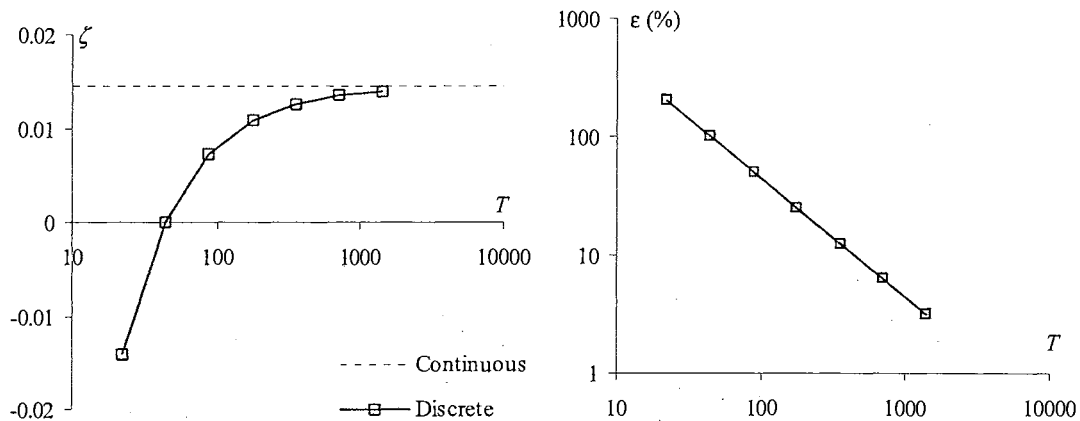


Figure 4.4: Comparison between continuous and discrete models for a one-dimensional aeroelastic model problem.

The data for Figure 4.4 was taken for sampling frequencies ranging from 22 to 1421 samples per period, and each successive data point was taken using fifty percent of the previous data point's sampling interval. If we extrapolate the error curve for this problem, a sampling frequency of nearly 4500 samples per period would be required to reach less than one percent error. It is important to note that the properties of the discrete-time system without feedback exactly matched the properties of the continuous-time system without feedback for each of sampling frequencies used to construct Figure 4.4. Hence, the sampling of the input force is solely responsible for the error that is evident in these plots.

Clearly the discrete-time solution to this one-dimensional model problem has a strong sensitivity to the size of the sampling interval. Surprisingly, it has not been uncommon to use sampling intervals of 100 samples per period or less when solving aeroelastic problems with the original STARS unsteady CFD algorithm. In light of the results presented for this model problem, the accuracy of these solutions is questionable.

It is expected that this model problem presents a best-case scenario since the aerodynamic model is exactly represented and does not itself possess a sensitivity to the sampling interval. If this is the case, then a problem that used a sampling interval of 100 samples/period would be off by more than 45%.

4.6.3 Higher Order Algorithms

A significant amount of effort was spent developing a CFD algorithm that would be both stable and accurate for relatively large global time steps. In Section 3.5.3, a second-order backwards difference operator was used to increase the time accuracy of the unsteady CFD algorithm. The result is a second-order discretization of our continuous-time aerodynamic model, the compressible Euler equations. A similar process can be used to improve the accuracy of the discrete-time structural model.

The zero-order hold used in the derivation of the discrete-time structural model is equivalent to using a zero-order integrator or a simple left-hand sum to integrate the input force for each sampling interval. An obvious enhancement to this scheme would be to substitute a higher order numerical integrator. Equations (4.38) and (4.39) define discrete-time models that use first-order and second-order integrators respectively.

$$(4.38) \quad \begin{Bmatrix} \mathbf{x}(k+1) \\ \dot{\mathbf{x}}(k+1) \end{Bmatrix} = \mathbf{G} \begin{Bmatrix} \mathbf{x}(k) \\ \dot{\mathbf{x}}(k) \end{Bmatrix} + \mathbf{H} \left[\frac{3}{2} \mathbf{f}_a(k) - \frac{1}{2} \mathbf{f}_a(k-1) \right]$$

$$(4.39) \quad \begin{Bmatrix} \mathbf{x}(k+1) \\ \dot{\mathbf{x}}(k+1) \end{Bmatrix} = \mathbf{G} \begin{Bmatrix} \mathbf{x}(k) \\ \dot{\mathbf{x}}(k) \end{Bmatrix} + \mathbf{H} \left[2\mathbf{f}_a(k) - \frac{3}{2} \mathbf{f}_a(k-1) + \frac{1}{2} \mathbf{f}_a(k-2) \right]$$

Notice that the only real cost of implementing a higher-order integrator is the small amount of additional memory required to store the force vectors from previous time intervals. Furthermore, these models were derived such that the core discrete-time

advancement scheme will remain essentially unchanged. All that is required is to swap in the appropriate force integrator during the structural advancement to the next time interval. Figure 4.5 presents a comparison of the error for the three different force integrators when applied to the one-dimensional model problem from Section 4.6.2.

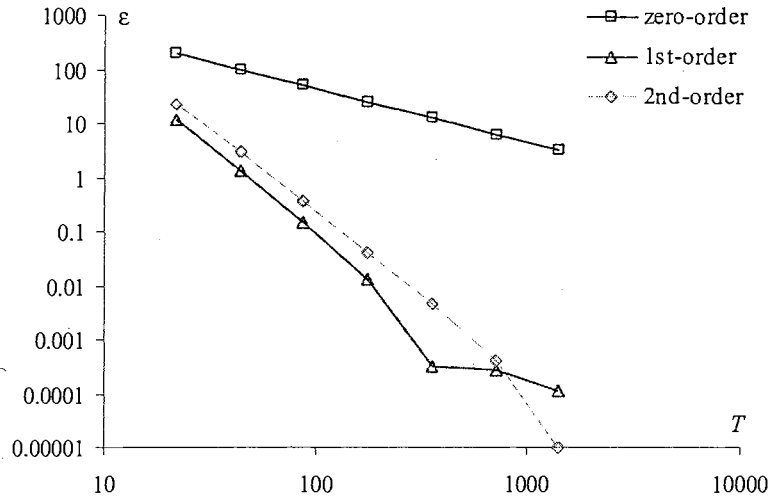


Figure 4.5: Error comparison between several discrete-time models for the one-dimensional aeroelastic model problem.

The results in Figure 4.5 show a significant improvement in the rate of convergence for the two higher-order integrators. Notice that the error for both the first and second-order methods is several orders of magnitude smaller than the zero-order method at the highest sampling frequency. Furthermore, we see that the first and second-order solution produce almost identical results. This result is somewhat expected for our model problem with its over-simplified linear force model.

For a better evaluation of our three integrators, let us consider the two-dimensional structural system defined by equation (4.40).

$$(4.40) \quad \begin{Bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \ddot{x}_1 \\ \ddot{x}_2 \end{Bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -0.0064 & 0 & -0.0032 & 0 \\ 0 & -0.1011 & 0 & -0.0127 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \dot{x}_1 \\ \dot{x}_2 \end{Bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 2.6093 & 0 \\ 0 & 8.7036 \end{bmatrix} \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix}$$

The structural coefficients for this system were taken from an actual aeroelastic model of a wing with two dominant structural modes. Equation (4.41) defines the quasi-steady aerodynamic model that will be used to complete this two-dimensional model problem.

$$(4.41) \quad f(t) = \begin{bmatrix} -0.0020 & -0.0078 & -0.0026 & -0.0015 \\ 0.0011 & 0.0032 & 0.0015 & -0.0009 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \dot{x}_1 \\ \dot{x}_2 \end{Bmatrix}$$

The aerodynamic coefficients for this force model were computed by applying a unit displacement and velocity to each degree of freedom and computing the resulting forces using the unsteady CFD solver. This leaves us with a system that can still be solved analytically, but is hopefully a closer approximation of a real aeroelastic system than the previous one-dimensional model problem.

Figure 4.6 presents a comparison of the modeling error at various sampling frequencies for the three force integrators when applied to the two-dimensional model problem.

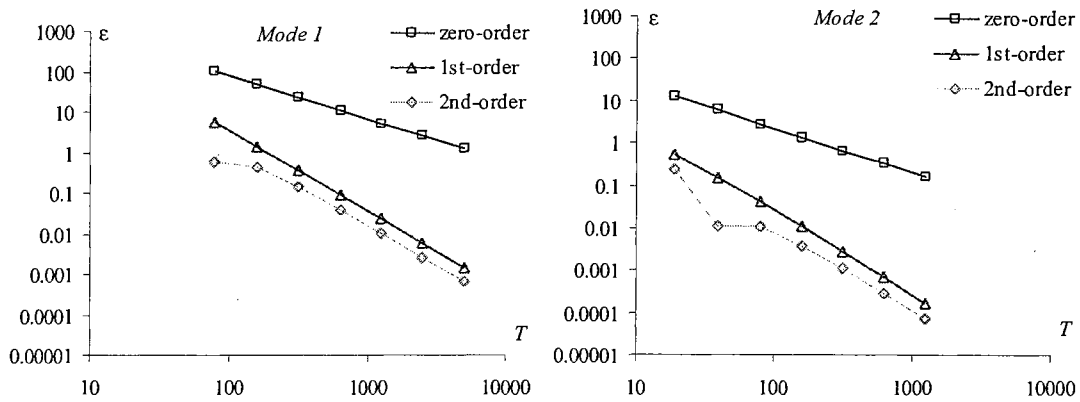


Figure 4.6: Error comparison between several discrete-time models for the two-dimensional aeroelastic model problem.

Once again, we see a similar trend in the convergence of the three models. The higher-order models clearly provide better accuracy over the zero-order model. However, we should note that the difference in rate of convergence between the three models is not as significant. For the results in Figure 4.5, the slope of the error curve for the higher-order models is approximately 200 percent greater than the slope for the zero-order model. This compares to only a 75 percent difference for the two-dimensional results of Figure 4.6.

The previous two model problems both relied on a linear force model where the force could be directly computed based on the generalized displacement and velocity of the structure at the current time step. This allowed us to compute the predicted response of the system directly and compare it to an exact analytical solution. What we are really interested in is the sensitivity of the system when we couple the structural dynamics integrators with the unsteady CFD solution. Figure 4.7 presents a comparison between the error of the zero-order and second-order integrator over a similar range of sampling

frequencies when the unsteady CFD solver is instead used to compute the aerodynamic forces acting on the structure. In this case an exact analytical solution for the damping ratio of each mode is not available, so the error is actually a percent difference between the computed value at each sampling frequency and an extrapolated value based on the trend of the two solutions. Although the improvement of the second-order solution is not as significant as the results shown in Figure 4.6, the trend here continues to be convergence of the solution as the sampling frequency is increased.

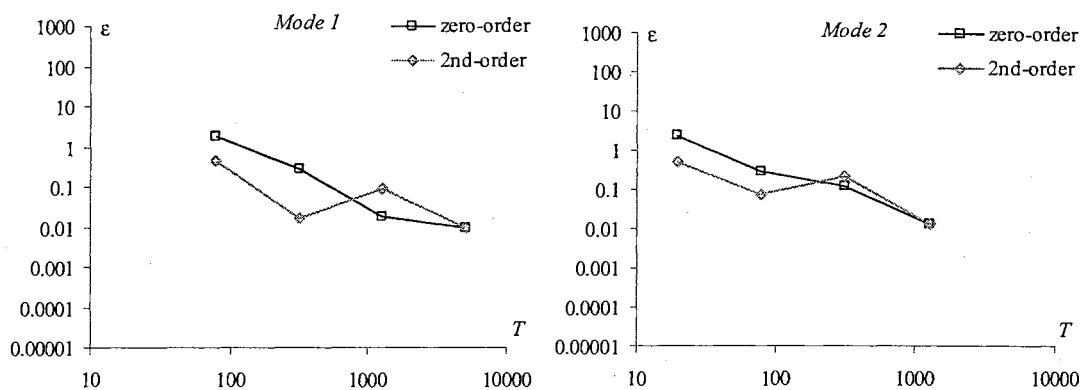


Figure 4.7: Error comparison between several discrete-time models for the two-dimensional aeroelastic CFD problem.

It would seem that the accumulation of errors between the CFD solver and the structural integrator has narrowed the gap between the two solutions. However, the benefit of having multiple structural integrators becomes more obvious if we plot the absolute difference between the damping ratio predicted by the zero-order and second-order integrators for each sampling frequency. Figure 4.8 presents plots of absolute difference, δ , versus sampling frequency, T , for each mode of the structural system.

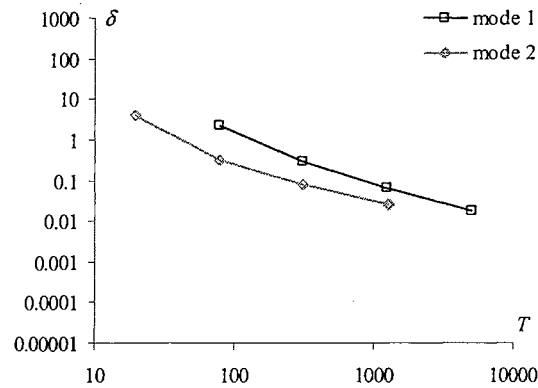


Figure 4.8: Plot of absolute difference between the zero-order and second-order integrators for each mode of the aeroelastic CFD problem.

By comparing the plot in Figure 4.8 with the error comparisons of Figure 4.7, we observe that the absolute difference between the two solutions is an indicator of the error present at that sampling frequency. This observation provides us with a way of directly evaluating the sensitivity of an aeroelastic system to the solution time step. A time step convergence study can actually be performed by running two solutions at the same time step with two different structural integrators and differencing the results rather than running three or more solutions at different time steps with the same structural integrator and searching for asymptotic convergence.

This is an important result because we expect that each aeroelastic system will exhibit a different degree of sensitivity to the sampling frequency based on the unique combination of structural and aerodynamic parameters that make up the system. At this point there is not enough data to make a generalization about the minimum sampling frequency required to attain a given level of convergence. However, it does appear that the best way of judging the convergence of the system is by comparing two solutions for a given time step.

4.7 Summary

We began this research with the goal of extending the capabilities of the STARS non-linear aeroelastic module by adding the capability for non-inertial CFD analysis. For a variety of reasons, a completely new CFD module was developed with non-inertial capabilities rather than simply modifying the existing CFD modules. Therefore, it is prudent at this point to compare the algorithm presented in this document with the previous CFD modules used by STARS. The fundamental differences between the two algorithms are summarized as follows:

1. The new CFD algorithm uses double precision for all floating-point calculations rather than single precision.
2. Element data is used to evaluate the Euler flux integrals rather than edge data.
3. A consistent mass formulation is used to evaluate the unsteady time flux rather than a lumped mass formulation.
4. The time advancement scheme has been re-worked, and no longer includes a multi-stage time stepping algorithm.
5. A non-inertial source term has been added to the unsteady solution residual to account for an arbitrarily rotating and accelerating frame of reference.
6. The transpiration boundary condition has been modified to correct for elastic deformations relative to a rotating frame of reference.
7. Local time steps are compared to the global time step for unsteady solutions to maintain stability for small global time steps.
8. Boundary conditions have been sorted and re-worked in order to achieve better computational efficiency.

9. Multigrid capability has not been included in the new solver since it is impractical for unsteady aeroelastic applications.
10. The structural matrix assembler has been modified to account for singular state-matrices in systems with rigid-body modes.
11. The structural dynamics algorithm was expanded to include the choice of three different integrators to improve the accuracy of unsteady aeroelastic solutions.

A comparison of the memory requirements and computational performance of the two codes is also provided the next two sections.

4.7.1 Memory Requirements

The CFD algorithm developed for this research actually requires more memory than the previous STARS CFD algorithm for two reasons. First, the new CFD algorithm is written in double precision, which requires exactly twice as much memory for each floating-point variable. In the new code, all floating-point variables are 8 bytes in size rather than the default 4 byte floating-point variables used by the original STARS CFD modules. Furthermore, the new algorithm requires both the element and edge data structures to accommodate the algorithmic enhancements indicated in the previous section, while the old CFD module utilized an edge-based data structure exclusively.

Combine these two factors together, and the new algorithm would require about four or five times the memory of the old algorithm. Fortunately, there were some redundant or superficial sets of data maintained in the old algorithm. These were removed to streamline our new memory requirements, but the resulting code still requires about a factor of three and a half times more memory (depending on the problem).

Equations (4.42) and (4.43) give the approximate memory requirements in bytes for the two-dimensional and three-dimensional solvers respectively.

$$(4.42) \quad mem_{2D} = 8 \cdot (37 \cdot nnd + 8 \cdot nel + 4 \cdot nsg + 2 \cdot (2 + nr) \cdot nwl + 5 \cdot nbe)$$

$$(4.43) \quad mem_{3D} = 8 \cdot (51 \cdot nnd + 14 \cdot nel + 5 \cdot nsg + 3 \cdot (2 + nr) \cdot nwl + 7 \cdot nbe)$$

In the above expressions, nnd is the total number of nodes, nel is the total number of elements, nsg is the number of segments, nr is the number of elastic modes, nwl is the number of wall nodes, and nbe is the number of boundary elements.

Note that the above equations actually assume that integers are 8 bytes as well, which may or may not be the default on different systems. Regardless, the proportion of integer arrays is small compared to the required floating-point arrays, so these equations would still be close. Furthermore, the memory requirements calculated above are only for the pointer workspace used to define the main data arrays. Obviously there are additional intermediate variables used to calculate various quantities, but these should also be a small proportion compared to the main data arrays. As an example, a problem that has 461,575 elements, 84,448 nodes, 17,838 boundary elements, and 8,923 boundary nodes requires about 105 MB of memory with the new algorithm, while it would have required about 29 MB, or around a third of the memory, with the old STARS CFD module. Fortunately, memory is less expensive than it was when the original STARS CFD module was written, and this increase in memory requirements is not seen as a major downfall of the new methodology.

4.7.2 Computational Performance

In this section we investigate the computational performance of the new solver and will compare and contrast its performance to the old STARS CFD solver. In particular, we are interested in the raw number crunching abilities of the two codes, and the overall rate of convergence of the two codes. For our comparison, we will run steady solutions for a three-dimensional geometry that is representative of the type of aerospace application we are interested in analyzing. Since unsteady solutions involve the solution of a modified steady problem at each step, we expect that the unsteady performance of the two codes is similar to that observed for steady solutions.

First we examine the relative CPU time required for a solution. Figure 4.9 presents a comparison between the relative computational time for the old solver and the new solver for various operating system/processor combinations. The relative CPU time is computed by dividing the total run time of the new solver by the total run time of the old solver for the same operating system. This simple normalization procedure provides an easy means of evaluating the performance gap between the two solvers.

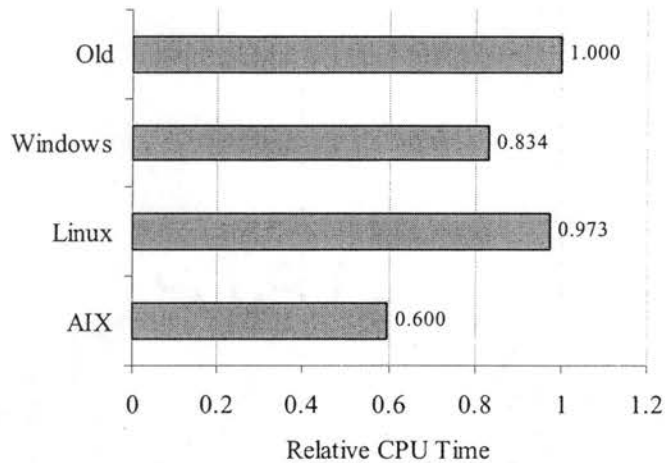


Figure 4.9: Comparison of computational performance of old solver and new solver for various operating system/processor combinations.

Notice that the new solver benefits from an improved computational performance on all three operating systems. However, the size of the computational gap is different for all three operating systems. This is most likely due to differences in how the compiler optimizes on each system and fundamental architecture differences, such as the system bus and floating point unit, between the three systems tested here.

The fact that the new solver is actually faster than the old solver is an interesting result considering the old solver utilized the faster edge-based data structure exclusively. Hence, the algorithmic enhancements that were made to the new code have more than made up for the quoted 30% difference between the element and edge-based data structures, since we have actually surpassed the performance of the old code. For our performance comparison, each solver was run using an equivalent set of control parameters so that the raw iterative speed is compared directly. In the case of the old steady solver this means that three stages for each steady cycle were used, while the new

solver ran three iterative cycles for each solution step. Both solvers used the same courant stability factor, $cf1 = 0.7$, and the high-order dissipation model.

Our next comparison considers the rate of convergence for each code. Figure 4.10 shows the residual convergence histories for the same problem run at two different Mach numbers with both the new and old solvers. Notice in both cases that the new solver convergences “farther” since it is coded using double precision variables.

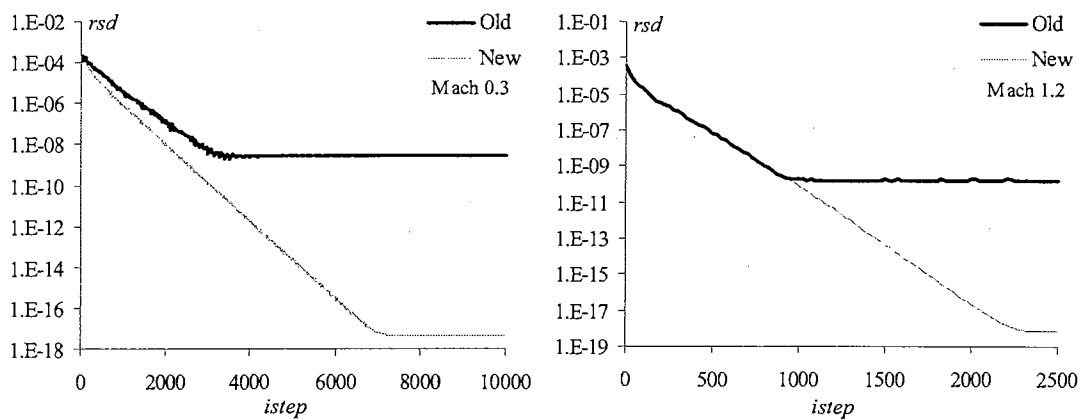


Figure 4.10: Comparison of residual convergence histories of old and new CFD solvers for two different free stream mach numbers.

Of particular interest is the slope of the residual convergence histories for each code. Notice that the new code actually converges faster than the old code for the subsonic Mach number, i.e. it converge to the same point in fewer iterations. For the supersonic Mach number, both codes converge at the same rate. Hence, the new code converges at least as fast as the old code when run with the same set of control parameters. This is not an entirely fair comparison though. The old STARS CFD module was written with a multi-stage time stepping algorithm and residual smoothing capability specifically for the purpose of allowing higher values of $cf1$ to be used. If the

old code is actually capable of running stably at higher values of `cf1`, then its rate of convergence would beat the new solver. However, the multi-stage time stepping algorithm and residual smoothing require extra CPU time for each step.

It is unclear exactly how to compare the total performance of the two codes when each is run with different sets of parameters. Especially because there is no way to factor in the amount of additional user time that is required to tweak the additional parameters available with the old solver. In fact, part of our design philosophy in developing the new solver has been to eliminate potentially confusing free parameters and go for a more direct solution of the fundamental equations. In this respect, the new solver should be easier to use, and thus save time with debugging and parameter tweaking for each individual problem.

Having compared the performance of the old and new steady solvers, we now examine the performance of the new unsteady solver for different combinations of control parameters. Specifically, we are interested in determining how much performance penalty is associated with non-inertial or higher-order accurate unsteady solutions when compared against the basic first-order accurate, inertial solution using one-point gauss quadrature for numerical integration. Table 4.1 summarizes the solver control settings for each of eight different tests used to investigate the performance of the unsteady solver.

Table 4.1: Summary of solver parameters for unsteady solution performance tests.

<i>Label</i>	<i>isol</i>	<i>idiss</i>	<i>npnt</i>	<i>idynm</i>
U11	1	1	1	.false.
U21	2	1	1	.false.
U14	1	1	4	.false.
U24	2	1	4	.false.

D11	1	1	1	.true.
D21	2	1	1	.true.
D14	1	1	4	.true.
D24	2	1	4	.true.

Using the above set of control parameters, eight unsteady solutions were completed for the geometry tested previously in the steady performance comparisons. Figure 4.11 shows the relative performance for each test, where the relative performance is computed by dividing the total CPU time for each solution by the total CPU time for the U11 solution.

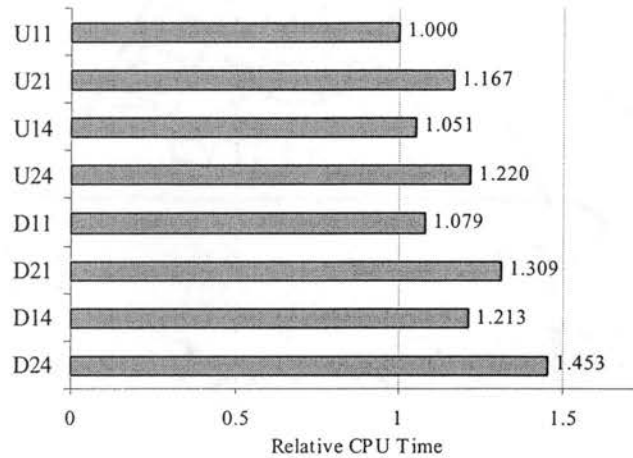


Figure 4.11: Comparison of computational performance for different types of unsteady solutions with new CFD solver.

From Figure 4.11, we can readily see the relative “cost” of running a non-inertial or higher-order accurate solution. Notice that the basic first-order accurate, non-inertial solution only requires about 8% more CPU time than the similar inertial solution for this problem. Considering the added functionality the non-inertial capability has added to the basic solver, this is not a significant performance penalty.

As a final performance comparison, Figure 4.12 presents a plot of CPU time versus number of elements for different problems analyzed using the new steady solver. There is some scatter in this data due to differences in type and number of boundary elements for each unique problem, but this plot shows the general trend of increasing CPU time as the number of elements in the computational domain increases. Figure 4.12 is intended to provide a gauge of the performance cost associated with grid refinement.

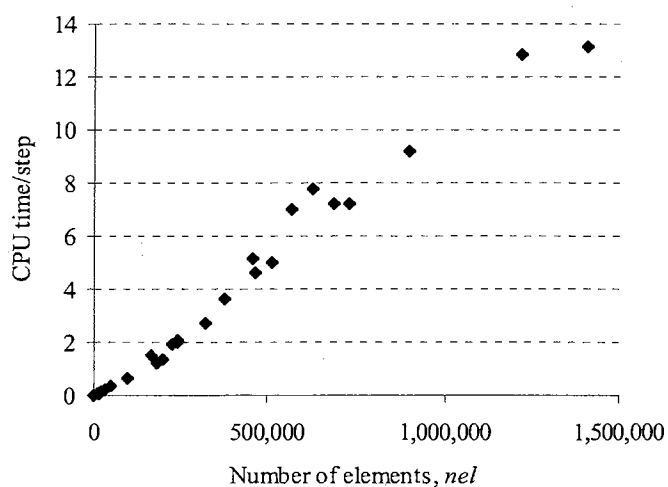


Figure 4.12: Comparison of CPU time for different geometries.

CHAPTER 5

5. RESULTS

In this Chapter, we present numerical examples intended to demonstrate the accuracy and performance of the finite element methodology presented thus far. Since our primary interest lies in solving three-dimensional aerospace applications, results are only presented for the new three-dimensional solver. The accuracy and performance of the two-dimensional solver has also been fully evaluated, but its primary purpose is for quick evaluation of simplified problems in preparation for a more complicated three-dimensional analysis. Hence, two-dimensional results are not presented here so that we may focus on more three-dimensional results.

The results in this section begin with a set of verification calculations for three-dimensional, steady and unsteady problems. This section is intended to demonstrate that the governing equations are being solved correctly and to investigate the order of convergence for the algorithm where possible. Following our verification of the algorithm, we examine validation problems. The validation problems are intended to demonstrate that the relevant flow physics are modeled correctly for the applications we are interested in solving. In particular, we investigate problems that demonstrate the application of the non-inertial reference frame to flight dynamics and spinning structures.

5.1 Verification

In the field of computational fluid dynamics, verification is typically referred to as solving the equations right.¹⁰ We have adopted that definition here and will attempt to demonstrate that our governing equations, the compressible Euler equations, are solved correctly. For the purposes of verification, our algorithm will be used to analyze a set of verification cases that have well known analytic solutions. Where ever possible, we will also compare our solutions to published solutions from other CFD codes that employ different algorithms. While this is a useful comparison to demonstrate accuracy, the primary comparison for verification will be with the analytic solution for each problem.

The verification results in this section are intended to provide a reasonable sampling of the verification problems that have been analyzed. A comprehensive verification procedure requires that we investigate problems that exercise all of the relevant flow physics and every possible combination of parameters for the algorithm. Obviously, this is a rather complicated task for a three-dimensional, unsteady CFD algorithm, and a comprehensive verification procedure will necessarily be a continuing process. However, the results presented here do provide a compelling verification of the accuracy for a variety of problems that are of interest.

The first several sub-sections pursue verification for steady problems. These problems demonstrate that the spatial variation of the governing equations is represented accurately for flow conditions ranging from supersonic down to subsonic. Next we tackle verification for unsteady problems where the temporal variation of the governing equations must be represented accurately. This is obviously a more complicated task, and there are fewer problems with analytical solutions for comparison. Especially when

we extend the verification to include non-inertial or elastic unsteady problems. However, the sampling of problems presented here do form a compelling argument for unsteady verification of the algorithm.

As much as possible, it is necessary in our verification process to include a grid convergence study that demonstrates the consistency of our discretization. A grid convergence study is intended to demonstrate that the solution approaches an exact solution to the governing equations as the measure of discretization for the problem approaches zero.¹⁰ For steady problems this is accomplished by using multiple grids with successively smaller elements. For unsteady problems this refinement will necessarily need to include successively smaller time increments as well.

A suitable grid convergence study will allow us to compute the observed order of convergence for the algorithm. Roache¹⁰ derives the order of convergence from the behavior of the error for the discrete solution, ε , as defined by Equation (5.1).

$$(5.1) \quad \varepsilon = f(\Delta) - f_{exact}$$

In the equation above, $f(\Delta)$ represents the discrete solution and f_{exact} represents the exact or analytic solution. For a well behaved problem, it is expected that the error in the discrete solution, ε , will be asymptotically proportional to Δ^p , where Δ is some measure of discretization and p is the order of convergence for the method. The order of convergence, p , can then be obtained from the slope of $\log(\varepsilon)$ versus $\log(\Delta)$.

Roache¹⁰ also reports that a more direct evaluation for the observed order of convergence may be obtained if the refinement is performed with a constant refinement ratio r . Equation (5.2) then defines the observed order of convergence, p , using three solutions that have been successively refined by a constant ratio r .

$$(5.2) \quad p = \ln \left(\frac{f_3 - f_2}{f_2 - f_1} \right) / \ln r$$

In the above expression, f_1 refers to the solution on the finest grid, f_2 refers to the solution on the intermediate grid, and f_3 refers to the solution on the coarsest grid.

Before beginning our verification it is appropriate to make some general comments about order of convergence. First, it is well known that the observed order of convergence will be less than the theoretical order of convergence. Of particular interest for our application, Roache¹⁰ cites a study on the observed order of convergence for the Euler equations with shocks. The results indicated that the observed order of convergence downstream of shocks is typically around $p = 1$ despite higher orders of convergence upstream of the shock. Along with this observation, we should make the distinction between local and global order of convergence. As the names imply, local order of convergence refers to one location within the grid, while global order of convergence involves the accumulation of errors throughout the entire grid. As such, the global order of convergence is typically one degree less than the local order of convergence.³³

5.1.1 Oblique Shock

This steady problem consists of supersonic flow over a wedge with a half angle of 15 degrees. The resulting flow field develops an oblique shock at the leading edge of the wedge. The results will be compared to the exact solution computed using the perfect gas equations. The first test will involve an upstream Mach number of 2.5, for which our theory predicts a downstream Mach number of $M_2 = 1.87353$, an oblique shock angle of θ

= 36.94 degrees, a pressure ratio across the shock of $p_2/p_1 = 2.46750$, and a density ratio across the shock of $\rho_2/\rho_1 = 1.86655$.

We take advantage of symmetry when defining this problem and represent only the upper half of the wedge. The layout of the computational domain, which covers the volume $x \in [0, 1.5]$, $y \in [0, 1]$ & $z \in [0, 0.1]$, is presented in Figure 5.1. Boundary conditions for the seven boundary surfaces enclosing this domain are specified as follows: surfaces 1, 2 and 3 are symmetry planes, surface 4 is a solid wall, and surfaces 5, 6, and 7 are far-field boundaries. Furthermore, all nodes on the line at the intersection of surfaces 3 and 4 are specified as singular to avoid ambiguity in the enforcement of boundary conditions.

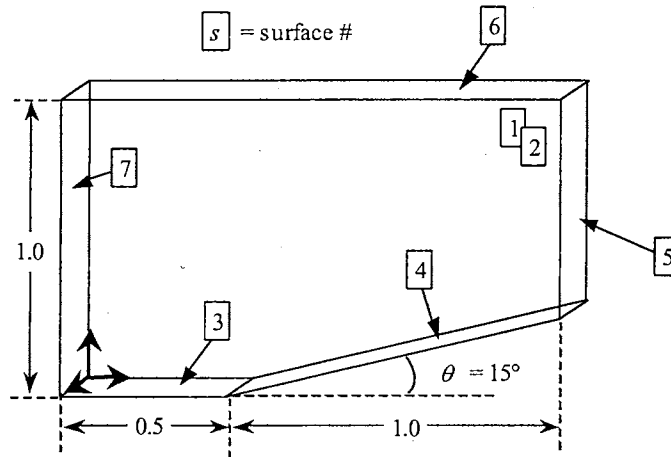


Figure 5.1: Layout of computational domain for oblique shock problem.

To solve this problem, we employ three grids that are refined successively using a constant refinement ratio of $r = 2$. Table 5.1 presents the grid spacing h , number of nodes nd , number of elements nel , and the average computational time required per iteration Δt_{cpu} for each grid. All three grids are made up of tetrahedral elements and are generated

using a uniform grid spacing. Figure 5.2 shows the surface triangulation of the coarse grid generated for this study.

Table 5.1: Summary of grid parameters for oblique shock problem.

	h	nnd	nel	Δt_{cpu}
<i>coarse</i>	0.04	3500	13743	0.0452 s
<i>medium</i>	0.02	21804	98580	0.3992 s
<i>fine</i>	0.01	145385	728564	3.6730 s

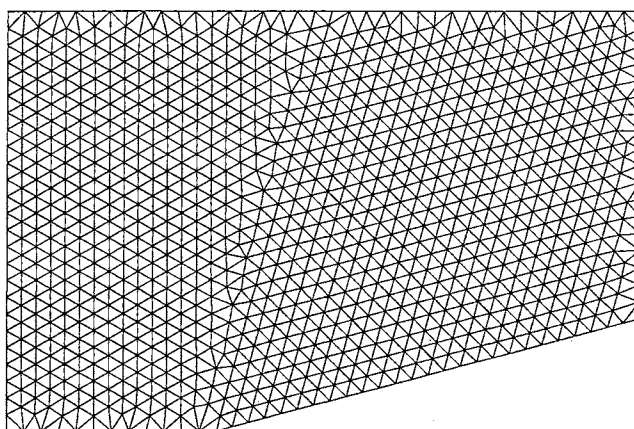


Figure 5.2: Representative surface triangulation for oblique shock.

The computational times presented in Table 5.1 are intended to show the relative increase in computer resources required as the grid resolution increases and are quoted for a steady solution using the low-order dissipation with one-point gauss quadrature. With this information, the total computational time required for an analysis of any grid is computed by multiplying the total number of iterations required for convergence, n_{stp} , by Δt_{cpu} for that grid. Unfortunately, the total computational time increases as the grid resolution increases not only because of an increase in Δt_{cpu} , but also because more iterations are required for convergence on the finer grids due to greater restrictions placed on local time increments for small elements.

In our analysis of this problem, four different solutions will be computed for each grid in order to evaluate the performance of the low-order and high-order dissipation models with both the one-point and four-point numerical integration options. A summary of the relevant solver parameters for this problem is provided in Figure 5.3. The solver control parameters are identical for all three grids except for the number of solution steps, *nstp*, since the number of steps required for numerical convergence increases as the grid resolution increases.

<i>Low-1pt</i>	<i>High-1pt</i>
gamma = 1.40d0	gamma = 1.40d0
mach = 2.50d0	mach = 2.50d0
diss = 0.80d0	diss = 1.00d0
cfl = 0.80d0	cfl = 0.80d0
nstp = 500	nstp = 500
nout = 500	nout = 500
ncyc = 3	ncyc = 3
isol = 0	isol = 0
idiss = 0	idiss = 1
ipnt = 1	ipnt = 1
<i>Low-4pt</i>	<i>High-4pt</i>
gamma = 1.40d0	gamma = 1.40d0
mach = 2.50d0	mach = 2.50d0
diss = 0.80d0	diss = 1.00d0
cfl = 0.80d0	cfl = 0.80d0
nstp = 500	nstp = 500
nout = 500	nout = 500
ncyc = 3	ncyc = 3
isol = 0	isol = 0
idiss = 0	idiss = 1
ipnt = 4	ipnt = 4

Figure 5.3: Summary of solver control parameters for oblique shock at Mach 2.5.

Figure 5.4 presents a plot of the residual convergence history for the four solutions on each grid. The coarse, medium, and fine grids were run for 600, 800 and 1000 iterations respectively in order to achieve full numerical convergence for each different solution.

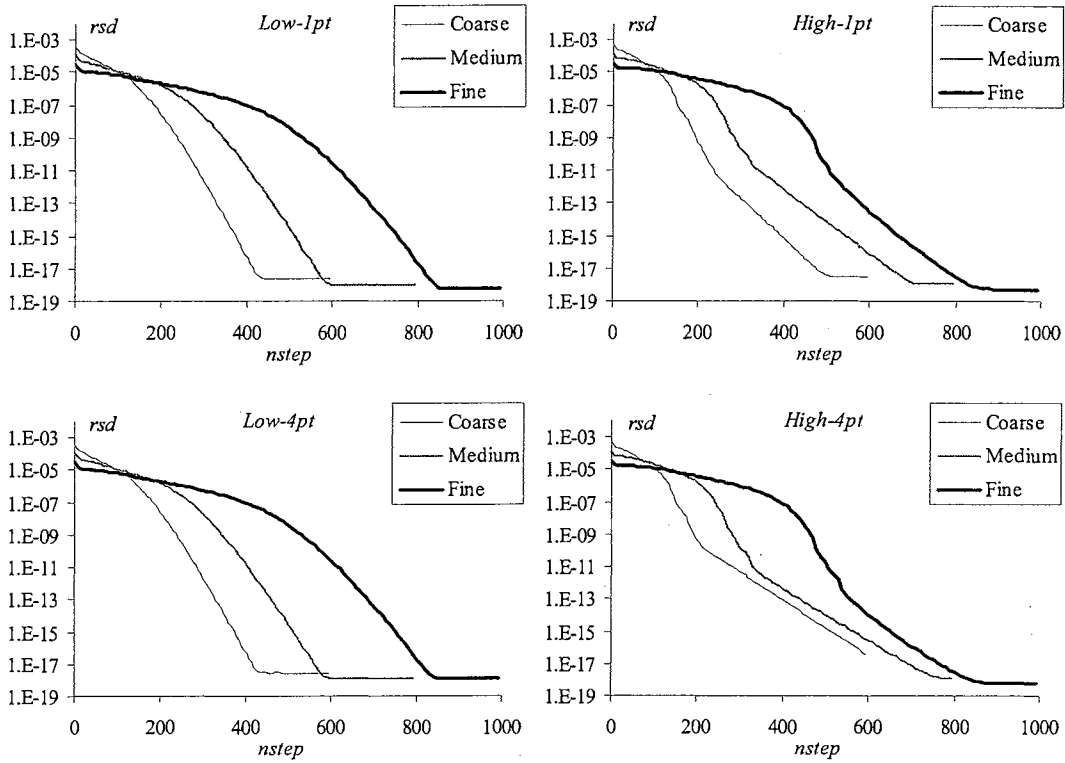


Figure 5.4: Plot of residual histories for oblique shock at Mach 2.5.

Results for these solutions are obtained along a cut-line defining the intersection of an xy -plane down the center of the computational domain, $z = 0.05$, with surfaces 3 and 4. Values of the pressure and local Mach number are obtained from this cut-line along the lower surfaces of the domain. Table 5.2 presents the average Mach error, $\bar{\epsilon}_M$, and average pressure error, $\bar{\epsilon}_p$, downstream of the shock for all twelve solutions. In each case, the average error is computed as the integral of the percent error along the cut-line divided by the length of the cut-line for $x > 0.5$.

Table 5.2: Summary of average error downstream of the shock for all solutions to the oblique shock at Mach 2.5.

<i>Solution</i>	<i>Coarse</i>		<i>Medium</i>		<i>Fine</i>	
	$\bar{\epsilon}_M$ (%)	$\bar{\epsilon}_p$ (%)	$\bar{\epsilon}_M$ (%)	$\bar{\epsilon}_p$ (%)	$\bar{\epsilon}_M$ (%)	$\bar{\epsilon}_p$ (%)
<i>Low-1pt</i>	2.933	3.535	2.490	3.146	1.918	1.367
<i>Low-4pt</i>	2.933	3.607	2.497	3.182	1.923	1.388
<i>High-1pt</i>	3.292	3.748	2.854	2.456	2.054	1.141
<i>High-4pt</i>	3.171	3.722	2.788	2.469	1.978	1.122

Based on the data given in Table 5.2, it appears that all four solutions are converging toward the correct theoretical value as the grid resolution increases. The observed order of convergence for these solutions is found by computing the slope for the plot of $\log(\epsilon)$ versus $\log(\Delta)$ as discussed in Section 5.1. For the one-point, low-order dissipation solution, the observed order of convergence is computed to be either $p = 0.98$ or $p = 1.15$ using the average Mach or pressure error respectively. This rate of convergence is comparable to the value reported by Roache for Euler solutions with shocks. The other three solutions also have a similar rate of convergence.

In addition to the error data, plots of the Mach number and pressure distributions along the cut-line are provided for the one-point low order and high-order dissipation solutions in Figure 5.5 and Figure 5.6 respectively. Both of the four-point solutions are indistinguishable from the similar one-point solutions and are not presented here.

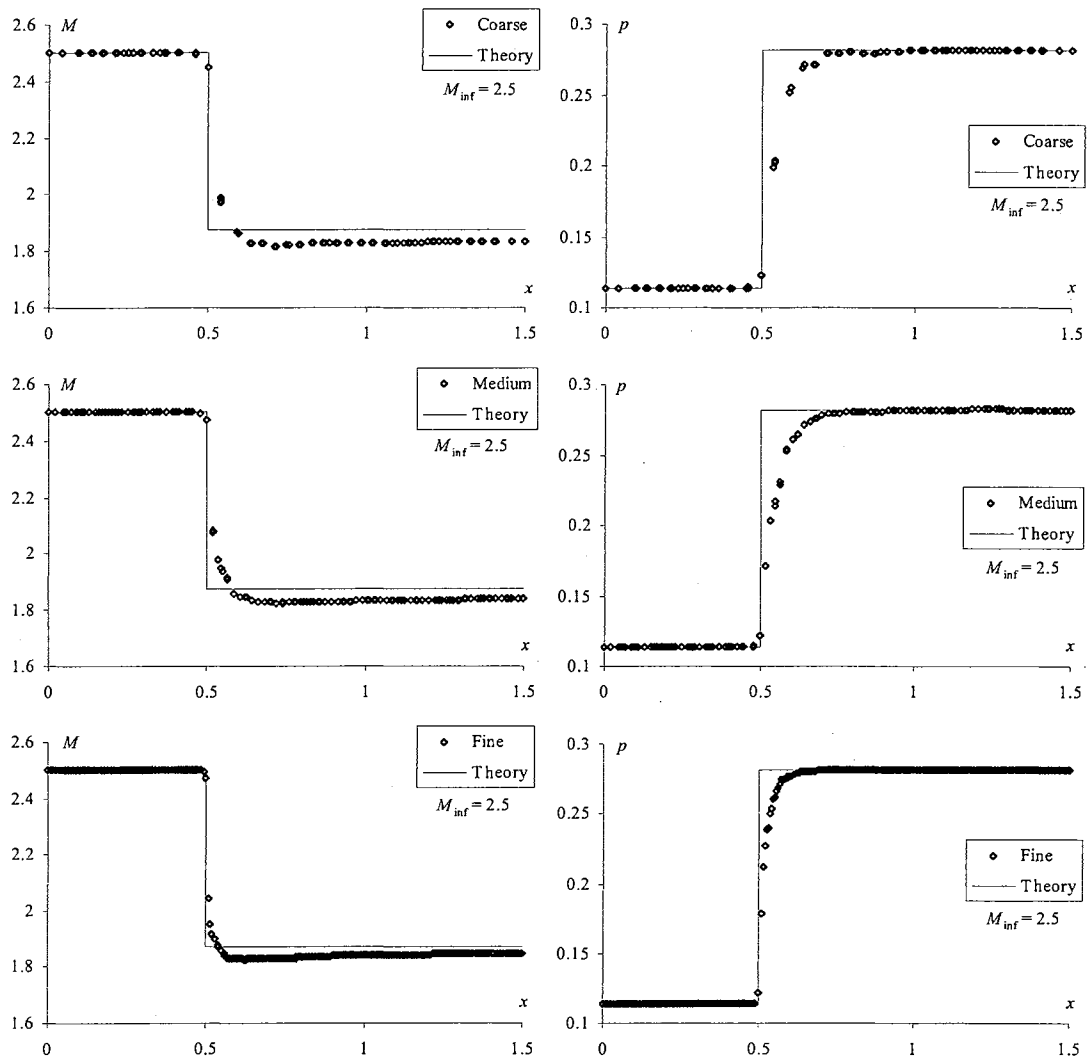


Figure 5.5: Pressure and Mach distributions for one-point, low-order dissipation solution to the oblique shock at Mach 2.5.

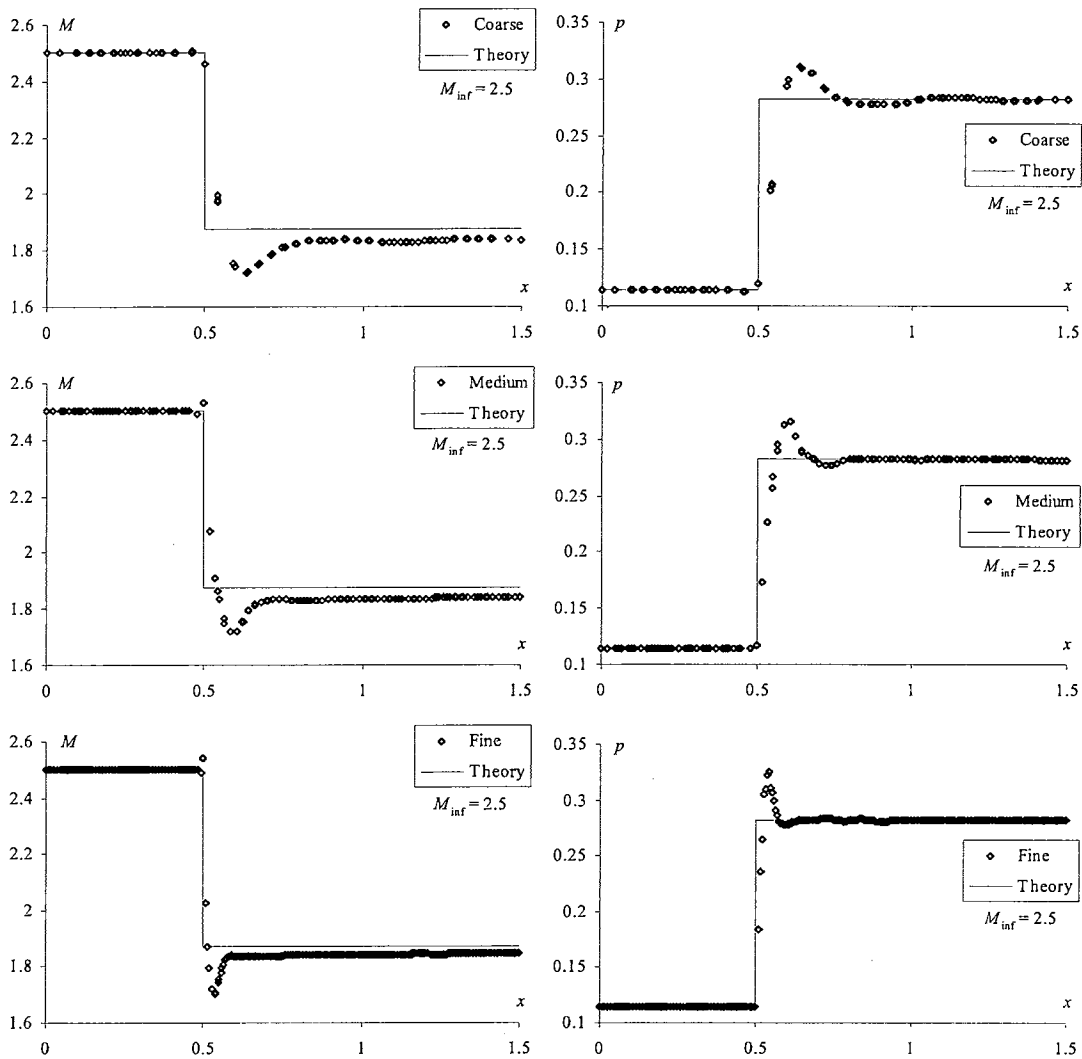


Figure 5.6: Pressure and Mach distributions for one-point, high-order dissipation solution to the oblique shock at Mach 2.5.

Notice that the high-order dissipation solution exhibits a severe overshoot in the vicinity of the shock for all three solutions. Increasing the dissipation factor will not help damp this overshoot because too much dissipation actually destabilizes the solution for this case. Because of this, the low-order dissipation solution is actually preferred since it produces a monotone shock profile. In fact, the high-order dissipation seems to be tuned

for subsonic and transonic applications, making it unsuitable for supersonic applications unless some additional dissipation in the form of a bulk viscosity is included.¹⁹ A bulk viscosity model was not implemented here since the low-order dissipation does perform satisfactorily as seen above.

As a final test with this geometry, the free-stream Mach number will be increased to 6.0 in order to investigate a solution in the upper Mach range for the solver. For an upstream Mach number of 6.0, ideal gas theory predicts a downstream Mach number of $M_2 = 3.99179$, an oblique shock angle of $\theta = 22.67$ degrees, a pressure ratio across the shock of $p_2/p_1 = 6.07$, and a density ratio across the shock of $\rho_2/\rho_1 = 3.10108$. Only the one-point, low-order dissipation solution will be presented for this problem because the high-order dissipation is incapable of producing a reasonable approximation for the strong shock that forms at Mach 6.0. Figure 5.7 presents a summary of the control parameters used for the solution to the oblique shock at Mach 6.0. As before, the solver control parameters are identical for all three grids except for the number of solution steps, *nstp*.

<i>Low-1pt</i>	
<i>gamma</i>	= 1.40d0
<i>mach</i>	= 2.50d0
<i>diss</i>	= 1.00d0
<i>cfl</i>	= 0.60d0
<i>nstp</i>	= 300
<i>nout</i>	= 300
<i>ncyc</i>	= 3
<i>isol</i>	= 0
<i>idiss</i>	= 0
<i>ipnt</i>	= 1

Figure 5.7: Summary of solver control parameters for oblique shock at Mach 6.0.

Figure 5.8 presents a plot of the residual convergence history for each grid. The coarse, medium, and fine grids were run for 300, 400 and 500 iterations respectively in order to achieve full numerical convergence.

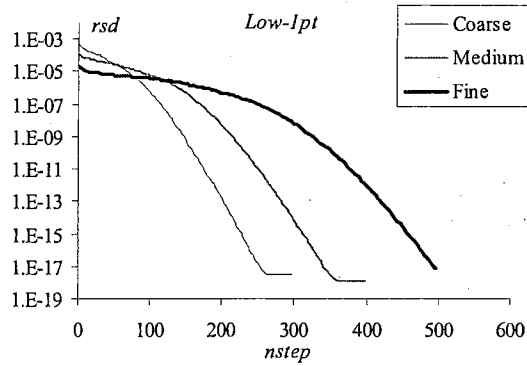


Figure 5.8: Plot of residual histories for oblique shock at Mach 6.0.

As with the Mach 2.5 solution, values of the local Mach number and pressure are extracted along a cut-line on the lower surfaces of the domain. Table 5.3 presents a summary of the average Mach and pressure error behind the shock along with the computed value of the local Mach number and pressure ratio on the cut-line at a point where at $x \approx 1.46$.

Table 5.3: Summary of one-point, low-order dissipation solution to the oblique shock at Mach 6.0.

	M_2	p_2/p_1	$\bar{\epsilon}_M$ (%)	$\bar{\epsilon}_p$ (%)
<i>coarse</i>	3.8163	5.5090	4.202	24.231
<i>medium</i>	3.8233	5.8268	3.693	20.514
<i>fine</i>	3.8628	6.0182	3.588	10.873

Notice that the average pressure error is quite a bit higher than that observed for the Mach 2.5 solution. The reason for this is evident in the computed Mach number and pressure distributions plotted in Figure 5.9.

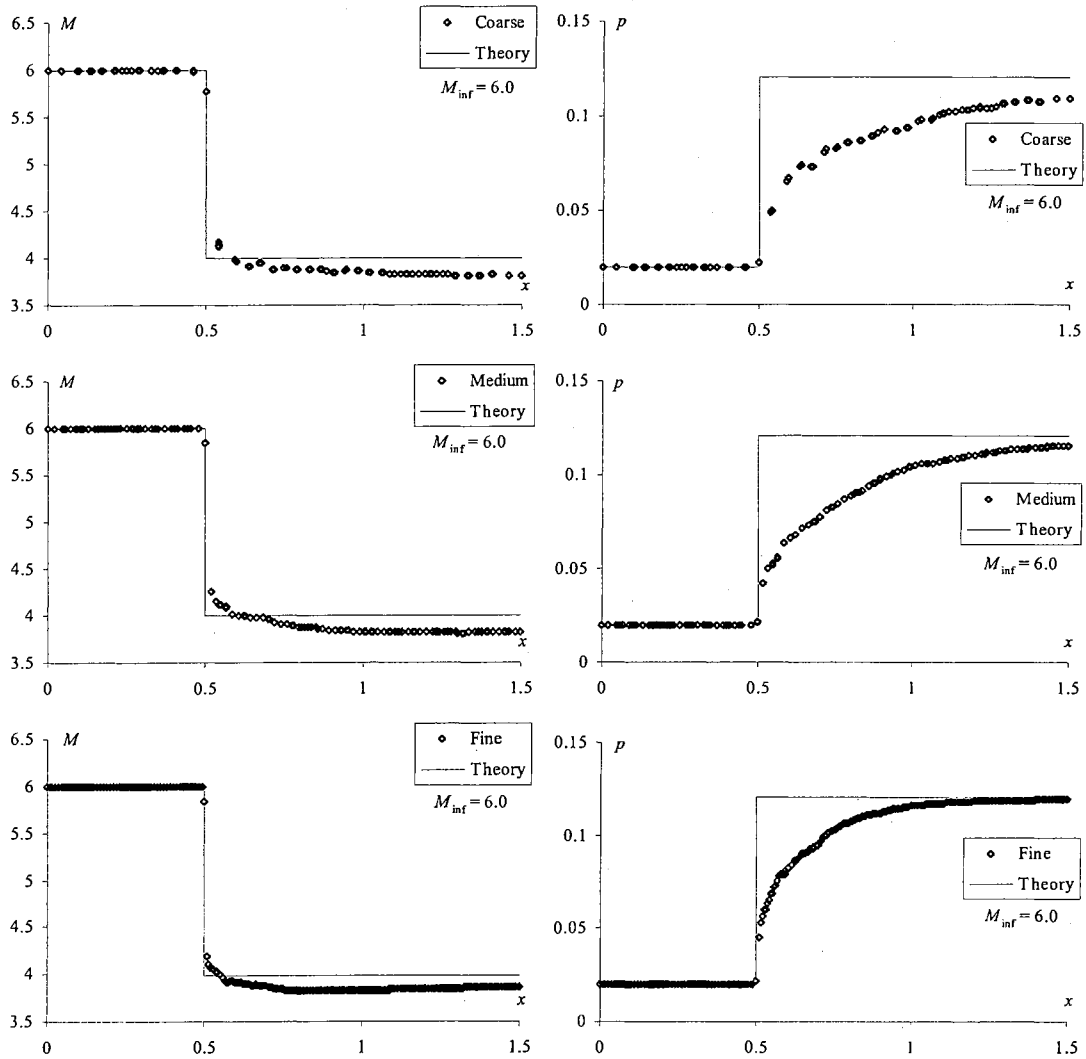


Figure 5.9: Pressure and Mach distributions for one-point, low-order dissipation solution to the oblique shock at Mach 6.0.

Unlike the Mach 2.5 solution, the pressure distributions in Figure 5.9 are in poor agreement with the sharp theoretical solution. In this case, stabilization has been

achieved at the expense of excessively smearing the shock. Despite this, the solution does converge toward the theoretical values as the grid resolution increases and the order of convergence is similar to that observed for the Mach 2.5 solution. This hypersonic Mach number seems to be the practical limit for the low-order dissipation model.

5.1.2 Prandtl-Meyer Expansion

This steady problem consists of a supersonic flow through an expansion corner at an angle of 15 degrees. The resulting flow develops a Prandtl-Meyer expansion fan on the backside of the 15 degree corner. The results will be compared to the exact solution computed using the perfect gas equations. This test will involve an upstream Mach number of 2.5, which results in a downstream Mach number of 3.23684, an expansion fan angle of 20.58 degrees, a downstream pressure ratio of 0.32743, and a downstream density ratio of 0.45046.

The layout of the computational domain, which covers the volume $x \in [0, 2]$, $y \in [-0.402, 1]$ & $z \in [0, 0.1]$, is presented in Figure 5.10. Boundary conditions for the seven boundary surfaces enclosing this domain are specified as follows: surfaces 1 and 2 are symmetry planes, surfaces 3 and 4 are solid walls, and surfaces 5, 6, and 7 are far-field boundaries.

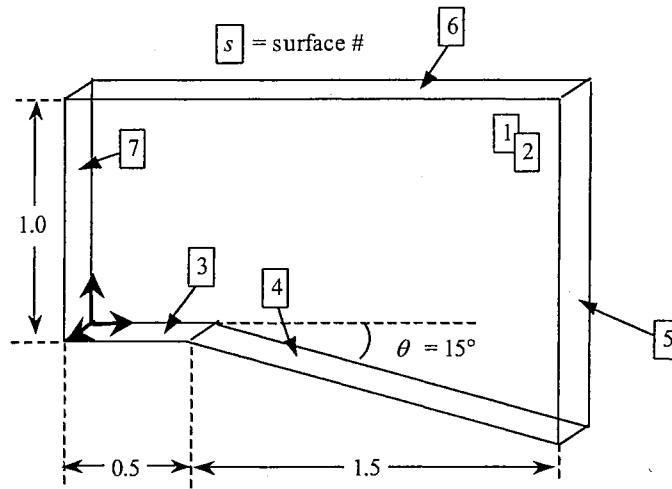


Figure 5.10: Layout of computational domain for Prandtl-Meyer expansion.

To solve this problem, we employ three grids that are refined successively using a constant refinement ratio of $r = 2$. Table 5.4 presents the grid spacing h , number of nodes nnd , number of elements nel , and the average computational time required per iteration Δt_{cpu} for each grid. All three grids are made up of tetrahedral elements and are generated using a uniform grid spacing. In order to maintain a reasonable element count, the thickness of the domain was cut in half when generating the fine grid. Figure 5.11 shows the surface triangulation of the coarse grid generated for this study.

Table 5.4: Summary of grid parameters for Prandtl-Meyer expansion.

	h	nnd	nel	Δt_{cpu}
<i>coarse</i>	0.04	5886	23313	0.0379 s
<i>medium</i>	0.02	36345	165581	0.1792 s
<i>fine</i>	0.01	142789	659119	0.7749 s

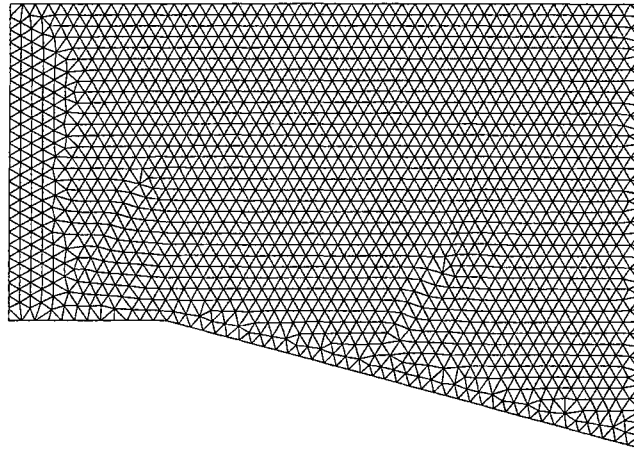


Figure 5.11: Representative surface triangulation for Prandtl-Meyer expansion.

As with the oblique shock problem, four different solutions will be computed for each grid in order to evaluate the performance of the low-order and high-order dissipation models with both the one-point and four-point numerical integration options. A summary of the relevant solver parameters for this problem is provided in Figure 5.12. The solver control parameters are identical for all three grids except for the number of solution steps, n_{stp} , since the number of steps required for numerical convergence increases as the grid resolution increases.

<i>Low-1pt</i>	<i>High-1pt</i>
gamma = 1.40d0	gamma = 1.40d0
mach = 2.50d0	mach = 2.50d0
diss = 0.60d0	diss = 0.90d0
cfl = 0.80d0	cfl = 0.70d0
nstp = 400	nstp = 500
nout = 400	nout = 500
ncyc = 3	ncyc = 3
isol = 0	isol = 0
idiss = 0	idiss = 1
ipnt = 1	ipnt = 1
<i>Low-4pt</i>	<i>High-4pt</i>
gamma = 1.40d0	gamma = 1.40d0
mach = 2.50d0	mach = 2.50d0
diss = 0.60d0	diss = 0.90d0
cfl = 0.80d0	cfl = 0.70d0
nstp = 400	nstp = 500
nout = 400	nout = 500
ncyc = 3	ncyc = 3
isol = 0	isol = 0
idiss = 0	idiss = 1
ipnt = 4	ipnt = 4

Figure 5.12: Summary of solver control parameters for oblique shock at Mach 2.5.

Figure 5.13 presents a plot of the residual convergence history for the four solutions on each grid. The low-order dissipation solution on the coarse, medium, and fine grids required 400, 500 and 600 iterations respectively in order to achieve full numerical convergence. The high-order dissipation solution required 500, 700, and 1200 iterations on the coarse, medium, and fine grids respectively.

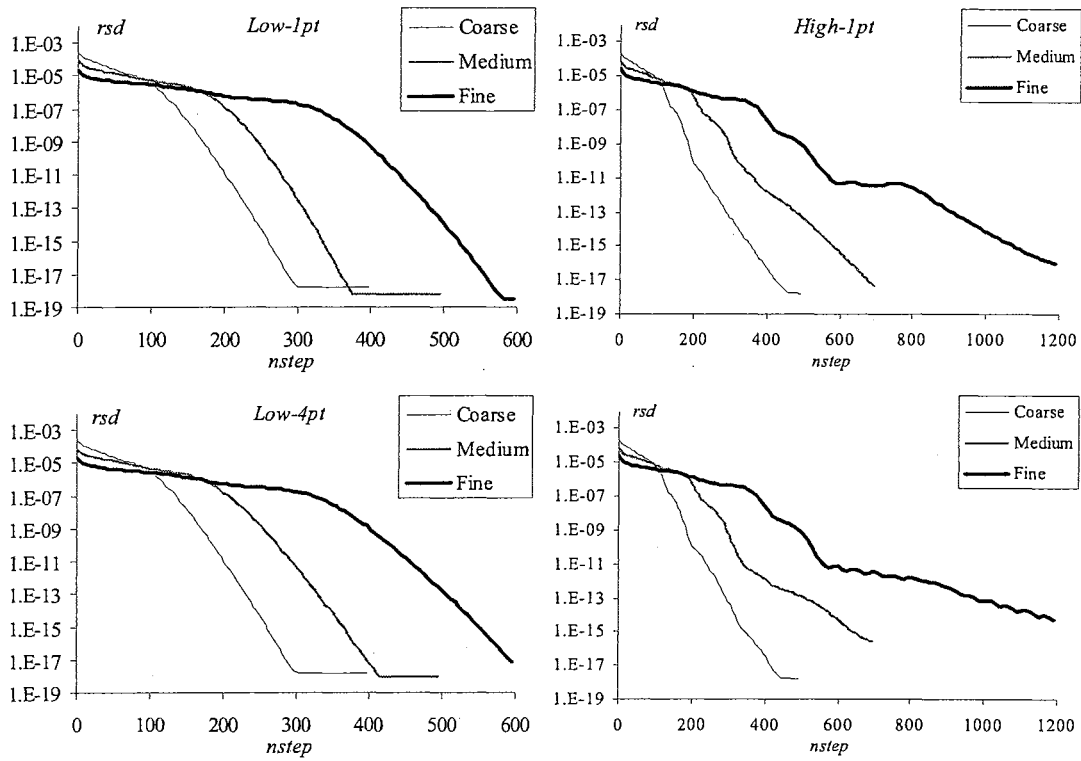


Figure 5.13: Plot of residual histories for Prandtl-Meyer expansion at Mach 2.5.

As with the oblique shock problem, results for these solutions are obtained for a cut-line along surfaces 3 and 4 at $z = 0.05$. Table 5.5 presents the average Mach error, $\bar{\mathcal{E}}_M$, and average pressure error, $\bar{\mathcal{E}}_p$, downstream of the shock for all twelve solutions. In each case, the average error is computed as the integral of the percent error along the cut-line divided by the length of the cut-line for $x > 0.5$.

Table 5.5: Summary of average error downstream of the shock for all solutions to the Prandtl-Meyer expansion at Mach 2.5.

<i>Solution</i>	<i>Coarse</i>		<i>Medium</i>		<i>Fine</i>	
	$\bar{\epsilon}_M$ (%)	$\bar{\epsilon}_p$ (%)	$\bar{\epsilon}_M$ (%)	$\bar{\epsilon}_p$ (%)	$\bar{\epsilon}_M$ (%)	$\bar{\epsilon}_p$ (%)
<i>Low-1pt</i>	5.890	13.166	5.247	8.660	4.243	4.899
<i>Low-4pt</i>	6.731	16.407	5.942	11.136	4.691	6.119
<i>High-1pt</i>	4.091	6.727	4.097	3.426	3.670	1.791
<i>High-4pt</i>	3.898	6.673	3.773	3.386	3.427	1.696

Based on the data given in Table 5.5, it appears that all four solutions are again converging toward the correct theoretical value as the grid resolution increases. The observed order of convergence for the one-point, low-order dissipation solution, is computed to be $p = 0.74$ using the Mach error or $p = 1.72$ using the pressure error data. In both cases, only the error data from the medium and fine grids was used.

In addition to the error data, plots of the Mach number and pressure distributions along the cut-line are provided for the one-point low order and high-order dissipation solutions in Figure 5.5 and Figure 5.6 respectively. Both of the four-point solutions are indistinguishable from the similar one-point solutions and are not presented here.

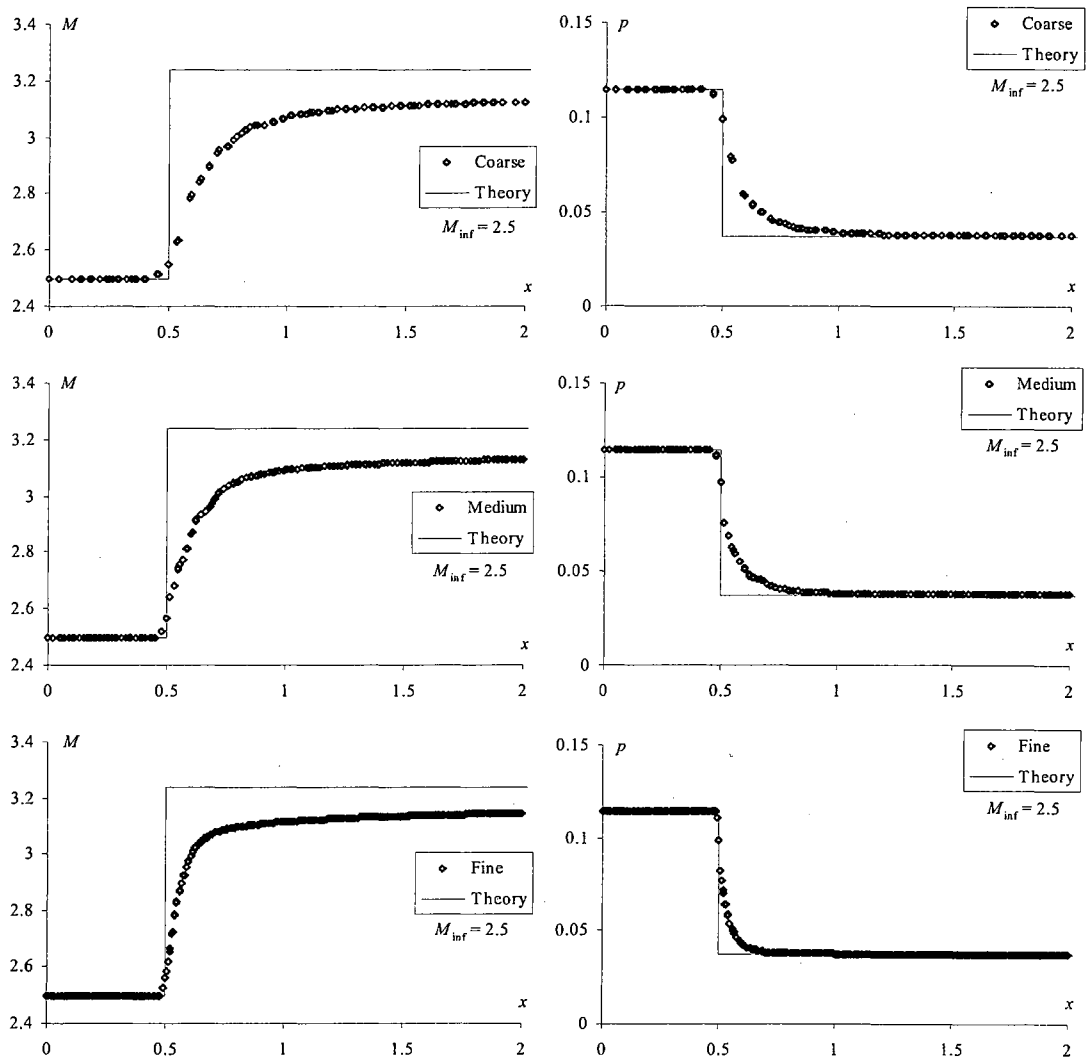


Figure 5.14: Pressure and Mach distributions for one-point, low-order dissipation solution to the Prandtl-Meyer at Mach 2.5.

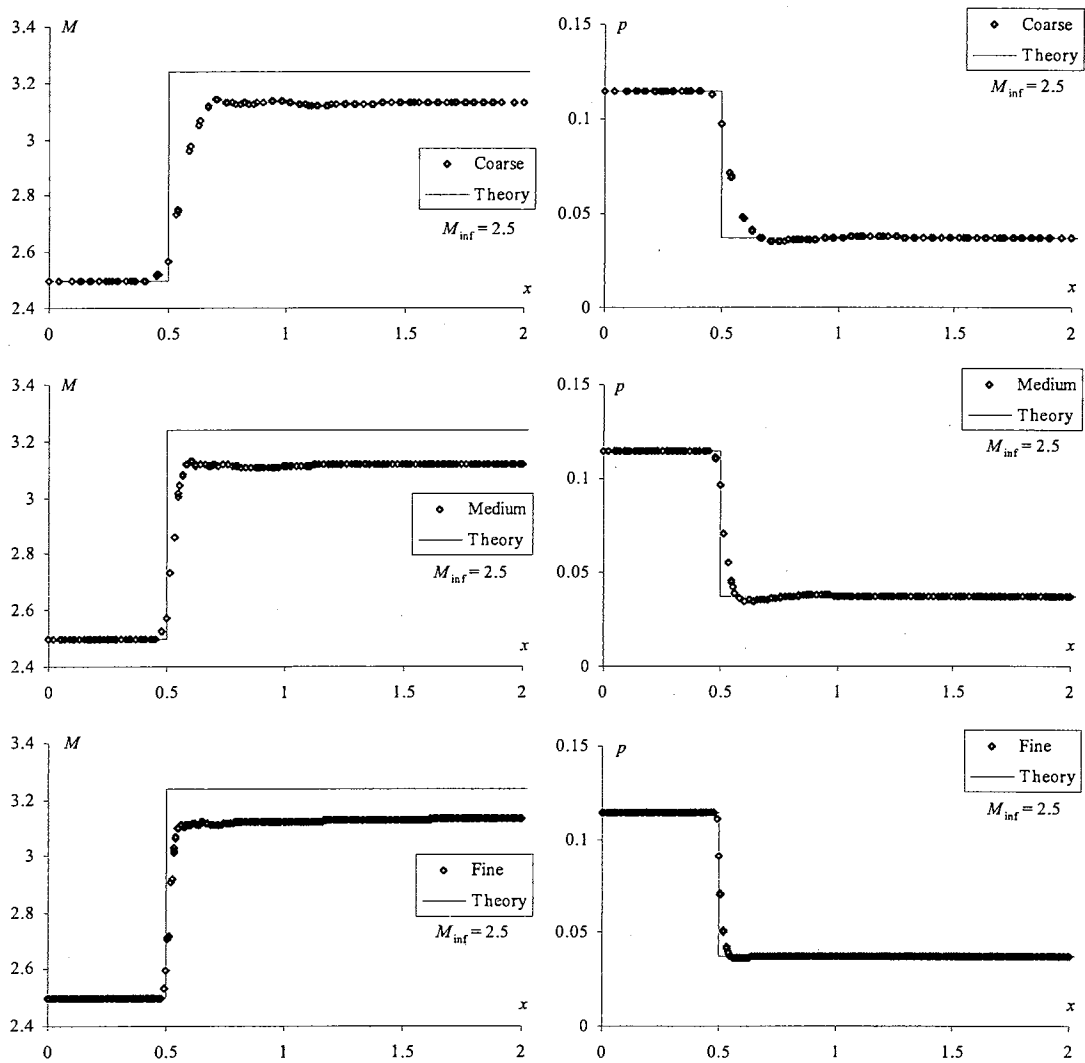


Figure 5.15: Pressure and Mach distributions for one-point, high-order dissipation solution to the Prandtl-Meyer at Mach 2.5.

For this problem, the high-order solution actually does a better job approximating the sharp jump in the theoretical Mach and pressure distributions. It does fall short of achieving the maximum Mach amplitude downstream of the expansion by approximately 4%. Despite this relatively large error, the solution is comparable to other CFD solutions

for the same problem. The NPARC alliance reports an average Mach error of over 5% for the WIND codes.³⁴

5.1.3 Converging-Diverging Nozzle

This steady problem consists of subsonic flow entering a converging-diverging nozzle whose cross-sectional area is defined by Equation (5.3).

$$(5.3) \quad A(x) = 1.0 + \frac{(x - 2.5)^2}{12.5}, \quad 0 \leq x \leq 5$$

The objective of this problem is to verify the transonic shock capturing capabilities of the solver by generating a solution where a shock wave forms between the throat and the exit plane. Since we do not have direct control over the back pressure at the exit plane with the three types of boundary conditions implemented in the solver, we must choose an initial condition that will lead to the formation of a shock wave in the nozzle. After some experimentation, a free-stream Mach number of 0.5 was chosen as our initial condition, which corresponds to enforcing an initial, dimensionless pressure as defined by Equation (2.43). This initial condition leads to the formation of a shock in the nozzle with the inlet and exit planes simply specified as far-field boundaries.

The layout of the computational domain, which covers the volume $x \in [0, 5]$, $y \in [-0.75, 0.75]$ & $z \in [0, 1]$, is presented in Figure 5.16. Boundary conditions for the six boundary surfaces enclosing this domain are specified as follows: surfaces 1 and 2 are symmetry planes, surfaces 3 and 5 are solid walls, and surfaces 4 and 6 are far-field boundaries.

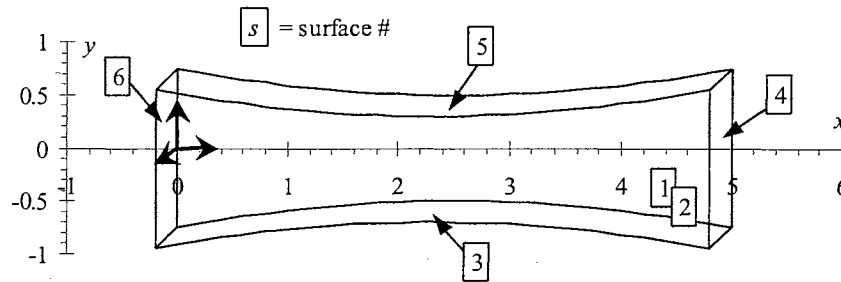


Figure 5.16: Layout of computational domain for converging-diverging nozzle.

To solve this problem, we employ two grids that are refined successively using a constant refinement ratio of $r = 2$. Table 5.6 presents the grid spacing h , number of nodes nnd , number of elements nel , and the average computational time required per iteration Δt_{cpu} for each grid. Both grids are made up of tetrahedral elements and are generated using a uniform grid spacing. Figure 5.17 shows the surface triangulation of the coarse grid generated for this study.

Table 5.6: Summary of grid parameters for converging-diverging nozzle.

	h	nnd	nel	Δt_{cpu}
<i>coarse</i>	0.100	7116	32774	0.1108 s
<i>fine</i>	0.050	48219	243643	1.0456 s

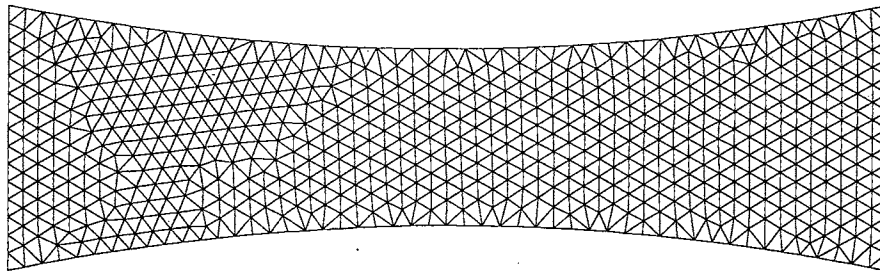


Figure 5.17: Representative surface triangulation for converging-diverging nozzle.

Two different solutions will be computed for each grid in order to evaluate the performance of the low-order and high-order dissipation models. A summary of the

relevant solver parameters for this problem is provided in Figure 5.18. The solver control parameters are identical for all three grids except for the number of solution steps, *nstp*.

<i>Low-1pt</i>		<i>High-1pt</i>	
gamma	= 1.40d0	gamma	= 1.40d0
mach	= 0.50d0	mach	= 0.50d0
diss	= 0.60d0	diss	= 1.00d0
cfl	= 0.80d0	cfl	= 0.70d0
nstp	= 400	nstp	= 500
nout	= 400	nout	= 500
ncyc	= 3	ncyc	= 3
isol	= 0	isol	= 0
idiss	= 0	idiss	= 1
ipnt	= 1	ipnt	= 1

Figure 5.18: Summary of solver control parameters for converging-diverging nozzle.

Figure 5.13 presents a plot of the residual convergence history for the two solutions on each grid. Both the low-order and high-order dissipation solutions on the coarse and fine grids required 4500 and 8000 iterations respectively in order to achieve full numerical convergence.

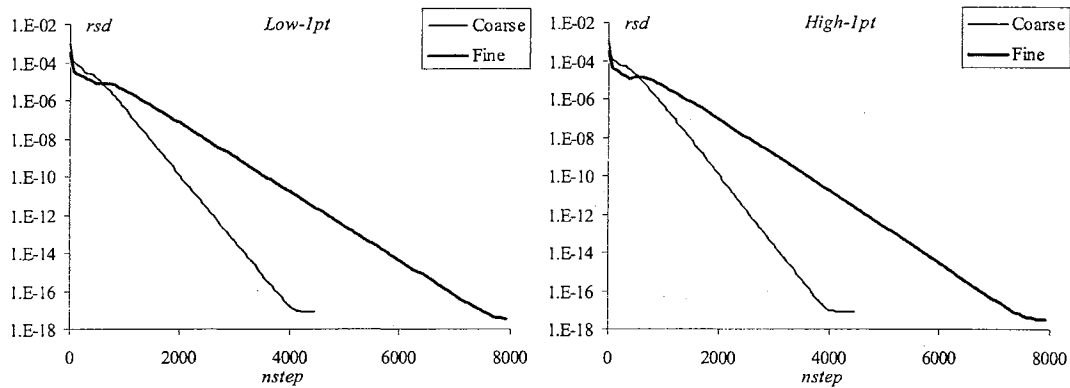


Figure 5.19: Plot of residual histories for converging-diverging nozzle.

Results for this problem are taken along a cut-line through the center of the nozzle on symmetry surface 1. The CFD results will be compared to the theoretical solution computed using the one-dimensional equations of gas dynamics. Since the CFD

geometry is not one-dimensional, we must consider the effect of the wall curvature when comparing the results. Figure 5.20 shows a plot of the Mach profile within the nozzle computed using the fine grid. As expected, the shock, as well as the overall Mach profile, has a slight curvature to it. The theoretical solution using one-dimensional gas dynamics will be valid along streamlines that are everywhere perpendicular to this curved Mach profile. This means that the Mach on the inlet and exit surface of the computational domain is not expected to be constant since they do not have the appropriate curvature.

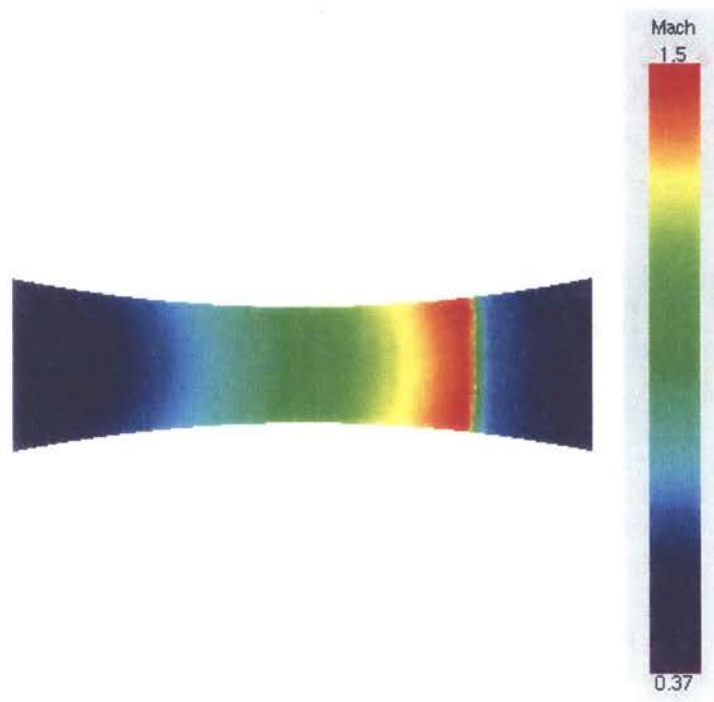


Figure 5.20: Colored Mach profile for converging-diverging nozzle.

Figure 5.21 shows a comparison of the computed Mach distribution and the theoretical Mach distribution for all four solutions. Since we have no direct control over the back pressure controlling the location of the shock in the nozzle, the theoretical

solution is computed based on the average Mach number computed at the exit of the converging-diverging nozzle for each case.

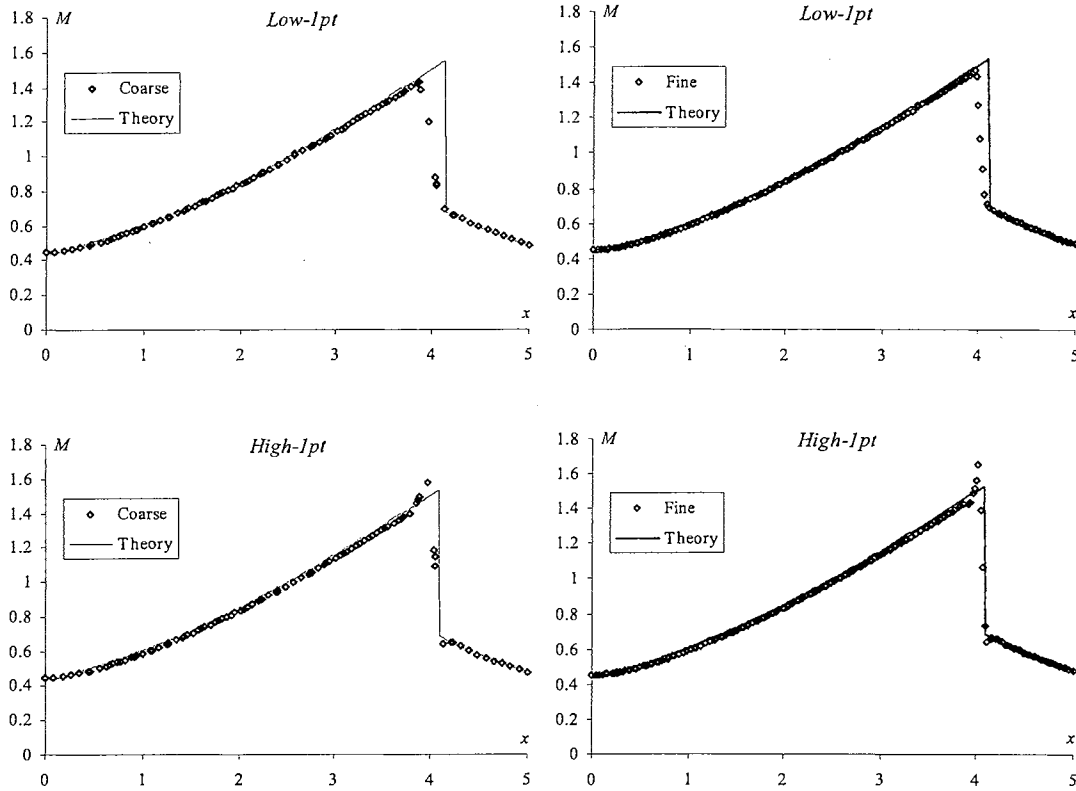


Figure 5.21: Mach distributions for one-point, low-order and high-order dissipation solutions to the converging-diverging nozzle.

Notice in Figure 5.21 that the high-order solutions predict the theoretical shock location better than the equivalent low-order solution. The low-order solution does provide reasonable accuracy for the overall Mach profile, and it converges toward the theoretical shock location as the grid resolution is increased. However, the low-order solution is not useful for most practical transonic applications because it requires a low value for the dissipation factor in order to provide accurate shock resolution. For this problem, the dissipation factor for both low-order solutions is specified as 0.5. In

practice, true three-dimensional solutions for more complicated geometries are not sufficiently stabilized using this low dissipation factor. Hence the high-order dissipation model will be preferred for most subsonic and transonic applications.

In order to quantify the order of convergence for this problem, we will consider the computed value of the inlet Mach number. The theoretical value for the inlet Mach number is directly related to the area ratio of the nozzle as defined by Equation (5.4).

$$(5.4) \quad \frac{A}{A^*} = \frac{\sqrt{\gamma} \left(\frac{\gamma+1}{2} \right)^{(\gamma+1)/(2-2\gamma)}}{M \sqrt{\gamma} \left(1 + \frac{\gamma-1}{2} M^2 \right)^{(\gamma+1)/(2-2\gamma)}}$$

Given that the ratio of the inlet area to the throat area is $A/A^* = 1.5$, the theoretical value for the inlet Mach number is numerically computed to be 0.430262. This theoretical value will be compared to the average Mach number at the inlet of the nozzle computed using the integral definition for the average value of a function. Table 5.7 presents a summary of the average Mach number at the inlet and exit of the nozzle for all four solutions.

Table 5.7: Summary of Mach number at inlet and exit of converging-diverging nozzle.

<i>Solution</i>	<i>Coarse</i>		<i>Fine</i>	
	\bar{M}_{in}	\bar{M}_{exit}	\bar{M}_{in}	\bar{M}_{exit}
<i>Low-1pt</i>	0.43190	0.48656	0.43112	0.48463
<i>High-1pt</i>	0.43059	0.48193	0.43059	0.48188

Notice that both the low-order and high-order dissipation solutions converge toward the theoretical inlet Mach number as the grid resolution increases. Using the inlet Mach number data in Table 5.7, the order of convergence for the high-order dissipation solution is computed to be $p = 2.10$.

5.1.4 Subsonic NACA-0012 Airfoil

This steady problem consists of a NACA 0012 airfoil in a subsonic, nearly incompressible flow. The results will be compared to the theoretical solution from ideal aerodynamics for a thin airfoil. The layout of the computational domain, which has the x -axis running along the airfoil's chord and y -axis along the span of the airfoil, is presented in Figure 5.16. Boundary conditions for the eight boundary surfaces enclosing this domain are specified as follows: surfaces 1 and 2 are solid walls, surfaces 3, 4, 5 and 6 are symmetry planes, and surfaces 7 and 8 are far-field boundaries.

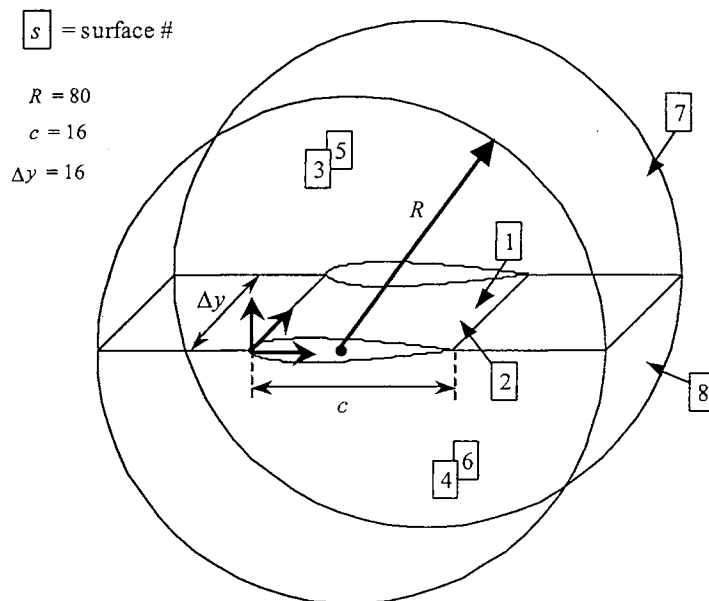


Figure 5.22: Layout of computational domain for NACA 0012 airfoil.

To solve this problem, we employ three grids that are refined successively using a constant refinement ratio of $r = 2$. This results in nearly 3 million elements for the third refinement. Hence, the width of the computational domain is cut in half for the fine grid in order to maintain a reasonable element count. Table 5.6 presents the number of nodes

nnd , number of elements nel , and the average computational time required per iteration Δt_{cpu} for all three grids. Each grid is made up of tetrahedral elements and is generated using a non-uniform grid spacing that concentrates elements near the surface of the airfoil where the strongest flow gradients reside. Figure 5.23 shows the overall surface triangulation of the coarse grid generated for this study, and Figure 5.24 shows a close-up of the surface triangulation on the surface of the airfoil for the coarse and medium grids.

Table 5.8: Summary of grid parameters for NACA 0012 airfoil.

	nnd	nel	Δt_{cpu}
<i>coarse</i>	8,815	43,552	0.2132 s
<i>medium</i>	59,011	311,060	2.3609 s
<i>fine</i>	262,859	1,406,731	13.1228 s

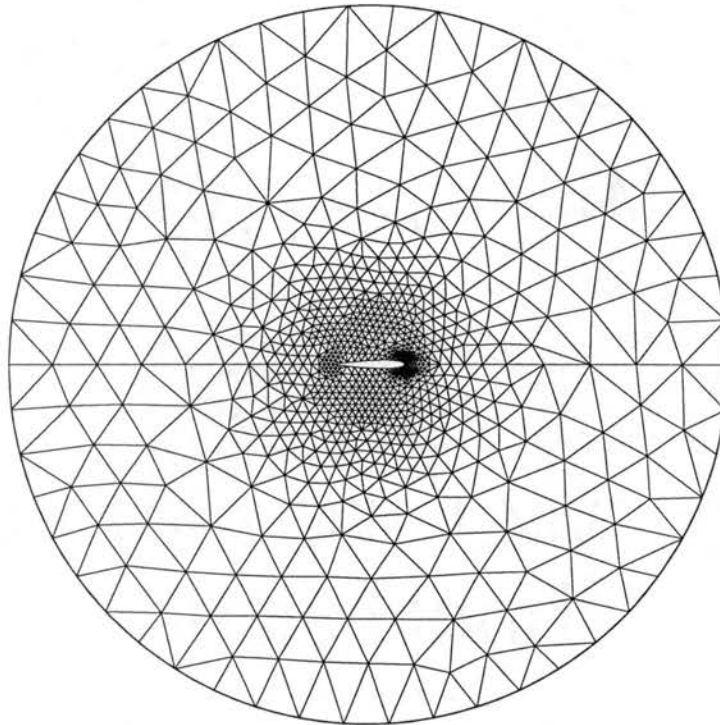


Figure 5.23: Representative grid for the NACA 0012 airfoil.

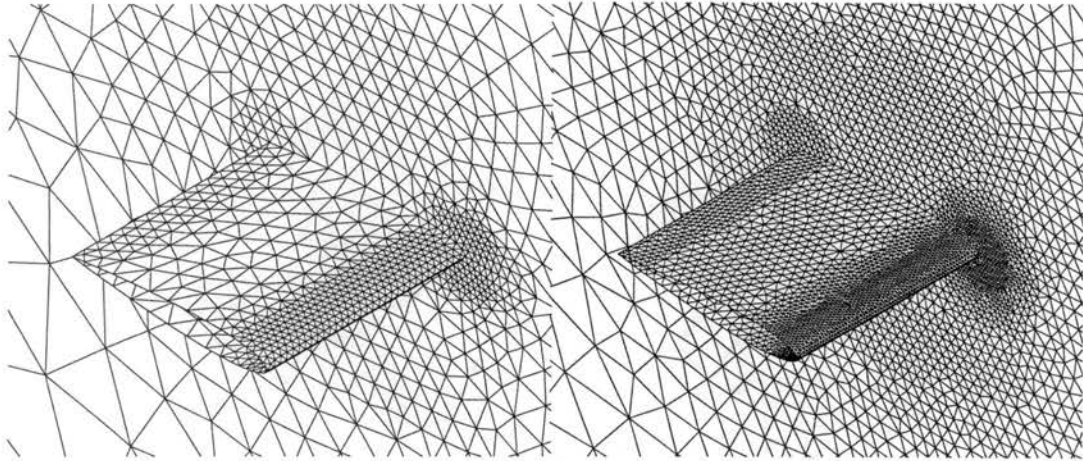


Figure 5.24: Close-up of coarse and medium surface grids near the NACA 0012 airfoil.

Only the high-order dissipation solution is investigated for this problem because the low-order model is overly dissipative at this low Mach number for all useful values of the dissipation factor. A summary of the relevant solver parameters for this problem is provided in Figure 5.25. Since the airfoil has a chord of 16, the reference dimension is specified as `refdim = 16.0` for this problem. The solver control parameters are identical for all three grids except for the number of solution steps, `nstp`.

<code>gamma</code>	<code>= 1.40d0</code>
<code>mach</code>	<code>= 0.30d0</code>
<code>refdim</code>	<code>= 16.0d0</code>
<code>diss</code>	<code>= 1.00d0</code>
<code>cfl</code>	<code>= 0.80d0</code>
<code>nstp</code>	<code>= 3000</code>
<code>nout</code>	<code>= 3000</code>
<code>ncyc</code>	<code>= 3</code>
<code>isol</code>	<code>= 0</code>
<code>idiss</code>	<code>= 1</code>
<code>ipnt</code>	<code>= 1</code>
<code>iaero</code>	<code>= .true.</code>

Figure 5.25: Summary of solver control parameters for NACA 0012 airfoil.

Figure 5.26 presents a plot of the residual convergence history for the three solutions. The solution on the coarse, medium and fine grids required 3000, 8000 and 16,000 iterations respectively in order to achieve numerical convergence.

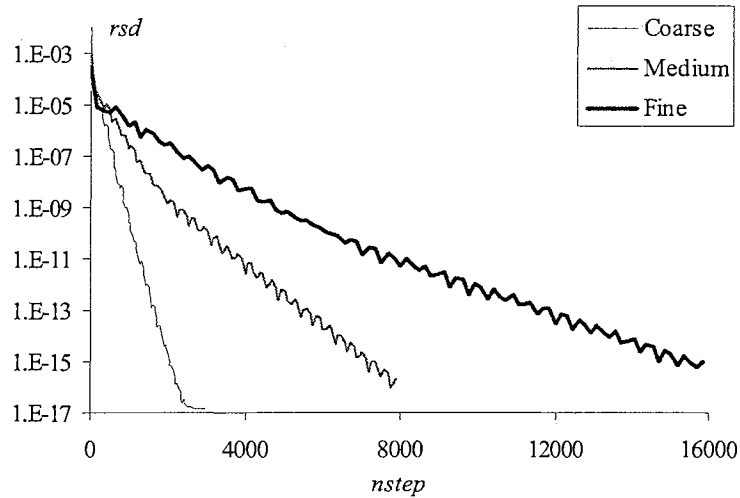


Figure 5.26: Plot of residual histories for the NACA 0012 airfoil at Mach 0.3 and zero angle of attack.

Results for this problem are taken along a cut-line defining the intersection of an xz -plane at $y = 8$ and the two surfaces defining the airfoil, surfaces 1 and 2. Figure 5.27 presents a comparison of the computed pressure coefficient distribution along the surface of the airfoil and the theoretical distribution computed using thin airfoil theory.³⁵ The theoretical solution has also been corrected for the effects of compressibility using the Prandtl-Glauert relation given in Equation (5.5).

$$(5.5) \quad C'_p = \frac{C_p}{\sqrt{1-M^2}}$$

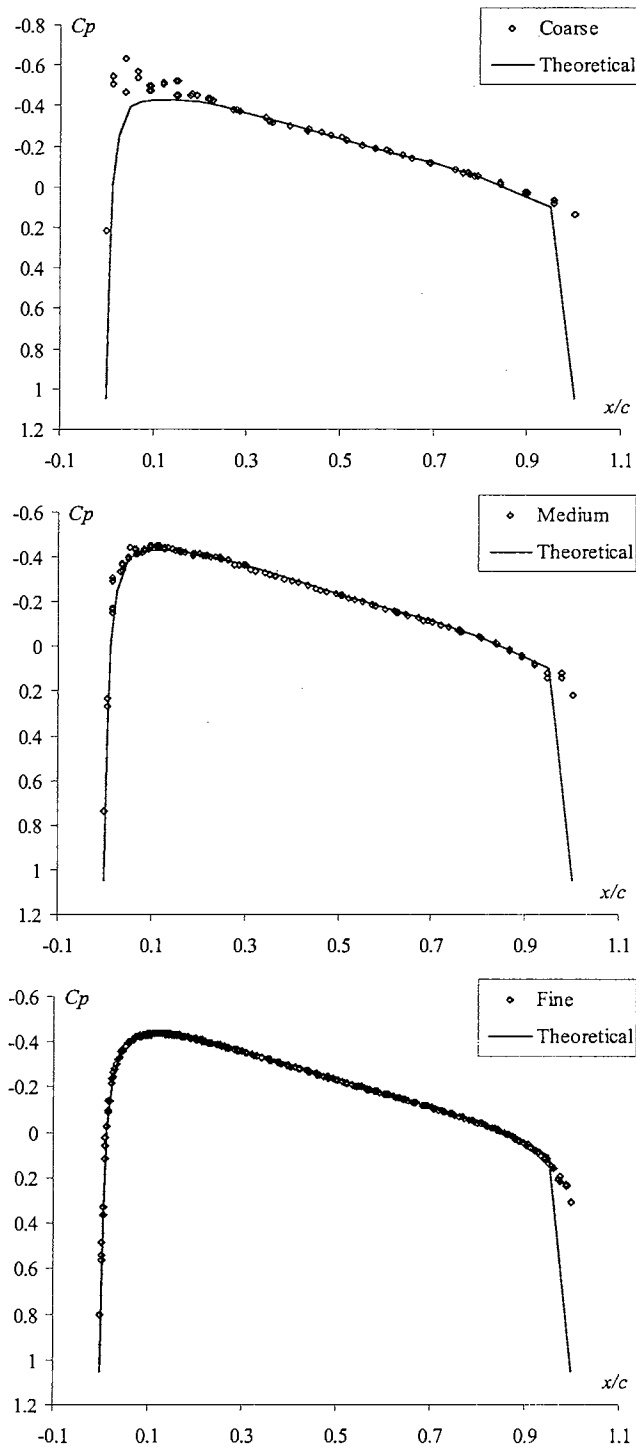


Figure 5.27: Comparison of pressure coefficient distributions for NACA 0012 airfoil at Mach 0.3 and zero angle of attack.

Notice that the computed pressure coefficient distribution for both the medium and fine grids are in excellent agreement with the theoretical solution. Even the coarse grid matches the theoretical solution aft of the leading edge, but it exhibits a large amount of scatter at the leading edge of the airfoil. Obviously the coarse grid does not have sufficient grid resolution at the leading edge of the airfoil and is not capable of accurately representing the strong flow gradients present there. In order to quantify the order of convergence for this solution, we consider the sectional lift and drag coefficients along the same cut-line used when extracting the pressure coefficient distributions in Figure 5.27. Table 5.9 presents a summary of the sectional lift and drag coefficients for each grid. The theoretical value for both sectional coefficients is zero for a symmetric airfoil in an ideal flow. Hence, the error in each case is simply the absolute value of the computed sectional coefficient.

Table 5.9: Summary of sectional lift and drag coefficients for the NACA 0012 airfoil at Mach 0.30 and zero angle of attack..

<i>Grid</i>	C_l	C_d	ϵ_{C_l}	ϵ_{C_d}
<i>Coarse</i>	0.08749	-0.58608	0.08749	0.58608
<i>Medium</i>	0.09775	-0.24861	0.09775	0.24861
<i>Fine</i>	-0.08432	-0.15792	0.08432	0.15792

With the exception of the sectional lift coefficient for the coarse grid, the numbers in Table 5.9 show convergence toward the correct theoretical values as the grid resolution increases. The sectional lift coefficient actually increases when changing from the coarse to the medium grid, but the large amount of scatter at the leading edge of the coarse grid's pressure distribution is an indication that this grid is probably beyond the lower resolution limit required for asymptotic convergence of this value. The order of

convergence for this problem is computed to be $p = 2.07$ using the sectional lift coefficients from the medium and fine grids or $p = 1.15$ using the sectional drag coefficients.

In addition to the zero angle of attack case, this airfoil geometry is also analyzed for an angle of attack of 5 degrees. This nonzero angle of attack will allow us to verify the steady transpiration boundary condition by simulating an angle of attack using elastic deformation vectors. Based on the results from the previous tests, we generated a new computational grid that is refined better than the medium grid, but uses fewer elements than the fine grid. As with the fine grid of the previous solutions, the width of the computational domain was also cut in half for this problem. The resulting grid consists of 165,943 elements and 33,995 nodes.

Next, two elastic mode shapes representing rigid-body pitch and plunge are manually generated. The mode shapes consist of a deformation vector for each solid wall node. The first mode represents a uniform translation of one chord length along the z -axis, while the second mode represents a rotation of one degree around the y -axis. These two mode shapes allow us to simulate an angle of attack in two different ways. The second mode with a generalized displacement, or scaling factor, of 5.0 is equivalent to a static angle of attack of five degrees. Alternatively, the first mode with a dimensionless generalized velocity of -0.087489 , which is simply $\tan(5^\circ)$, is also equivalent to an angle of attack of five degrees. Results from both of these transpiration solutions will be compared to the theoretical solution from thin airfoil theory as well as a steady solution using $\alpha = 5.0$.

For an elastic problem, there are two important reference conditions that must be specified in the solver control file: the free-stream speed of sound a_{inf} and free-stream density ρ_{inf} . These reference conditions are used to non-dimensionalize the structural matrices and initial conditions specified in the elastic vectors file. Since we are not solving a coupled aeroelastic problem, we are only interested in the dimensionless quantities output by the CFD solver. In this case, it makes sense to input dimensionless quantities as our initial conditions as well. Specifically, we want to input a dimensionless generalized velocity for mode one, which corresponds to a 5.0 degree angle of attack. The reference conditions will have no effect on the other elastic solution for this problem since the initial condition is a generalized displacement, and generalized displacements are dimensionless by definition.

Using the non-dimensionalization given in Equation (4.24) for a generalized velocity, a free-stream speed of sound is computed such that the dimensional and dimensionless generalized velocity are equivalent, or $L/u_{inf} = 1.0$. Figure 5.28 provides a summary of the solver control parameters required for the transpiration solutions.

gamma	= 1.40d0
mach	= 0.30d0
alpha	= 0.00d0
refdim	= 16.0d0
diss	= 1.00d0
cfl	= 0.80d0
nstp	= 5000
nout	= 5000
ncyc	= 3
isol	= 0
idiss	= 1
ipnt	= 1
iaero	= .true.
ielast	= .true.
nr	= 2
ainf	= 53.3333d0
rhoinf	= 1.0d0

Figure 5.28: Summary of solver control parameters for transpiration solutions with the NACA 0012 airfoil.

With the free-stream speed of sound chosen appropriately, the dimensionless generalized velocity corresponding to a five degree angle of attack is input directly as the initial condition for mode one in the elastic vectors file. Figure 5.29 presents the head of the elastic vectors file up to the comment line that the elastic mode vectors themselves follow.

```

$ Number of elastic modes (nr)
  2
$ Mass matrix for elastic modes (nr x nr)
  1.0d0  0.0d0
  0.0d0  1.0d0
$ Damping matrix for elastic modes (nr x nr)
  1.0d0  0.0d0
  0.0d0  1.0d0
$ Stiffness matrix for elastic modes (nr x nr)
  1.0d0  0.0d0
  0.0d0  1.0d0
$ ICs for elastic modes (x1...xn, vx1...vxn)
  0.0d0  0.0d0  -0.087489  0.0d0
$ IBXN for elastic modes (nr)
  2      1
$ Elastic modes vectors (nwl 2) x nr

```

Figure 5.29: Head of elastic vectors file for the NACA 0012 airfoil with a specified mode one generalized velocity.

Figure 5.30 presents a plot of the residual convergence history for the three solutions. The labels Alpha 5, Transpiration 5 and Transpiration h_{dot} refer to the solution using $\alpha = 5$, the transpiration solution using a mode two generalized displacement of 5.0, and the transpiration solution using a mode one generalized velocity of -0.087489 respectively. All three solutions were run for 3000 total steps, which led to a sufficiently converged numerical solution.

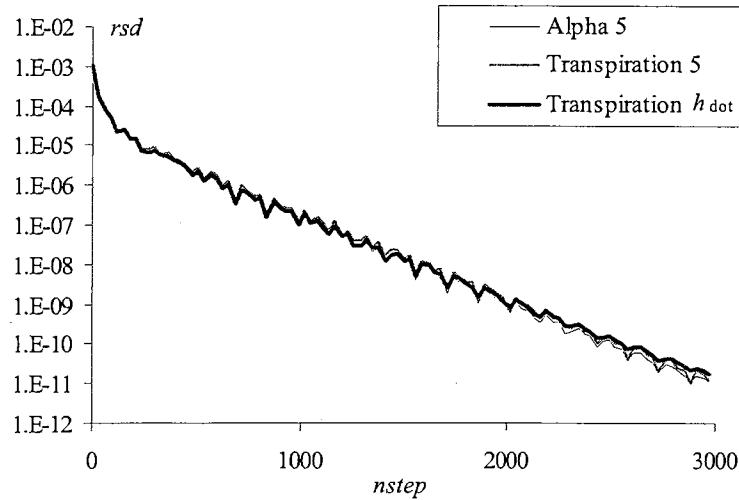


Figure 5.30: Plot of residual histories for the NACA 0012 airfoil at Mach 0.3 and 5.0 degree angle of attack.

Results for this problem are again taken along a cut-line defining the intersection of an xz -plane down the center of the domain and the two surfaces defining the airfoil, surfaces 1 and 2. Figure 5.31 presents a comparison of the computed pressure coefficient distributions along the surface of the airfoil and the theoretical distribution computed using thin airfoil theory. Once again, the theoretical solution has been corrected for the effects of compressibility using the Prandtl-Glauert relation.

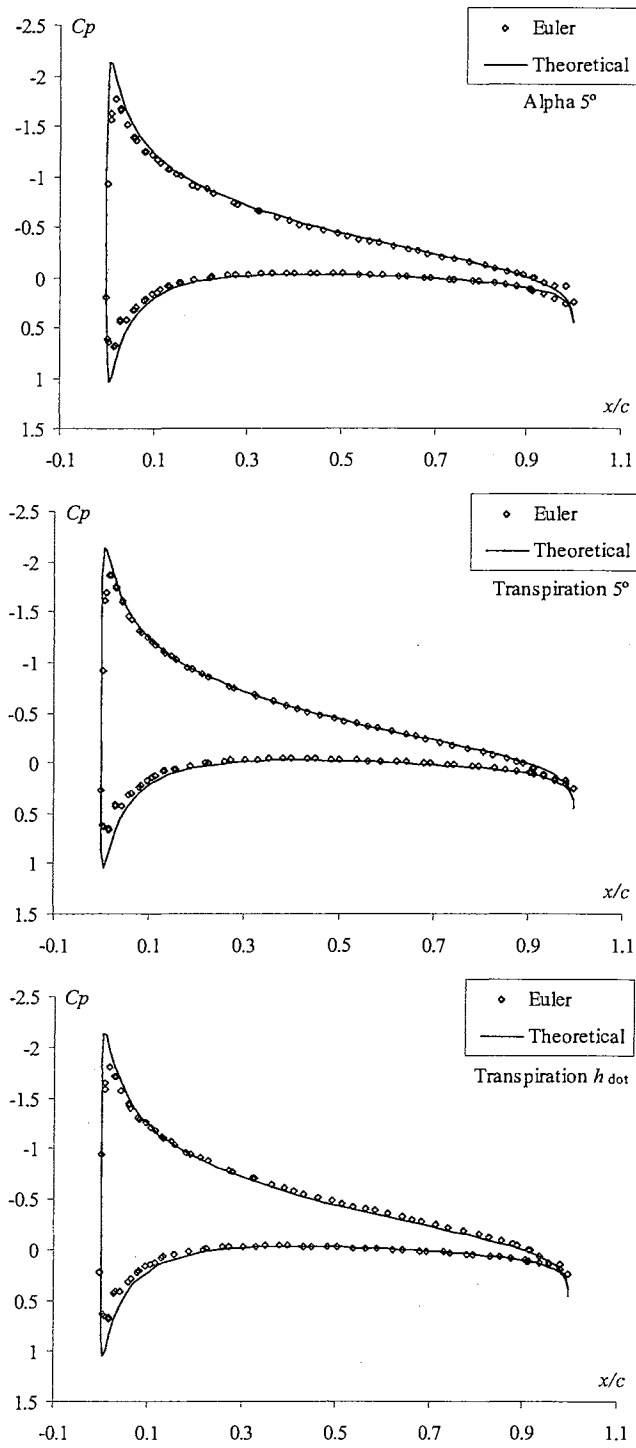


Figure 5.31: Comparison of pressure coefficient distributions for NACA 0012 airfoil at Mach 0.3 and 5.0 degree angle of attack.

Notice that all three solution are in excellent agreement with the theoretical solution. There is some deviation in the pressure peaks at the leading edge of the airfoil that could be improved by further refinement of the grid. However, the primary comparison here should be between the three CFD solutions themselves. The two transpiration solutions will not match the theoretical solutions any better than the solution computed with the “real” angle of attack since the only difference is in how the angle of attack is enforced. To that extent, these three solutions verify that the transpiration boundary condition is accurately enforcing both generalized displacements and generalized velocities since all three solutions are virtually indistinguishable.

In order to further quantify the comparison between the three solutions, we consider the sectional lift coefficient computed by the solver. Table 5.10 presents a summary of the sectional lift coefficient, C_l , for each solution. The theoretical value for the sectional lift coefficient is defined from thin airfoil theory using Equation (5.6).

$$(5.6) \quad C_l = \frac{2\pi\alpha}{\sqrt{1-M^2}} = 0.5748$$

Table 5.10: Summary of sectional lift coefficients for the NACA 0012 airfoil at Mach 0.30 and 5.0 degree angle of attack..

<i>Solution</i>	C_l	ϵ_{C_l} (%)
<i>Alpha 5°</i>	0.5788	0.698
<i>Transpiration 5°</i>	0.5816	1.179
<i>Transpiration h_{dot}</i>	0.6133	6.707

The numbers in Table 5.10 show that even the integrated pressure distributions for the three solution are in reasonable agreement. On first inspection, the error of the transpiration solution using the enforced plunge velocity has grown to nearly 7%.

However, this result is not entirely correct. By imposing an additional velocity component on the airfoil, the local velocity has increased from a free-stream velocity of 1.0 to 1.00382. This means that the local Mach number seen by the airfoil is higher than the free-stream velocity specified in the solver control file. Specifically, the local Mach number of the airfoil is computed to be 0.301146. This discrepancy is relatively small in terms of the Prandtl-Glauert relation, but it illustrates a significant difference between the effects of static transpiration and dynamic transpiration. With dynamic transpiration, the generalized velocity is changing the physics of the problem locally. Based on this problem, it appears that the influence of a generalized velocity degrades the accuracy of the solution more rapidly than generalized displacements.

Regardless, both elastic solutions are still reasonable considering that any small errors in the pressure profile will be magnified when integrated across the surface of the airfoil. Furthermore, the angle of attack chosen for this problem is relatively large for a transpiration simulation. Transpiration is typically applied to aeroelastic problems with small deflections on the order of one degree or less. Realistically, the five degree angle of attack used for this problem is approaching the limit of applicability for the transpiration method, and larger deflections beyond this point will lead to poor accuracy for the simulated deflections.

5.1.5 Conical Shock on a 10 degree Cone

This is the first truly three-dimensional verification problem. It consists of a steady supersonic flow over a cone with a semi-vertex angle of 10 degrees. The resulting conical flow field consists of an attached shock at the vertex of the cone with conical rays

of constant properties emanating from the vertex. The results will be compared to the analytical solution which is obtained by fitting the oblique shock equations to each point along the conical wave. The result is a set of nonlinear differential equations known as the Taylor-Maccoll differential equations, the numerical solution of which is tabulated in most books on compressible flows. This test will involve an upstream Mach number of 2.35, which produces a Mach number of $M_2 = 2.1469$, a pressure ratio of $p_2/p_1 = 1.4234$, and a density ratio of $r_2/r_1 = 1.2867$ on the surface of the cone.

We take advantage of symmetry when defining this problem and represent only the upper-half of the cone. The layout of the computational domain is presented in Figure 5.32. Boundary conditions for the six boundary surfaces enclosing this domain are specified as follows: surface 1 is a solid wall, surfaces 2 and 3 are symmetry planes, and surfaces 4, 5 and 6 are far-field boundaries. Furthermore, the node at the apex of the cone is specified as singular.

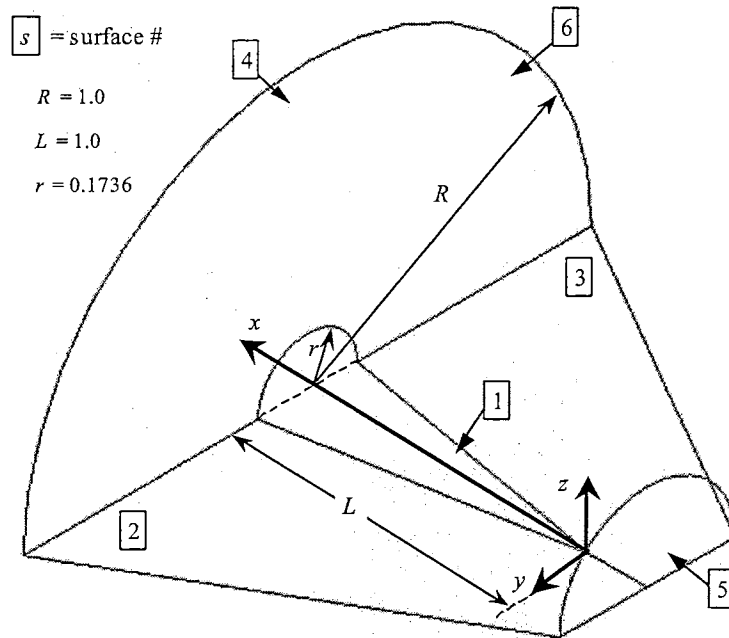


Figure 5.32: Layout of computational domain for the 10 degree cone.

In order to accurately represent the curved surfaces of this geometry, a fine grid is required near the surface of the cone with the grid spacing along the length of the cone defined as a function of its local radius. Since the radius of the cone at its apex is zero, the grid spacing would need to approach zero at that point. Obviously it is impossible to generate elements with zero size, so the curvature of the cone at the apex will not be accurately represented. This will be evident in the solutions we examine later in this section. Figure 5.33 presents the surface triangulation generated for this cone and Figure 5.34 shows a close up of the surface triangulation at the apex of the cone. The grid generated for this problem consists of 187,615 elements and 35,825 nodes. Grid refinement will not be pursued for this problem since the current grid is at the limit of our grid generator's capabilities.

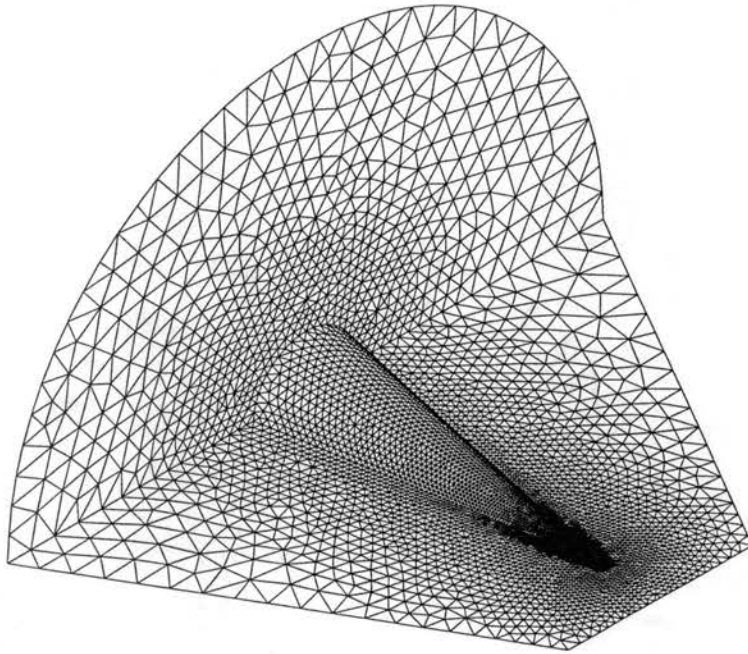


Figure 5.33: Surface triangulation for 10 degree cone.

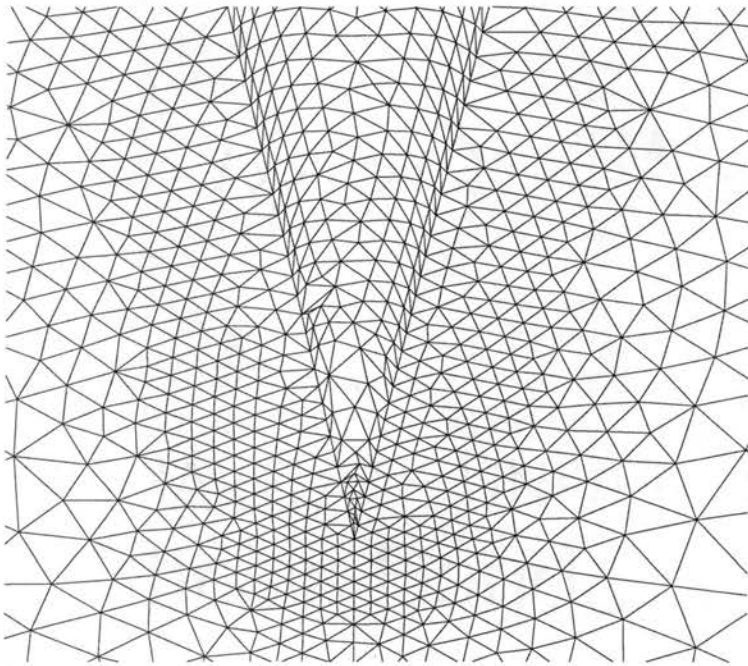


Figure 5.34: Close-up of surface triangulation at the apex of 10 degree cone.

Two different solutions are again computed for this geometry in order to evaluate the performance of the low-order and high-order dissipation models. A summary of the relevant solver parameters for this problem is provided in Figure 5.18.

<i>Low-1pt</i>		<i>High-1pt</i>	
gamma	= 1.40d0	gamma	= 1.40d0
mach	= 2.35d0	mach	= 2.35d0
diss	= 0.80d0	diss	= 1.00d0
cfl	= 0.80d0	cfl	= 0.80d0
nstp	= 600	nstp	= 800
nout	= 600	nout	= 800
ncyc	= 3	ncyc	= 3
isol	= 0	isol	= 0
idiss	= 0	idiss	= 1
ipnt	= 1	ipnt	= 1

Figure 5.35: Summary of solver control parameters for 10 degree cone.

Figure 5.13 presents a plot of the residual convergence history for the two solutions. The low-order dissipation solution required 600 iterations in order to achieve full numerical convergence, while the high-order dissipation solution required 800 iterations.

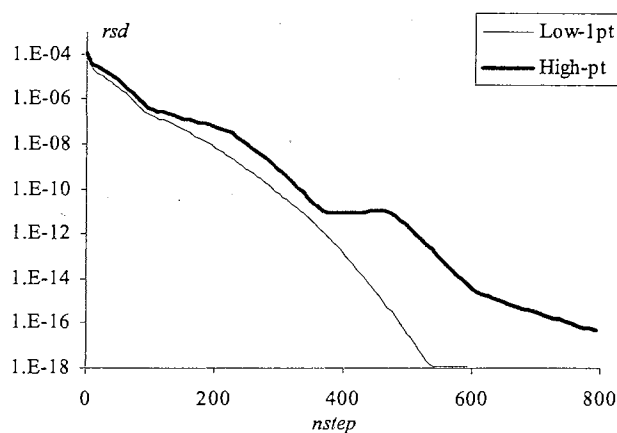


Figure 5.36: Plot of residual histories for 10 degree cone.

Results for this problem are taken along a cut-line across the length of the cone, which is surface 1 of our geometry. Figure 5.37 presents a comparison of the computed and theoretical distributions of the pressure and Mach number along the surface of the cone.

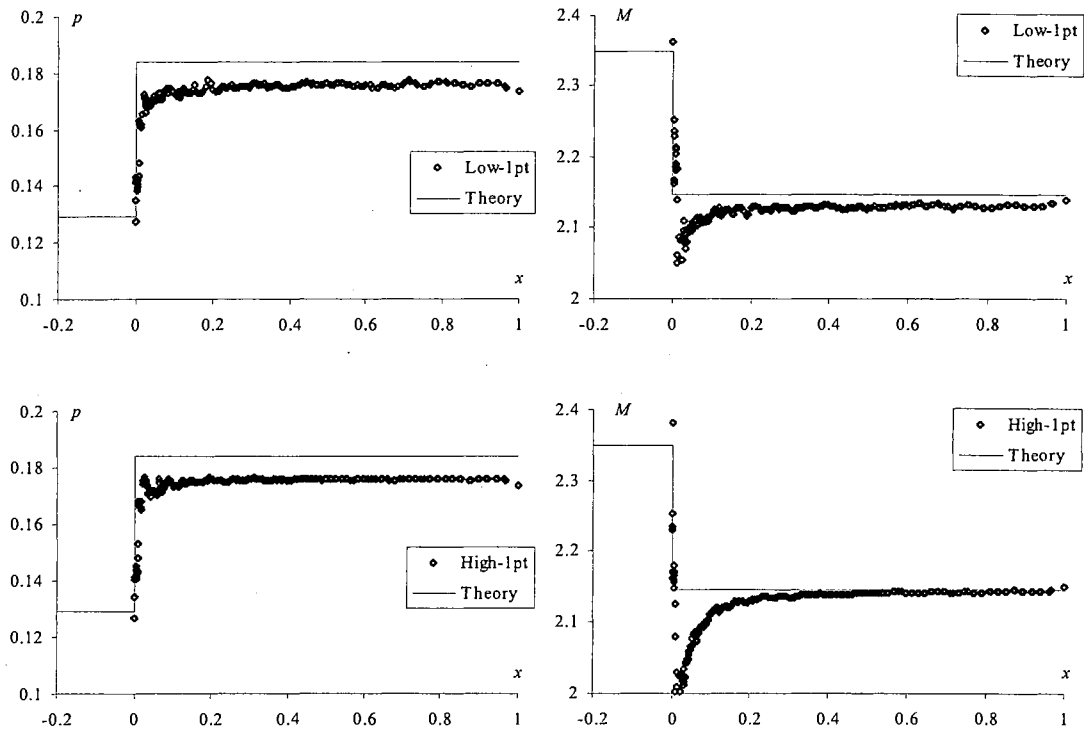


Figure 5.37: Pressure and Mach distributions on the surface of the 10 degree cone at Mach 2.35.

Both the low-order and high-order dissipation solutions show reasonable agreement with the theoretical values. The Mach number deviates significantly at the apex of the cone due its irregular surface triangulation, but asymptotically approaches the theoretical value aft of the apex. In order to quantify the accuracy of this solution, the average pressure and Mach number on the surface of the cone is computed using the integral definition for the average value of a function. Table 5.11 presents a summary of

the average pressure ratio and Mach number on the surface of the cone and the corresponding percent error for both solutions.

Table 5.11: Summary of average pressure and Mach number on the surface of the 10 degree cone at Mach 2.35.

	\bar{p}_2/p_1	\bar{M}	ε_p (%)	ε_M (%)
<i>Low-1pt</i>	0.1754	2.1321	4.747	0.688
<i>High-1pt</i>	0.1757	2.1421	4.547	0.212

Notice that the average Mach error is less than one percent, but the pressure error is nearly five percent. Despite the relatively large error in the predicted pressure on the surface of the cone, this solution is comparable to other CFD solutions for the same problem. The NPARC alliance reports an average pressure error of approximately 3.5% for the WIND codes, but an average Mach error of only 0.007%.³⁶

5.1.6 Unsteady Shock Tube

Unlike the problems considered thus far, this verification problem will investigate the time accuracy of our numerical algorithm for an unsteady flow problem. An unsteady shock tube consists of a high pressure and low pressure gas initially separated by a diaphragm in a tube. The solution starts when the diaphragm is removed, resulting in the formation of a normal shock that moves from the high pressure region into the low pressure region. The geometry consists of a simple rectangular tube, which covers the volume $x \in [-0.5, 0.5]$, $y \in [-0.1, 0.1]$ and $z \in [-0.02, 0.02]$. The layout of the computational domain is presented in Figure 5.38. Boundary conditions for the six

boundary surfaces enclosing this domain are specified as follows: surfaces 1 and 2 are symmetry planes and surfaces 3, 4, 5 and 6 are solid walls.

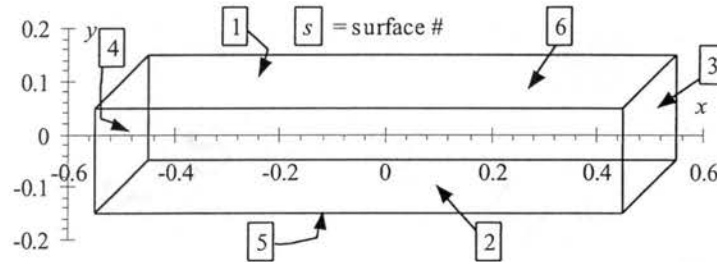


Figure 5.38: Layout of computational domain for unsteady shock tube.

The grid generated for this problem uses a constant grid spacing of 0.01 and consists of 48,334 elements and 11,239 nodes. Grid refinement is neglected for this problem in favor of investigating the effect of time step refinement for an unsteady solution. Figure 5.39 shows the surface triangulation used to represent the shock tube geometry.

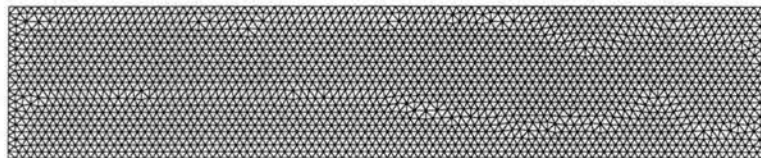


Figure 5.39: Surface triangulation for unsteady shock tube.

Two different types of solutions are computed for this problem in order to evaluate the performance of the first-order and second-order unsteady solutions. Both solutions will use the low-order dissipation model with one-point numerical integration. Figure 5.40 presents a summary of the control parameters used for the two solutions to this problem.

<i>1st-Order Unsteady</i>		<i>2nd-Order Unsteady</i>	
dt	= 0.001d0	dt	= 0.001d0
gamma	= 1.40d0	gamma	= 1.40d0
diss	= 0.50d0	diss	= 0.50d0
cfl	= 0.80d0	cfl	= 0.80d0
nstp	= 200	nstp	= 200
nout	= 200	nout	= 200
ncyc	= 20	ncyc	= 20
isol	= 1	isol	= 2
idiss	= 0	idiss	= 0
ipnt	= 1	ipnt	= 1
istrt	= .true.	istrt	= .true.

Figure 5.40: Summary of solver control parameters for unsteady shock tube.

The initial conditions for this problem must be generated manually. The dimensionless initial conditions for this problem are summarized in Figure 5.41. Every node in the computational domain is assigned an appropriate value for the solution unknowns based on its x -coordinate.

$x < 0.0$		$x > 0.0$	
$\rho = 1.0$	$u = 0.0$	$\rho = 0.125$	$u = 0.0$
$p = 1.0$	$v = 0.0$	$p = 0.1$	$v = 0.0$
$h = 3.5$	$w = 0.0$	$h = 2.8$	$w = 0.0$

Figure 5.41: Initial conditions for unsteady shock tube.

Our first two solutions use the exact set of control parameters given in Figure 5.40. The unsteady solutions restarts from the specified initial conditions and advances 200 steps using 20 iterative cycles per step. The large courant stability factor is appropriate here due to the relatively small global time step that was chosen for advancing the solution. Figure 5.42 presents a plot of the solution residuals for the two unsteady solutions. Notice that the solution residuals do not converge as far as seen for

the steady problems solved in the previous sections. For an unsteady solution, it is not practical to solve for that level of convergence because several hundred iterative cycles, or more depending on the size of the global time step, would be required for each step of the solution. Rather, the number of cycles is chosen such that the unsteady solution residual converge to some appropriate level at each step, typically around 1E-08 or 1E-09. In practice, there is no significant change in the solution at each step beyond this level of converge, so there is nothing to gain by increasing the number of cycles other than the satisfaction of having a super-converged solution.

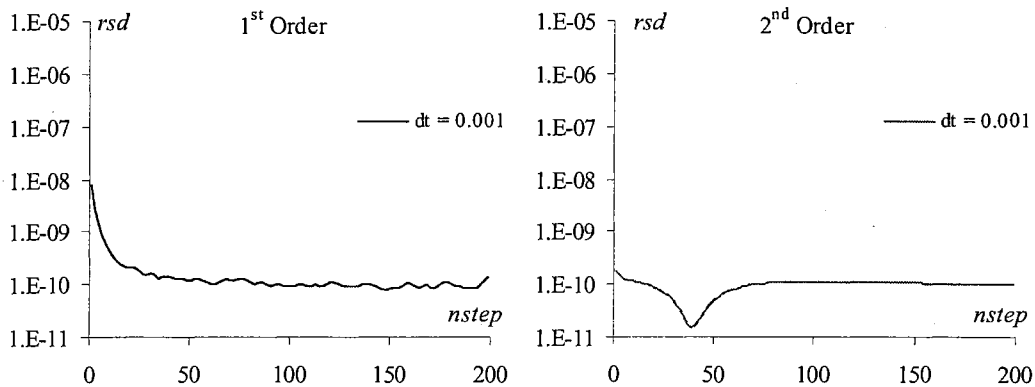


Figure 5.42: Plot of residual histories for 1st and 2nd order solution to the unsteady shock tube using a dimensionless time step of 0.001.

The final CFD results at $t^* = 0.2$ will be compared to the theoretical solution using the one-dimensional gas dynamics equations. Figure 5.43 presents a comparison of the computed density, pressure and Mach profile in the shock tube with the theoretical solution for each flow variable.

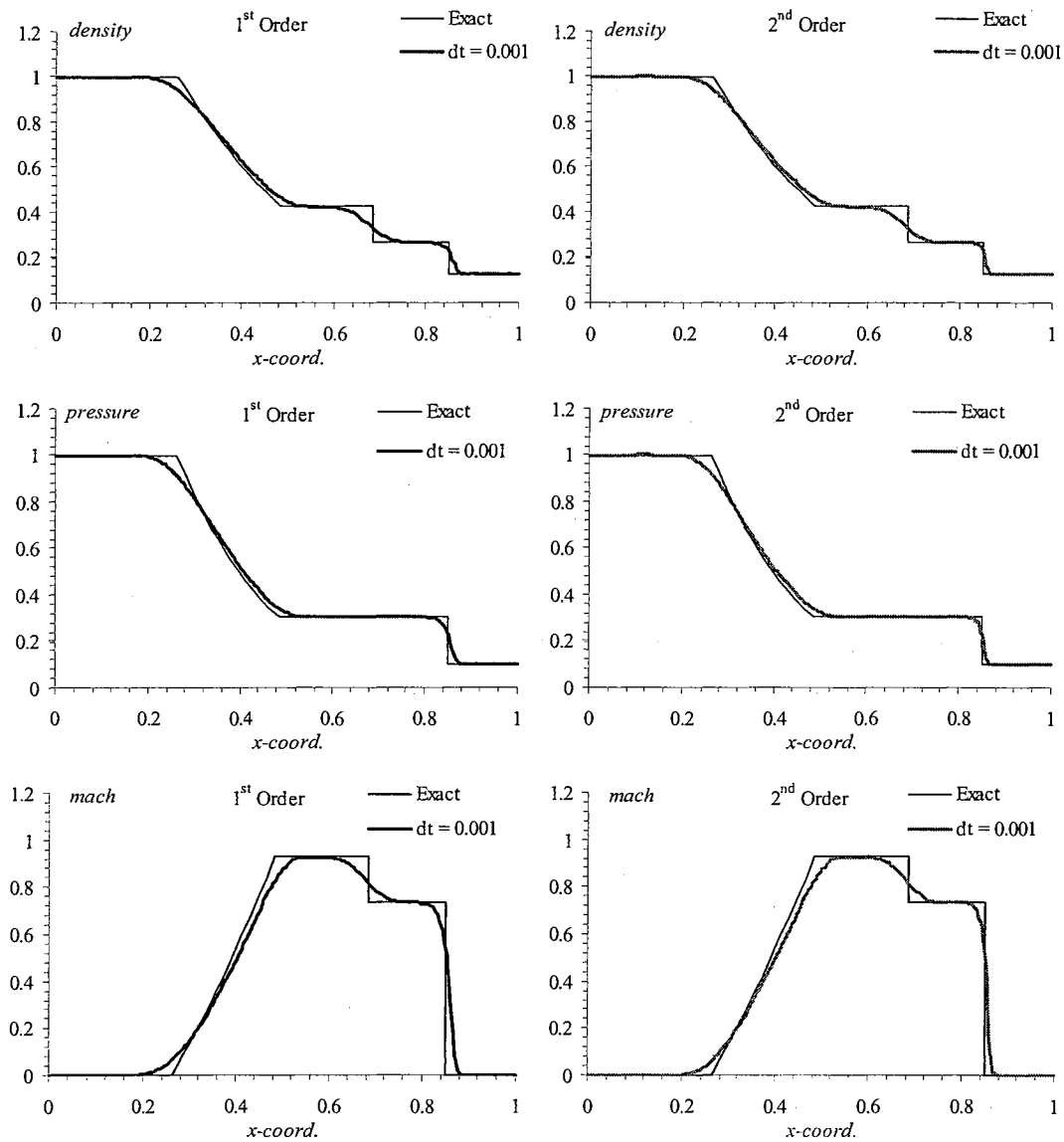


Figure 5.43: Density, pressure, and Mach distributions for 1st and 2nd order solution to the unsteady shock tube at $t^* = 0.2$ using a dimensionless time step of 0.001.

Notice that both the first-order and second-order unsteady solutions are in excellent agreement with the theoretical solution for this small dimensionless time step. Furthermore, the first-order and second-order solutions are virtually indistinguishable from each other. In order to investigate the convergence characteristics for an unsteady

solution, the first-order and second-order unsteady solutions are recomputed using successively larger time steps. As the time step increases, the total number of solution steps will decrease, but the number of iterative cycles required to achieve a reasonable level of numerical convergence for each step will typically increase. Furthermore, it is often necessary to decrease the courant stability factor for large global time steps in order to maintain the stability of the iterative algorithm. Table 5.12 presents a summary of the relevant control parameters used for the five time steps used in this study as well as the total computational time required for each solution. In each case, the number of cycles was chosen such that all five solutions reached a similar level of convergence after each step.

Table 5.12: Summary of solver control parameters used for time step refinement with unsteady shock tube.

dt	nstep	ncyc	cfl	CPU-time	
				1 st Order	2 nd Order
0.001	200	20	0.8	441.16 s	575.44 s
0.002	100	20	0.8	213.16 s	287.83 s
0.01	20	50	0.6	110.45 s	141.19 s
0.02	10	80	0.5	86.42 s	112.95 s
0.05	4	200	0.4	85.04 s	112.03 s

In general, the total computational time decreases as the global time step is increased since fewer number of steps are required to advance the solution to $t^* = 0.2$. However, there is obviously a point of diminishing return as the number of cycles increases dramatically for excessively large global time steps. Figure 5.44 presents a plot of the residual convergence histories for the first-order and second-order solutions using the five different time steps.

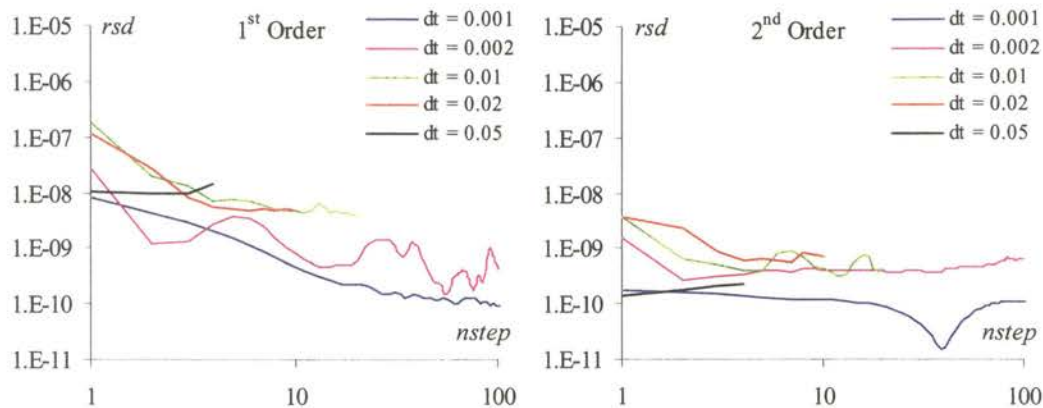


Figure 5.44: Plot of residual histories for 1st and 2nd order solution to the unsteady shock tube using various dimensionless time steps.

Notice that the second-order solution typically achieves a higher level of convergence at each step than the first-order solution for the same number of iterative cycles. Although the second-order solution suffers from longer computational times relative to the first-order solution, the higher-order accuracy seems to allow it to converge faster. Thus the computational penalty can be alleviated through the use of fewer iterative cycles with the second-order solution. In order to actually evaluate the performance of the two solutions, Figure 5.45 presents a comparison of the density, pressure and Mach distributions in the shock tube for the first-order and second-order unsteady solutions using the four new values of the global time step.

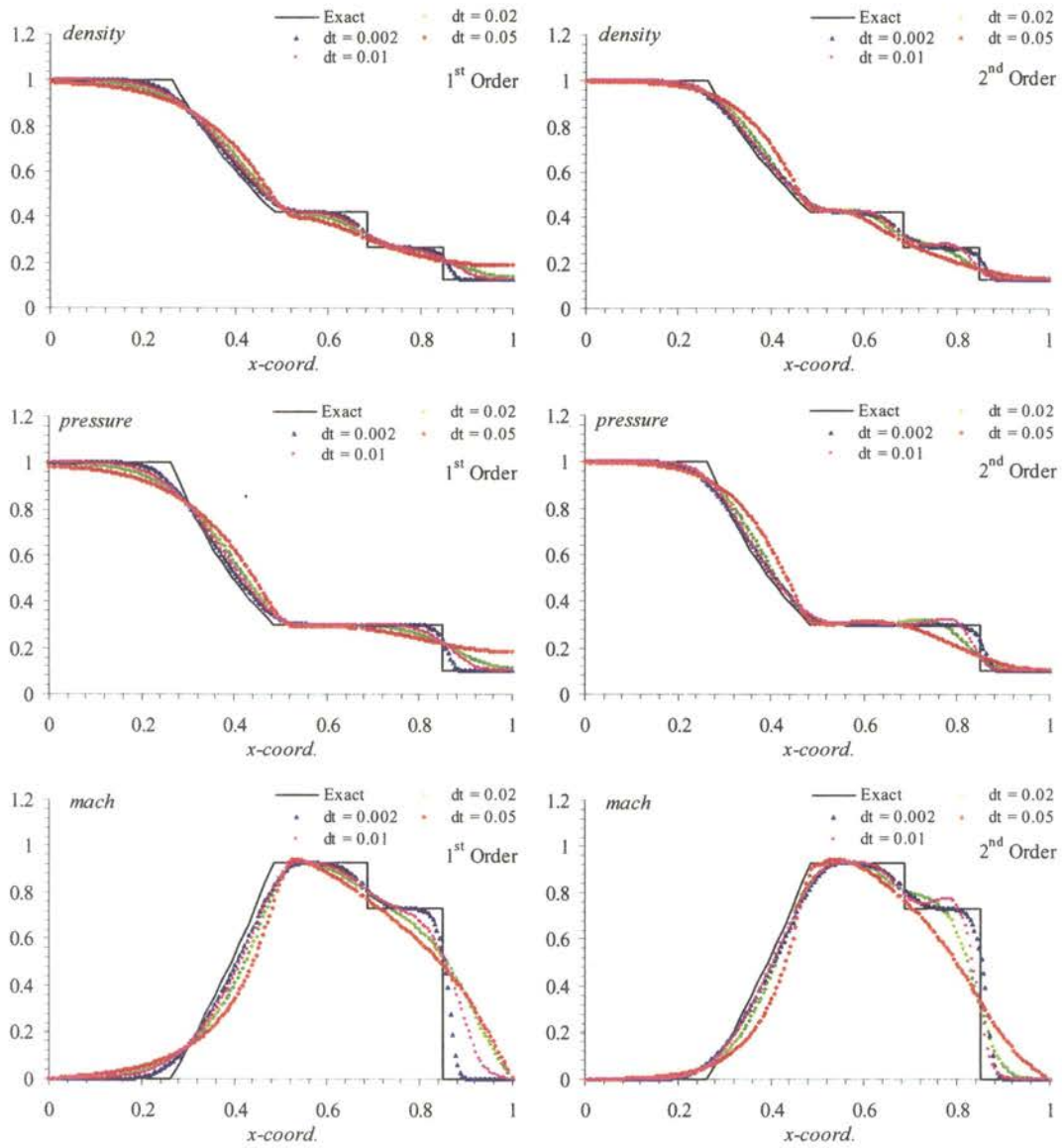


Figure 5.45: Density, pressure, and Mach distributions for 1st and 2nd order solution to the unsteady shock tube at $t^* = 0.2$ using various dimensionless time steps.

Notice that the solution accuracy decreases rapidly as the global time step increases. For the time step $\Delta t = 0.002$, which is double the size of our initial time step, both the first-order and second-order unsteady solutions are in excellent agreement with the theoretical density, pressure, and Mach distributions. However, the resolution of the

sharp shock and expansion regions computed using the first-order unsteady solution is degraded for the larger time steps. Use of the second-order unsteady solution does improve the time accuracy of the solutions with the larger time steps. In fact, the second-order unsteady solution is still in reasonable agreement with the theoretical distributions for $dt = 0.01$.

This problem clearly demonstrates the effect of increasing the global time step on the accuracy of an unsteady solution. The strong unsteady effects of a moving shock wave dominate the physics of this problem, making it very sensitive to the size of the global time step. In order to quantify this effect, we compare the velocity of the shock to the size of the global time step. At $t^* = 0.2$, the shock is located at $x^* = 0.850431$, or it has traveled a total distance of $\Delta x^* = 0.350431$. This equates to an average velocity of $u^* = 1.75$ for the shock during the total time interval of the solution. The computational grid for this problem was generated using a uniform grid spacing of 0.01, which means the shock moves across an average of one element in about $\Delta t^* = 0.006$. Based on the solution results presented in Figure 5.45, the best accuracy for this problem is achieved when the global time step is smaller than the time it takes the shock to move across one element.

Fortunately, other unsteady problems will not typically be this sensitive to the global time step. This will allow the use of larger global time steps while still maintaining the time accuracy of the solution. However, it is necessary to consider the physics of each problem when selecting the global time step for an unsteady solution. Otherwise the resulting solution might be inaccurate as demonstrated by the results of this problem.

5.1.7 Impulsively Accelerated Airfoil

The unsteady variation of lift coefficient for an impulsively accelerated airfoil is one of the most basic examples of unsteady aerodynamics. For this problem, the NACA 0012 airfoil geometry presented in Section 5.1.4 is suddenly accelerated to a constant velocity corresponding to Mach 0.3 at an angle of attack of 5.0 degrees. The results will be compared to the theoretical solution solved by Wagner^{37,38} for the infinite acceleration of a two-dimensional, thin airfoil.

For this problem, the transient response of the airfoil is sensitive to the unsteady wake that develops downstream. Therefore, the grids generated for the NACA 0012 airfoil in Section 5.1.4 are locally refined in the region downstream of the airfoil. Grid convergence will not be attempted for this unsteady problem, so only one computational grid is required. Figure 5.46 shows the overall surface triangulation generated for this study, and Figure 5.47 shows a close-up of the surface triangulation on the surface of the airfoil.

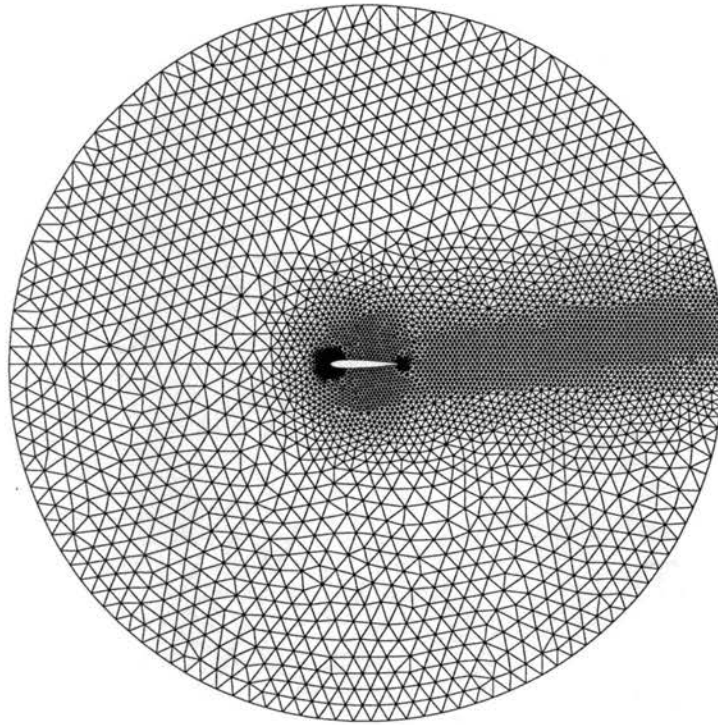


Figure 5.46: Surface triangulation with wake refinement for impulsively accelerated airfoil.

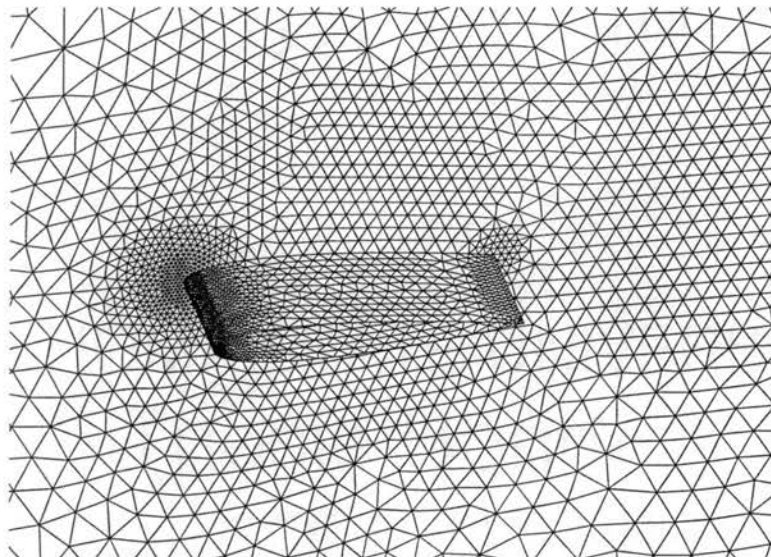


Figure 5.47: Close-up of surface triangulation for impulsively accelerated airfoil.

As with the unsteady shock tube, this problem is sensitive to our choice of global time step. The theoretical solution involves an infinite acceleration at $t^* = 0.0$. However, our numerical solution to this problem uses finite time steps, which results in a finite acceleration rate. This finite acceleration rate has been demonstrated to moderately increase the lift.³⁸ Furthermore, the large initial acceleration will cause some stability and convergence problems for the initial solution steps. This will require the use of low values for the courant stability factor and extra iterative cycles for convergence.

Figure 5.48 presents the solver control parameters chosen for the first solution to the impulsively accelerated airfoil. Since our reference dimension is equal to the chord of the airfoil, the dimensionless time step measures the distance traveled by the airfoil in chord lengths for each global time step.

dt	= 0.050d0
gamma	= 1.40d0
mach	= 0.30d0
alpha	= 5.0d0
refdim	= 16.0d0
diss	= 1.00d0
cfl	= 0.60d0
nstp	= 140
nout	= 20
ncyc	= 50
isol	= 2
idiss	= 1
ipnt	= 1
istrt	= .false.
iaero	= .true.

Figure 5.48: Summary of solver control parameters for impulsively accelerated airfoil.

As with the previous unsteady problem, the effects of time step refinement are investigated here using several solutions with varying time steps. Table 5.13 presents the combinations of time step size, solution steps, iterative cycles, and courant stability factor

used for the four unsteady solutions. In each case, the number of cycles was chosen so that the solution for each time step attained a similar level of convergence.

Table 5.13: Summary of solver control parameters used for time step refinement with the impulsively accelerated airfoil.

Δt	nstep	ncyc	cfl	CPU-time
0.050	140	50	0.5	6612.24 s
0.100	70	70	0.5	4638.86 s
0.200	35	100	0.4	3305.47 s
0.500	14	140	0.4	1892.38 s

Figure 5.49 presents a plot of the residual convergence histories for the unsteady solutions using the four different time steps. As expected, the level of convergence for the initial time steps is relatively low due to the large acceleration at $t^* = 0.0$.

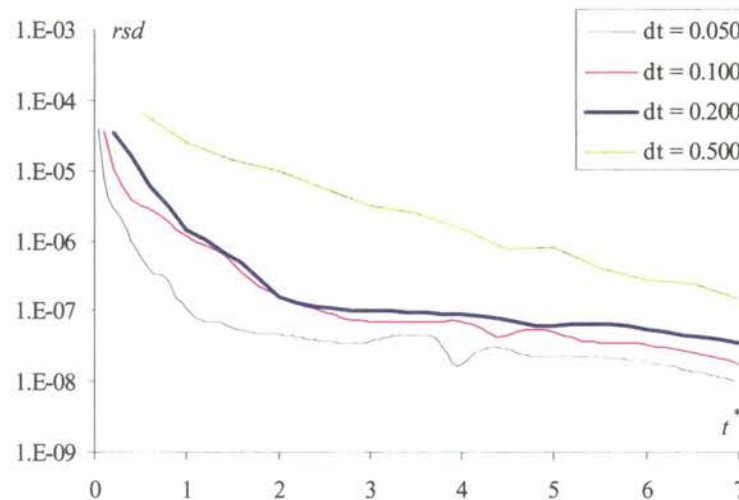


Figure 5.49: Plot of residual histories for impulsively accelerated airfoil using various dimensionless time steps.

The transient response of the airfoil for each solution is plotted as $C_L(t)/C_L$, where $C_L(t)$ is the instantaneous lift coefficient computed for the airfoil at time t , and C_L is the

lift coefficient of the airfoil at infinity or the steady state lift coefficient. The steady state lift coefficient was computed directly using a steady CFD solution for this airfoil. Figure 5.50 presents a comparison of the transient lift coefficient computed using the four different dimensionless time steps and the theoretical Wagner solution.

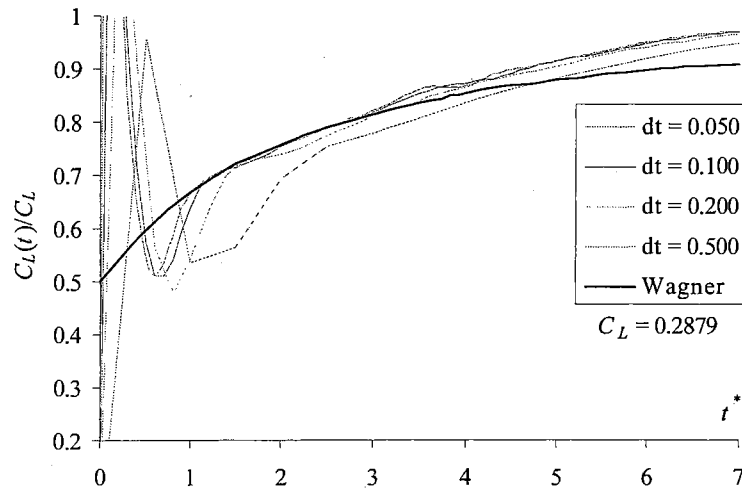


Figure 5.50: Plot of lift coefficient variation for the impulsively accelerated airfoil using various dimensionless time steps.

The results in Figure 5.50 show reasonable agreement with the theoretical solution. As expected, there is a large discrepancy in the initial transient resulting from the sudden acceleration of the airfoil. This is in part due to our discretization of the problem, but it is also affected by the applied initial condition for the solution, which is simply free-stream velocity and pressure throughout the domain. The numerical solutions do asymptotically approach the steady state lift coefficient, but do so more rapidly than the theoretical response curve predicts. This discrepancy between the numerical and theoretical solutions is most likely caused by the cumulative errors from an imperfect initial condition, our spatial discretization, and compressibility. The effects

of compressibility are small at this low Mach number. Nevertheless, compressibility has the effect of stretching the coordinate system and would diminish the impact of the unsteady wake more rapidly, which is consistent with the results presented in Figure 5.50.

In order to further visualize the unsteady behavior of this problem, Figure 5.51 shows the colored Mach profiles at six sequential times in the solution.

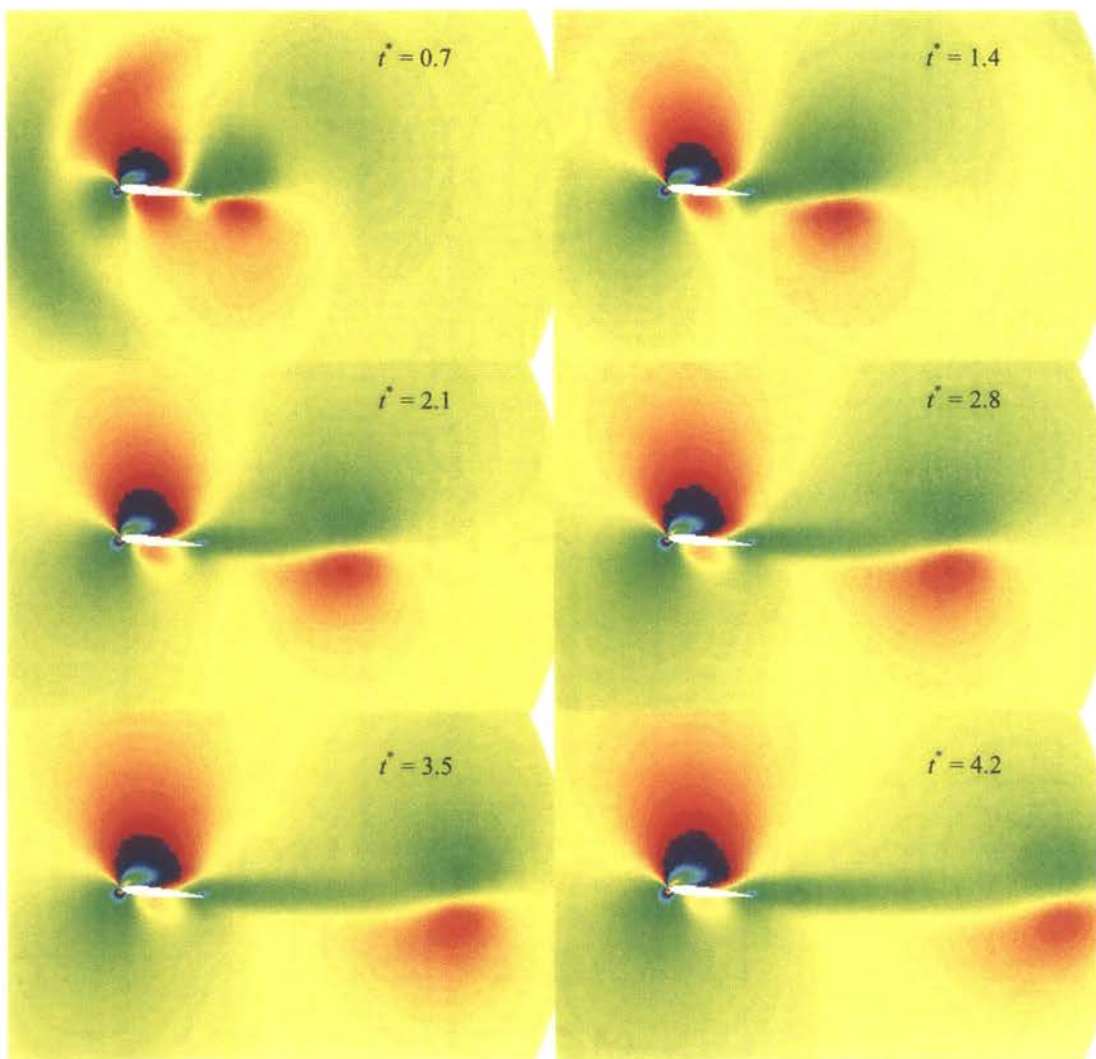


Figure 5.51: Colored Mach profile showing unsteady wake development for the impulsively accelerated airfoil.

The unsteady wake development is clearly visible in the six pictures of Figure 5.51. As expected, the wake reaches the far-field boundary at approximately $t^* = 5.0$, since the far-field boundary is exactly five chord lengths from the trailing edge of the airfoil.

As a final verification using this geometry, the same impulsively accelerated airfoil is also solved in a non-inertial formulation. This is accomplished by turning off the free-stream velocity and applying a dynamic initial condition to the non-inertial frame that corresponds to the identical conditions used in the inertial solution. Figure 5.52 presents the solver control parameters for the non-inertial solution, while Figure 5.53 presents the dynamics file with the correct initial conditions for the non-inertial frame.

```
dt          = 0.100d0
gamma       = 1.40d0
mach        = 0.30d0
alpha       = 0.0d0
refdim      = 16.0d0
diss        = 1.00d0
cfl         = 0.50d0
nstp        = 70
nout        = 70
ncyc        = 70
isol        = 2
idiss       = 1
ipnt        = 1
istrt       = .false.
iaero       = .true.
idynm       = .true.
ifree       = .false.
ainf        = 3.333333d0
```

Figure 5.52: Summary of solver control parameters for impulsively accelerated airfoil in a non-inertial frame.

```

$ 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 accels )
0.0d0, 0.0d0, 0.0d0, 0.0d0, 5.0d0, 0.0d0
-1.0d0, 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 (6)
2, 1, 1, 1, 1, 1

```

Figure 5.53: Dynamics file for the impulsively accelerated airfoil in a non-inertial frame.

Notice that the initial velocity vector for the origin of the non-inertial frame is specified as $\{-1.0, 0.0, 0.0\}^T$. The translational velocity and acceleration of the non-inertial frame is always expressed relative to the inertial coordinate system. Furthermore, the specified velocity for the non-inertial frame is dimensionless since the reference velocity, $U_0 = \text{mach} \cdot a_{\text{inf}}$, is identically one. For this problem, the non-inertial frame is also oriented at a 5.0 degree pitch angle with respect to the inertial frame so that we account for the angle of attack of the airfoil.

The non-inertial aspects of this problem are trivial since both the acceleration and the angular rates are all zero. Hence, the inertial and non-inertial solutions are expected to be identical. Figure 5.54 presents a comparison of the transient lift coefficient

computed using the inertial and non-inertial formulations with the theoretical Wagner solution.

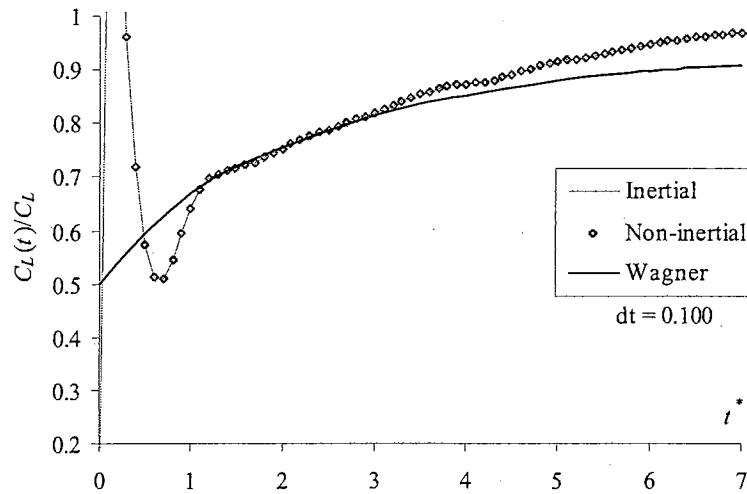


Figure 5.54: Comparison of transient lift coefficient for the impulsively accelerated airfoil solved using inertial and non-inertial formulations.

Notice that the inertial and non-inertial solutions are identical. This solution verifies the non-inertial implementation for the simplest case of a solution with constant velocity and orientation.

5.1.8 Pitching and Plunging Airfoil

For this problem, we use the NACA 0012 airfoil of Section 5.1.4 again to study the unsteady variation of lift coefficient generated by a sinusoidal pitching and plunging motion. The unsteady, harmonic motion of the airfoil is generated by dynamically moving the computational domain in a non-inertial solution. The results will be compared to the theoretical solution solved by Theodorsen's method³⁹ for a thin airfoil undergoing harmonic pitching and plunging motion in a uniform, incompressible flow.

As with the problem of Section 5.1.7, the solution for this problem is sensitive to the unsteady wake that develops downstream. Therefore, a grid similar to the one used in Section 5.1.7 with local refinement in the region downstream of the airfoil is again employed. The only grid used for this problem consists of 41,400 nodes and 182,076 elements. Grid convergence will not be investigated for this unsteady problem.

The first unsteady solution involves a dynamically plunging airfoil. A sinusoidal plunging motion is generated by specifying an appropriate mass and stiffness for the z -axis in the dynamics input file and clamping the other five degrees of freedom. If the aerodynamic forces are turned off in the dynamics solver and no structural damping is specified, the airfoil will oscillate at its natural frequency with a plunge amplitude equal to whatever displacement is specified in the initial conditions. Figure 5.55 presents the solver control parameters chosen for the first solution to the plunging airfoil, while Figure 5.56 presents the dynamics input file utilized for the same solution.

dt	=	0.050d0
gamma	=	1.40d0
mach	=	0.30d0
alpha	=	0.0d0
refdim	=	16.0d0
diss	=	1.00d0
cfl	=	0.60d0
nstp	=	500
nout	=	50
ncyc	=	40
isol	=	2
idiss	=	1
ipnt	=	1
istrt	=	.true.
iaero	=	.false.
idynm	=	.true.

Figure 5.55: Summary of solver control parameters for the plunging airfoil.

```

$ Position vector to origin of non-inertial frame (rx, ry, rz)
8.0d0, 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 1200.0d0    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)
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
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
$ 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    0.4d0    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 accels )
0.0d0, 0.0d0, 3.2d0, 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 (6)
1,      1,      0,      1,      1,      1

```

Figure 5.56: Dynamics input file for the plunging airfoil.

The combination of parameters given in Figure 5.55 and Figure 5.56 define a plunging airfoil with a reduced natural frequency approximately equal to 0.487, where the reduced natural frequency, k , is defined by Equation (5.7).

$$(5.7) \quad k = \frac{\omega c}{2U_\infty}$$

Figure 5.57 presents the plunge response time history computed for this combination of physical parameters.

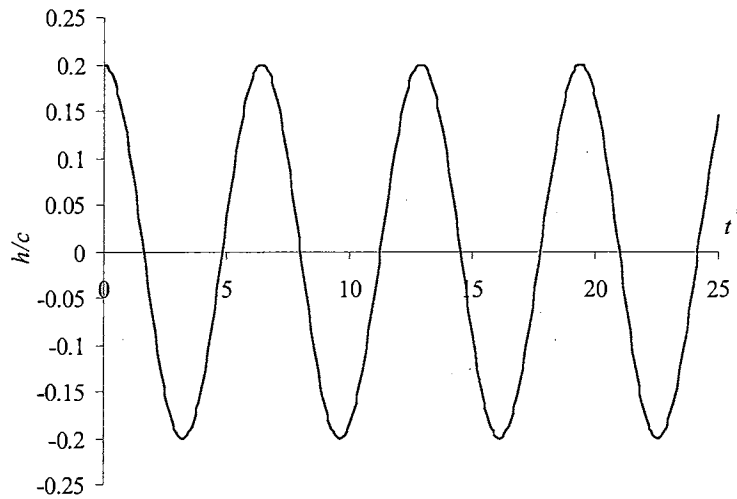


Figure 5.57: Response time history for the plunging airfoil.

In addition to the time step presented in Figure 5.55, a second solution using a larger time step of 0.20 combined with 80 iterative cycles was also executed. A plot of the residual time histories for each of the solutions is presented in Figure 5.58, while a comparison of the theoretical and predicted lift coefficient time histories for the same solutions is presented in Figure 5.59.

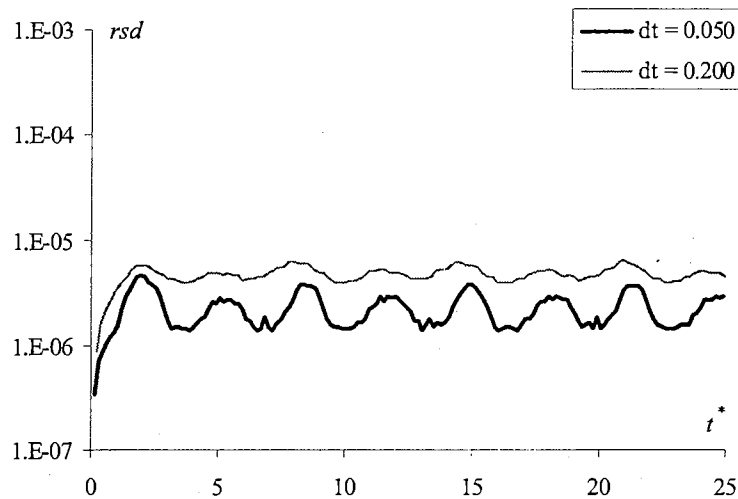


Figure 5.58: Plot of residual time histories for the plunging airfoil solution.

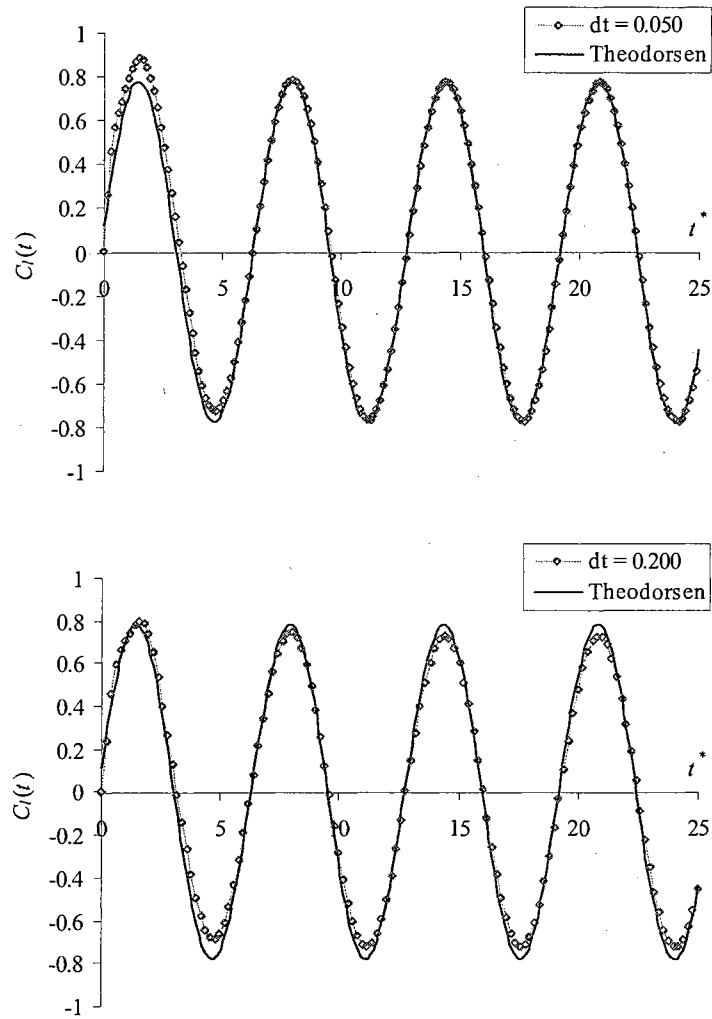


Figure 5.59: Comparison of lift coefficient time histories for the plunging airfoil.

Notice that the solution with the smaller time step, $dt = 0.05$, is in excellent agreement with the theoretical solution computed using Theodorsen's method. As expected, the discrepancy visible in this plot for the first cycle of oscillation vanishes as a proper wake develops downstream. Even the solution using the larger time step, $dt = 0.20$, shows reasonable agreement with the theoretical solution, but falls short of capturing the proper amplitude. To quantify this comparison, the amplitude of the lift

coefficient time history predicted using a time step of 0.05 was approximately 0.774. This differs from the theoretical value of approximately 0.779 by only 0.64%.

For our next solution, we switch to a pitching airfoil with the same basic physical parameters used for the plunging airfoil. The solver control parameters will remain unchanged from the previous solution, while Figure 5.60 presents the new dynamics input file for the pitching airfoil solution. Notice that the airfoil is setup to pitch about a non-inertial y -axis located at the midpoint of the airfoil, since the origin of the coordinate system was originally setup at the leading edge of the airfoil.

```

$ Position vector to origin of non-inertial frame (rx, ry, rz)
8.0d0, 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 1200.0d0    0.0d0
0.0d0    0.0d0    0.0d0    0.0d0    0.0d0    1.0d0
$ Damping matrix for non-inertial frame (6 x 6)
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
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
$ 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    0.4d0    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 accels )
0.0d0, 0.0d0, 0.0d0, 0.0d0, 1.0d0, 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 (6)
1,      1,      1,      1,      0,      1

```

Figure 5.60: Dynamics input file for the pitching airfoil.

Using the same time steps as the previous solutions, a plot of the residual time histories is presented in Figure 5.61, while a comparison of the theoretical and predicted lift coefficient time histories is presented in Figure 5.62.

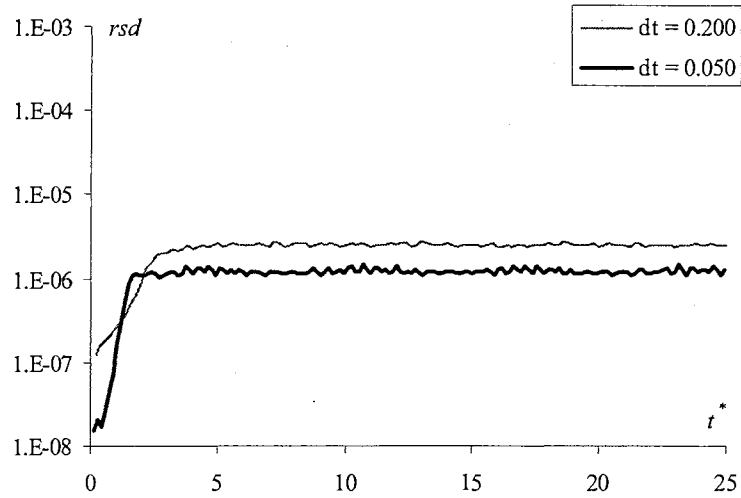


Figure 5.61: Plot of residual time histories for the pitching airfoil solution.

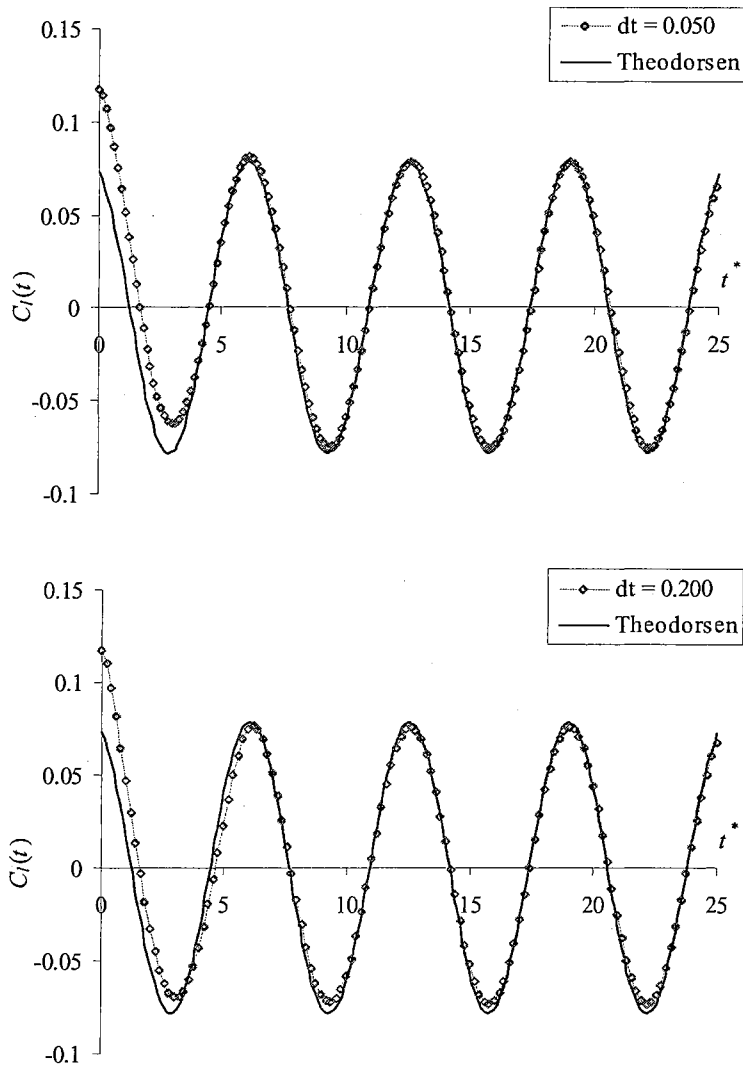


Figure 5.62: Comparison of lift coefficient time histories for the pitching airfoil.

Once again, the solution for a time step of 0.05 is in excellent agreement with the theoretical solution computed using Theodorsen's method. For this problem, notice that the wake effect has a more pronounced influence on the amplitude of the dynamic lift coefficient. The initial conditions for this solution at $t^* = 0$ were obtained by solving a steady solution for this airfoil pitched at a 1° constant angle of attack, which predicted a steady state value of approximately 0.117 for the lift coefficient. For the pitching airfoil,

the predicted amplitude of the dynamic lift coefficient is approximately 0.0780, or a decrease of approximately 33% from the steady state lift coefficient. The theoretical amplitude for the dynamic lift coefficient is approximately 0.0782, which means the solution found here is within 0.32% of the expected value.

5.1.9 Spinning Centrifuge

The problems of the previous section demonstrated the capabilities of the non-inertial algorithm for relatively small amplitude harmonic motion. This verification problem investigates the accuracy of the non-inertial algorithm for a rapidly spinning geometry that undergoes several complete revolutions. The shock tube geometry of Section 5.1.6 is rotated about its z -axis in order to simulate a spinning centrifuge. It is expected that the pressure distribution within the tube will follow an exponential profile as defined by Equation (5.8).

$$(5.8) \quad \frac{P_2}{P_1} = e^{\frac{1}{2}\omega^2 r^2 / RT}$$

Unfortunately, simply starting the tube rotating with an initially uniform pressure distribution inside will result in a solution where pressure waves oscillate back and forth along the length of the tube. It is expected that these pressure waves would eventually damp out under the influence of the artificial viscosity in the algorithm, but this may take several thousand revolutions. Instead, we will start the solution with proper initial conditions, which represent the correct theoretical solution for the spinning tube. Using Equation (5.8), a proper set of initial conditions is generated for each node of the

computational domain using the following set of physical constants: $p_1 = 101.33$ kPa, $\rho_1 = 1.225$ kg/m³, $R = 286.9$ J/kg·K, $T = 288.15$ K, and $\omega = 91.8792$ rad/s or 5264.29deg/s.

Figure 5.63 and Figure 5.64 present the solver control parameters and dynamics input file used to analyze this problem.

dt	= 0.005d0
gamma	= 1.40d0
mach	= 0.30d0
diss	= 1.00d0
cfl	= 0.50d0
nstp	= 5000
nout	= 100
ncyc	= 20
isol	= 2
idiss	= 1
ipnt	= 1
istrt	= .true.
iaero	= .false.
idynm	= .true.
ifree	= .false.
ainf	= 204.176d0
rhoinf	= 1.225d0

Figure 5.63: Summary of solver control parameters for spinning centrifuge.

```

$ 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 accels )
0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0
0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0, 5264.29d0
0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0
$ IBXD for non-inertial frame (6)
1, 1, 1, 1, 1, 2

```

Figure 5.64: Dynamics file for the spinning centrifuge.

The dimensionless angular rate specified for this problem is computed to be 1.5. Given that the time step chosen for the solution is 0.005, the tube will complete one revolution every 838 solution steps. Figure 5.65 presents a comparison of the computed and theoretical pressure and density distribution within the tube after 5000 steps or approximately 6 revolutions. The results clearly demonstrates the accuracy of the non-inertial algorithm for the extreme case of a spinning body that undergoes multiple revolutions.

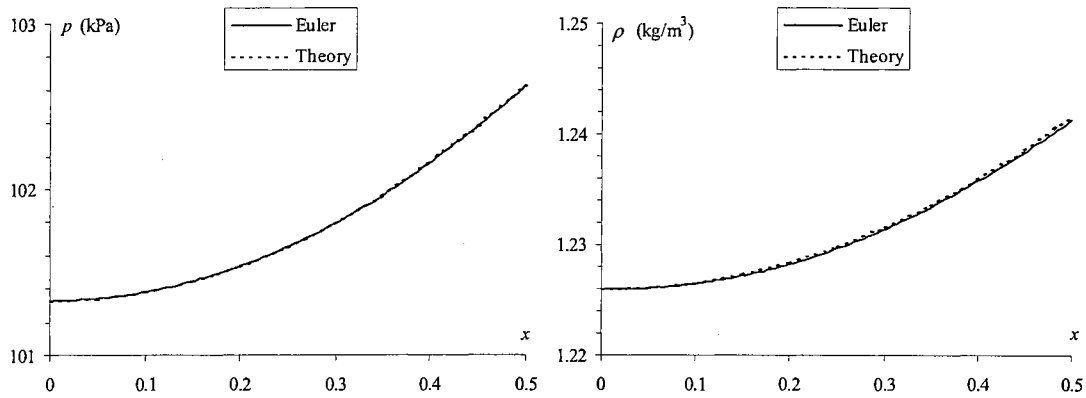


Figure 5.65: Comparison of computed and theoretical pressure and density distributions along the centerline of the spinning centrifuge after four revolutions.

5.2 Validation

Having completed numerous verification tests for both steady and unsteady problems, our focus shifts to a validation of our numerical algorithm for practical aerospace applications. In contrast to verification where we attempted to demonstrate that the algorithm is solving the equations right, validation is typically referred to as solving the right equations.¹⁰ We have adopted that definition here and will attempt to demonstrate that our governing equations, the compressible Euler equations, are capable of accurately representing the dominant flow physics for relevant aerospace applications. In addition, we must demonstrate that the assumptions of our numerical approximations, specifically the transpiration method, are accurate and applicable to the types of problems we are solving.

Our primary comparison for validation solutions is experimental data. Often this is a somewhat difficult comparison to make because there are just as many difficulties

and inaccuracies associated with experiments as there are with numerical models. Although many modern experiments typically take advantage of non-intrusive measurement techniques, much of the experimental data available for our selected aerospace applications is legacy data obtained before such techniques were widely in use and or even available. Hence, it is necessary to be cautious in our evaluation of the numerical results and take in to consideration any experimental uncertainties if such information is even available. Roache¹⁰ borrowed an interesting quote from Aeschliman and Oberkampf who said, “the general point is that as one progresses down the list to more difficult quantities for CFD to predict, the experimental uncertainty generally increases also.” This proves particularly true in our case for aeroelastic applications where the combination of uncertainties in both the structural and fluid dynamics data can couple adversely to magnify the differences between experiment and numerical prediction.

The sections that follow present a sample of validation problems intended to demonstrate the types of aerospace applications our numerical algorithm is capable of solving. Our primary interest is in the modeling of transonic flow problems since good theoretical models are not readily available for advanced three-dimensional applications in this flow regime. As such, we begin with a steady validation case that demonstrates the transonic shock capturing capabilities of the solver. This leads in to several unsteady validation tests for aeroelastic problems in the transonic flow regime, as well as problems that exercise the non-inertial terms in our formulation since that was the primary goal of this research.

5.2.1 RAE 2822 Transonic Airfoil

This steady problem consists of a RAE 2822 airfoil in a transonic flow. The results will be compared to an experimental solution obtained for Mach 0.729 and an angle of attack of 2.31 degrees.⁴⁰ The layout of the computational domain, which is similar to that of the NACA 0012 airfoil from Section 5.1.4, is presented in Figure 5.66. Boundary conditions for the eight boundary surfaces enclosing this domain are specified as follows: surfaces 1, 2, 3 and 4 are symmetry planes, surfaces 5 and 6 are far-field boundaries, and surfaces 7 and 8 are solid walls. The curve defining the trailing edge of the airfoil is also specified as singular since the local surface normals are undefined along that curve.

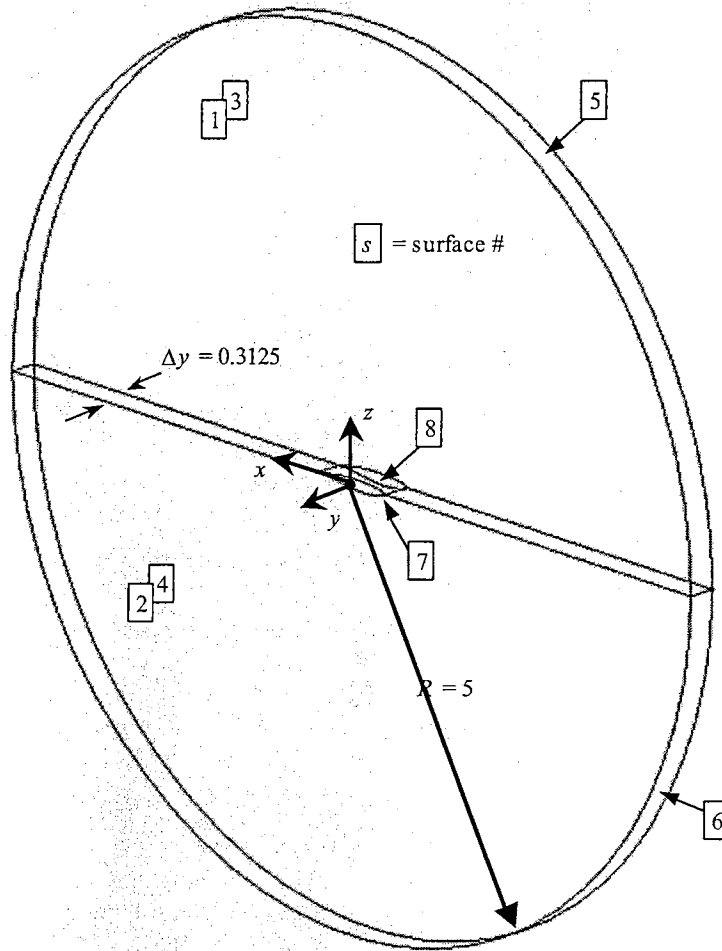


Figure 5.66: Layout of computational domain for the RAE 2822 transonic airfoil.

The computational grid generated for this problem consists of 198,528 elements and 38,571 nodes, with local refinement of the elements near the surface of the airfoil. Figure 5.67 shows a close-up of the surface triangulation near the airfoil.

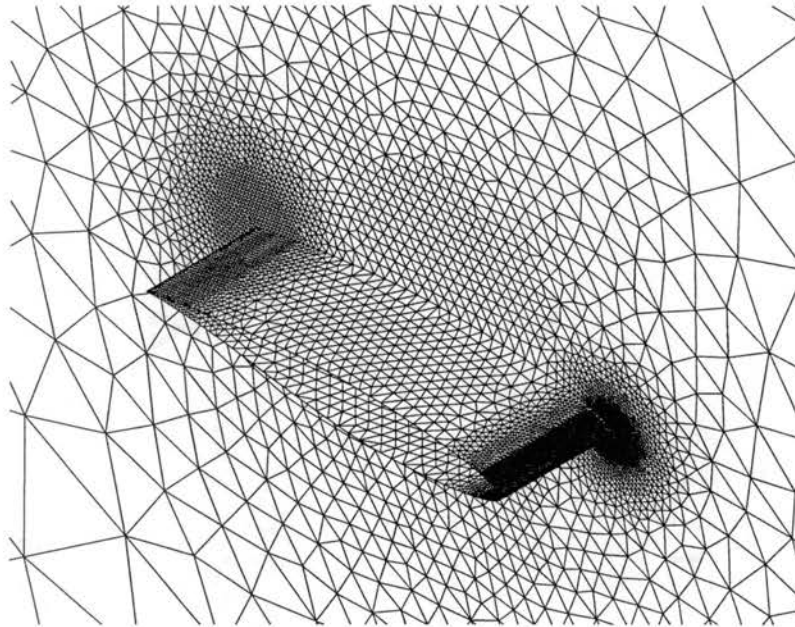


Figure 5.67: Close-up of surface grid near the RAE-2822 airfoil.

Our solution to this problem uses the high-order dissipation model with a dissipation factor of 1.0. The complete set of solver control parameters used in the solution of this problem is given in Figure 5.68.

gamma	= 1.40d0
mach	= 0.729d0
alpha	= 2.31d0
diss	= 1.00d0
cfl	= 0.80d0
nstp	= 8000
nout	= 8000
ncyc	= 3
isol	= 0
idiss	= 1
ipnt	= 1
iaero	= .true.

Figure 5.68: Summary of solver control parameters for transonic airfoil solution.

Figure 5.13 presents a plot of the residual convergence history for the solution. A total of 8000 iterations were required in order to attain a reasonable degree of numerical convergence.

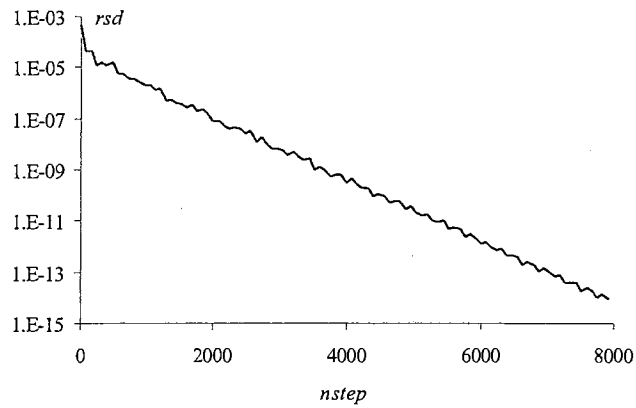


Figure 5.69: Plot of residual histories for transonic airfoil solution.

Results for this problem are taken along a cut-line defining the intersection of an xz -plane at $y = -0.15625$ and the two surfaces defining the airfoil, surfaces 7 and 8. Figure 5.70 presents a comparison of the computed pressure coefficient on the surface of the airfoil and the experimental data for Mach 0.729. Notice that the computed results are in excellent agreement with the experimental data, including the predicted location of the shock.

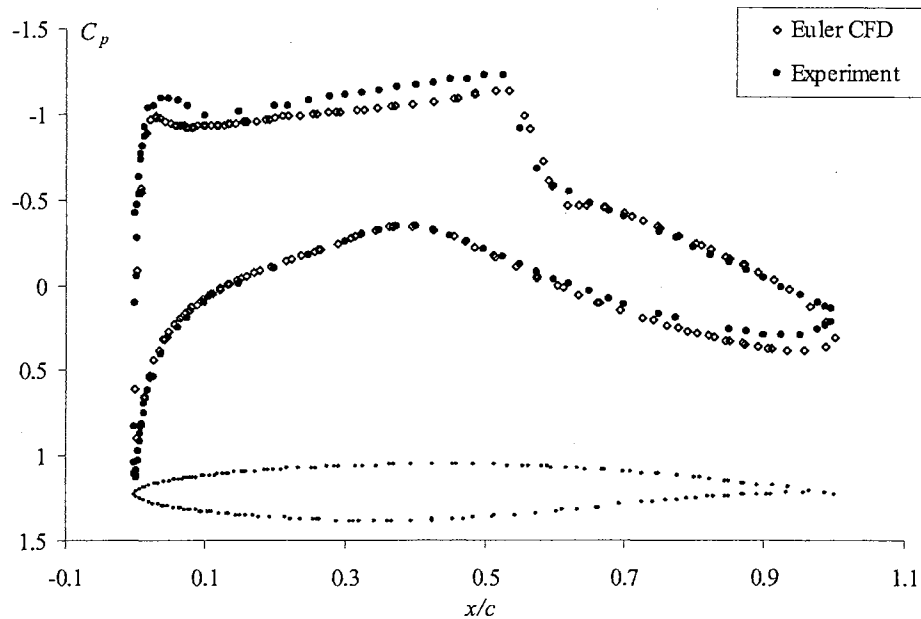


Figure 5.70: Plot of pressure coefficient distribution on the surface of the transonic airfoil at Mach 0.729 and angle of attack 2.31 degrees.

5.2.2 AGARD 445.6 Aeroelastic Test Wing

Having validated the CFD solver for steady transonic problems, we will now investigate an unsteady transonic problem. The AGARD 445.6 wing configuration is an aeroelastic test case that was investigated experimentally in the 16-foot Transonic Dynamics Tunnel (TDT) at NASA Langley Research Center.⁴¹ Experimental flutter boundaries are reported for transonic Mach numbers ranging from 0.499 up to 1.141. The wing is made from an NACA 65A004 airfoil cross section with a chord of 1.833 feet, a semi-span of 2.5 feet, a quarter-chord sweep angle of 45 degrees, a panel aspect ratio of 1.65, and a taper ratio of 0.66.

This particular wing configuration was analyzed previously using the STARS unsteady CFD solver.⁵ For this validation problem, we will simply re-use the original elastic CFD model generated for the previous STARS analysis of this wing geometry. This will allow us to compare our results to the previous STARS solution and validate our implementation of the transpiration boundary condition and elastic structural dynamics solver for a three-dimensional aeroelastic solution. The structural model consists of two elastic mode shapes that represent the first bending and torsion modes of the wing. The frequencies for these two modes are 9.60 and 38.20 Hz respectively. The CFD model consists of 69,630 nodes and 373,798 elements. Figure 5.71 shows a close-up of the surface triangulation on the wing. The wing is attached to the center of a rectangular computational domain that measures approximately 32 feet long, 32 feet high, and 16 feet wide.

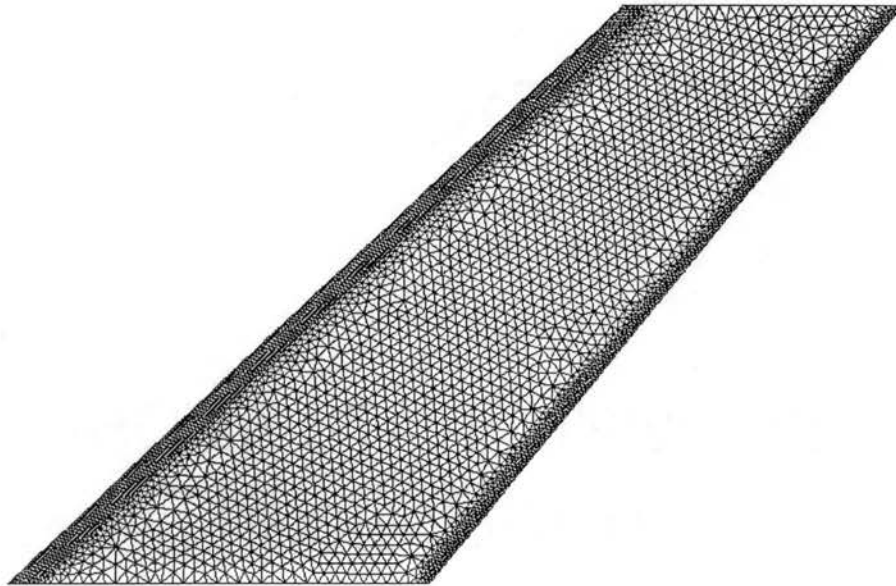


Figure 5.71: Close-up of surface grid for AGARD 445.6 aeroelastic test wing.

The first step in an aeroelastic analysis is to generate proper initial conditions for the problem by completing a steady solution for the CFD model. Figure 5.72 presents a summary of the solver control parameters used for the Mach 0.96 steady solution for this problem.

gamma	= 1.40d0
mach	= 0.96d0
diss	= 1.00d0
cfl	= 0.80d0
nstp	= 4000
nout	= 4000
ncyc	= 3
isol	= 0
idiss	= 1
ipnt	= 1

Figure 5.72: Summary of solver control parameters for the AGARD steady solution.

Figure 5.73 presents plots of the steady pressure coefficient distribution along the surface of the AGARD wing at Mach 0.96. Although there were no steady experimental results published for this wing geometry, the results presented here are consistent with the steady computational results published by Batina.⁴¹

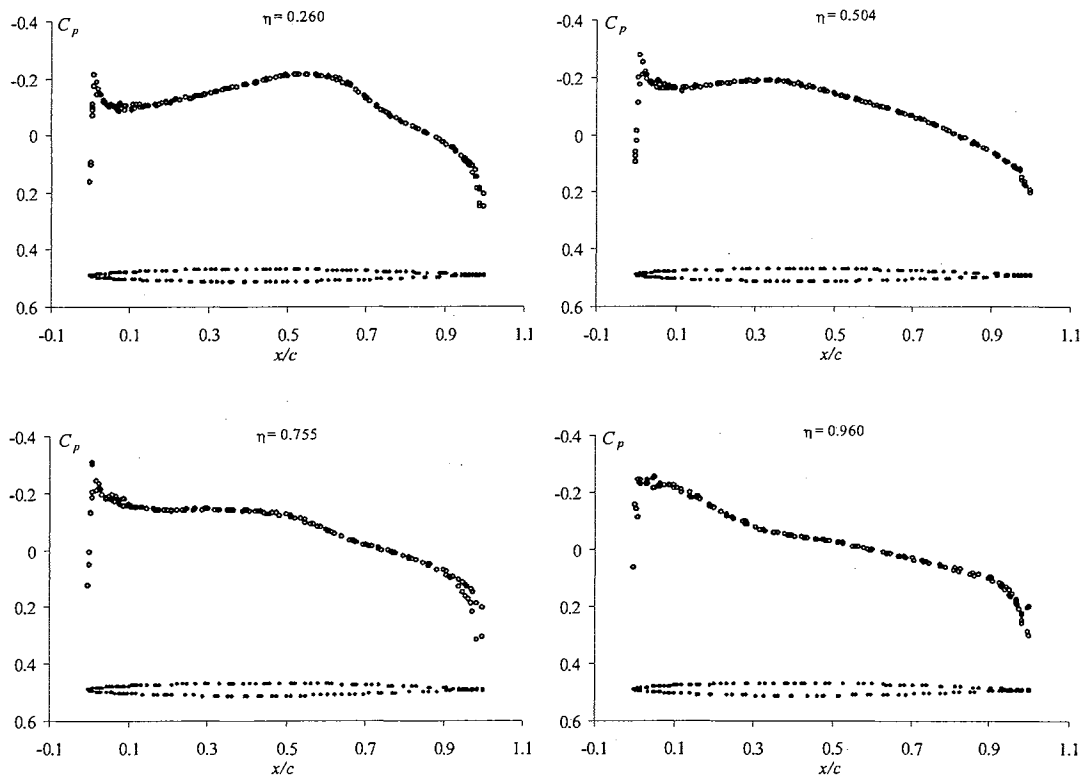


Figure 5.73: Plot of steady pressure coefficient distribution on the surface of the AGARD wing at Mach 0.96 for various spanwise locations, η .

Having completed a steady solution at Mach 0.96, we are now ready to begin an unsteady solution at Mach 0.96. Figure 5.74, Figure 5.75, and Figure 5.76 present the solver control parameters, elastic vectors input file, and external force input file respectively. All three of these input files are required for an unsteady aeroelastic solution. The dimensionless time step and all other solver parameters specified here were chosen to match those used in the original STARS analysis of this problem as close as possible. This will allow us to make a direct comparison between time history data from the two solvers as a verification of the aeroelastic predictions of the new solver.

```

dt      = 15.71d0,
gamma   = 1.40d0,
mach    = 0.960d0,
refdim  = 1.0d0,
cfl     = 0.50d0,
diss    = 1.0d0,
nstp    = 500,
ncyc    = 80,
nout    = 500,
idiss   = 1,
isol    = 2,
idsol   = 0,
ipnt    = 1,
istrt   = .true.,
iaero   = .true.,
ielast  = .true.,
idynm   = .false.,
iforce  = .true.,
nr      = 2,
ainf    = 12571.08,
rhoinf  = 3.40E-09

```

Figure 5.74: Summary of solver control parameters for the AGARD aeroelastic solution.

```

$ Number of elastic modes (nr)
  2
$ Mass matrix for elastic modes (nr x nr)
1.205600000000000E-003  0.000000000000000E+000
0.000000000000000E+000  3.614300000000000E-004
$ Damping matrix for elastic modes (nr x nr)
2.908949000000000E-003  0.000000000000000E+000
0.000000000000000E+000  3.467299000000000E-003
$ Stiffness matrix for elastic modes (nr x nr)
4.386490000000000  0.000000000000000E+000
0.000000000000000E+000  20.783400000000000
$ ICs for elastic modes (x1...xn, vx1...vxn)
0.0d0 0.0d0 0.0d0 0.0d0
$ IBXN for elastic modes (nr)
  0  0
$ Elastic modes vectors (nwl 2) x nr
0.7326388801E-10  -0.5034722292E-10  0.6874974370E+00
  ⋮  ⋮  ⋮

```

Figure 5.75: Header of elastic vectors input file for the AGARD aeroelastic solution.

0	0.0d0	0.0d0
1	0.0d0	0.0d0
2	0.0d0	0.0d0
3	10.0d0	10.0d0
4	10.0d0	10.0d0
5	10.0d0	10.0d0
6	0.0d0	0.0d0

Figure 5.76: External force input file for the AGARD aeroelastic solution.

Based on the previous computational results generated using the STARS unsteady solver, we expect that the flutter boundary for Mach 0.96 will occur at a dynamic pressure of approximately 0.233 psi. The combination of parameters given in Figure 5.74 equate to a dynamic pressure of approximately 0.247 psi, so we expect that the response time history will prove to be unstable. Our first comparison will be between a set of time history data computed by the new euler3d and old STARS unsteady CFD solvers. Figure 5.78 presents a plot of generalized displacement and force for both modes as computed by the two solvers.

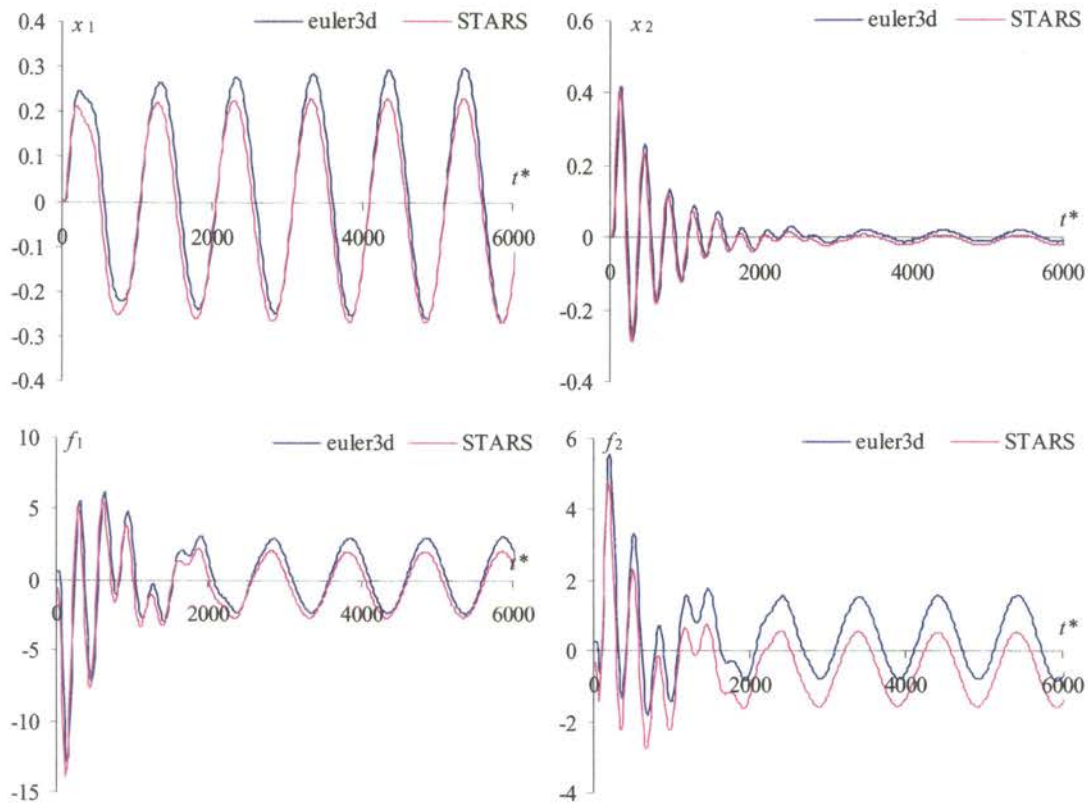


Figure 5.77: Comparison between time history data computed by STARS and euler3d solvers for the AGARD wing at Mach 0.96.

The most noticeable difference between the time history data in the plots of Figure 5.77 is caused by a discrepancy in the predicted steady state load condition. The steady state load for mode two as predicted by the STARS steady solver differs from the predicted value from the euler3d solver by nearly 200%. This has the obvious effect of shifting the time history data so that the two solutions oscillate about different neutral points, as well as impacting the relative amplitude of the generalized displacement time histories. Aside from this difference, the two sets of time history data are qualitatively the same. Both solutions show a system that is mildly unstable, with approximately the same frequency and damping for each mode. At this point, there are enough differences

between the two unsteady CFD solvers that we should expect some small differences between the two solutions. Hence, the excellent agreement between the time history data from each solver serves to verify the new aeroelastic solver.

The next step in predicting the flutter boundary for an aeroelastic problem would normally be to run the same solution at a different dynamic pressure. Based on the agreement between the euler3d and STARS time history data, we expect that the new solver would predict approximately the same flutter boundary as the previously published value. The more interesting test will come from a comparison between the predicted flutter boundary using both the zero-order and new second-order structural dynamics integrators. The previous solution used the zero-order integrator for comparison with the old STARS solution, which uses the same integrator. Figure 5.78 presents a comparison between time history data for mode one generalized displacement computed using the new euler3d solver with both the zero-order and second-order structural dynamics integrators. The only solver input parameter that differs between the two solutions is *idsol*, which is equal to zero or two for the zero-order and second-order integrators respectively.

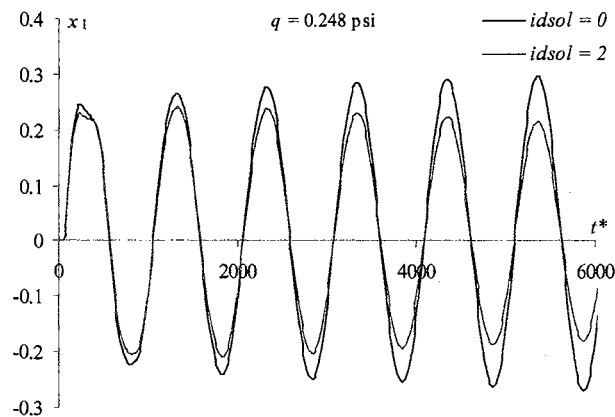


Figure 5.78: Comparison of time history data for AGARD wing at Mach 0.96.

Notice that there is a relatively large discrepancy between the time history data predicted by the zero-order and second-order structural dynamics solutions. The two sets of time history data are not even qualitatively the same. The time history data from the zero-order integrator shows a system that is mildly unstable, while the time history data from the second-order integrator shows a system that is clearly stable. As discussed in Section 4.6.3, this indicates that the time step chosen for this problem was too large. In order to refine the solution and determine the correct flutter boundary, we will need to decrease the time step. A complete set of time history data for $dt = 16.0, 4.0, 1.0$ and 0.25 is presented in Appendix C. Some of the solver control parameters necessarily change as the time step is refined. Table 5.14 presents a summary of the relevant parameters that were changed for each time step to construct the plots in Appendix C.

Table 5.14: Summary of solver control parameters used for time step refinement with AGARD wing at Mach 0.96.

dt	nstep	ncyc	cfl
16.0	500	80	0.40
4.0	2000	35	0.40
1.0	12000	12	0.50
0.25	32000	4	0.60

In order to compute the predicted flutter boundary for each set of solutions, we approximate the relative damping factor for the mode one generalized displacement data, and interpolate to find the dynamic pressure at which the system is neutrally stable. Figure 5.79 presents a plot of the dimensionless flutter velocity, V_f , versus dimensionless time step, dt^* , for the two different structural integrators, along with a reference line

representing the experimental value for Mach 0.96. The dimensionless flutter velocity is defined by the following equation:

$$(5.9) \quad V_f = \frac{U_f}{bw_\alpha \sqrt{\mu}}$$

where, U_f is the flutter velocity, b is the root semi-chord, ω_α is the uncoupled natural frequency of the wing first torsion mode, and μ is the mass ratio.

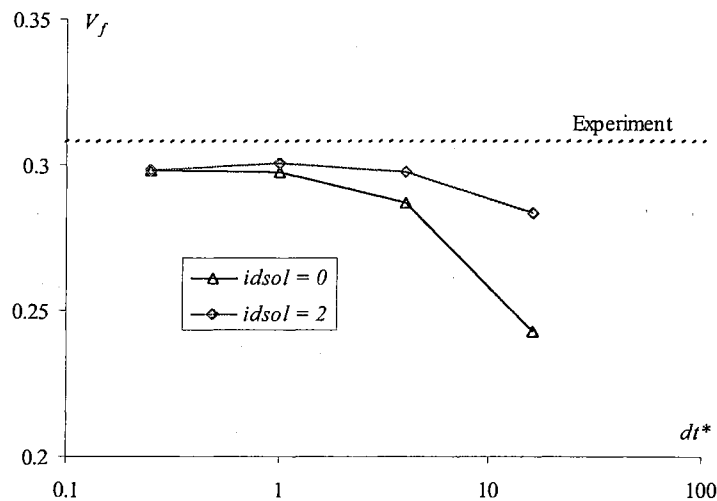


Figure 5.79: Plot of flutter velocity versus time step for the AGARD wing at Mach 0.96.

Based on these results, our initial choice of time step was too large by an order of magnitude or more. Our initial solution, which used the zero-order integrator with $dt = 16.0$, was off by approximately 21% when compared to the experimental value, whereas the final solution differs from the experimental value by only 3%. This is a significant refinement in the predicted flutter speed, with even further refinement a possibility with an improved computational grid since we limited ourselves to the grid used in the original STARS analysis. For a more direct comparison of each solution, Table 5.15 presents a summary of the flutter velocity and percent error for each set of solutions.

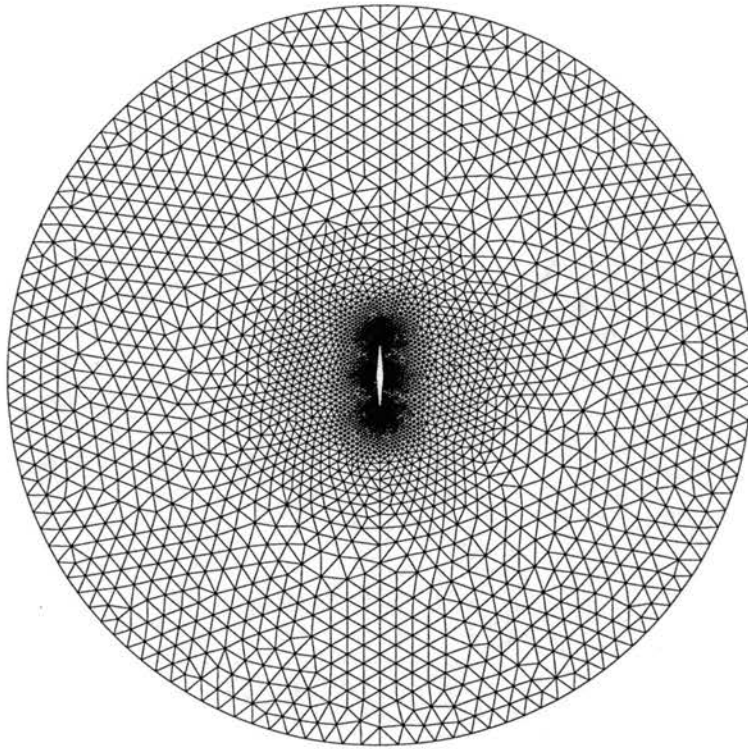


Figure 5.88: Surface triangulation around wedge-shaped body.

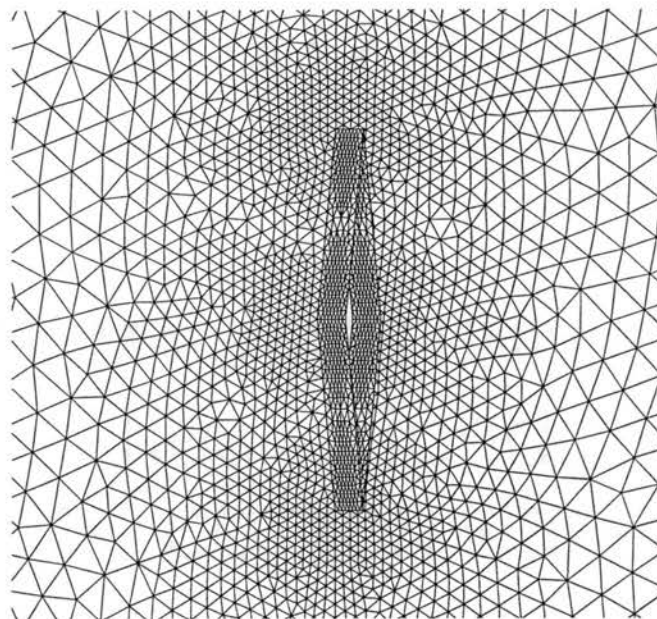


Figure 5.89: Close-up of surface triangulation near wedge-shaped body.

The total height of the wedge-shaped cylinder at the center of the computational domain is equal to one unit, while its width is equal to five percent of the total height. The outer radius of the computational domain was initially chosen to be six units, while the thickness of the computational domain was set to 0.2 units. With these dimensions, the computational grid generated for this study consists of 110,431 elements and 27,264 nodes. Boundary conditions for the ten surfaces enclosing this domain are specified as follows: the surfaces 1 through 4 define the sides of the computational domain and are symmetry planes, surfaces 5 and 6 are the far field boundaries along the outer radius of the computational domain, and surfaces 7 through 10 define the surface of the wedge-shaped body and are solid walls. Furthermore, the sharp edges at the top and bottom of the wedge are specified as singular since the local surface normal is undefined along those edges.

For this unsteady solution, we select a low mach number such that the effects of compressibility will be minimized. At a mach number of 0.3, it is expected that these effects will be less than five percent. Figure 5.90 presents a summary of all solver control parameters selected for this problem.

dt	= 0.040d0,
gamma	= 1.40d0,
mach	= 0.30d0,
cfl	= 0.50d0,
diss	= 1.0d0,
nstp	= 4000,
ncyc	= 50,
nout	= 30,
idiss	= 1,
isol	= 1,
ipnt	= 1,
istrt	= .false.,
iaero	= .true.,
ielast	= .false.,
idynm	= .false.

Figure 5.90: Summary of solver control parameters for the wedge solution.

Figure 5.91 presents a plot of pressure contours showing the vortex wake that develops downstream of the wedge-shaped cylinder. The alternating vortices propagate downstream as they are shed from the upper and lower edges of the body and eventually pass through the outer edge of the computational domain, which uses the far-field boundary condition. The fully developed flow reaches an oscillatory steady-state with a constant vortex shedding frequency.

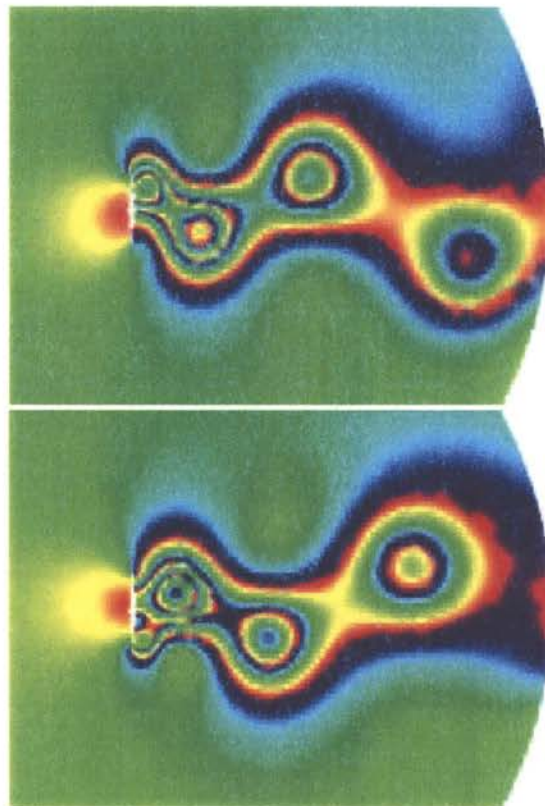


Figure 5.91: Contour plot showing vortex generation behind a wedge-shaped cylinder.

The vortex shedding frequency is exactly equivalent to the Strouhal number for this flow, which is simply a dimensionless frequency as defined by Equation (5.10).

$$(5.10) \quad St = \frac{f \cdot D}{U_0}$$

where f is the frequency of vortices shed in a vortex street, D is the length scale, and U_0 is the fluid velocity. The frequency of the vortices is typically determined by measuring the frequency of the lift coefficient time history data. Since the unsteady CFD solver already outputs all time history data with dimensionless values, the Strouhal number for the flow will be exactly equal to the frequency we measure for the time history data. Figure 5.92 presents a plot of the lift coefficient time history data for the wedge solution.

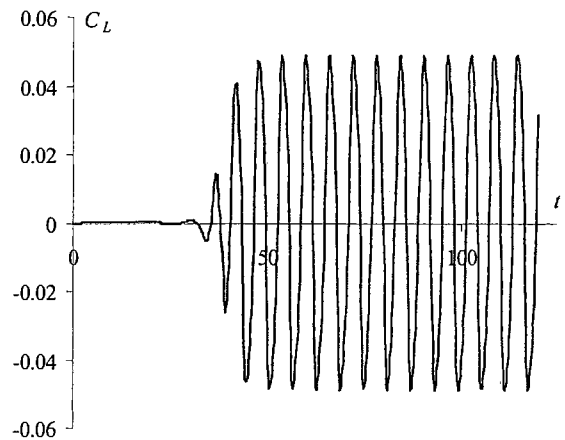


Figure 5.92: Plot of lift coefficient time history data for the wedge solution.

The computed Strouhal number for the time history shown in Figure 5.92 is 0.164. This compares to the observed value of 0.15 for a thin plate oriented perpendicular to the flow.⁴³ Given the differences in geometry between a flat plate and the wedge-shaped body examined here as well as the lack of viscous effects in the unsteady Euler solution, we do not expect the computed Strouhal number for this flow to match the experimental value for the thin plate. In this case, the comparison to an experimental value is simply to verify that the computed Strouhal number for this problem is reasonable.

Based on the computed Strouhal number for this problem, we have approximately 150 discrete solution steps per cycle. No significant changes in the flow solution were observed with further refinement of the time step or grid spacing. However, enlarging the outer radius of the computational domain, which provides more space for the vortices to propagate downstream before passing through the outer boundary, does yield significant changes in the flow solution. Figure 5.93 presents a comparison between time history data for outer radii ranging from six units to eighteen units.

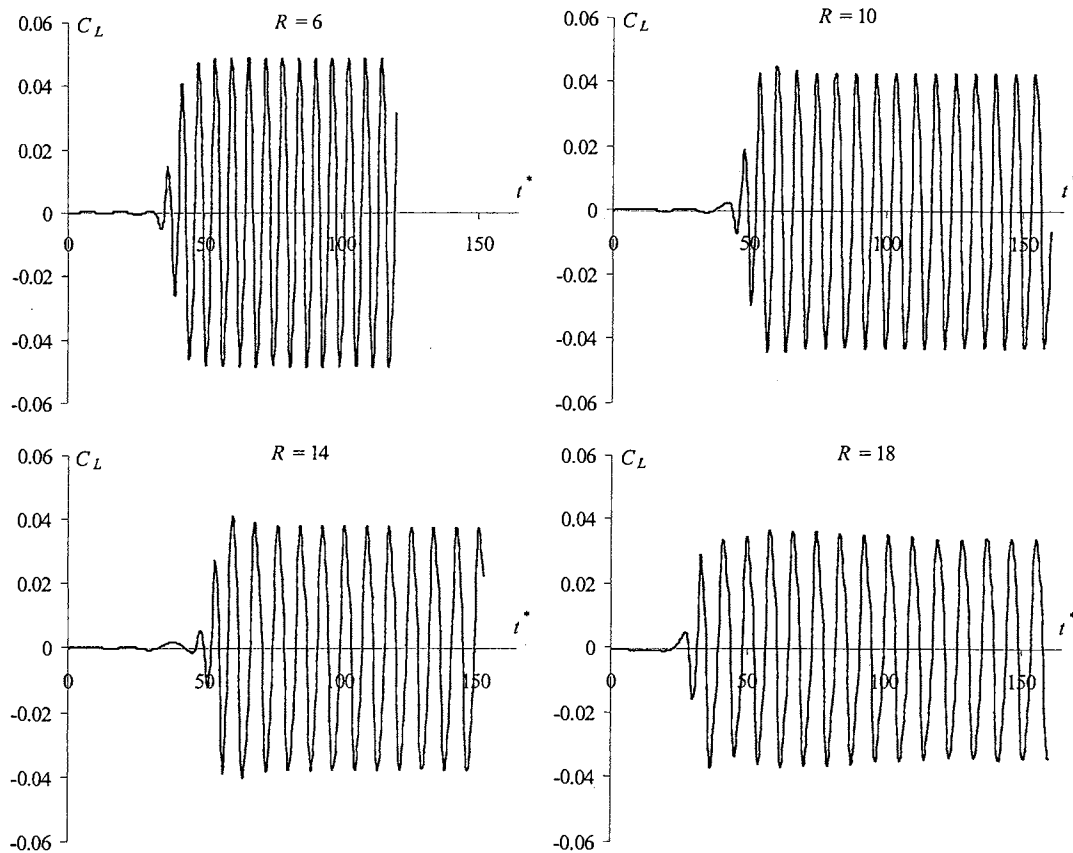


Figure 5.93: Comparison of lift coefficient time history data for the wedge solution with various outer radius dimensions.

As the outer radius of the computational domain increases, we observe a significant change in the Strouhal number for the flow. The Strouhal number ranges from the initially computed value of 0.164 for a radius of six units down to a value of 0.110 for a radius of twenty-two units. Table 5.18 presents a summary of the computed Strouhal number for all outer radii used in this study, and Figure 5.94 presents a plot of the raw numerical data. Figure 5.94 demonstrates proper convergence of the solution to the final Strouhal number of 0.110, which still compares reasonably well to the observed value of 0.15 for a thin plate oriented perpendicular to the flow.

Table 5.18: Summary of computed Strouhal numbers for the wedge solution.

R	St
6	0.164
10	0.138
14	0.121
18	0.111
22	0.110

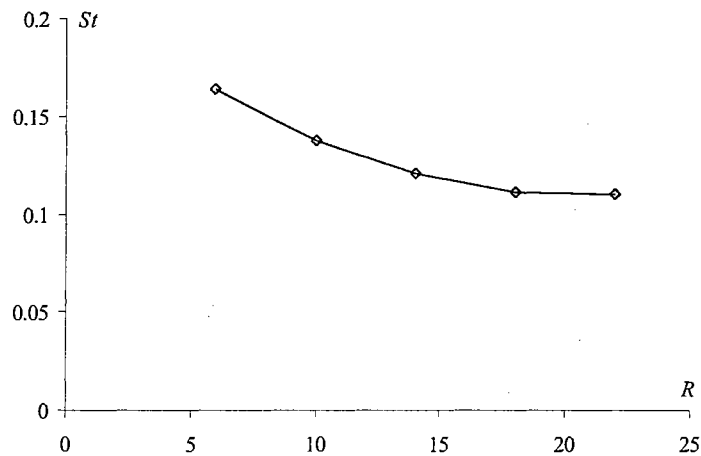


Figure 5.94: Plot of Strouhal number versus outer radius of the computational domain for the wedge solution.

The results of this section demonstrate the sensitivity of some solutions to the size of the computational domain. In the case of this problem, the vortices appear to generate a small disturbance as they pass through the outer boundary. This disturbance propagates up stream through the subsonic flow field and influences the vortex shedding frequency. The effect of the disturbance on the solution is minimized by choosing a sufficiently large computational domain. Hence, care should be taken when choosing the size of the computational domain for low mach number flows.

5.2.5 80 Degree Delta Wing

Having demonstrated the capacity to model vortex dominated flow problems in the previous section, this section investigates a relevant aerospace application known as wing-rock. The wing-rock phenomenon is a self-induced limit cycle rolling oscillation, which occurs for delta wings at high angles of attack. There has been considerable experimental and computational research into the basic physics of the unsteady, vortical flow that drives the wing-rock phenomenon.^{9,47,48,49} These studies indicate that the wing-rock phenomenon persists in all flow regimes from subsonic to supersonic and will provide us with the necessary data to validate the results obtained in this section.

For this study, we select a sharp-edged delta wing with a leading-edge sweep of 80° , a root chord equal to one unit, and a thickness equal to 0.01504 units. Figure 5.95 shows the cylindrical computational domain and surface triangulation generated for this geometry, while Figure 5.96 shows a close-up of the surface triangulation on the delta wing.

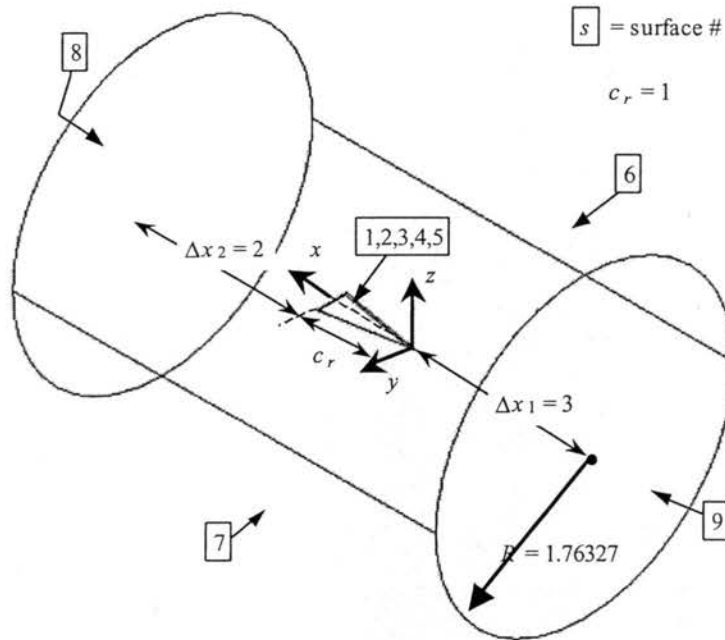


Figure 5.95: Layout of computational domain for 80 degree delta wing.

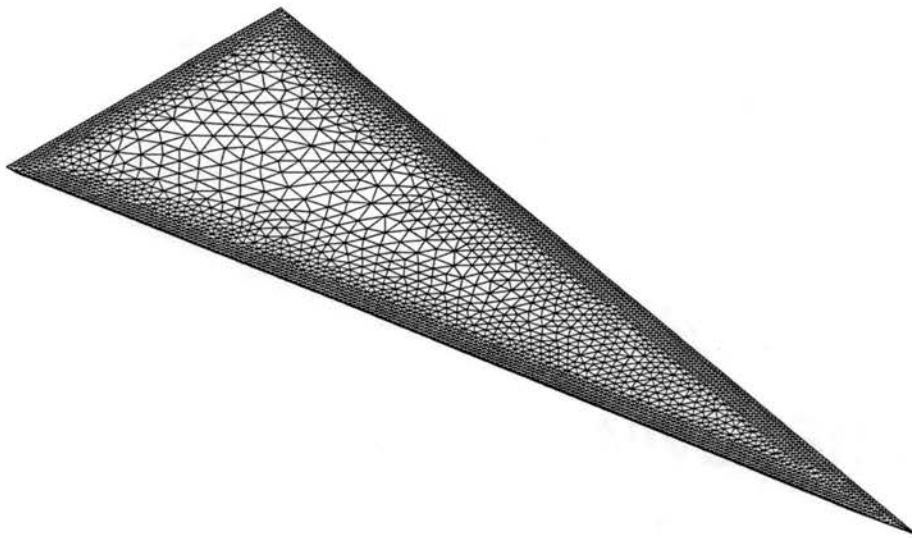


Figure 5.96: Surface triangulation for 80 degree delta wing.

The computational grid generated for this study consists of 461,575 elements and 84,448 nodes. Boundary conditions for the nine surfaces enclosing this domain are

specified as follows: surfaces 1 through 5 are solid walls and surfaces 6 through 9 are far-field boundaries. Furthermore, the sharp edges around the top-side of the delta wing are specified as singular since the local surface normal is undefined along those edges.

Our analysis of the delta wing configuration begins with a steady flow solution for a fixed angle of attack and various roll angles. Experimental results for a similar 80° delta wing are available for subsonic flow and an angle of attack of 30° from Arena.⁴⁷ These experimental measurements serve as a baseline to validate our computational results and verify that the grid is sufficiently refined before performing an unsteady analysis. The flow conditions for the initial subsonic solutions are defined by a freestream Mach number of 0.3, an angle of attack of 30° and roll angles ranging from 0° to 70°. Figure 5.97 presents a summary of the relevant solver control parameters for these solutions, and Table 5.19 presents a summary of the proper combination of solver angle parameters, alpha and beta, required to achieve the roll angles, ϕ , ranging from 0° to 70° for a fixed angle of attack of 30°.

gamma	= 1.40d0,
mach	= 0.30d0,
alpha	= 30.0d0,
beta	= 0.0d0,
refdim	= 1.0d0,
cfl	= 0.70d0,
diss	= 1.0d0,
nstp	= 10000,
ncyc	= 3,
nout	= 500,
idiss	= 1,
isol	= 0,
ipnt	= 1,
iaero	= .true.

Figure 5.97: Summary of solver control parameters for the steady delta wing solutions.

Table 5.19: Summary of solver angle parameters for various roll angles with a constant 30° angle of attack.

ϕ	alpha	beta
0°	30.000000	0.000000
10°	29.498704	5.725105
20°	28.481238	11.170229
30°	25.658906	16.102114
40°	22.521012	20.360575
50°	18.747237	23.858655
60°	14.477512	26.565051
70°	9.846552	28.481238

Figure 5.98 presents a plot of the velocity vectors along a normal cut plane at the $x/c = 0.60$ chord location for roll angles 0° and 40°. These plots are used to visualize the vortex structure that devolps over the wing.

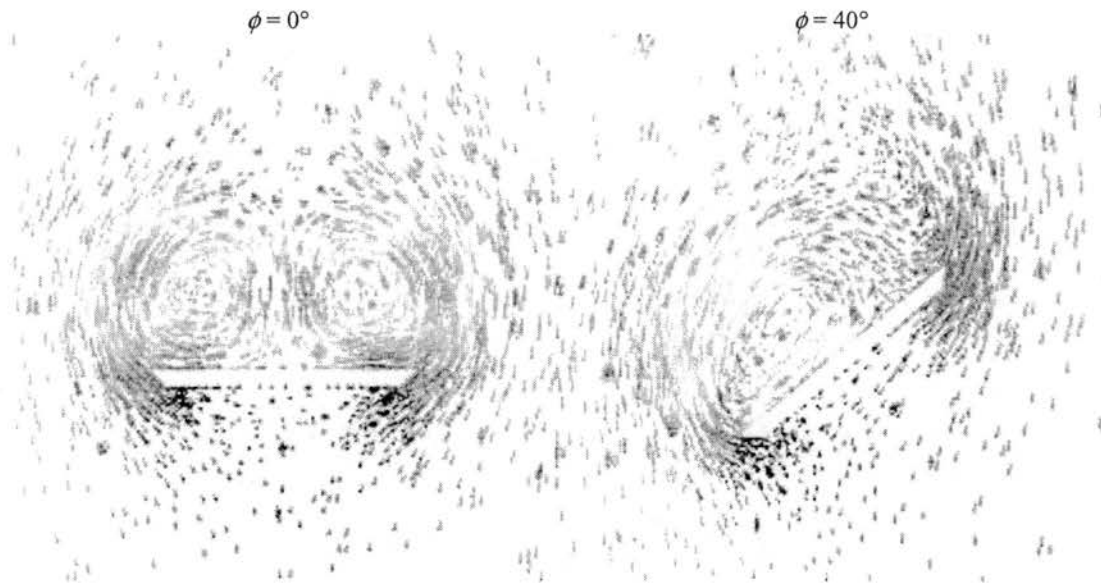


Figure 5.98: Plot of velocity vectors along a normal cut plane at $x/c = 0.60$ for the Mach 0.30 delta wing.

The approximate size and location of the primary vortices is consistent with the experimental results published by Arena⁴⁷, as well as the computational results from Gornier.⁴⁹ A complete set of plots comparing the computed surface pressure with the experimental measurements is also provided in Appendix E for all roll angles. These plots demonstrate reasonable agreement between the computed and experimental results for roll angles up to about 40°. The computed results capture the primary flow features, with a slightly underpredicted pressure peak. For increasing roll angles, we see that the computational model begins to diverge from the low Reynolds number experimental results. This is due to the development of secondary and tertiary vortices, which the inviscid flow solver is not capable of reproducing. Despite these differences, the similarity of the primary flow features lends credibility to these steady-state computational results.

Given that wing-rock behavior is expected to exist for high mach number flows as well, we prefer to use a supersonic mach number for our unsteady solution in order to take advantage of the better convergence characteristics of such a solution. Lee and Batina⁴⁸ have analyzed a similar 75° delta wing in Mach 1.2 flow with a conical euler solution and demonstrated the existence of wing rock. Figure 5.99 presents a plot of the velocity vectors for Mach 1.20 flow along a normal cut plane at the $x/c = 0.60$ chord location for roll angles 0° and 40° at the same 30° angle of attack. Additionally, Appendix E contains a set of steady surface pressure plots for Mach 1.20 flow at various roll angles. Unfortunately, there are no experimental results available for comparison with these solutions.

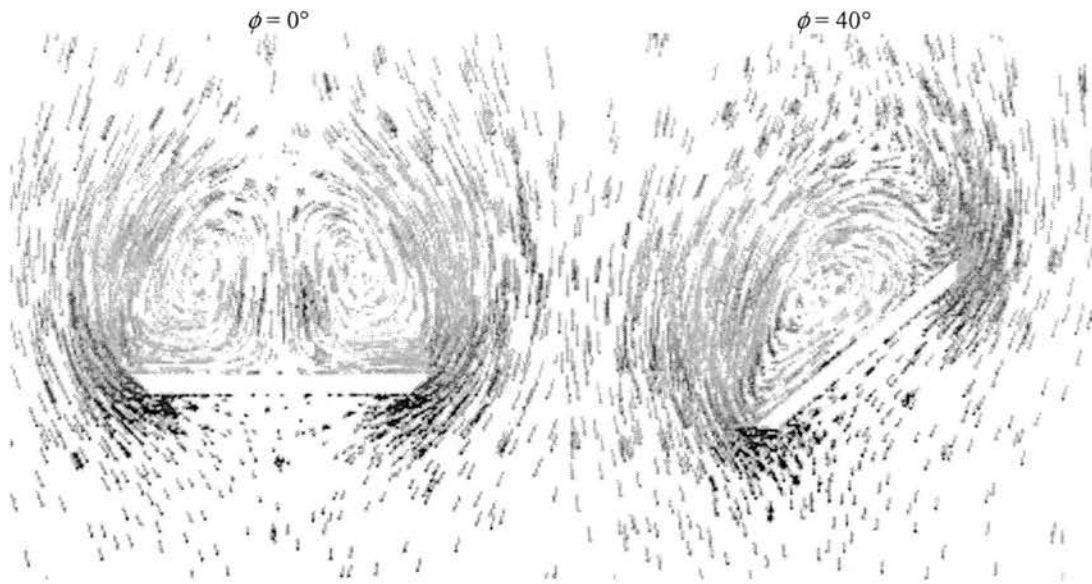


Figure 5.99: Plot of velocity vectors along a normal cut plane at $x/c = 0.60$ for the Mach 1.20 delta wing at a 30° angle of attack.

The unsteady computational results of Lee and Batina⁴⁸ indicate the existence of limit cycle wing-rock motion for a 75° delta wing in Mach 1.20 flow at an angle of attack of 30° . In contrast, their results predict that the same delta wing held at an angle of attack of 10° in Mach 1.20 flow will exhibit a stable free-to-roll response. The primary goal here will be to validate this physical phenomenon using the larger 80° delta wing geometry generated for this study. Figure 5.100 and Figure 5.101 present a summary of the relevant solver control parameters and dynamics inputs for the unsteady free-to-roll solutions. Dimensional parameters were taken from the results of Lee and Batina and scaled to match the physical dimensions of the delta wing generated for this study. The initial conditions for the first solution are for a delta wing at a 30° pitch and 40° roll angle with the roll axis free to rotate.

```

dt      = 0.010d0,
gamma   = 1.40d0,
mach    = 1.20d0,
alpha   = 0.0d0,
beta    = 0.0d0,
refdim  = 1.0d0,
cfl     = 0.60d0,
diss    = 1.0d0,
nstp    = 25000,
ncyc    = 8,
nout    = 200,
idiss   = 1,
isol    = 2,
idsol   = 2,
ipnt    = 1,
istrt   = .true.,
iaero   = .true.,
ielast  = .false.,
idynm   = .true.,
ainf    = 312.0d0,
rhoinf  = 0.526d0

```

Figure 5.100: Summary of solver control parameters for the unsteady free to roll delta wing solutions.

```

$ 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  0.005211d0  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  0.0d0  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  0.0d0  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 accels )
0.0d0, 0.0d0, 0.0d0, 40.0d0, 30.0d0, 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 (6)
1, 1, 1, 0, 1, 1

```

Figure 5.101: Dynamics input file for the unsteady free to roll delta wing solution.

The time step for this solution was chosen after some experimentation to find the point at which the sensitivity of the roll dynamics to time step was effectively eliminated. As in previous sections, this was accomplished by comparing time history data from the zero-order and second-order dynamics integrators for increasingly smaller time steps until the two solutions approximately matched. Figure 5.102 presents two plots of roll time history data for Mach 1.20 delta wing using initial roll angles of 40° and 10° . These two plots show the solution converging to the same limit cycle wing-rock motion with an amplitude of 23° and dimensionless frequency of 0.095.

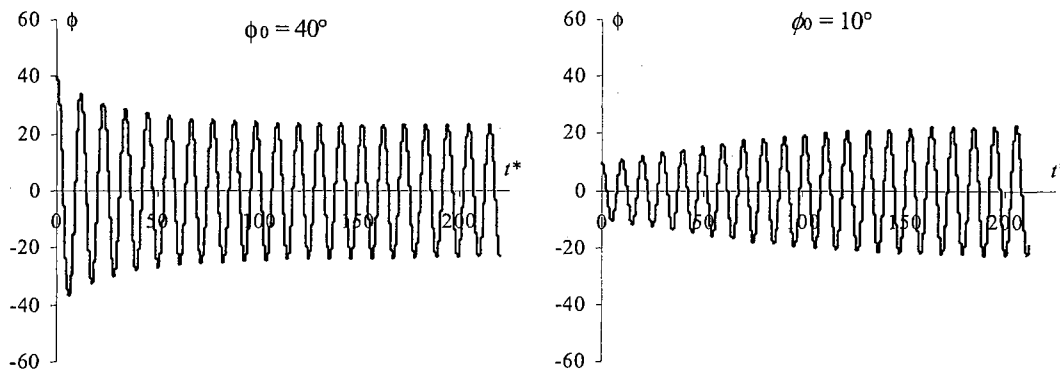


Figure 5.102: Roll time history data for the Mach 1.20 delta wing at a 30° angle of attack.

For wing-rock cases, it is typically more enlightening to study a plot of roll moment versus roll angle such as the plot presented in Figure 5.103. This plot illustrates the driving physics behind the delta-wing's limit cycle rolling motion. Clockwise oriented loops in the plot produce an unstable response, while counter-clockwise loops have a stabilizing effect on the roll response. For an initial roll angle of 40° , the area of counter-clockwise loops at the extreme values of roll angle is greater than the clockwise loop at the center of the plot for smaller roll angles. This causes the amplitude of the

response to decay until it eventually reaches an equilibrium point where the unstable and stable loops balance each other.

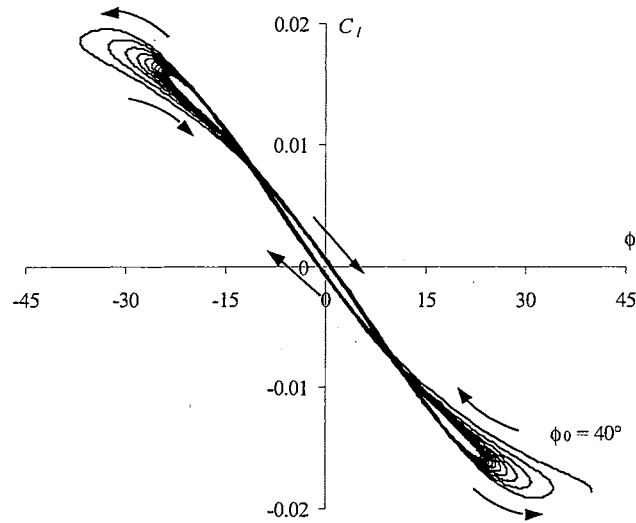


Figure 5.103: Plot of roll moment coefficient versus roll angle for the Mach 1.20 delta wing at a 30° angle of attack.

The last set of unsteady delta wing solutions is for a 10° angle of attack. In this case, the same basic set of solver parameters is reused with different initial conditions. Figure 5.104 presents a plot of roll time history data for this solution using an initial roll angle of 40°, while Figure 5.105 presents the plot of roll moment versus roll angle from the same solution. As expected this solution exhibits a stable response that eventually converges to a neutral roll angle. In this case, the plot of roll moment versus roll angle consists entirely of counter-clockwise, stabilizing loops, which become successively smaller with time.

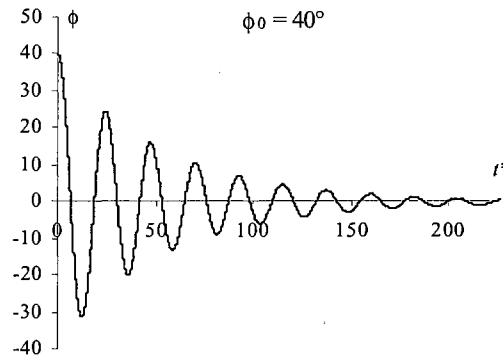


Figure 5.104: Roll time history data for the Mach 1.20 delta wing at a 10° angle of attack.

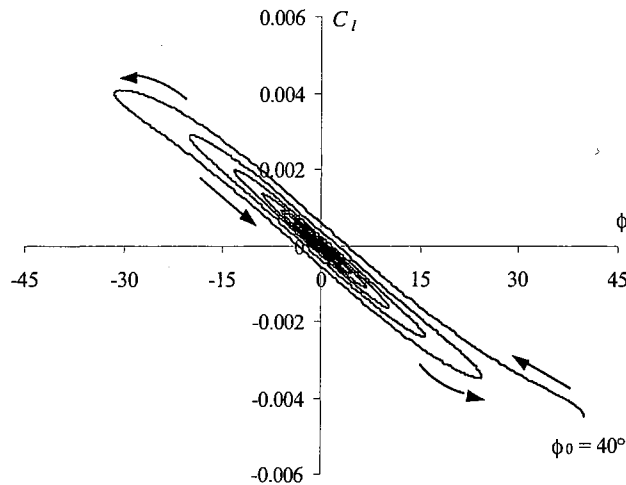


Figure 5.105: Plot of roll moment coefficient versus roll angle for the Mach 1.20 delta wing at a 10° angle of attack.

The results of this section show reasonable qualitative agreement with the expected dynamic properties of a rolling delta wing. Further consideration should be given to this problem to determine its sensitivity to grid resolution and overall size of the computational domain. Some time was spent optimizing the grid for the initial steady solutions, but only time step sensitivity was considered for the unsteady solutions.

5.2.6 Hovering Rotor

Our final validation problem investigates the unsteady dynamics of a hovering rotor. This type of flow problem is considered to be suitable for Euler codes because the essential physics are expected to be inviscid and separation does not typically occur.⁵¹ The geometry for this problem is the two-bladed rotor used in the experiments of Caradonna and Tung.⁵⁰ Since the publication of their original experimental results, this rotor has been the basis of numerous computational simulations ranging from panel methods⁵² to fully three-dimensional CFD solutions.^{53,54} One might even consider this to be the standard validation case for a non-inertial CFD solution considering the wealth of computational data that is available for it.

The rotor consists of two cantilever-mounted, rectangular blades with NACA 0012 cross-sections. The blades are untwisted and untapered with an aspect ratio of six and a precone of one-half degree. Figure 5.106 shows the cylindrical computational domain and surface triangulation generated for this geometry, while Figure 5.107 shows a close-up of the surface triangulation for the rotor. The total diameter of the rotor disk is 7.5 feet, the chord of the blades is 0.625 feet, and the cut-out radius at the center of the rotor is equal to one chord. The rotor is positioned in the computational domain such that two-thirds of the volume is below the plane of the rotor for capturing the unsteady wake that will develop in that region.

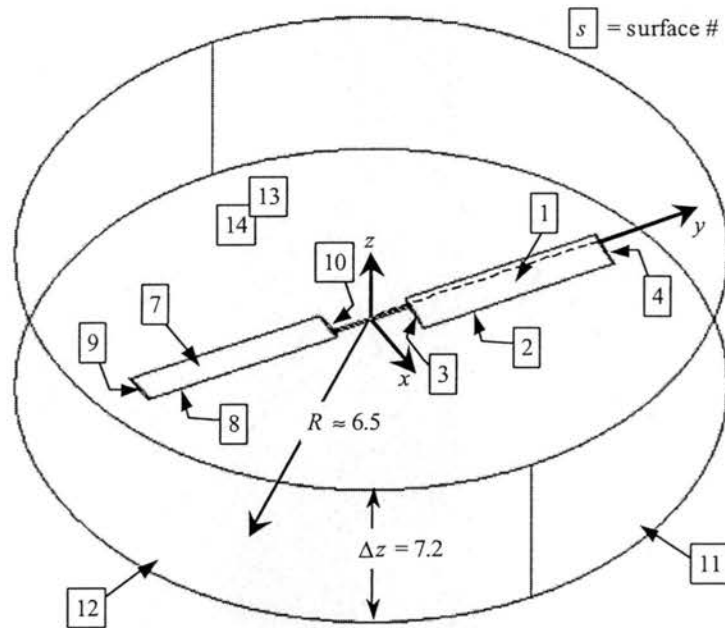


Figure 5.106: Layout of computational domain for hovering rotor.

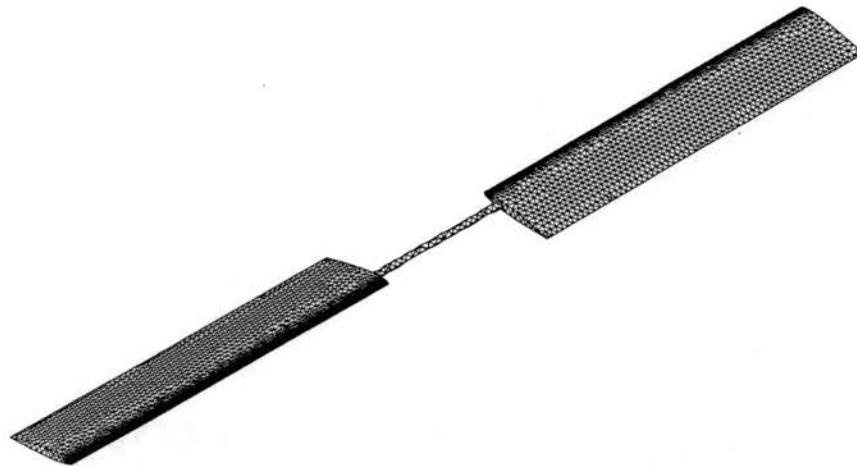


Figure 5.107: Close-up of surface grid for hovering rotor.

The computational grids generated for this study consist of approximately 1.5 million elements. The highest density of elements are in the region below the plane of the rotor for accurate representation of the wake. Initial tests with fewer elements showed that the wake would not be accurately modeled, and the problem behaved more

like a simple rectangular wing in a non-uniform, linear velocity field. Boundary conditions for the fourteen surfaces enclosing this domain are specified as follows: surfaces 1 through 10 are solid walls and surfaces 11 through 14 are far-field boundaries. Furthermore, the sharp edges at the two trailing edges of the rotor are specified as singular since the local surface normal is undefined along those edges.

The geometry for the rotor is defined such that the collective pitch angle for the rotor blades can be easily modified prior to generating the computational grid by adjusting a single parameter in a spreadsheet. Our first solution will be for a hover case with a tip Mach number of 0.520 and a collective pitch angle of two degrees. Figure 5.108 and Figure 5.109 present a summary of the solver control parameters and dynamics input file used for this solution.

dt	=	0.040d0,
gamma	=	1.40d0,
mach	=	0.520d0,
alpha	=	0.0d0,
beta	=	0.0d0,
refdim	=	0.625d0,
cfl	=	0.60d0,
diss	=	0.90d0,
nstp	=	10000,
ncyc	=	16,
nout	=	50,
idiss	=	1,
isol	=	2,
ipnt	=	1,
istrt	=	.false.,
iaero	=	.true.,
ielast	=	.false.,
idynm	=	.true.,
ifree	=	.false.,
ainf	=	1.0d0

Figure 5.108: Summary of solver control parameters for the hovering rotor with tip Mach 0.520.

```

$ 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 accels )
0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0
0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0, 7.94501
0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0, 0.0d0
$ IBXD for non-inertial frame (6)
1, 1, 1, 1, 1, 2

```

Figure 5.109: Dynamics input file for the hovering rotor with tip Mach 0.520.

In this case, we have disabled the usual free-stream velocity with `ifree = .false.`, since the velocity field will be generated exclusively by spinning the computational domain. The required tip Mach number is achieved by specifying `mach = 0.520`, and then computing a rotational velocity that will produce a dimensionless velocity of one unit at the tip of the rotor blade. The yaw rate that satisfies this condition is 7.94501 degrees per second, which equates to a dimensionless rate of 0.16667. Given this rate of rotation, the dimensionless time step chosen here equates to a resolution of approximately 942 global steps for each complete revolution of the rotor blades.

Figure 5.110 presents two plots of z-force time history data for this solution. The first plot shows a close-up of the convergence characteristics exhibited by this problem. After one-half revolution, we see a relatively large jump in the force time history data,

which coincides with the rotor blades' first encounter with the wake generated by the opposite blade. The discrete jumps in the time history data continue every half revolution until final convergence is achieved after approximately eight revolutions. Figure 5.111 shows the corresponding change in surface pressure distribution at the 0.68 span location for three points along the convergence profile.

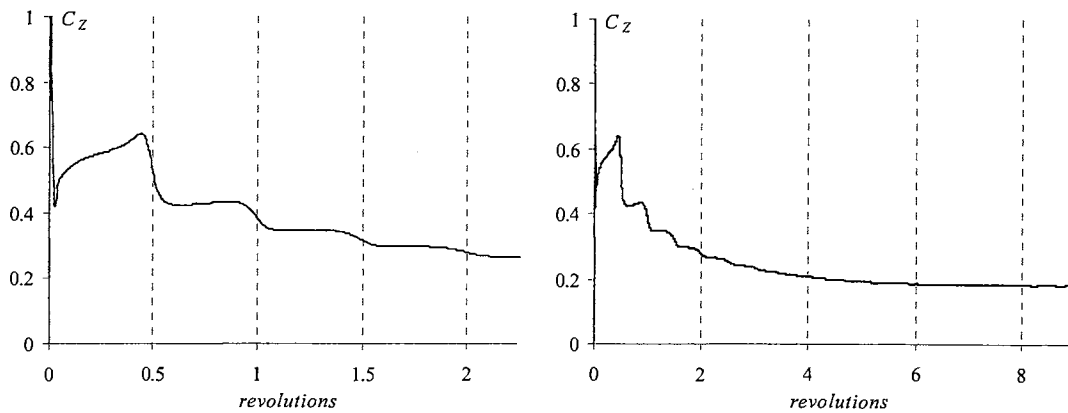


Figure 5.110: Plots of z -force time history for the hovering rotor with a 2° collective pitch angle and a tip Mach number of 0.520.

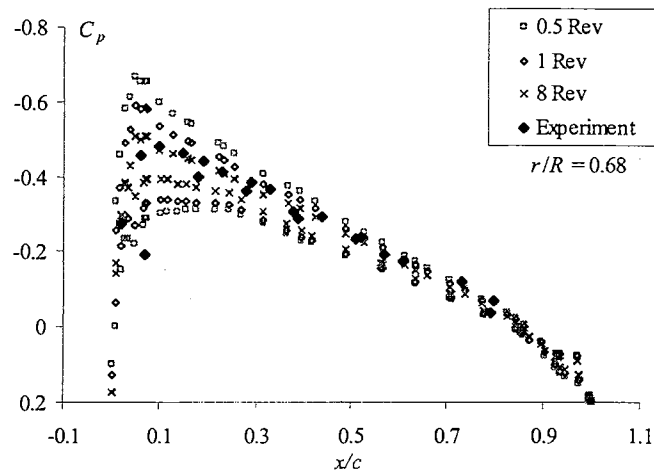


Figure 5.111: Comparison of surface pressure distributions after various revolutions for a hovering rotor with a 2° collective pitch and tip Mach number of 0.520.

The results of Figure 5.110 and Figure 5.111 illustrate the importance of allocating sufficient grid resolution to resolve the rotor wake. Preliminary tests with fewer elements produced convergence profiles that behaved more like the impulsively accelerated airfoil previously investigated in Section 5.1.7. Unfortunately, the grid resolution required for this problem has quickly reached the effective limits of our computer resources. The current problem consumes approximately 300 Meg of RAM and requires 27 hours of CPU time on a 2.4 GHz Pentium 4 to complete one revolution. Further refinement of the grid is impractical at this time, however the results of Figure 5.112 show reasonable agreement with the experimental data for all span stations.

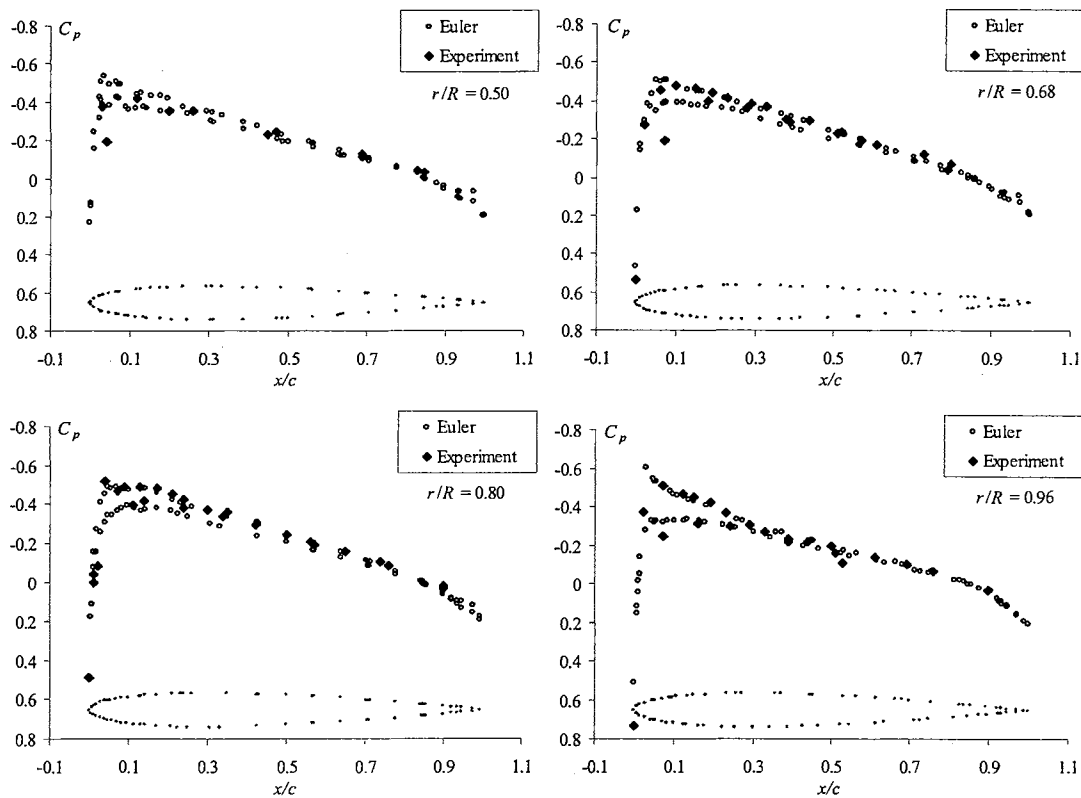


Figure 5.112: Plot of surface pressure distribution for a hovering rotor with a 2° collective pitch and tip Mach number of 0.520 after eight revolutions.

Our next solution will be for a hover case with a collective pitch angle of eight degrees and tip Mach number of 0.439, which corresponds to yaw rate of 6.70743 degrees per second. Figure 5.113 presents two plots of z-force time history data for this solution, and Figure 5.114 shows the corresponding change in surface pressure distribution at the 0.68 span location for three points along the convergence profile. The convergence trend is similar to the previous solution, except that final convergence is achieved in fewer revolutions. Figure 5.115 presents a comparison of the converged set of computed pressure distributions with the experimental values at four span stations. Once again we have reasonable agreement between the computed and experimental pressure distributions for all span stations. We expect that further refinement of the grid would enhance the agreement with experiment for both the two-degree and eight-degree solutions. Ideally, a grid similar to the fine grid used for the NACA 0012 verification solution of Section 5.1.4 would be used for this problem. However, this would require an estimated 24 million elements, which is obviously impractical at this time.

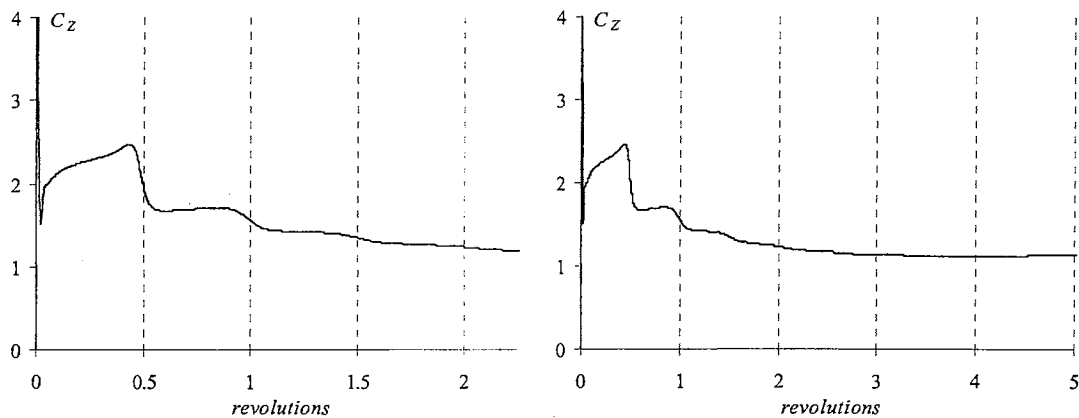


Figure 5.113: Plots of z-force time history for the hovering rotor with an 8° collective pitch angle and a tip Mach number of 0.439.

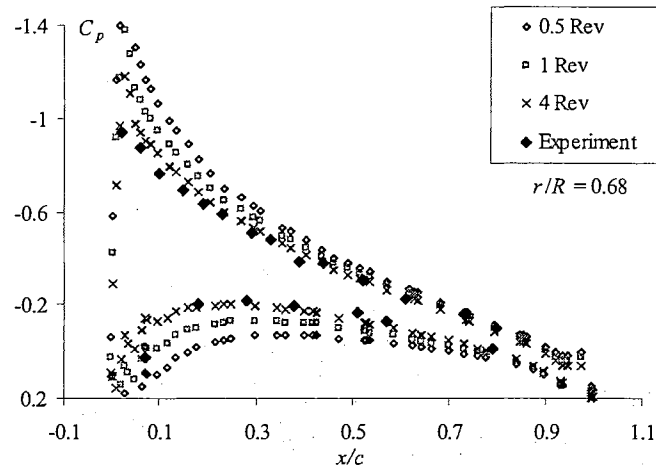


Figure 5.114: Comparison of surface pressure distributions after various revolutions for a hovering rotor with an 8° collective pitch and tip Mach number of 0.439.

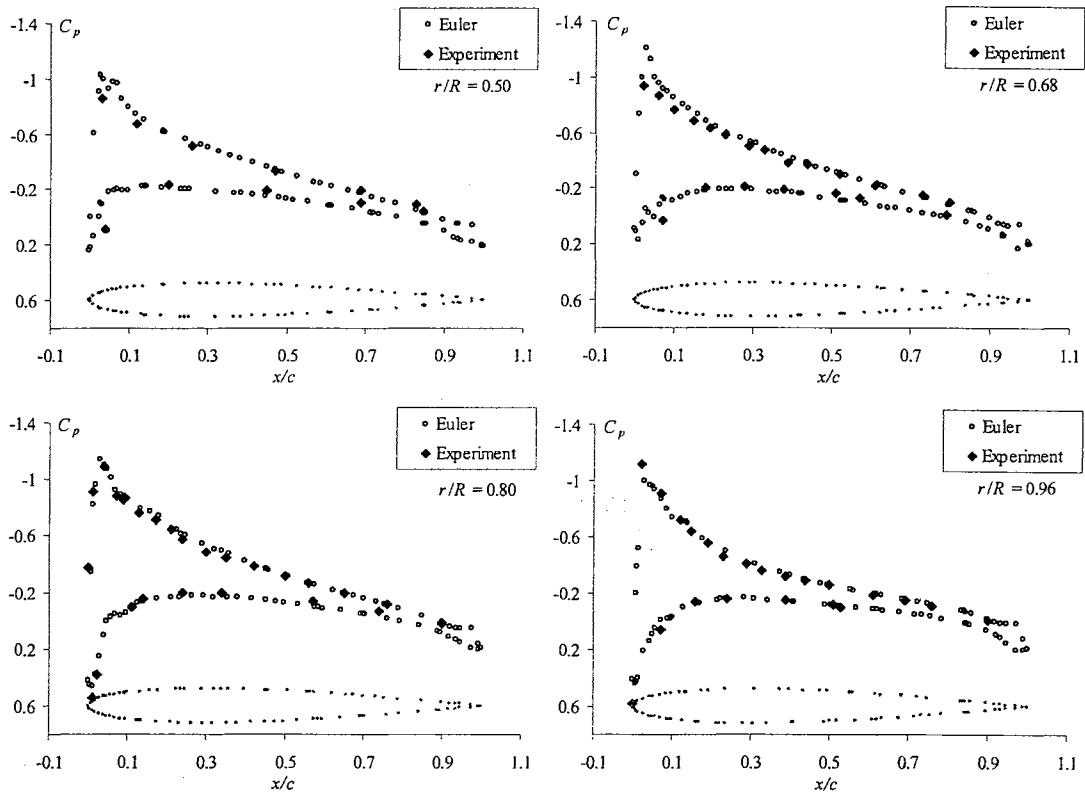


Figure 5.115: Plot of surface pressure distribution for a hovering rotor with an 8° collective pitch and tip Mach number of 0.439 after five revolutions.

Our next test with this model will use the transpiration boundary condition to simulate the previous two solutions using a baseline grid with a zero degree collective pitch. The required collective pitch angle for each solution is applied through an user generated mode shape that represents a uniform twisting of each blade to the proper orientation. We then perform an elastic solution with the structure clamped at an initial generalized displacement of one unit. Figure 5.116 presents a comparison of the computed surface pressure coefficient for a two degree collective pitch using transpiration to simulate the collective pitch angle and the previous solution where the blades were actually twisted.

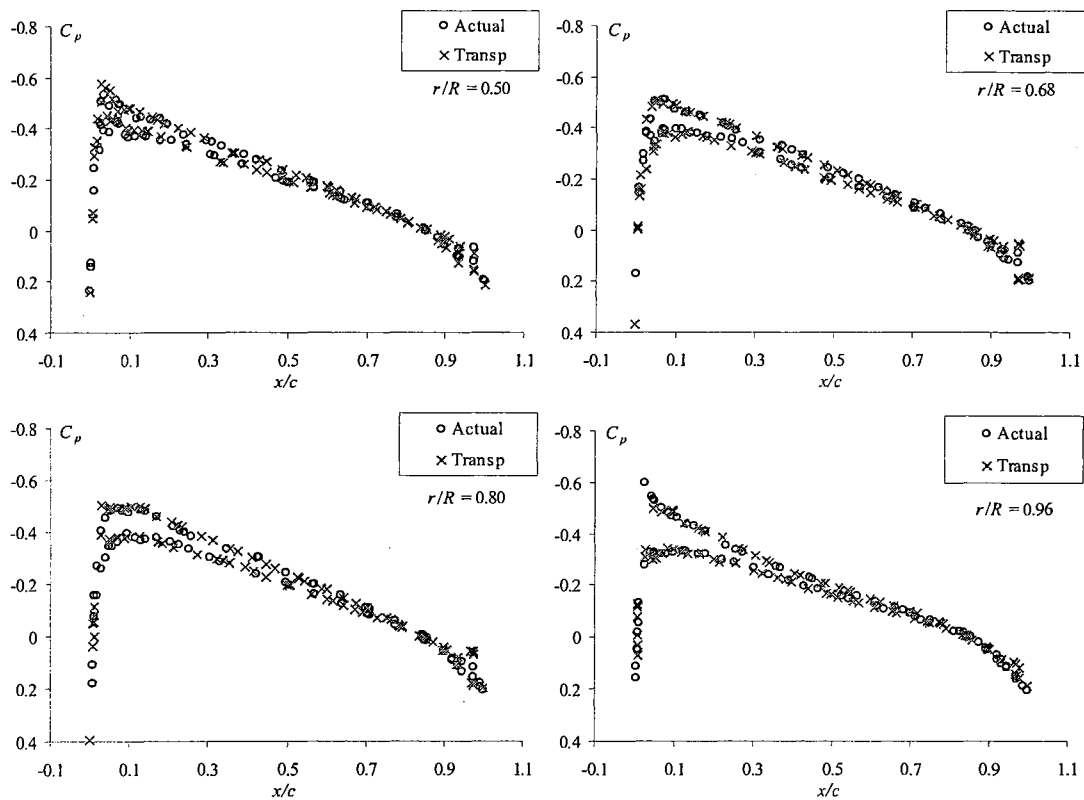


Figure 5.116: Comparison of computed surface pressure distribution for a hovering rotor using an actual and simulated 2° collective pitch angle for a tip Mach number of 0.520.

The results of Figure 5.116 are for the fully converged case after approximately eight revolutions of the rotor blade. Figure 5.117 presents a similar comparison for an eight degree collective pitch after approximately one revolution of the rotor blades. Even for this relatively large pitch angle, the transpiration solution is in reasonable agreement with the actual twist case. The primary comparison for both of these solutions is between the two sets of computational results. We are interested in the extent to which the transpiration boundary condition is capable of simulating an actual deformation of the surface grid. Hence, the experimental results have been neglected from these plots.

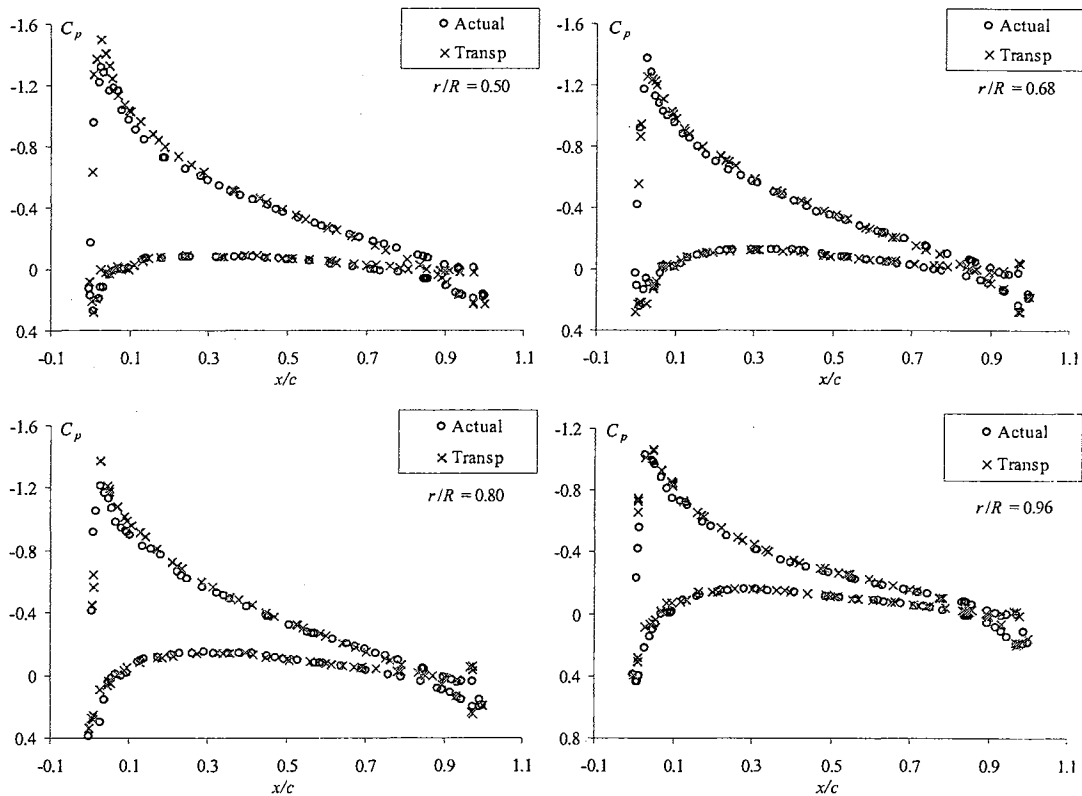


Figure 5.117: Comparison of computed surface pressure distribution for a hovering rotor using an actual and simulated 8° collective pitch angle for a tip Mach number of 0.439.

CHAPTER 6

6. CONCLUSIONS

The general goal of this research was to develop an efficient finite element methodology for non-inertial unsteady CFD solutions that would serve as the foundation for aeroservoelastic analysis of a super-maneuvering or spinning structure. To achieve this goal, the STARS ASE analysis routine was re-formulated based on a space-time finite element formulation for the compressible Euler equations expressed in non-inertial coordinates. The new algorithm takes advantage of the dissipation model and boundary conditions from the original STARS unsteady CFD algorithm, modified to account for a non-inertial coordinate system. This includes a modified transpiration boundary condition for simulating elastic deformations of the surface grid in a rotating frame of reference.

Additional enhancements in the new solver include improved structural integration techniques, increased stability for small time steps, and a general improvement in computational efficiency. We conclude this research effort with a general discussion of results to summarize experiences gained from analyzing the numerous problems presented in Chapter 5. In addition, the final section of this chapter outlines the remaining challenges for future work.

6.1 Discussion of Results

The verification and validation of a three-dimensional, unsteady CFD algorithm is never totally finished. Rather it is an on going process where we eventually achieve a certain level of confidence in the accuracy and validity of the algorithm, and that level of confidence continues to rise asymptotically as more verification and validation work is completed successfully. When we consider the number of components, parameters and boundary conditions, as well as the wide range of flow regimes that must be tested, verification and validation of the algorithm for all possible solutions is an imposing task. However, the verification and validation results presented in Chapter 5 make a compelling case for the accuracy and validity of the numerical algorithm developed here.

The verification cases analyzed here illustrate the performance and applicability of the algorithm for a wide range of flow regimes and a variety of different geometries. These solutions were for carefully controlled problems where the exact theoretical solution was known beforehand. Unfortunately, exact theoretical solutions for complicated three-dimensional, unsteady flow problems are rare, so most of our verifications problems were essentially two-dimensional solutions with a trivial third dimension used only to define the thickness of the computational domain. Regardless, these verification solutions demonstrate the accuracy of the implemented algorithm under carefully executed grid convergence studies as well as time step refinement studies for unsteady solutions. In all cases, convergence to the exact theoretical solution has been demonstrated with the exception of some difficulties resolving shocks at hypersonic Mach numbers, which continues to remain beyond of the realistic limit of the current algorithm.

The dissipation model of the current algorithm seems best suited for transonic and subsonic applications, although the rate of convergence is relatively slow when using a compressible algorithm to analyze subsonic flow fields. Subsonic solutions also seem to require a higher grid density than comparable transonic solutions to accurately resolve relevant flow gradients. However, subsonic verification solutions do demonstrate superior agreement between computed and theoretical results given sufficient refinement of the computational grid.

The validation solutions analyzed here demonstrate the applicability of the current algorithm for more complicated three-dimensional steady and unsteady problems. Difficulties arise in the validation of aeroelastic solutions where modeling errors for complicated fluid-structure interactions accumulate into the final result. Significant tuning of aeroelastic parameters within the level of uncertainty is typically necessary to achieve even a reasonable agreement with experiment. As a result, obtaining high quality experimental results that include a proper uncertainty analysis will continue to be one of the primary challenges in validating aeroelastic CFD algorithms.

6.2 Future Challenges

Brute force unsteady CFD solutions for complicated three-dimensional aeroelastic problems, such as those presented in Section 5.2, will continue to be impractical without significant reduction of the overall CPU requirements for such a solution. It is estimated that the verification and validation problems presented here required well over 12,000 CPU hours to generate. This includes failed solutions used to determine grid resolution and unsteady time step requirements for each problem, as well as experimentation with

the other solver control parameters that are used to tune the convergence and accuracy of the solution. Obviously, the main burden associated with time-marched aeroelastic CFD solutions is the processing time required to run multiple convergence and tuning solutions as well as multiple unsteady solutions for interpolating the instability boundary.

Although processing power has been increasing at an exponential rate since the introduction of the personal computer 25 years ago, the increased power has simply increased the size of the problems we are trying to solve. It is expected that we will continue to use up increases in processor speeds by increasing the grid density of our current problems, since we are constantly striving for more accurate solutions. The rotor model of Section 5.2.6 is a perfect example of a problem where we could easily add several million more elements to improve the wake resolution and overall accuracy of the solution. Hence, it will not be possible to simply wait for processor speeds to catch-up with the size of our current problems. With this in mind, we conclude this research effort with suggestions for improving the performance and accuracy of the solution scheme as well as extending its current capabilities.

- **Parallel Processing:** In the near future, the most significant advances in processing power will likely be gained from parallel processing. This trend is already becoming apparent in the interactive entertainment industry, where the next generation of Sony's consumer hardware may be based on the new "cell microprocessor" technology being developed jointly by Sony, IBM and Toshiba. This would mark the first intrusion of large scale parallel processing into the mainstream consumer market. Of course, even the most basic personal computer benefits from the performance savings of splitting some processing between the

graphics processor and CPU. In order to benefit from the trend toward parallel processing, our CFD algorithm will need to be written to take advantage of the extra processors by splitting the computational work into multiple tasks or threads. This area of research has the potential to provide the most significant reduction in overall processing time for a single CFD solution.

- **Convergence Requirements:** Iterative convergence of the unsteady solution for each time step is currently controlled by two constant control parameters: `cfl`, which affects the stability and rate of convergence, and `ncycl`, which simply specifies the number of iterative cycles for each global step. The choice of values for these two constants has a significant impact on the accuracy and processing requirements for an unsteady solution. Using fewer iterative cycles significantly reduces the cumulative processing time for an unsteady solution at the expense of degrading the overall accuracy of the solution, and the opposite is true when using more iterative cycles. However, there is a point of diminishing return, where extra iterative cycles will require extra processing time without significantly improving the overall accuracy of the solution. With the current implementation, the only precise method for choosing an acceptable value for the iterative convergence parameters is to run multiple solutions with different parameters and compare results. The solution residuals output by the solver are not a sufficient indicator of the level of convergence because it is unclear how small the residuals must be for the solution to be adequately converged. Future efforts should focus on determining a generic convergence criteria using a properly scaled set of solution residuals that is an accurate indicator of how much the solution is

actually changing for each iterative cycle. Given a proper convergence criteria, the number of iterative cycles could then be variable, such that the minimum number of iterative cycles are used for each global step, and the solution runs at the optimum accuracy to performance ratio.

- **Time Step Requirements:** Even if the unsteady CFD solution is properly converged for each global step, we have observed that the accuracy of time-marched aeroelastic solutions is dependent on the sensitivity of the structural integration to the global time step. This often means that we must use smaller global time steps than would normally be necessary for a time-accurate unsteady CFD solution with a rigid structure. It also means that several solutions must be run to determine the optimum time step for each problem. Future efforts should focus on developing a general technique for identifying the optimum time step for each problem without requiring multiple unsteady CFD solutions, as well as developing improved numerical algorithms for increasing the size of the minimum time step. It should be possible to use a simplified aerodynamic model along with the actual structural parameters for each problem to estimate the expected integration error and identify the largest allowable time step.
- **Grid Generation:** The accuracy of our solutions is often limited by our ability to generate high quality computational grids that do not exceed the available computer resources. It is necessary to find a balance between adding more elements to gain accuracy and removing more elements to save processing time. Furthermore, the processing time required to generate these grids, which was as high as eight hours for the largest problems evaluated here, has not been included

in the estimated processing cost provided at the beginning of this section. The current set of grid generation tools used in conjunction with STARS are out dated and should be updated or replaced with newer tools that use faster grid generation algorithms and use interfaces that allow better control over the local concentration of elements. The generic data format used by the new CFD algorithm was designed to allow easier conversion of models from the data format used by any readily available grid generation tool. Of course, we will always be limited by our own ability to accurately model the complex features of a three-dimensional aircraft, which is why most research problems are restricted to simple wing geometries.

- **Improved Dissipation Model:** As discussed in Section 3.6, some sort of artificial stabilization is a necessity for any discrete solution to the fluid dynamics equations of motion. Despite the general agreement of the solutions presented here with theoretical and experimental results, the artificial dissipation model adopted for this research effort is a source of error in our weighted residual formulation because the exact solution does not satisfy the algorithmic residual.²² Future research into improved stabilization methods should focus on identifying residual based dissipation models that are suitable for large-scale, unsteady CFD solutions. Residual based methods maintain the consistency of the finite element formulation and would eliminate a source of error in the algorithm.
- **Rigid-Body Dynamics:** A fully coupled rigid-body dynamics solver was not implemented for this work due to the need for further research into optimized non-inertial solutions for flexible structures. The non-inertial form of the

transpiration boundary condition implemented here should continue to be applicable to such a solution. The primary challenge will be updating the center of gravity for the structure, as well as the origin of the rotating coordinate system, if necessary.

- **Aerodynamic Modeling:** One of the primary limitations of aeroelastic CFD solutions is the processing time necessary to complete not just one unsteady CFD solution for a complicated three-dimensional problem, but multiple solutions at various dynamic pressures. Aerodynamic modeling techniques used in conjunctions with unsteady CFD results are beginning to make this limitation essentially disappear. The primary challenge will be to extend such a technique to the current non-inertial formulation for future research into aeroelastic solutions for spinning structures. Even if non-inertial aeroelastic solutions are neglected, such a technique could be valuable for extracting stability derivatives from a set of unsteady CFD time history data for a rigid-body non-inertial solution.

BIBLIOGRAPHY

1. Hunter, J.P. and Arena, A.S., "An Efficient Method for Time-Marching Supersonic Flutter Prediction Using CFD," *AIAA Paper 97-0733*, January 1997.
2. Ballhaus, W.F. and Goorjian, P.M., "Computation of Unsteady Transonic Flows by the Indicial Method," *AIAA Journal*, February 1978, pp. 117-124.
3. Cowan, T.J., Arena, A.S., and Gupta, K.K., "Accelerating CFD-Based Aeroelastic Predictions Using System Identification," *AIAA Paper 98-4152*, August 1998.
4. Gupta, K.K., "An Integrated, Multidisciplinary Finite Element Structural, Fluids, Aeroelastic, and Aeroservoelastic Analysis Computer Program," Users and Verification Manual, December 1995.
5. Gupta, K.K., "Development of a Finite Element Aeroelastic Analysis Capability," *Journal of Aircraft*, Vol. 33, No. 5, September-October, 1996: pp. 995-1002.
6. Stephens, C.H., Arena, A.S., and Gupta, K.K., "CFD-Based Aeroservoelastic Predictions With Comparisons To Benchmark Experimental Data," *AIAA Paper 99-0766*, January 1999.
7. Stephens, C.H., and Arena, A.S., "Application of the Transpiration Method for Aeroservoelastic Prediction Using CFD," *AIAA Paper 98-2071*, April 1998.
8. Kandil, O.A., and Chuang, H.A., "Unsteady Transonic Airfoil Computation Using Implicit Euler Scheme on Body-Fixed Grid," *AIAA Journal*, Vol. 27, No. 8, August 1989: pp. 1031-1037.
9. Kandil, O.A., and Chuang, H.A., "Computation of Vortex-Dominated Flow for a Delta Wing Undergoing Pitching Oscillation," *AIAA Journal*, Vol. 28, No. 9, September 1990: pp. 1589-1595.
10. Roache, P.J., *Verification and Validation in Computational Science and Engineering*, Hermosa Publishers, 1998.
11. Moin, P. and Kim, J., "Tackling Turbulence with Supercomputers," *Scientific American*, January 1997: pp. 62-68.
12. Moin, P. and Mahesh, K., "Direct Numerical Simulation: A Tool in Turbulence Research," *Annual Review of Fluid Mechanics*, Vol. 30, 1998: pp. 539-579.

13. Le, H., Moin, P., and Kim, J. "Direct Numerical Simulation of Turbulent Flow Over a Backward Facing Step," *Journal of Fluid Mechanics*, Vol. 330, January 1997: pp. 349-374.
14. Steger, J.L. and Bailey, H.E., "Calculation of Transonic Aileron Buzz," *AIAA Paper* 79-0134, 1979.
15. Nelson, R.C., *Flight Stability and Automatic Control*, Second Edition, McGraw-Hill, 1998.
16. Peiro, J., "A Finite Element Procedure for the Euler Equations on Unstructured Meshes," *Ph.D. Thesis*, University College of Swansea, September 1989.
17. Shakib, F., "Finite Element Analysis of the Compressible Euler and Navier-Stokes Equations," *Ph.D. Dissertation*, Stanford University, November 1988.
18. Morgan, K., Peraire, J., and Peiró, J., "Unstructured Grid Methods for the Simulation of 3D Transient Flows," Final Report NASA Research Grant No. NAGW-2962, Jun 1994.
19. Peiro, J., Peraire, J., and Morgan, K., "FELISA System Reference Manual: Basic Theory," December 3, 1993.
20. Shapiro, R.A., and Murman, E.M., "Higher-Order and 3-D Finite Element Methods for the Euler Equations," *AIAA Paper* 89-0655, 1989.
21. Katz, J. and Plotkin, A., *Low Speed Aerodynamics: From Wing Theory to Panel Methods*, McGraw-Hill, Inc., 1991.
22. Hughes, T.J.R., "Recent Progress in the Development and Understanding of SUPG Methods with Special Reference to the Compressible Euler and Navier-Stokes Equations," *International Journal for Numerical Methods in Fluids*, Vol. 7, 1987.
23. Shakib, F., Hughes, T.J.R., and Johan, Z., "A New Finite Element Formulation for Computational Fluid Dynamics: X. The Compressible Euler and Navier-Stokes Equations," *Computer Methods in Applied Mechanics and Engineering*, Vol. 89, 1991.
24. Masud, A. and Hughes, T.J.R., "A Space-Time Galerkin/Least-Squares Finite Element Formulation of the Navier-Stokes Equations for Moving Domain Problems," *Computer Methods in Applied Mechanics and Engineering*, Vol. 146, 1997.
25. Aliabadi, S.K. and Tezduyar, T.E., "Space-Time Finite Element Computation of Compressible Flows Involving Moving Boundaries and Interfaces," *Computer Methods in Applied Mechanics and Engineering*, Vol. 107, 1993.
26. Chandrupatla, T.R. and Belegundu, A., *Introduction to Finite Elements in Engineering*, Prentice Hall, 1991.

27. Jinyun, Y., "Symmetric Gaussian Quadrature Formulae for Tetrahedral Regions," *Computer Methods in Applied Mechanics and Engineering*, Vol. 43, 1984.
28. Keast, P., "Moderate Degree Tetrahedral Quadrature Formulas," *Computer Methods in Applied Mechanics and Engineering*, Vol. 55, 1986.
29. Shakib, F. and Hughes, T.J.R., "A New Finite Element Formulation for Computational Fluid Dynamics: IX. Fourier Analysis of Space-Time Galerkin/Least-Squares Algorithms," *Computer Methods in Applied Mechanics and Engineering*, Vol. 87, 1991.
30. Jameson, A., "Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows Past Airfoils and Wings," *AIAA Paper 91-1596*, 1991.
31. Hughes, T.J.R., "Finite Element Methods for Fluids," *AGARD Special Course on Unstructured Grid Methods for Advection Dominated Flows*, AGARD-R-787, May 1992.
32. Le Beau, G.J., Ray, S.E., Aliadabi, S.K., and Tezduyar, T.E., "SUPG Finite Element Computation of Compressible Flows with the Entropy and Conservation Variables Formulations," *Computer Methods in Applied Mechanics and Engineering*, Vol. 104, 1993.
33. NPARC Alliance, "Performing a Grid Convergence Study," *CFD Verification and Validation Website*, <http://www.grc.nasa.gov/www/wind/valid/gridconv.html>.
34. NPARC Alliance Validation Archive, "Prandtl-Meyer 15 Degree Corner Expansion at Mach 2.5," <http://www.grc.nasa.gov/www/wind/valid/pm15/pm15.html>.
35. Abbott, I.H. and Von Doenhoff, A.E., *Theory of Wing Sections: Including a Summary of Airfoil Data*, Dover Publications, New York: 1959.
36. NPARC Alliance Validation Archive, "10 Degree Cone at Mach 2.35," <http://www.grc.nasa.gov/www/wind/valid/cone10/cone10.html>.
37. Wagner, H. "Über die Entstehung des Dynamischen Auftriebes von Tragflugeln," *Zeitschrift für Angewandte Mathematik und Mechanik*, Vol. 5, No. 1, pp. 17-35, 1925.
38. Katz, J. "Calculation of the Aerodynamic Forces on Automotive Lifting Surfaces," *ASME Journal of Fluids Engineering*, Vol. 107, pp. 438-443, 1985.
39. Bisplinghoff, R., Ashley, H., and Halfman, R., *Aeroelasticity*, Dover Publications, Inc., 1996.
40. Cook, P.H., McDonald, M.A. and Firmin, M.C.P., "Aerofoil RAE 2822 - Pressure Distributions, and Boundary Layer and Wake Measurements," *Experimental Data Base for Computer Program Assessment*, AGARD Report AR 138, 1979.

41. Lee-Rausch, E.M. and Batina, J.T., "Calculation of AGARD Wing 445.6 Flutter Using Navier-Stokes Aerodynamics," *AIAA Paper 93-3476*, American Institute of Aeronautics and Astronautics, 1993.
42. Stephens, C.H., "CFD-Based Aeroservoelastic Predictions on a Benchmark Configuration Using the Transpiration Method," *Masters Thesis*, Oklahoma State University, 1998.
43. Rivera, J.A., Dansberry, B.E. and Durham, M.H., "Pressure Measurements on a Rectangular Wing with a NACA 0012 Airfoil During Conventional Flutter," *NASA TM 104211*, National Aeronautics and Space Administration, 1992.
44. Rivera, J.A., Dansberry, B.E. Bennett, R.M. and Durham, M.H., "NACA 0012 Benchmark Model Experimental Flutter Results With Unsteady Pressure Distributions," *NASA TM 107581*, National Aeronautics and Space Administration, 1992.
45. Waszak, M.R., "Modeling of the Benchmark Active Controls Technology Wind-Tunnel Model for Application to Flutter Suppression," *AIAA Paper 96-3437*, American Institute of Aeronautics and Astronautics, 1996.
46. Naudascher, E. and Rockwell, D., *Flow-Induced Vibrations: An Engineering Guide*, A.A. Balkema Publishers, 1993.
47. Arena, A.S., "An Experimental and Computational Investigation of Slender Wings Undergoing Wing Rock," *Ph.D. Dissertation*, Department of Aerospace and Mechanical Engineering, Notre Dame, April 1992.
48. Lee, E.M. and Batina, J.T., "Conical Euler Methodology for Unsteady Vortical Flows About Rolling Delta Wings," *AIAA Paper 91-0730*, January 1991.
49. Gordnier, R.E. and Visbal, M.R., "Numerical Simulation of Delta-Wing Roll," *AIAA Paper 93-0554*, January 1993.
50. Caradonna, F.X. and Tung, C. "Experimental and Analytical Studies of a Helicopter Rotor in Hover," *NASA TM 81232*, September 1981.
51. Caradonna, F.X. and Tung, C. "Finite-Difference Computations of Rotor Loads," *NASA TM 86682*, April 1985.
52. Katz, J. and Maskew, B., "Unsteady Low-Speed Aerodynamic Model for Complete Aircraft Configurations," *Journal of Aircraft*, pp. 302-310, April 1988.
53. Yang, G. and Zhuang, L., "Numerical Simulation of Rotor Flow in Hover," *Journal of Aircraft*, Vol. 37, No.2, pp. 221-226, March-April 2000.

54. Modi, A., Sezer-Uzol, N., Long, L.N. and Plassmann, P.E., "Scalable Computational Steering System for Visualization of Large-Scale CFD Simulations," *AIAA Paper 2002-2750*, June 2002.

APPENDICES

APPENDIX A: Summary of 2-D File Formats

This Appendix provides a summary of the input/output file formats used by the two-dimensional CFD solver developed for this research.

The following *input* files are defined:

- `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 nodal values of the primitive flow variables (density, velocity, and pressure) 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 dynamic solution. (ASCII)
- `case.vec` (optional) contains the elastic mode matrices, initial conditions, and vectors for the boundary surfaces as required for an aeroelastic solution. (ASCII)

The following *output* files are defined:

- `case.un1` contains the nodal values of the primitive flow variables (density, velocity, and pressure) for each node of the computational mesh. (Binary)
- `case.rsd` contains a history of the solution residuals for the conservation variables (density, momentum, and total energy). (ASCII)
- `case.lds` contains a history of the dimensionless aerodynamic forces acting on the solid walls of the CFD 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 and velocities for an unsteady, aeroelastic solution. (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=1,6 )
```

Line of Text

```
COOR(i,1) COOR(i,2)
           ( i = 1,...,nnd )
```

Line of Text

```
IELM(i,1) IELM(i,2) IELM(i,3)
           ( i = 1,...,nel )
```

Line of Text

```
ISEG(i,1) ISEG(i,2)
           ( i = 1,...,nsg )
```

Line of Text

```
IBEL(i,1) IBEL (i,2)
           ( 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) boundary element
starting/stopping indexes for three 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) boundary curve containing
boundary element *i*

Comments

- This is a plain text (ASCII) file.
- Nodal data 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 specified in the solver control file.
- The element connectivity data must define elements that are oriented clockwise.
- 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 makeg2d is used to convert a standard STARS surface triangulation file and modified boundary conditions file into an appropriately sorted two-dimensional geometry file.

Sample File

```
$ nnd, nel, nsg, nbe, nbp, nwl, nsd
   8   6  13   8   8   3   0
$ LBE(6)
   1   2   3   2   3   8
$ 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
   3   2
   2   6
   6   4
   4   7
   7   5
   5   8
   8   1
```

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,
  ncy         = 3,
  isol        = 0,
  idiss       = 0,
  ipnt        = 1,

  istrtr      = .false.,
  iaero       = .false.,
  idynm       = .false.,
  ielast      = .false.,
  ifree       = .true.,
  iforce      = .false.,

  nr          = 0,
  ainf        = 1.0d0,
  rhoinf      = 1.0d0,
/
```

Definition of Terms

dt : (real) dimensionless global time step
gamma : (real) ratio of specific heats
diss : (real) dissipation factor or constant
cfl : (real) local time step stability factor

mach : (real) free-stream mach number
alpha : (real) free-stream angle of attack
refdim : (real) reference dimension

nstp : (int) total solution steps
nout : (int) output frequency, steps/output
ncyc : (int) iterative cycles per solution step
isol : (int) CFD solution type
idiss : (int) dissipation type
ipnt : (int) number of points for numerical integration of flux/source vectors

istrtr : (logical) restart flag
iaero : (logical) aerodynamic forces flag
idynm : (logical) dynamic/non-inertial flag
ielast : (logical) elastic flag
ifree : (logical) free-stream velocity flag
iforce : (logical) external force flag

nr : (int) number of elastic modes
ainf : (real) dimensional free-stream sonic speed
rhoinf : (real) dimensional free-stream density

Comments

- This is a plain text (ASCII) file formatted as a Fortran namelist.
- The default values for each parameter are given in the basic file format above.
- The global time step is only used for unsteady solutions.
- 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 typically acceptable for most problems. For unsteady solutions, the stability factor is typically in the range $0.3 \leq \text{cfl} \leq 0.8$.
- The values of refdim, mach, ainf, and rhoinf are used to non-dimensionalize all values read in by 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 3 for steady solutions. For unsteady solutions, use a sufficient number of cycles to allow for an appropriate level of convergence at each step.
- 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 available dissipation types defined as follows:
 - `idiss = 0` is a low order dissipation
 - `idiss = 1` is a high order dissipation with gradient limiters
- The low-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 high-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.
- There are two types of numerical integration defined as follows:
 - `ipnt = 1` uses a one-point gauss quadrature
 - `ipnt = 3` uses a three-point symmetric gauss quadrature
- When the restart flag 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 free-stream velocity flag 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 the external force 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 global time steps if it was non-zero.

Solution Unknowns Input/Output File (case.un*)

Basic File Format

```
nnd gam xmi alp ref t  
( ( UN(i,j), i=1,nnd ), j=1,5 )
```

Definition of Terms

nnd: (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) 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 this file are relative quantities referenced to the body-fixed coordinate system.

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-coordinate for origin of rotation

R0(2) : (real) y-coordinate 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 coord. system

y : (real) initial y-position of coord. system

q : (real) initial orientation of coord. system

vx : (real) initial x-velocity of coord. system

vy : (real) initial y-velocity of coord. system

vq : (real) initial angular velocity of coord. system

ax : (real) initial x-acceleration of coord. system

ay : (real) initial y-acceleration of coord. system

aq : (real) initial angular accel. of coord. system

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 vector defining the origin of rotation is subtracted directly from the nodal coordinates defined in the geometry input file after it is non-dimensionalized by the reference dimension.
- The mass matrix 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.
- Initial conditions for the rotational degree of freedom should have units of degrees, degrees/sec, etc.
- 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 with zero velocity and acceleration.
- $IBXD = 2$ is a constant acceleration, uncoupled response, i.e. integrates acceleration and velocity to compute new position.

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)
0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0
0.0d0 0.0d0 0.0d0
$ 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 0.00d0 0.0d0
0.0d0 0.00d0 0.0d0
0.0d0 0.00d0 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,nw1*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

IBXN(i) : (int) dynamics flag for i^{th} mode

PHIA(i*2-1,j) : x -component of displacement vector for mode j at node i

PHIA(i*2,j) : y -component of 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 mass matrix 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 generalized displacement and velocity.
 - IBXN = 1 is a clamped condition, i.e. hold at initial generalized displacement with zero velocity.
 - IBXN = 2 is a constant velocity, uncoupled response, i.e. integrates generalized velocity to compute new displacement.

- $IBXN = 3$ is a forced multistep response used for system identification purposes.
- Do not combine $IBXN = 0$ with $IBXN \neq 0$ for different modes if there are coupling or off-diagonal terms in the mass, damping or stiffness matrices.
- A limited set of simple modal vectors representing standard rigid-body degrees of freedom can be generated using the program `makevec2d`.

Sample File

```
$ Number of elastic modes (nr)
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)
0.0d0    0.0d0    0.0d0
0.0d0    0.0d0    0.0d0
0.0d0    0.0d0    0.0d0
$ 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
  ⋮    ⋮
```

Solution 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	0.57348E-08	0.25187E-07	0.16390E-07	0.90629E-07
2	0.44739E-08	0.18578E-07	0.12373E-07	0.67703E-07
3	0.34518E-08	0.13871E-07	0.96347E-08	0.50632E-07
4	0.26323E-08	0.10501E-07	0.77174E-08	0.37989E-07
5	0.19826E-08	0.80732E-08	0.63226E-08	0.28663E-07
6	0.14732E-08	0.63101E-08	0.52640E-08	0.21805E-07
7	0.10788E-08	0.50186E-08	0.44300E-08	0.16766E-07
8	0.77813E-09	0.40643E-08	0.37547E-08	0.13063E-07
9	0.55413E-09	0.33531E-08	0.31987E-08	0.10335E-07
10	0.39393E-09	0.28190E-08	0.27371E-08	0.83161E-08
11	0.28851E-09	0.24147E-08	0.23530E-08	0.68136E-08
12	0.23033E-09	0.21061E-08	0.20339E-08	0.56868E-08
13	0.20827E-09	0.18682E-08	0.17699E-08	0.48346E-08
14	0.20688E-09	0.16825E-08	0.15525E-08	0.41839E-08
15	0.21322E-09	0.15354E-08	0.13745E-08	0.36818E-08
16	0.22042E-09	0.14165E-08	0.12294E-08	0.32900E-08
17	0.22583E-09	0.13186E-08	0.11114E-08	0.29802E-08
18	0.22883E-09	0.12363E-08	0.10156E-08	0.27318E-08
19	0.22955E-09	0.11656E-08	0.93758E-09	0.25294E-08
20	0.22840E-09	0.11038E-08	0.87375E-09	0.23617E-08
⋮	⋮	⋮	⋮	⋮

Aerodynamic Loads Output File (case.l ds)

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_i: (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 in this output file are dimensionless values based on the reference conditions specified in the solver control file.
- 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-01	0.00000E+00	0.00000E+00	0.00000E+00
2	0.20000E-01	0.00000E+00	0.00000E+00	0.00000E+00
3	0.30000E-01	0.00000E+00	0.00000E+00	0.00000E+00
4	0.40000E-01	0.00000E+00	0.00000E+00	0.00000E+00
5	0.50000E-01	0.00000E+00	0.00000E+00	0.00000E+00
6	0.60000E-01	0.00000E+00	0.00000E+00	0.00000E+00
7	0.70000E-01	0.00000E+00	0.00000E+00	0.00000E+00
8	0.80000E-01	0.00000E+00	0.00000E+00	0.00000E+00
9	0.90000E-01	0.00000E+00	0.00000E+00	0.00000E+00
10	0.10000E+00	0.00000E+00	0.00000E+00	0.00000E+00
11	0.11000E+00	0.00000E+00	0.00000E+00	0.00000E+00
12	0.12000E+00	0.00000E+00	0.00000E+00	0.00000E+00
13	0.13000E+00	0.00000E+00	0.00000E+00	0.00000E+00
14	0.14000E+00	0.00000E+00	0.00000E+00	0.00000E+00
15	0.15000E+00	0.00000E+00	0.00000E+00	0.00000E+00
16	0.16000E+00	0.00000E+00	0.00000E+00	0.00000E+00
17	0.17000E+00	0.00000E+00	0.00000E+00	0.00000E+00
18	0.18000E+00	0.00000E+00	0.00000E+00	0.00000E+00
19	0.19000E+00	0.00000E+00	0.00000E+00	0.00000E+00
20	0.20000E+00	0.00000E+00	0.00000E+00	0.00000E+00
:	:	:	:	:

Dynamic Output File (xd.dat)

Basic File Format

```
0 0.0 (XD(i), i=1,6) (A0(i), i=1,3)
:   :   :           :           :
:   :   :           :           :
istp t_istp (XD(i), i=1,6) (A0(i), i=1,3)
:   :   :           :           :
:   :   :           :           :
nstp t_nstp (XD(i), i=1,6) (A0(i), i=1,3)
```

Definition of Terms

istp: (int) current solution step
nstp: (int) total or last solution step
 t_i : (real) dimensionless time at step i

XD (1) : (real) x-position
XD (2) : (real) y- position
XD (3) : (real) pitch angle
XD (4) : (real) x-velocity
XD (5) : (real) y- velocity
XD (6) : (real) pitch rate

A0 (1) : (real) x-acceleration
A0 (2) : (real) y- acceleration
A0 (3) : (real) pitch acceleration

Comments

- This is a plain text (ASCII) file.
- The dynamic data in this output file are dimensionless values based on the reference conditions specified in the solver control 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.

Elastic Output File (xn . dat)

Basic File Format

```
0 0.0 (XN(i), i=1,nr*2) (FA(i), i=1,nr)
:    :      :
:    :      :
istp t_istp (XN(i), i=1, nr*2) (FA(i), i=1,nr)
:    :      :
:    :      :
nstp t_nstp (XN(i), i=1, nr*2) (FA(i), i=1,nr)
```

Definition of Terms

istp: (int) current solution step
nstp: (int) total or last solution step
 t_i : (real) dimensionless time at step i

XN(i): (real) gen. displ. for mode i
XN($i+nr$): (real) gen. vel. for mode i

FA(i): (real) gen. force for mode i

Comments

- This is a plain text (ASCII) file.
- The elastic data in this output file are dimensionless values based on the reference conditions specified in the solver control file.
- The sample file on the following page is for a two mode solution.

APPENDIX B: Summary of 3-D File Formats

This Appendix provides a summary of the input/output file formats used by the three-dimensional CFD solver developed for this research.

The following *input* files are defined:

- `case.g3d` (required) contains the geometry data structures representing the computational volume mesh as required by the flow solver. (Binary)
- `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, and pressure) 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 dynamic solution. (ASCII)
- `case.vec` (optional) contains the elastic mode matrices, initial conditions, and vectors for the boundary 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)

The following *output* files are defined:

- `case.un1` contains the nodal values of the primitive flow variables (density, velocity, and pressure) for each node of the computational mesh. (Binary)
- `case.rsd` contains a history of the solution residuals for the conservation variables (density, momentum, and total energy). (ASCII)
- `case.lds` contains a history of the dimensionless aerodynamic forces acting on the solid walls of the CFD 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 and velocities for an unsteady, aeroelastic solution. (ASCII)

Geometry Input File (case.g3d)

Basic File Format

```
nnd nel nsg nbe nbp nwl nsd nsf
( LBE(i), i = 1,6 )
( ( COOR(i,j), i=1,nnd ), j=1,3 )
( ( IELM(i,j), i=1,nel ), j=1,4 )
( ( ISEG(i,j), i=1,nsg ), j=1,2 )
( ( IBEL(i,j), i=1,nbe ), j=1,4 )
```

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) boundary element
starting/stopping indexes for three 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) boundary surface containing
boundary element *i*

Comments

- This is an unformatted (binary) file.
- Nodal data 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 specified in the solver control file.
- 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 `makeg3d` is used to convert a standard STARS surface triangulation file, tetrahedral volume file, and modified boundary conditions file into an appropriately sorted three-dimensional geometry file.

Solver Control 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        = 3,
  isol        = 0,
  idsol       = 2,
  idiss       = 0,
  ipnt        = 1,

  istrtr      = .false.,
  iaero       = .false.,
  idynm       = .false.,
  ielast      = .false.,
  ifree       = .true.,
  iforce      = .true.,

  nr          = 0,
  ainf        = 1.0d0,
  rhoinf      = 1.0d0,
/
```

Definition of Terms

dt : (real) dimensionless global time step
gamma : (real) ratio of specific heats
diss : (real) dissipation constant
cfl : (real) local time step stability factor

mach : (real) free-stream mach number
alpha : (real) 1st free-stream orientation angle
beta : (real) 2nd free-stream orientation angle
refdim : (real) reference dimension

nstp : (int) total solution steps
nout : (int) output frequency, steps/output
ncyc : (int) iterative cycles per solution step
isol : (int) CFD solution type
idsol : (int) dynamics solution type
idiss : (int) dissipation type
ipnt : (int) number of points for numerical integration of flux/source vectors

istrtr : (logical) restart flag
iaero : (logical) aerodynamic forces flag
idynm : (logical) dynamic/non-inertial flag
ielast : (logical) elastic flag
ifree : (logical) free-stream velocity flag
iforce : (logical) external force flag

nr : (int) number of elastic modes
ainf : (real) dimensional free-stream sonic speed
rhoinf : (real) dimensional free-stream density

Comments

- This is a plain text (ASCII) file formatted as a Fortran namelist.
- The default values for each parameter are given in the basic file format above.
- The global time step is only used for unsteady solutions.
- 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 typically acceptable for most problems. For unsteady solutions, the stability factor is typically in the range $0.3 \leq \text{cfl} \leq 0.8$.

- The values of `refdim`, `mach`, `ainf`, and `rhoinf` are used to non-dimensionalize all values read in by the flow solver.
- The free-stream orientation angles are ignored for dynamic (non-inertial) problems.
- The number of iterative cycles should be set to 3 for steady solutions. For unsteady solutions, use a sufficient number of cycles to allow for an appropriate level of convergence at each step.
- 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 dynamics solution types defined as follows:
 - `idsol = 0` uses a zero-order integrator for the applied forces
 - `idsol = 1` uses a first-order integrator for the applied forces
 - `idsol = 2` uses a second-order integrator for the applied forces
- There are two available dissipation types defined as follows:
 - `idiss = 0` is a low order dissipation
 - `idiss = 1` is a high order dissipation with gradient limiters
- The low-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 high-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.
- There are two types of numerical integration defined as follows:
 - `ipnt = 1` uses a one-point gauss quadrature
 - `ipnt = 4` uses a three-point symmetric gauss quadrature
- When the restart flag 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 free-stream velocity flag 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 the external force 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 global time steps if it was non-zero.

Solution Unknowns Input/Output File (case.un*)

Basic File Format

```
nnd gam xmi alp bet ref t
( ( UN(i,j), i=1,nnd ), j=1,6 )
```

Definition of Terms

nnd: (int) number of nodes
gam: (real) ratio of specific heats
xmi: (real) free stream mach number
alp: (real) 1st free-stream orientation angle
bet: (real) 2nd free-stream orientation 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) 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 this file are relative quantities referenced to the body-fixed coordinate system.

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=1,6 )
```

Definition of Terms

R0 (1) : (real) x-coordinate for origin of rotation

R0 (2) : (real) y-coordinate for origin of rotation

R0 (3) : (real) z-coordinate 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 coord. system

y : (real) initial y-position of coord. system

z : (real) initial z-position of coord. system

p : (real) initial roll angle of coord. system

q : (real) initial pitch angle of coord. system

r : (real) initial yaw angle of coord. system

vx : (real) initial x-velocity of coord. system

vy : (real) initial y-velocity of coord. system

vz : (real) initial z-velocity of coord. system

vp : (real) initial roll rate of coord. system

vq : (real) initial pitch rate of coord. system

vr : (real) initial yaw rate of coord. system

ax : (real) initial x-acceleration of coord. system

ay : (real) initial y-acceleration of coord. system

az : (real) initial z-acceleration of coord. system

ap : (real) initial roll accel. of coord. system

aq : (real) initial pitch accel. of coord. system

ar : (real) initial yaw accel. of coord. system

IBXD (i) : (int) dynamics flag for i^{th} 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 vector defining the origin of rotation is subtracted directly from the nodal coordinates defined in the geometry input file after it is non-dimensionalized by the reference dimension.
- The mass matrix defined in this file cannot be singular.
- Initial conditions for 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.

- Initial conditions for the three rotational degrees of freedom should have units of degrees, degrees/sec, etc., and are true Euler angles and rates expressed relative to the body-fixed coordinate system.
- 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 with zero velocity and acceleration.
 - IBXD = 2 is a constant acceleration, uncoupled response, i.e. integrates acceleration and velocity to compute new position.

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)
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
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
$ 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 accels )
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 (6)
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,nwl*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

IBXN(i): (int) dynamics flag for i^{th} mode

PHIA(i*3-2,j): x -component of displacement vector for mode j at node i

PHIA(i*3-1,j): y -component of displacement vector for mode j at node i

PHIA(i*3 ,j): z -component of 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 mass matrix 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 generalized displacement and velocity.
 - IBXN = 1 is a clamped condition, i.e. hold at initial generalized displacement with zero velocity.
 - IBXN = 2 is a constant velocity, uncoupled response, i.e. integrates generalized velocity to compute new displacement.

- $IBXN = 3$ is a forced multistep response used for system identification purposes.
- Do not combine $IBXN = 0$ with $IBXN \neq 0$ for different modes if there are coupling or off-diagonal terms in the mass, damping or stiffness matrices.
- A limited set of simple modal vectors representing standard rigid-body degrees of freedom can be generated using the program `makevec3d`.

Sample File

```
$ Number of elastic modes (nr)
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)
0.0d0    0.0d0    0.0d0
0.0d0    0.0d0    0.0d0
0.0d0    0.0d0    0.0d0
$ 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*3) x nr
0.0d0  1.0d0  1.0d0
0.0d0  1.0d0  1.0d0
0.0d0  1.0d0  1.0d0
0.0d0  1.0d0  1.0d0
1.0d0  0.0d0  1.0d0
0.0d0  1.0d0  1.0d0
0.0d0  1.0d0  1.0d0
  ⋮      ⋮      ⋮
```

External Force Input File (case . frc)

Basic File Format

```
1 (FD(i), i=1,6) (FE(j), j=1,nr)
:      :      :
istp (FD(i), i=1,6) (FE(j), j=1,nr)
:      :      :
nstp (FD(i), i=1,6) (FE(j), j=1,nr)
```

Definition of Terms

istp: (int) current solution step
nstp: (int) total or last solution step
nr: (int) number of elastic modes

FD (1) : (real) x force applied at *istp*
FD (2) : (real) y force applied at *istp*
FD (3) : (real) z force applied at *istp*
FD (4) : (real) roll moment applied at *istp*
FD (5) : (real) pitch moment applied at *istp*
FD (6) : (real) yaw moment applied at *istp*
FE (j) : (real) force applied to elastic mode j

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.
- Forces applied to the three translational degrees of freedom are specified relative to the inertial coordinate system.
- 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.

Sample File

1	0.0d0	0.0d0	0.0d0	0.0d0	0.0d0	0.0d0
2	0.0d0	0.0d0	0.0d0	0.0d0	0.0d0	0.0d0
3	0.0d0	0.0d0	1.0d0	0.0d0	1.0d0	0.0d0
4	0.0d0	0.0d0	1.0d0	0.0d0	2.0d0	0.0d0
5	0.0d0	0.0d0	1.0d0	0.0d0	1.0d0	0.0d0
6	0.0d0	0.0d0	1.0d0	0.0d0	0.0d0	0.0d0
7	0.0d0	0.0d0	0.0d0	0.0d0	0.0d0	0.0d0

Solution 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 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	0.12222E-07	0.10626E-07	0.16687E-07	0.12295E-07	0.22126E-07
2	0.11570E-07	0.10062E-07	0.15729E-07	0.11602E-07	0.20890E-07
3	0.10921E-07	0.95097E-08	0.14773E-07	0.10908E-07	0.19662E-07
4	0.10275E-07	0.89463E-08	0.13814E-07	0.10215E-07	0.18436E-07
5	0.96242E-08	0.83818E-08	0.12854E-07	0.95216E-08	0.17203E-07
6	0.89786E-08	0.78260E-08	0.11892E-07	0.88276E-08	0.15977E-07
7	0.83298E-08	0.72699E-08	0.10929E-07	0.81336E-08	0.14748E-07
8	0.76831E-08	0.67176E-08	0.99662E-08	0.74405E-08	0.13527E-07
9	0.70441E-08	0.61708E-08	0.90087E-08	0.67538E-08	0.12320E-07
10	0.64162E-08	0.56360E-08	0.80584E-08	0.60769E-08	0.11139E-07
11	0.57970E-08	0.51153E-08	0.71208E-08	0.54130E-08	0.99765E-08
12	0.51854E-08	0.46001E-08	0.61957E-08	0.47627E-08	0.88260E-08
13	0.45869E-08	0.40968E-08	0.52837E-08	0.41320E-08	0.76967E-08
14	0.40076E-08	0.36189E-08	0.43870E-08	0.35263E-08	0.66145E-08
15	0.34579E-08	0.31776E-08	0.35119E-08	0.29571E-08	0.56028E-08
16	0.29562E-08	0.27873E-08	0.26754E-08	0.24518E-08	0.47029E-08
17	0.25236E-08	0.24650E-08	0.19320E-08	0.20618E-08	0.39779E-08
18	0.21945E-08	0.22406E-08	0.14353E-08	0.18541E-08	0.35322E-08
19	0.20227E-08	0.21516E-08	0.14507E-08	0.18860E-08	0.34857E-08
20	0.20517E-08	0.22121E-08	0.19558E-08	0.21498E-08	0.38474E-08
⋮	⋮	⋮	⋮	⋮	⋮

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_i: (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 in this output file are dimensionless values based on the reference conditions specified in the solver control file.
- For dynamic (non-inertial) problems, the force coefficients stored in this file are referenced to the body-fixed coordinate system.

Dynamic Output File (xd.dat)

Basic File Format

```
0 0.0 (XD(i), i=1,12) (A0(i), i=1,6)
:   :   :           :
istp t_istp (XD(i), i=1,12) (A0(i), i=1,6)
:   :   :           :
nstp t_nstp (XD(i), i=1,12) (A0(i), i=1,6)
```

Definition of Terms

istp: (int) current solution step
nstp: (int) total or last solution step
 t_i : (real) dimensionless time at step i

XD (1) : (real) x-position
XD (2) : (real) y- position
XD (3) : (real) z- position
XD (4) : (real) roll angle
XD (5) : (real) pitch angle
XD (6) : (real) yaw angle
XD (7) : (real) x-velocity
XD (8) : (real) y- velocity
XD (9) : (real) z- velocity
XD (10) : (real) roll rate
XD (11) : (real) pitch rate
XD (12) : (real) yaw rate

A0 (1) : (real) x-acceleration
A0 (2) : (real) y- acceleration
A0 (3) : (real) z- acceleration
A0 (4) : (real) roll acceleration
A0 (5) : (real) pitch acceleration
A0 (6) : (real) yaw acceleration

Comments

- This is a plain text (ASCII) file.
- The dynamic data in this output file are dimensionless values based on the reference conditions specified in the solver control file.
- The position, velocity, and acceleration vectors in this file are defined relative to the global coordinate system, while the rotational vectors are defined as rotations about the local or body-fixed coordinate system.

Elastic Output File (xn.dat)

Basic File Format

```
0 0.0 (XN(i), i=1,nr*2) (FA(i), i=1,nr)
:   :   :   :
istp t_istp (XN(i), i=1, nr*2) (FA(i), i=1,nr)
:   :   :   :
nstp t_nstp (XN(i), i=1, nr*2) (FA(i), i=1,nr)
```

Definition of Terms

istp: (int) current solution step
nstp: (int) total or last solution step
t_i: (real) dimensionless time at step *i*

XN(i): (real) gen. displ. for mode *i*
XN(i+nr): (real) gen. vel. for mode *i*

FA(i): (real) gen. force for mode *i*

Comments

- This is a plain text (ASCII) file.
- The elastic data in this output file are dimensionless values based on the reference conditions specified in the solver control file.
- The sample file on the following page is for a two mode solution.

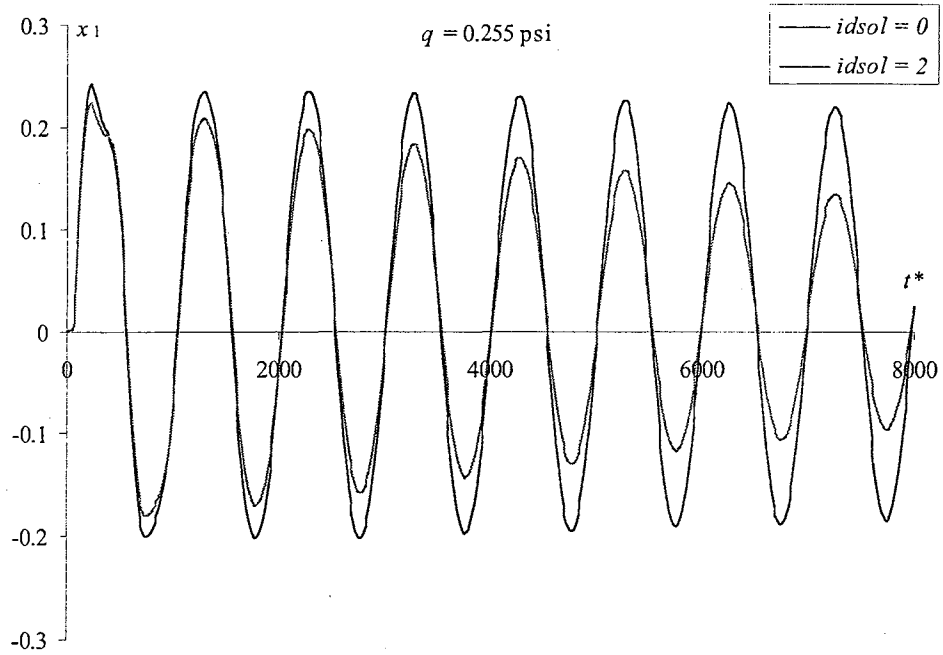
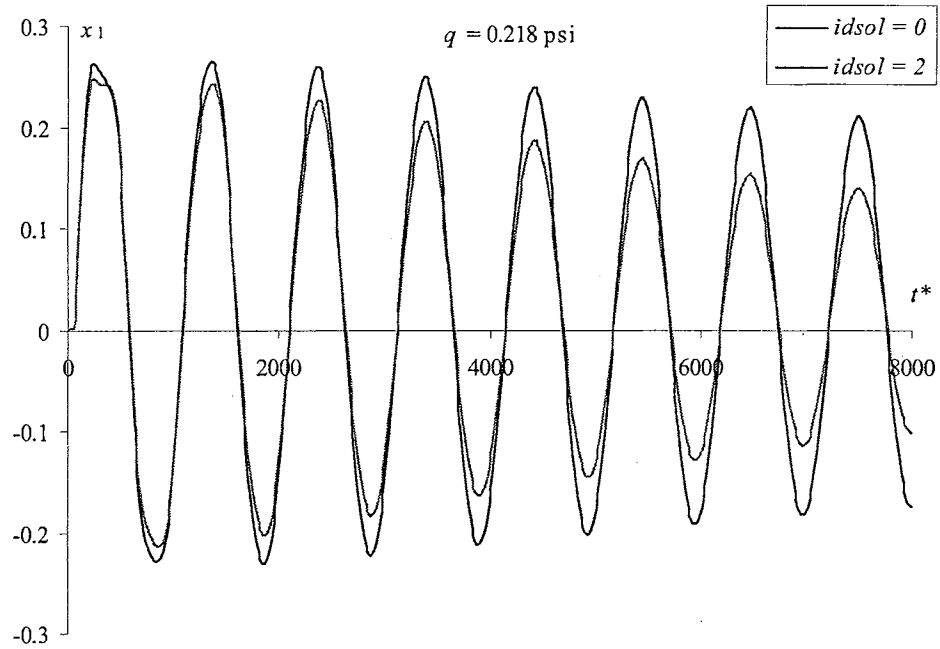
Sample File

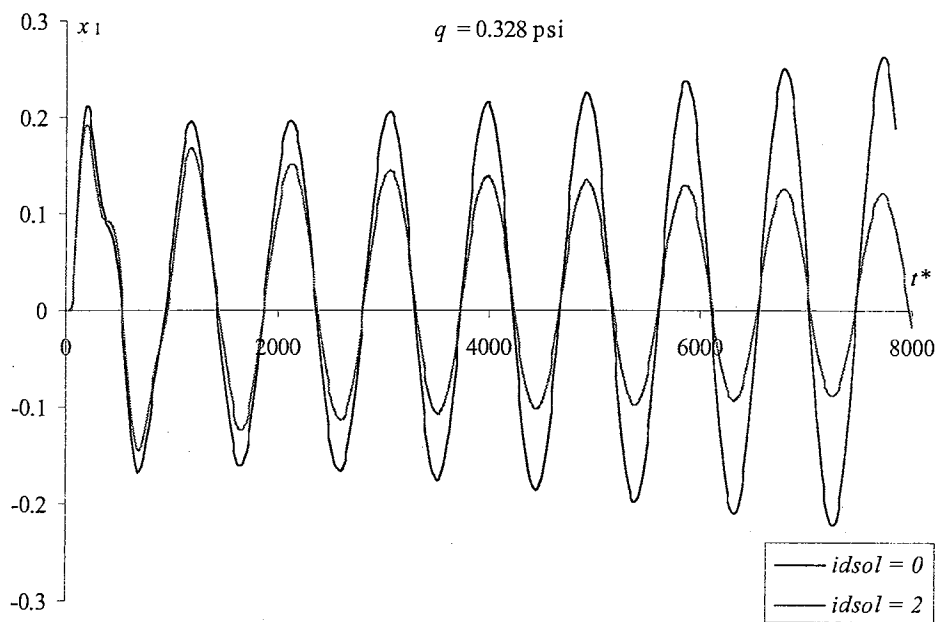
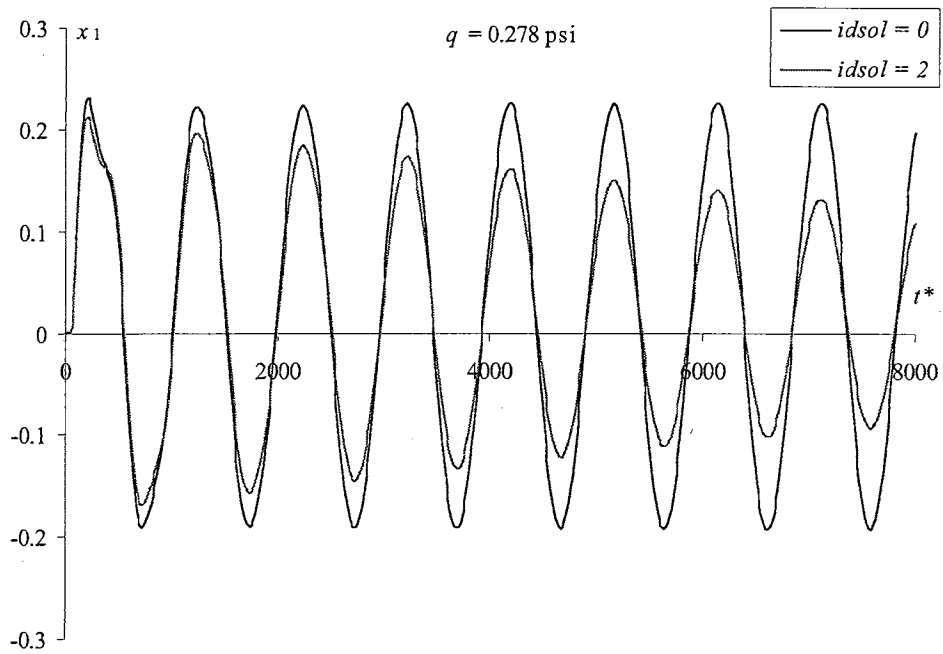
```
0 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
1 0.10000E-01 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
2 0.20000E-01 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
3 0.30000E-01 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
4 0.40000E-01 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
5 0.50000E-01 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
6 0.60000E-01 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
7 0.70000E-01 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
8 0.80000E-01 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
9 0.90000E-01 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
10 0.10000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
11 0.11000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
12 0.12000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
13 0.13000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
14 0.14000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
15 0.15000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
16 0.16000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
17 0.17000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
18 0.18000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
19 0.19000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
20 0.20000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
⋮           ⋮           ⋮           ⋮           ⋮           ⋮           ⋮
```

APPENDIX C: Time History Data For AGARD Wing

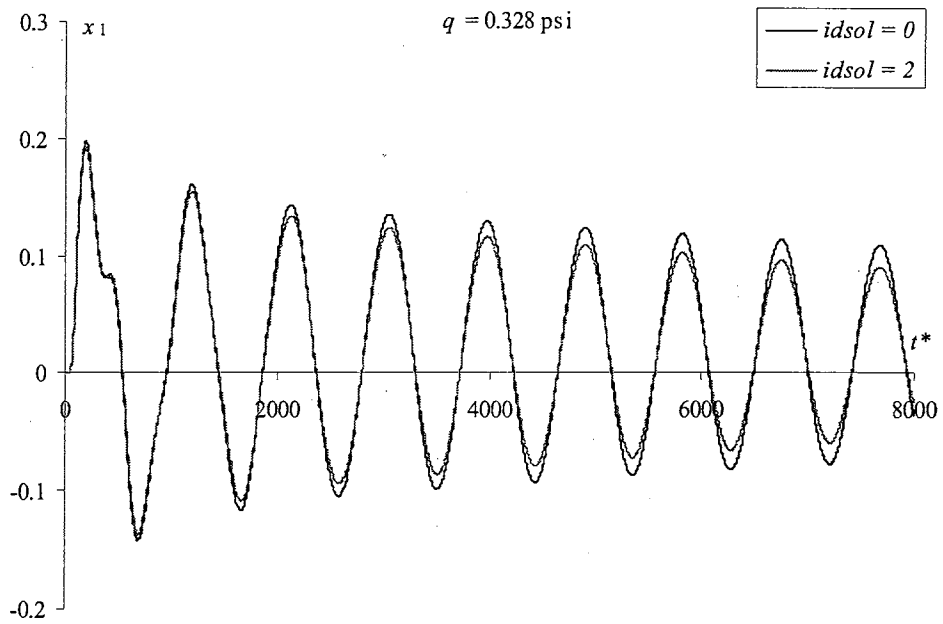
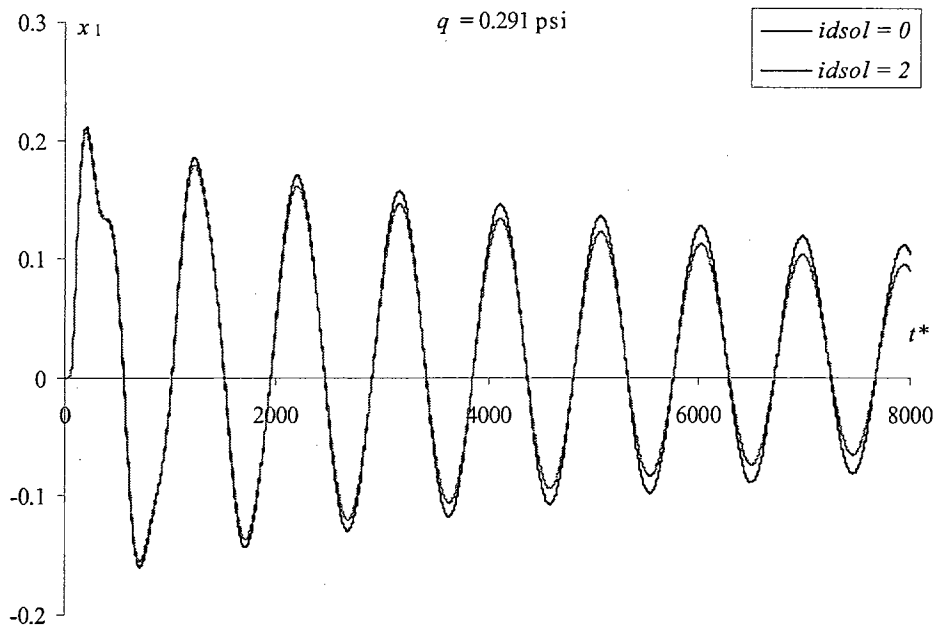
This Appendix provides a complete set of time history data for the AGARD wing at Mach 0.96 from Section 5.2.2. The time history data presented in this Appendix is for time steps ranging from $dt = 16.0$ through 0.25 using both the zero-order, $idsol = 0$, and second-order, $idsol = 2$, structural dynamics solvers.

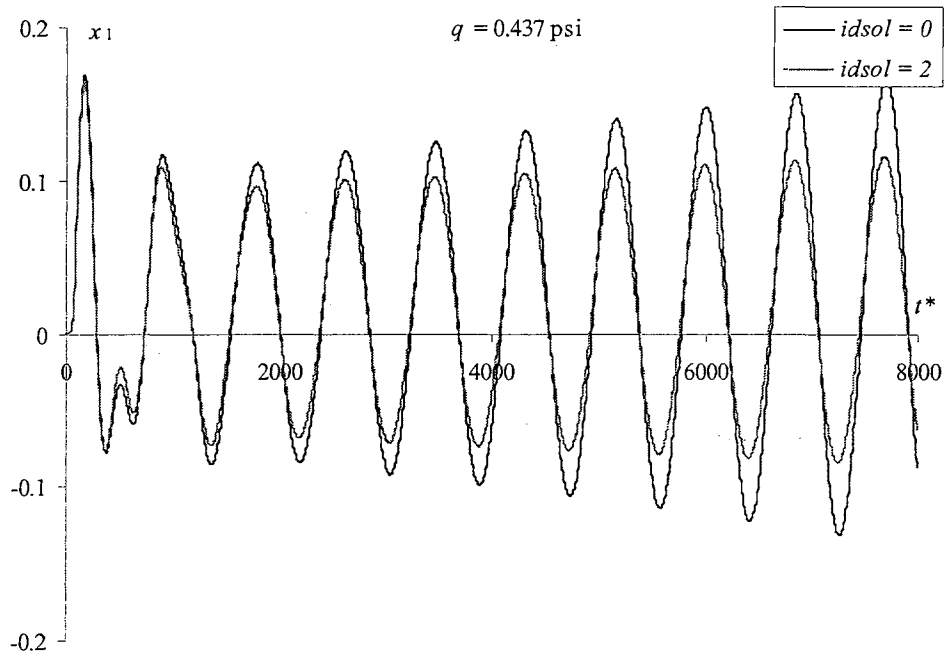
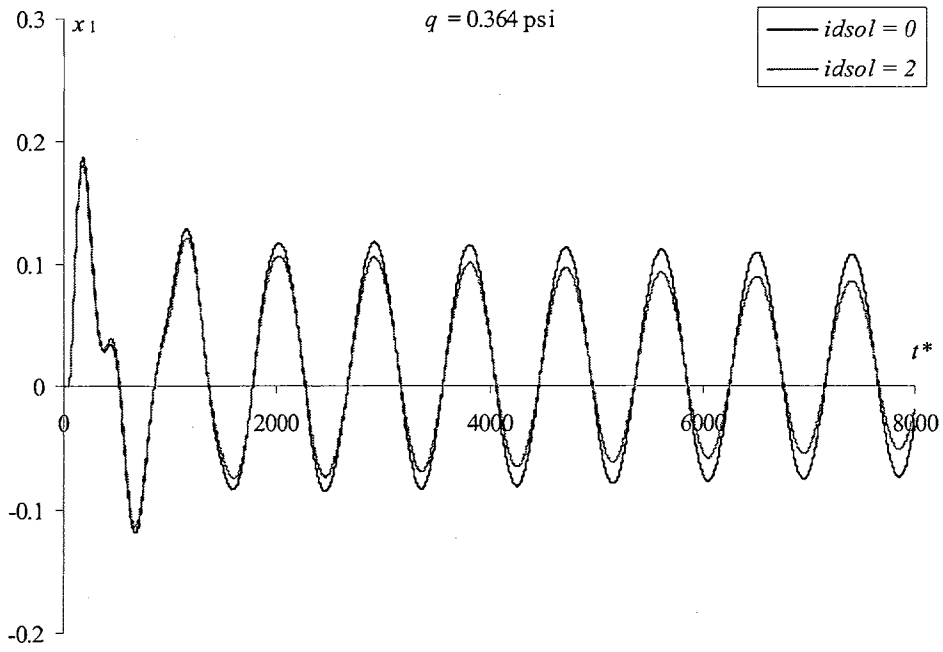
Time History Data for Mach 0.96, $dt = 16.0$



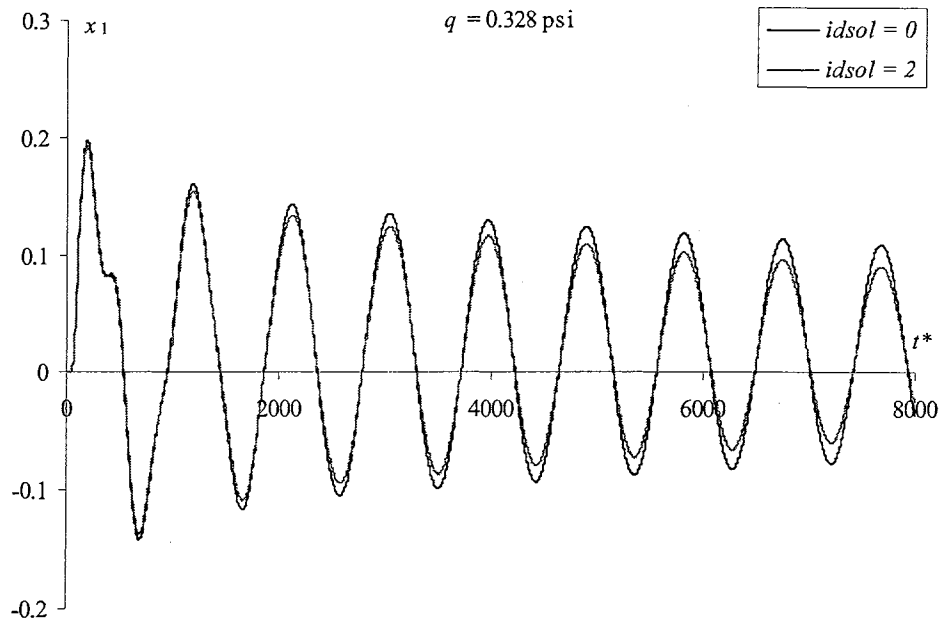
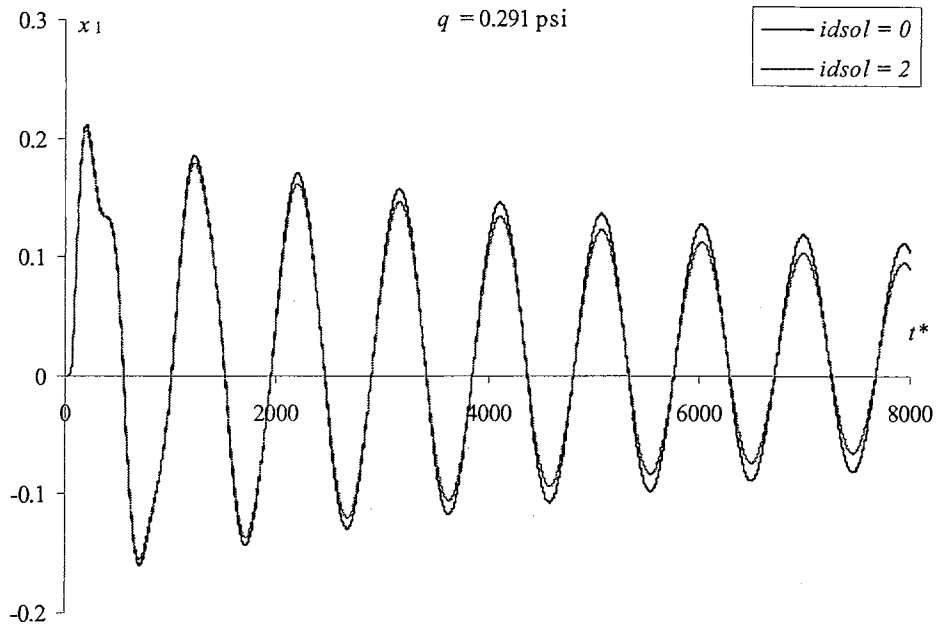


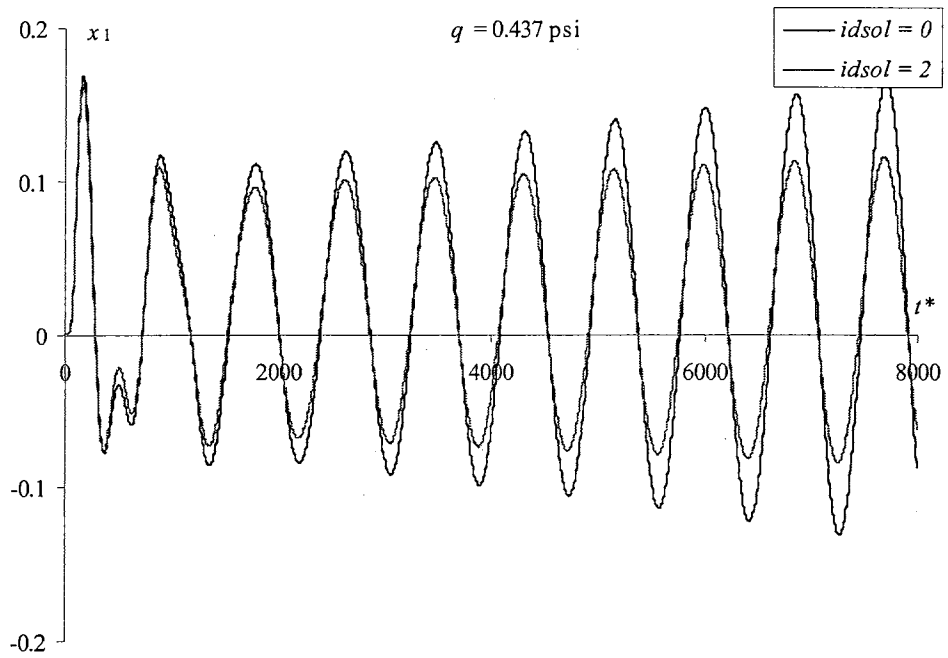
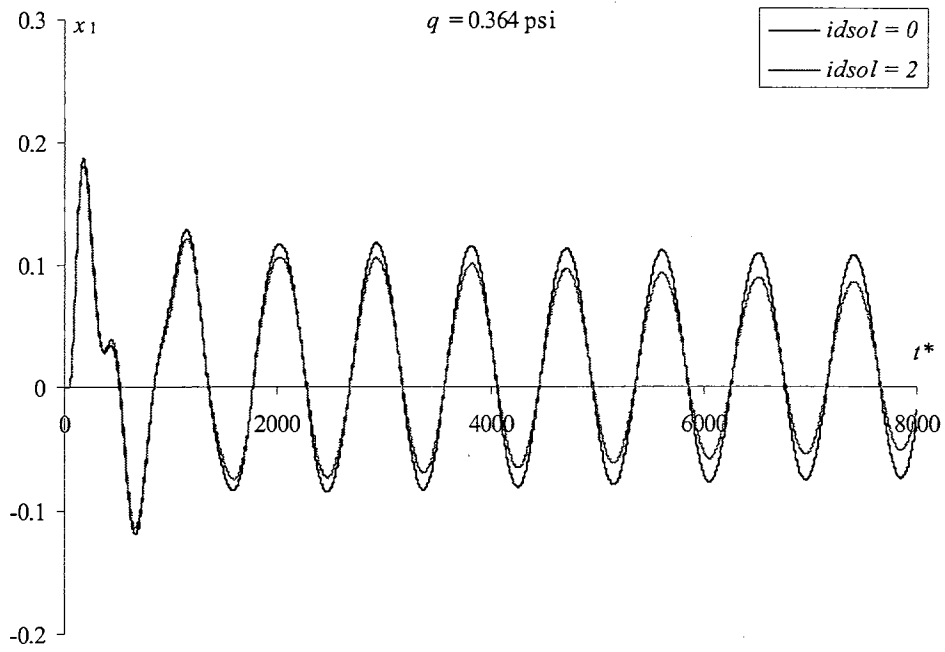
Time History Data for Mach 0.96, $dt = 4.0$



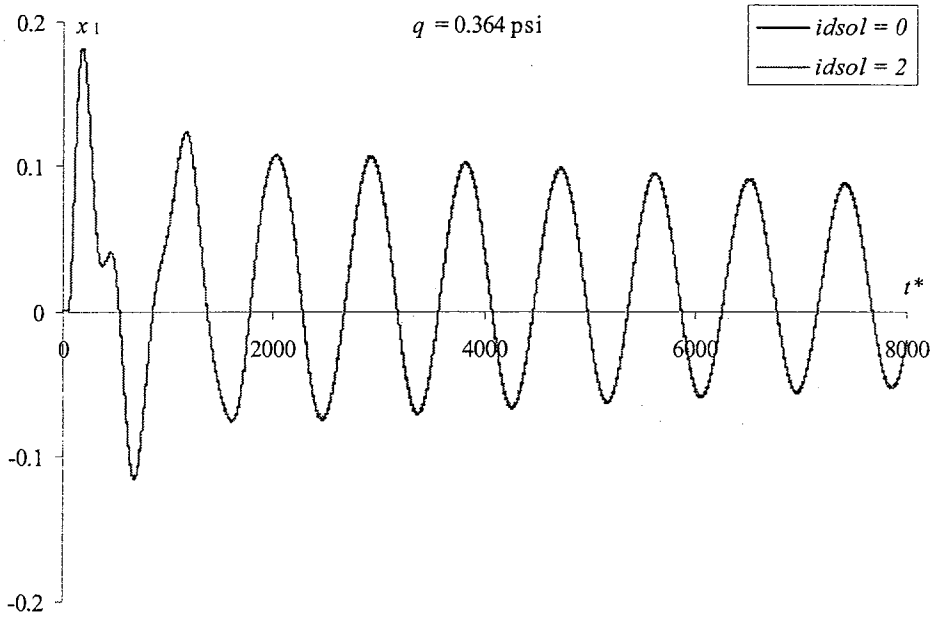
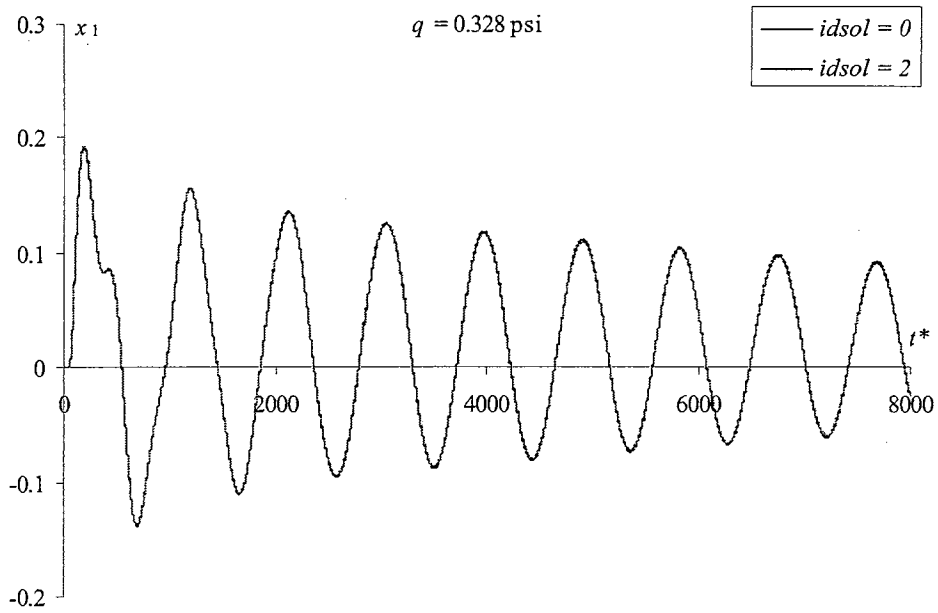


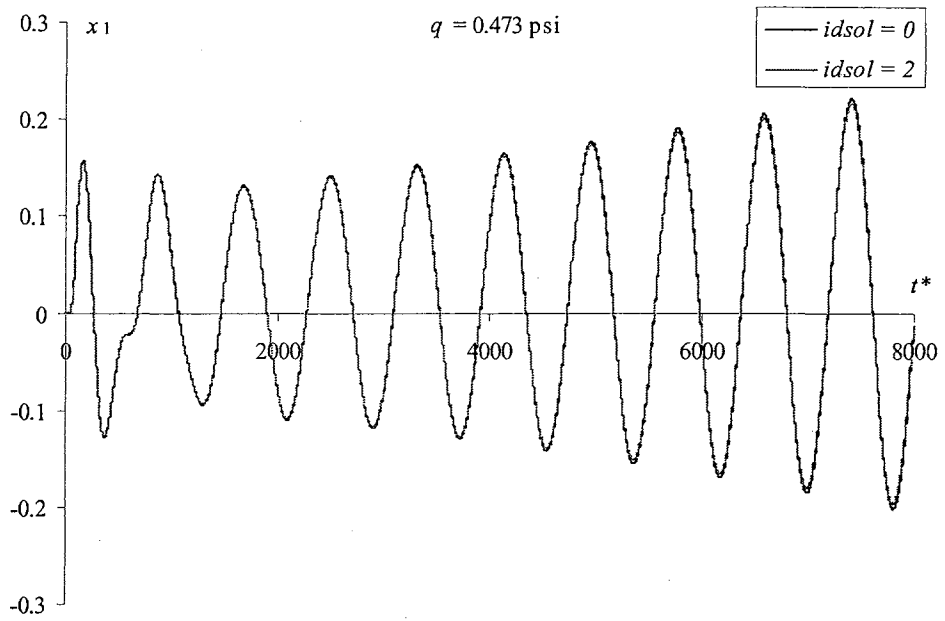
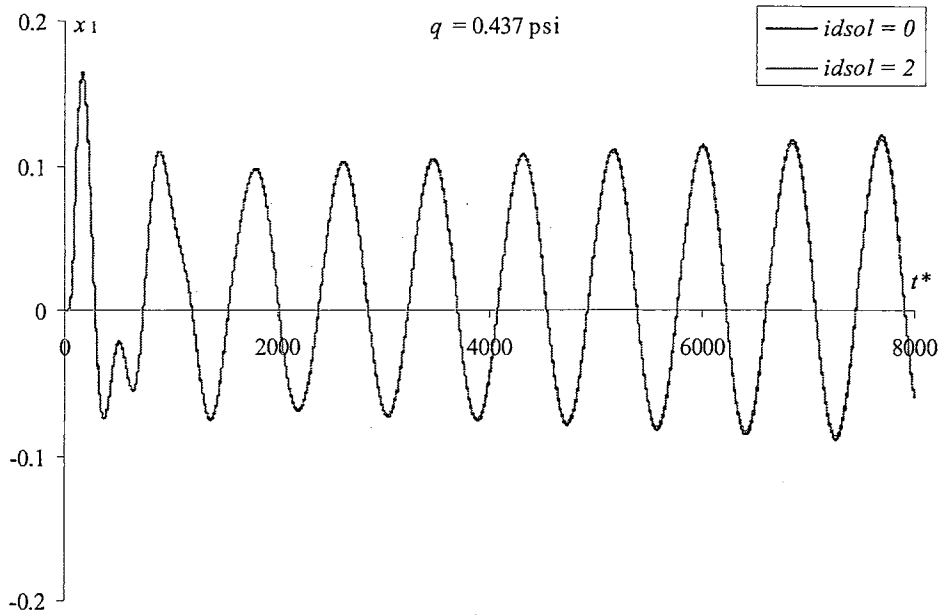
Time History Data for Mach 0.96, $dt = 1.0$





Time History Data for Mach 0.96, $dt = 0.25$

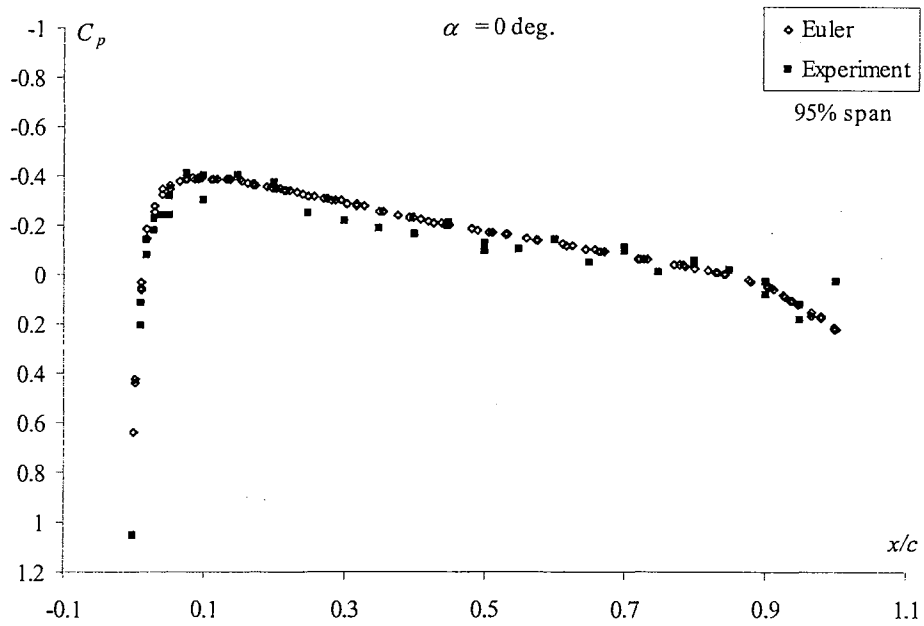
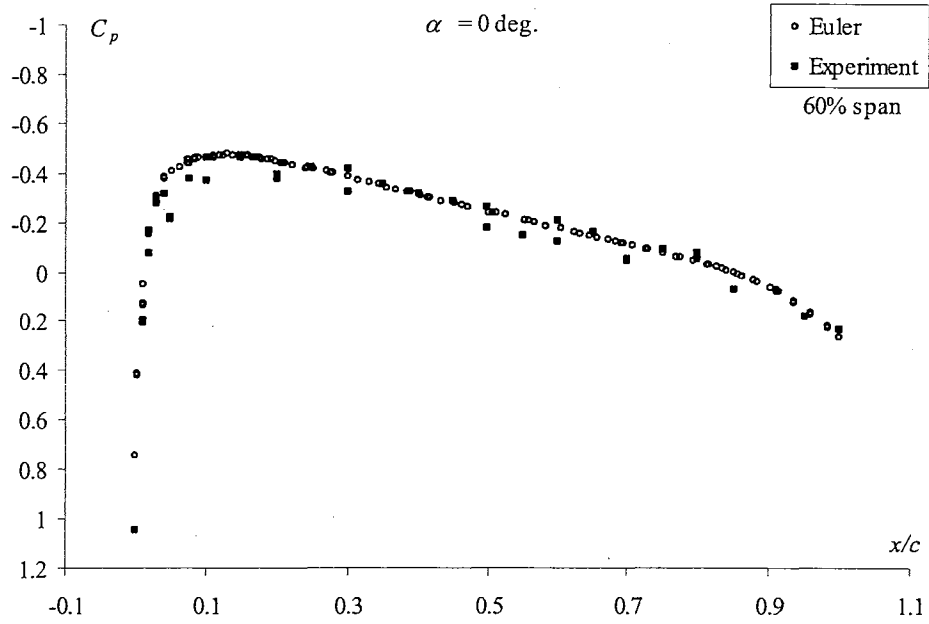


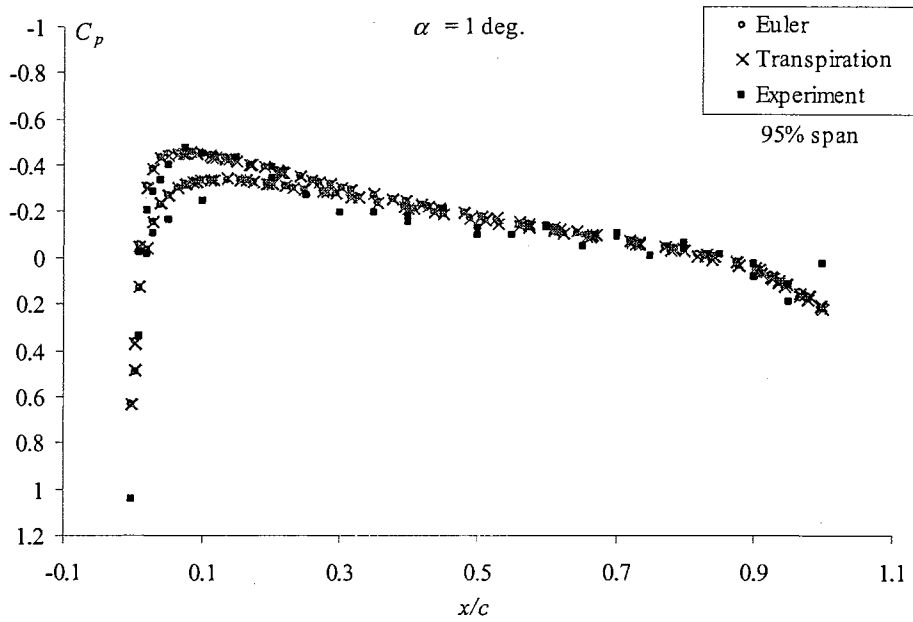
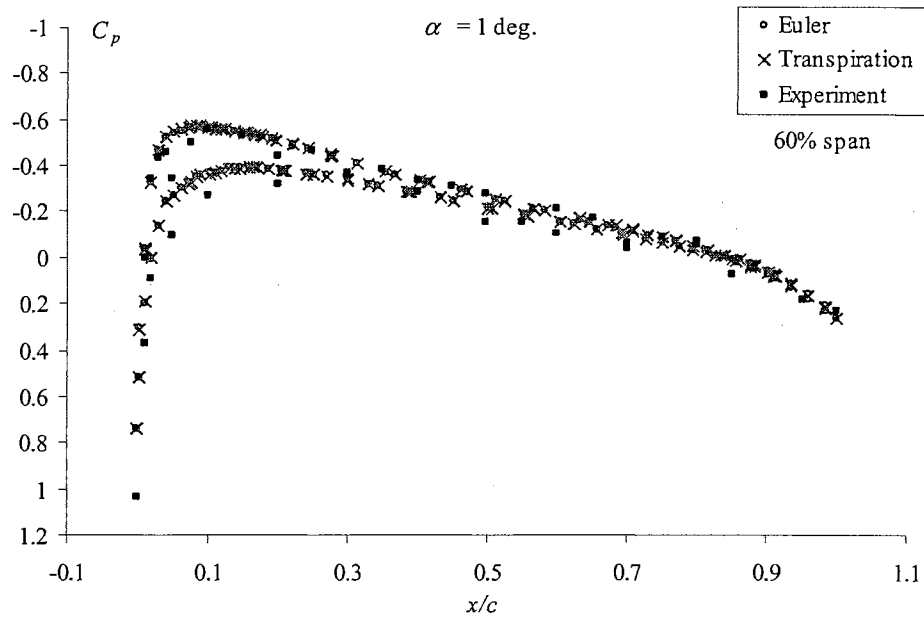


APPENDIX D: Steady Validation Data For BACT Wing

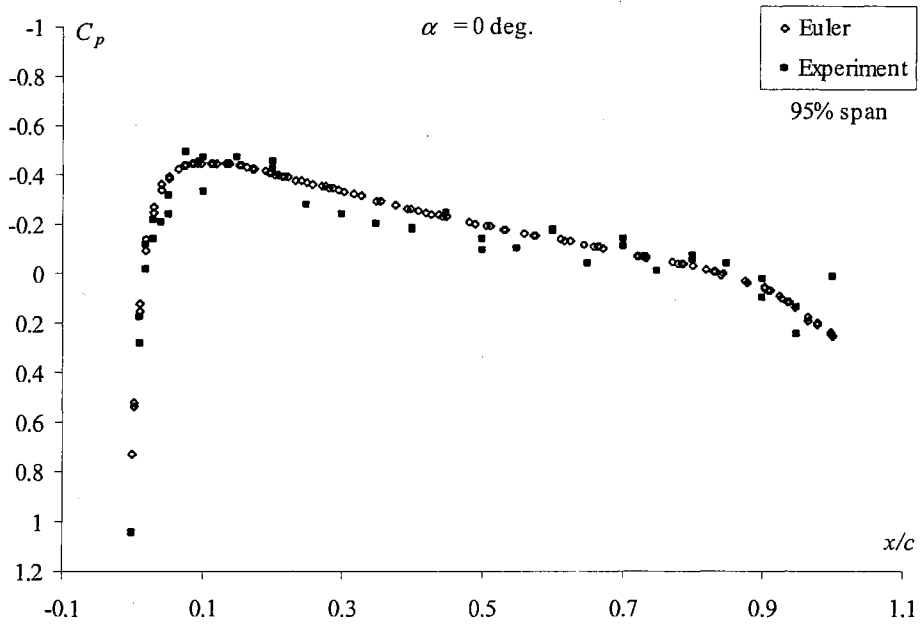
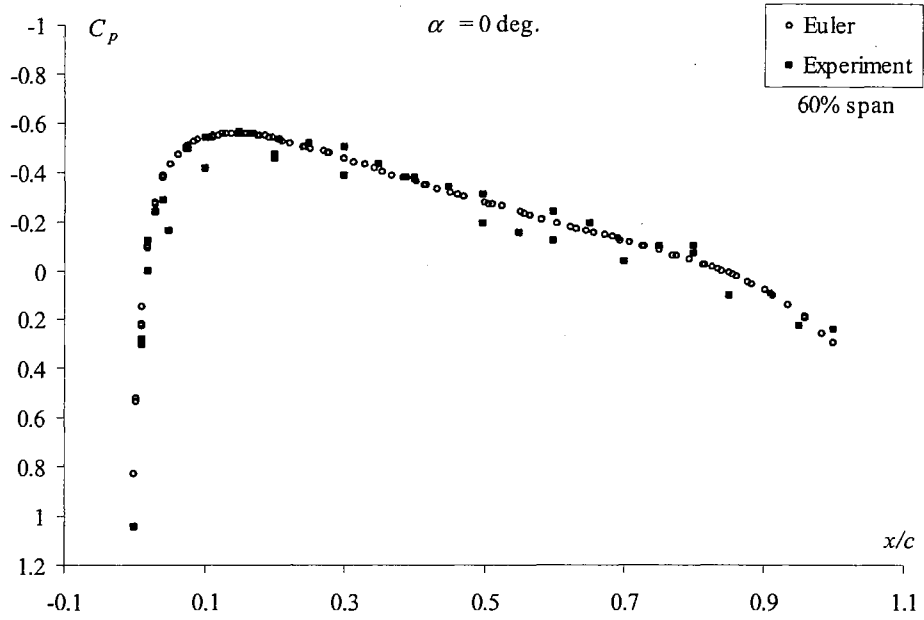
This Appendix provides a set of steady validation data for the BACT wing at each of the Mach numbers investigated in the unsteady aeroelastic analysis of Section 5.2.3. This data is important for two reasons. First, it demonstrates that the grid resolution is sufficient for an accurate representation of the flow field whether the solution is steady or unsteady. Second, steady data serves as the initial conditions for any unsteady solution that is computed. The data presented in this Appendix shows chordwise surface pressure distributions at two span locations for the wing, the 60% and 95% span locations. Each set of results is compared to the appropriate experimental data from reference 43.

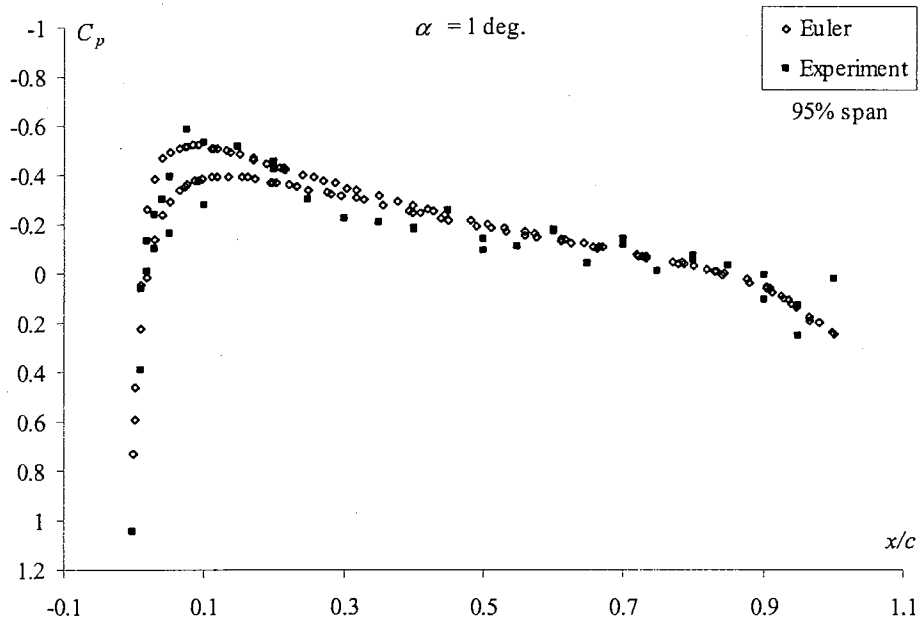
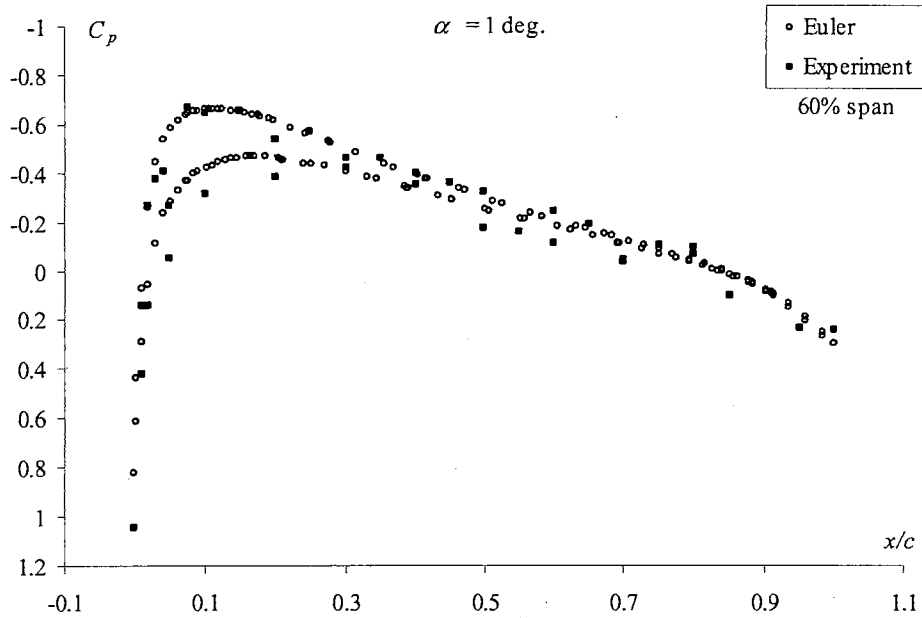
C_p Data for Mach 0.51



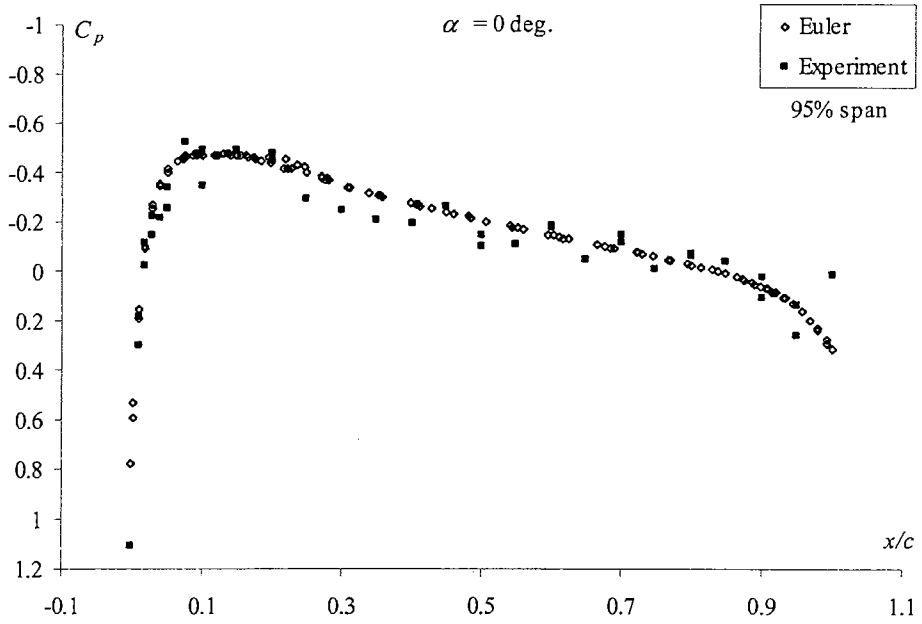
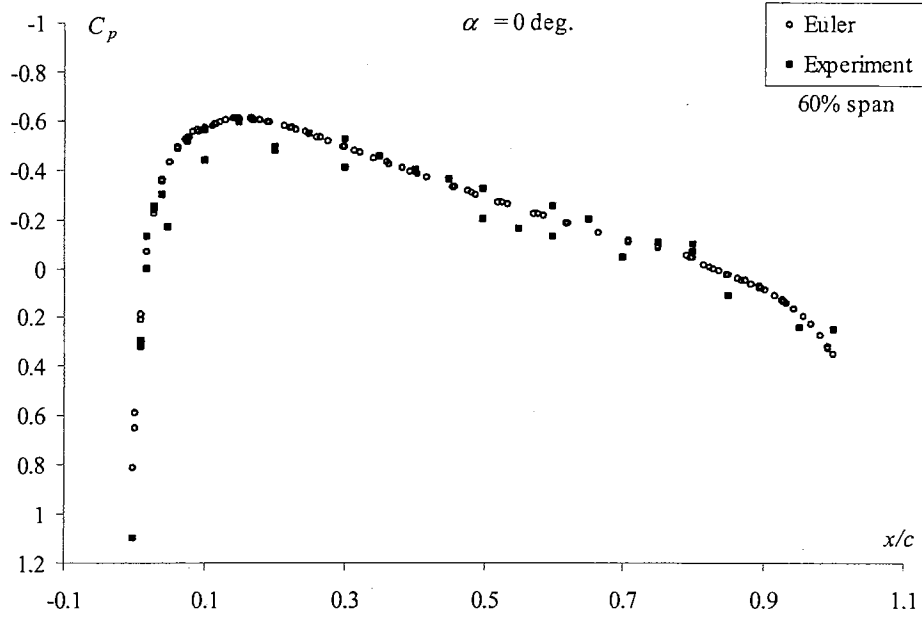


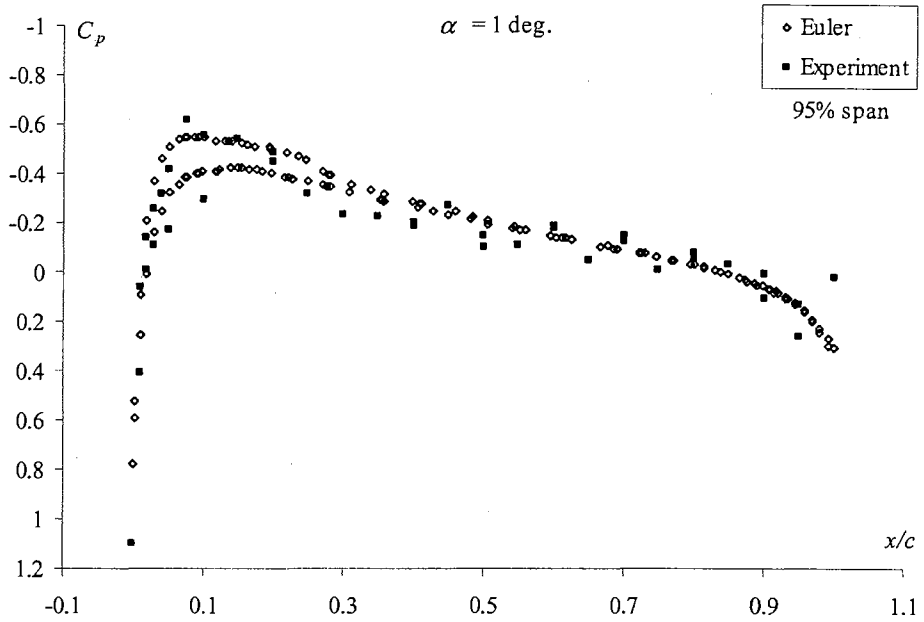
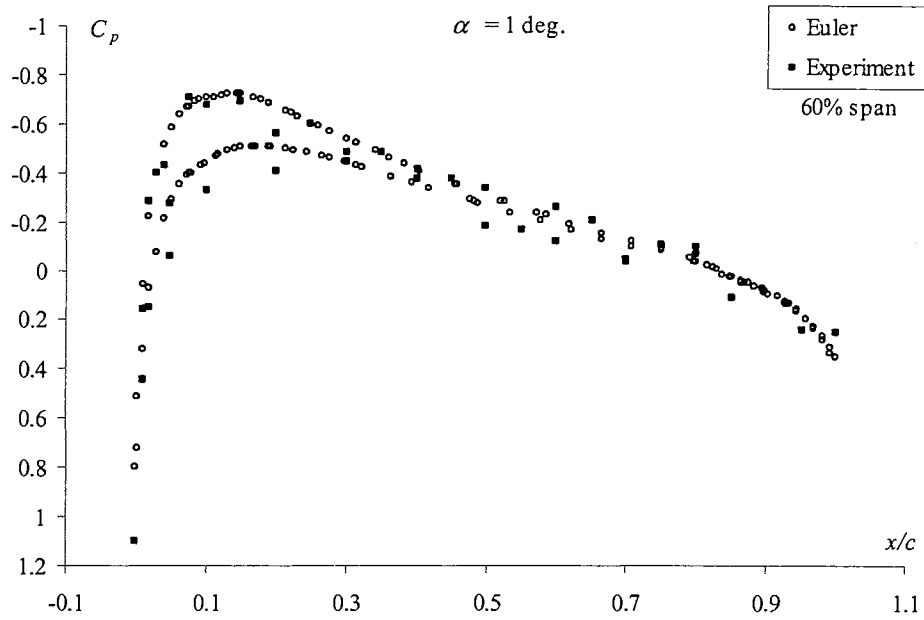
C_p Data for Mach 0.67



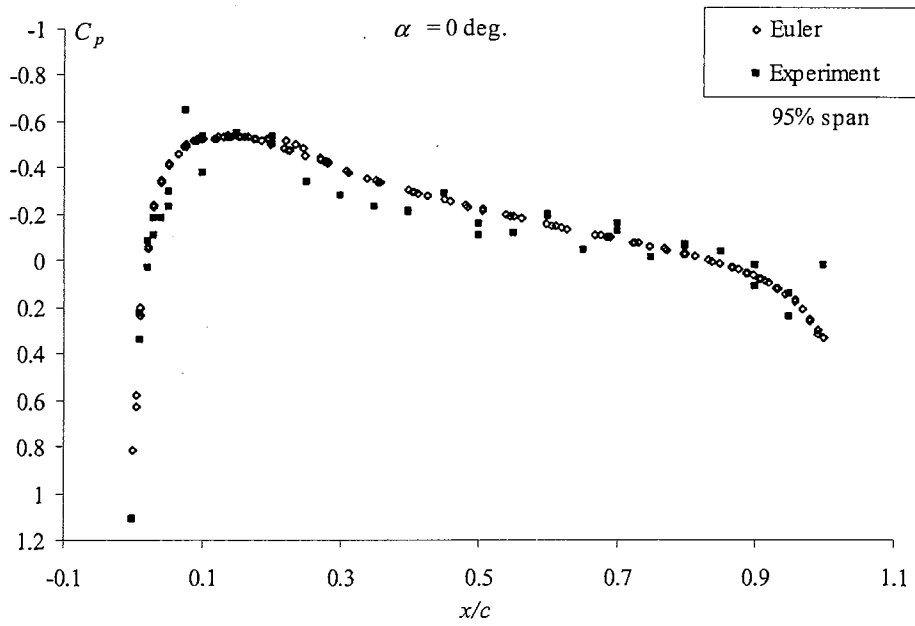
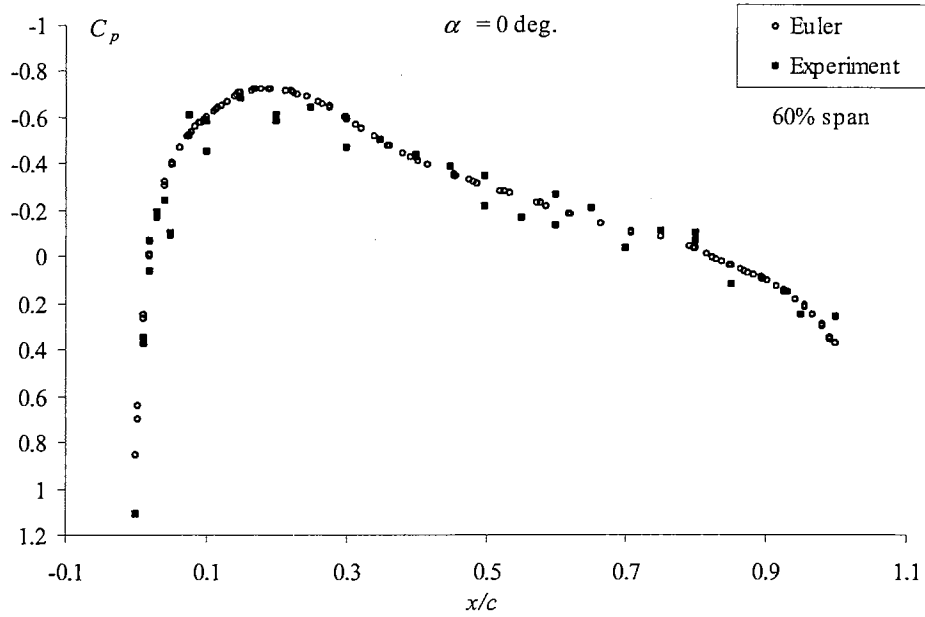


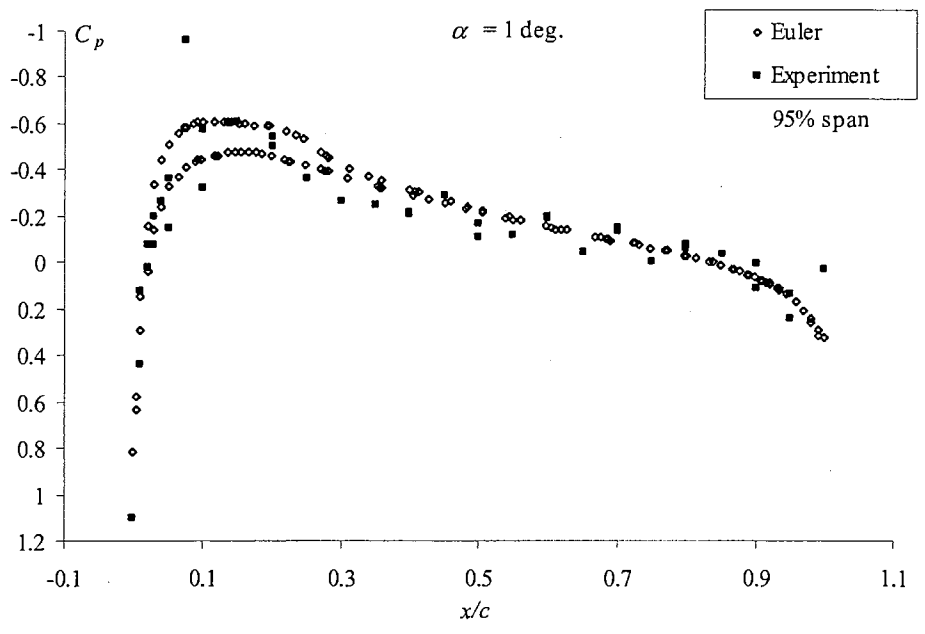
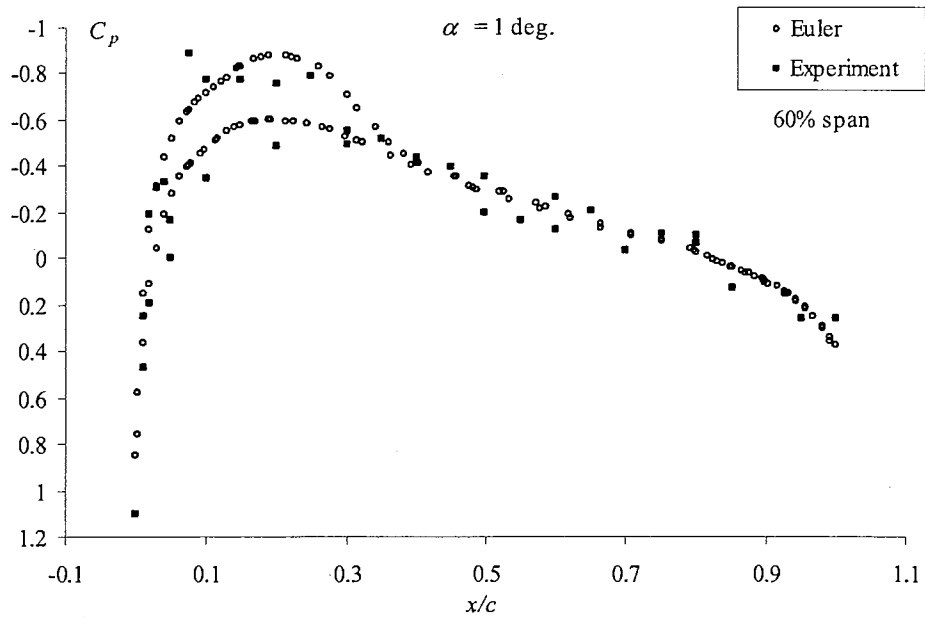
C_p Data for Mach 0.71



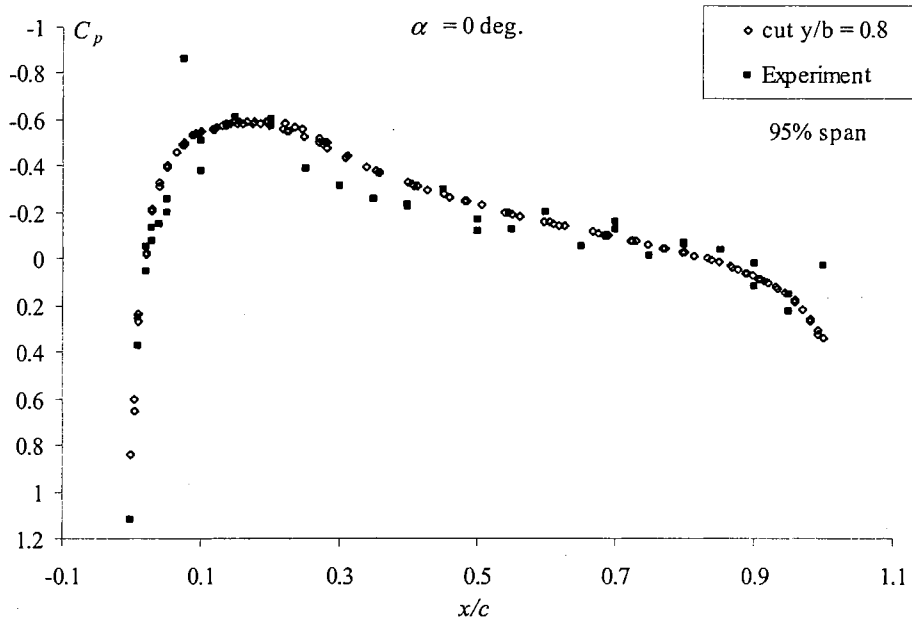
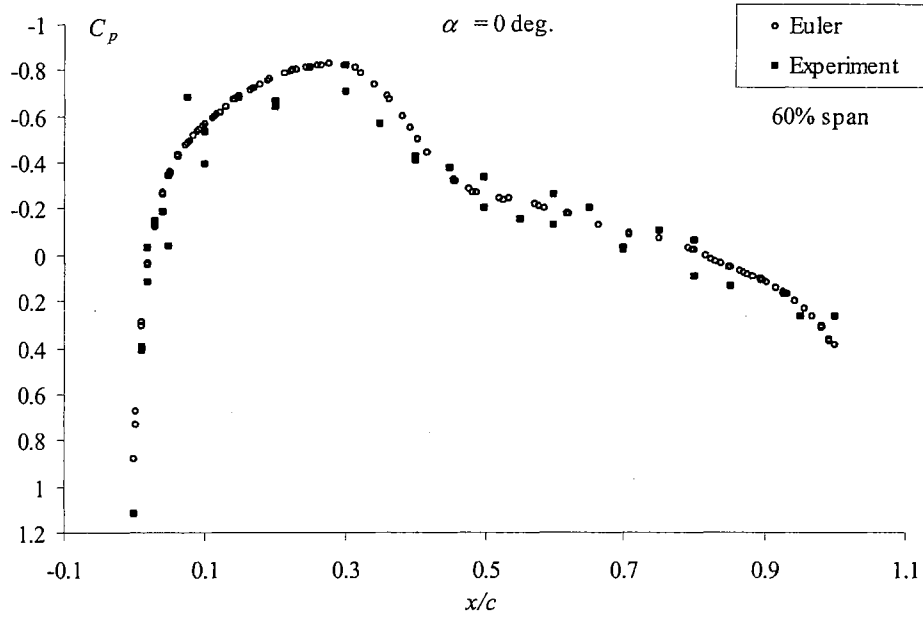


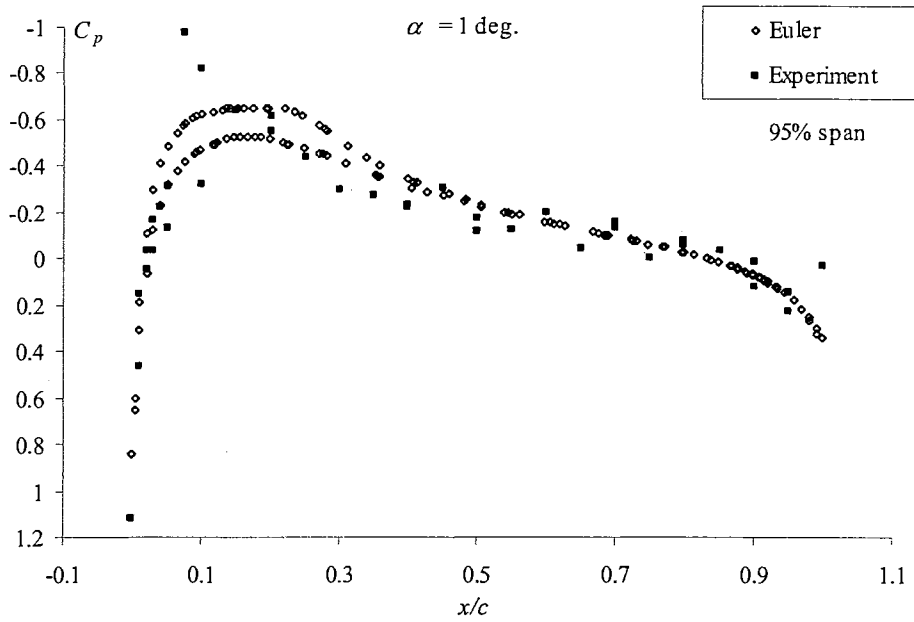
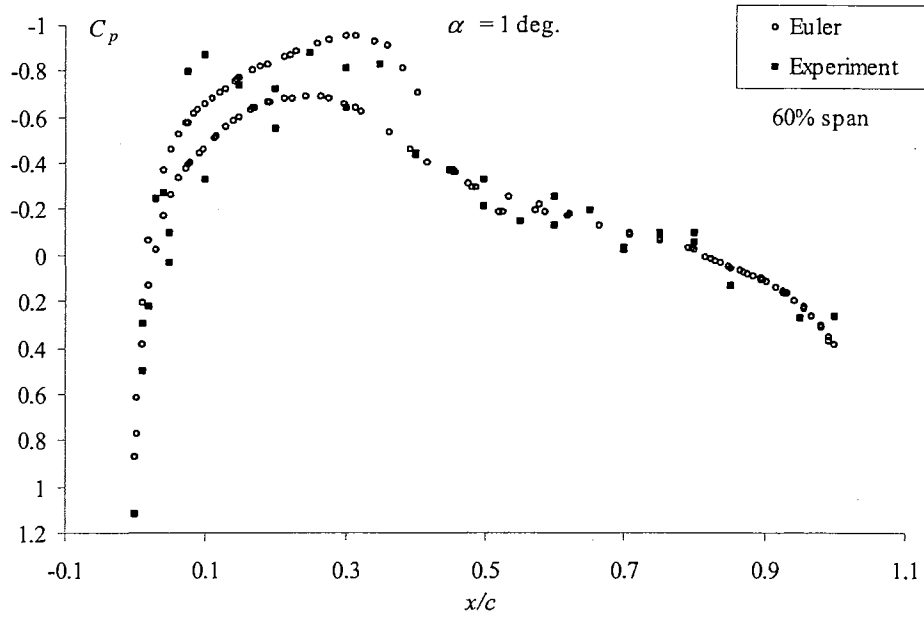
C_p Data for Mach 0.77



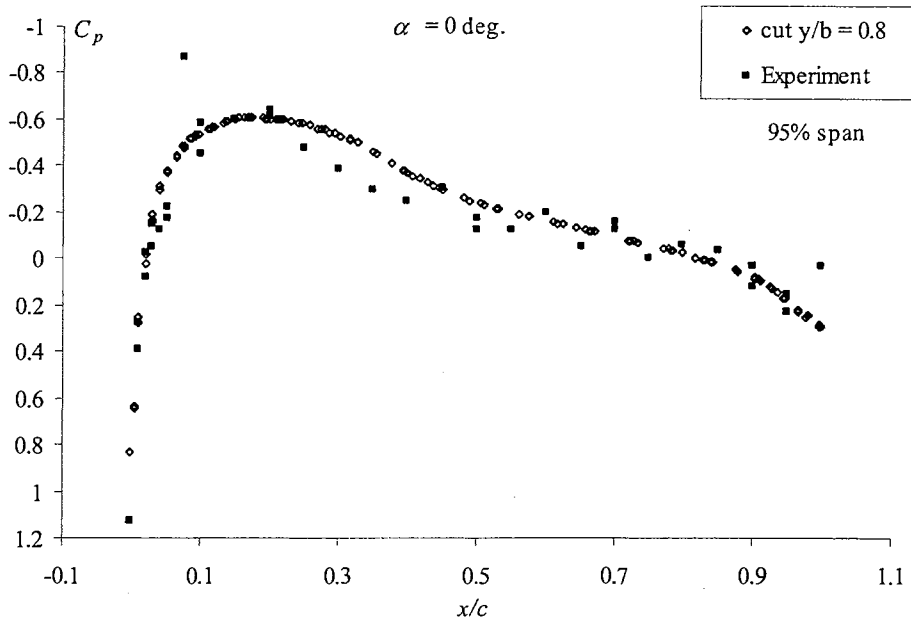
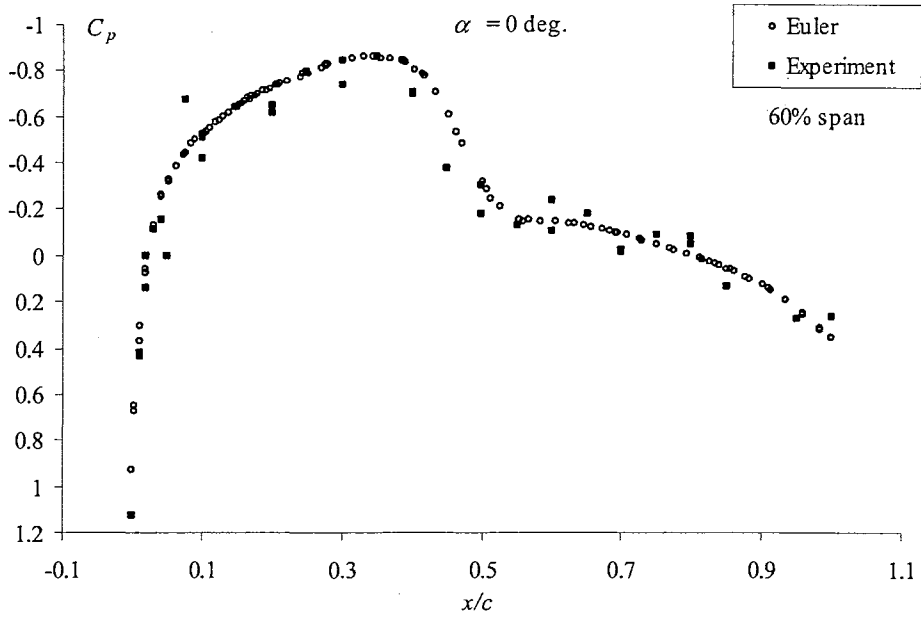


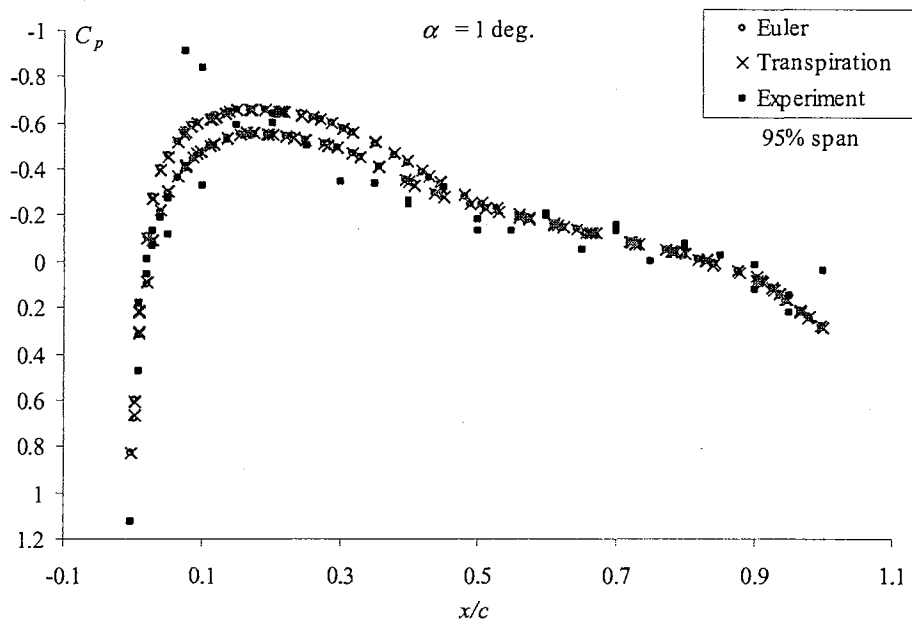
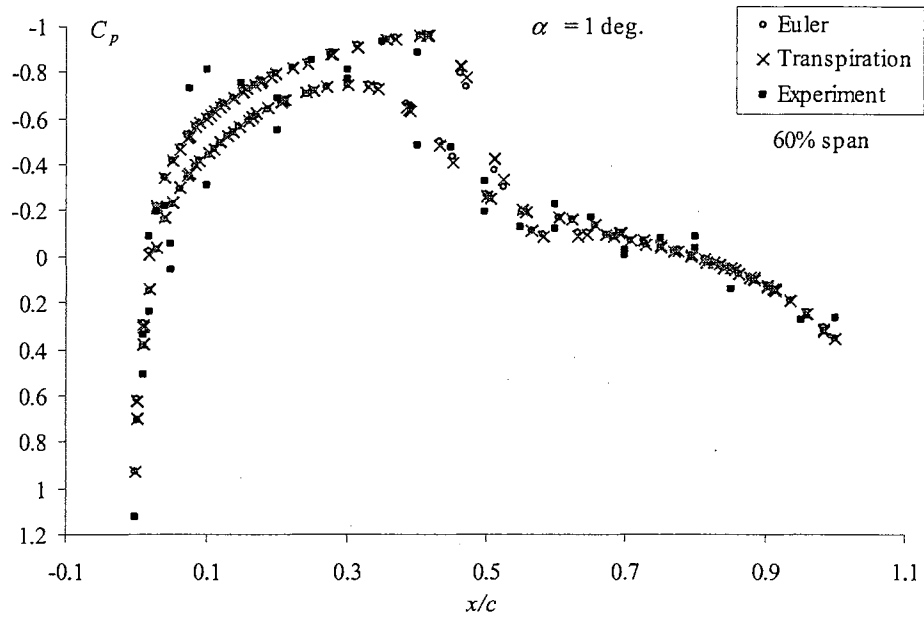
C_p Data for Mach 0.80





C_p Data for Mach 0.82

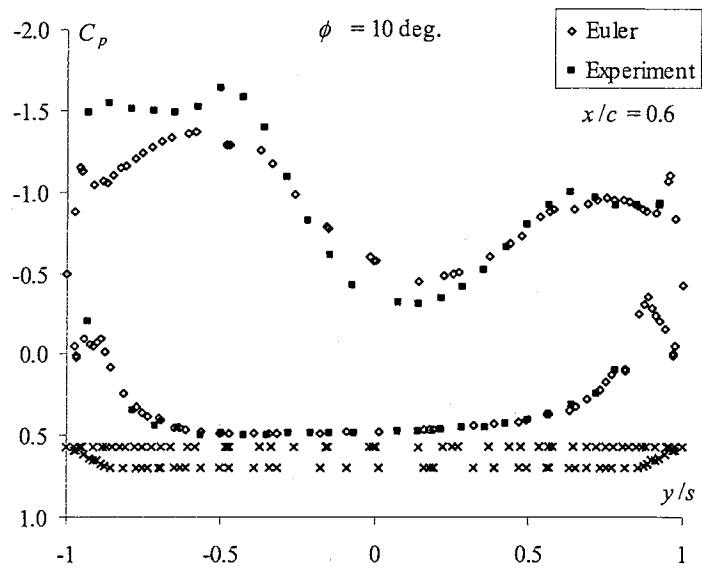
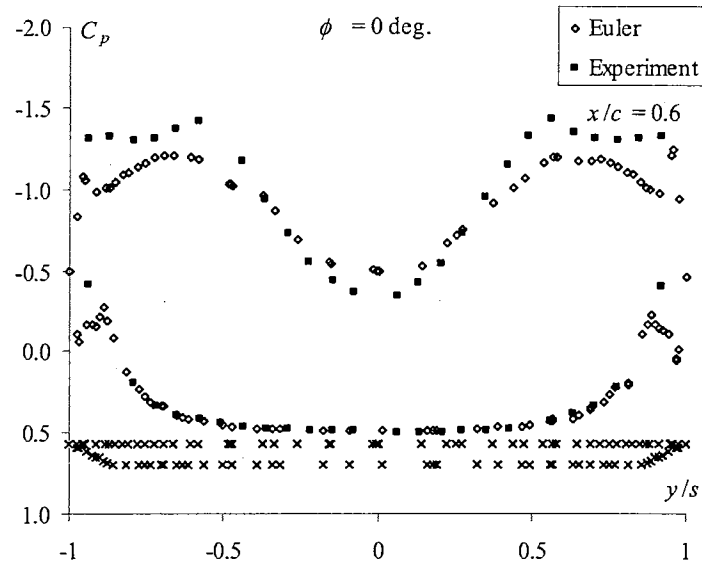




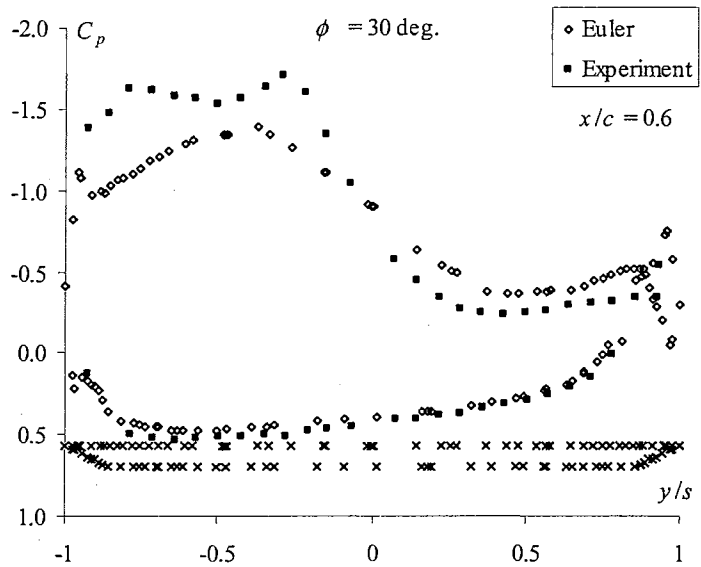
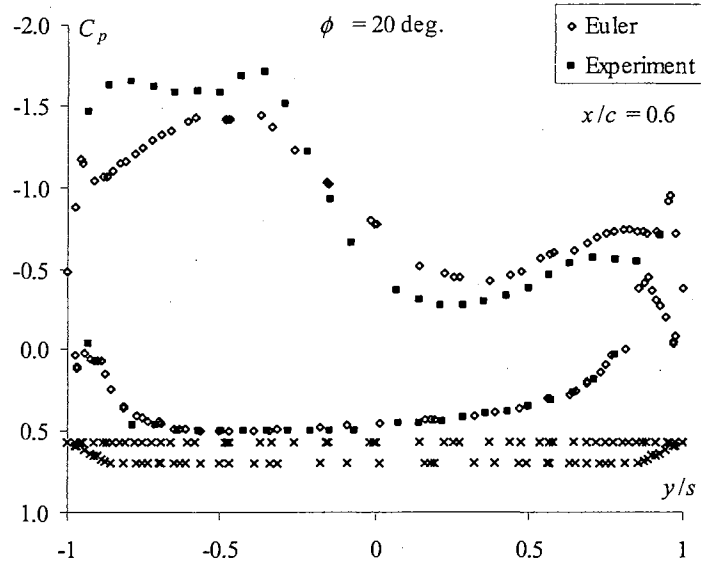
APPENDIX E: Steady Validation Data For 80 Degree Delta Wing

This Appendix provides a set of steady validation data for the delta wing of Section 5.2.5 at an angle of attack of 30 degrees and various roll angles from 0 degrees up to 70 degrees. This data is important for two reasons. First, it demonstrates that the grid resolution is sufficient for an accurate representation of the flow field whether the solution is steady or unsteady. Second, steady data serves as the initial conditions for any unsteady solution that is computed. The data presented in this Appendix shows spanwise surface pressure distributions at one chord location for the wing, the 60% root chord location. The computed data for Mach 0.3 is compared to the experimental data from reference 47 corrected for the effects of compressibility using the Prandtl-Glauert relation, Equation (5.5). There is no experimental data for comparison with the computed data for Mach 1.2.

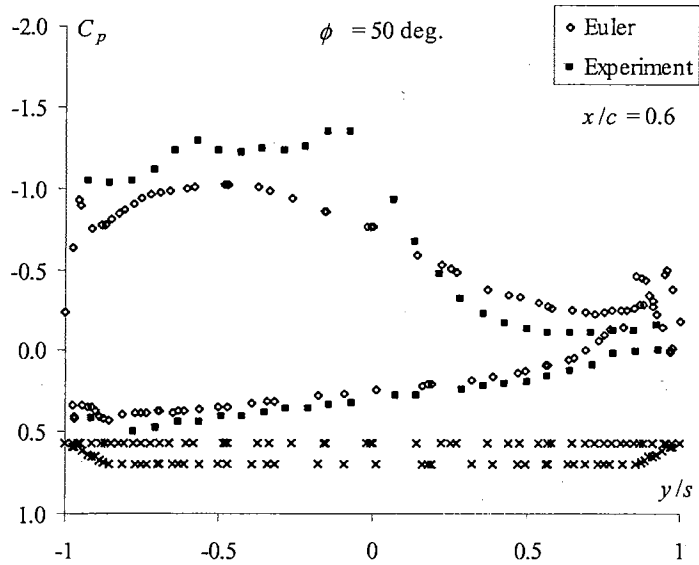
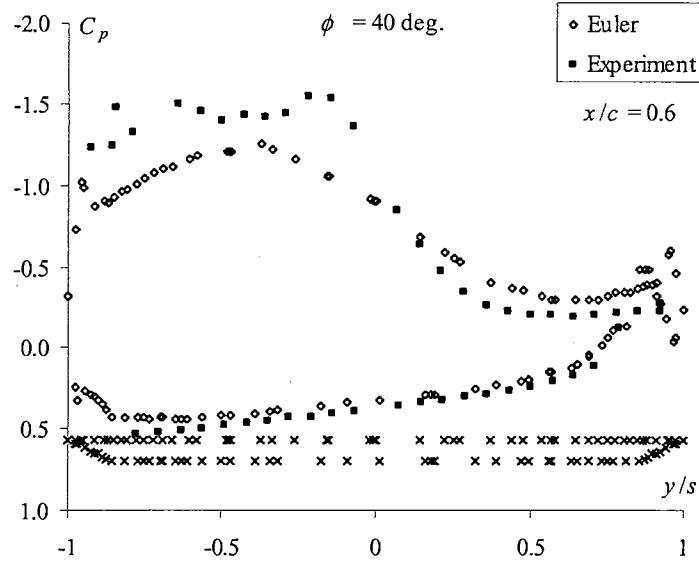
C_p Data for Mach 0.30, Roll Angles of 0 and 10 Degrees



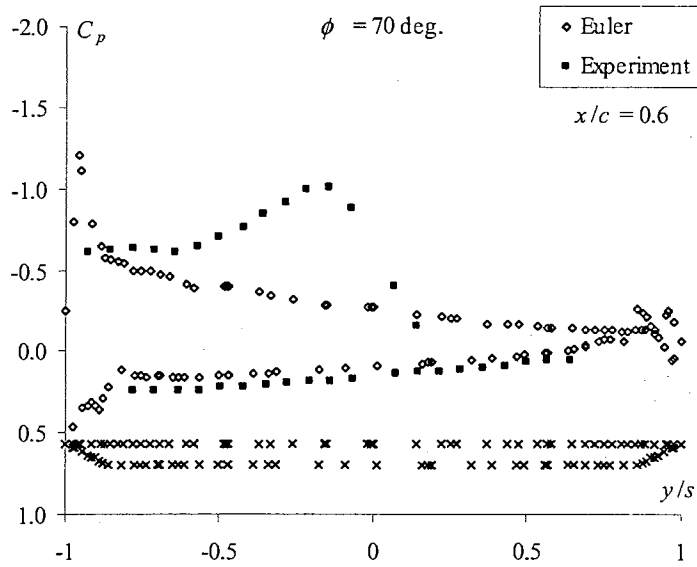
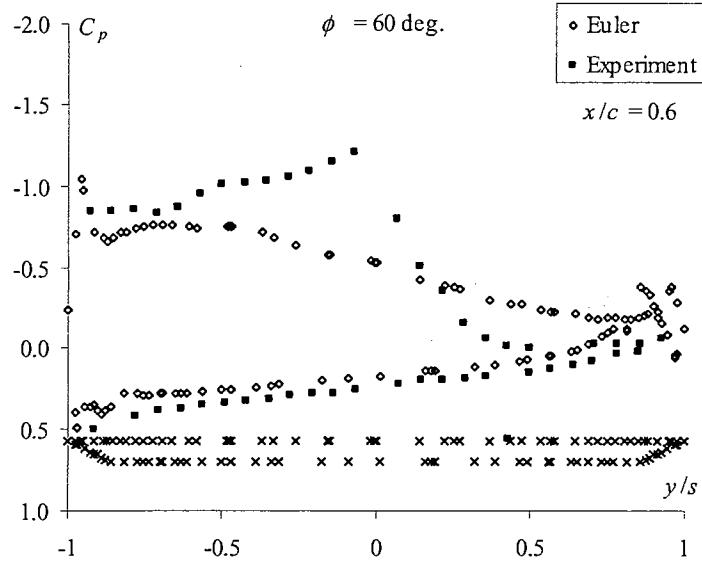
C_p Data for Mach 0.30, Roll Angles of 20 and 30 degrees



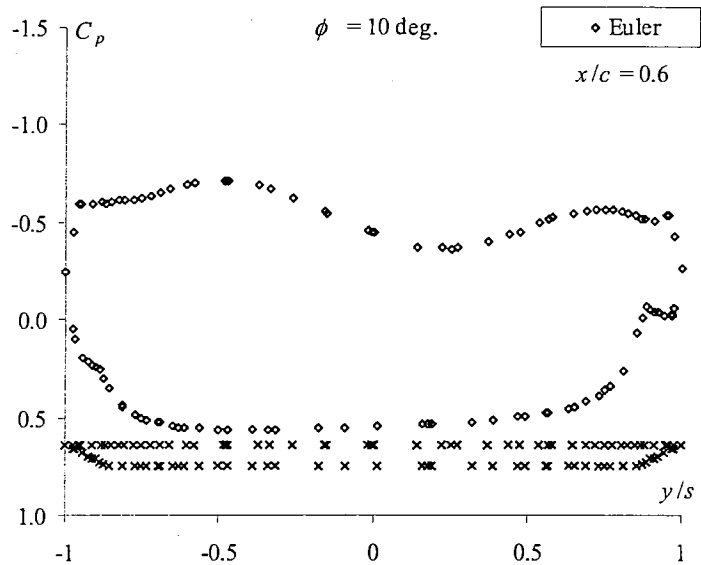
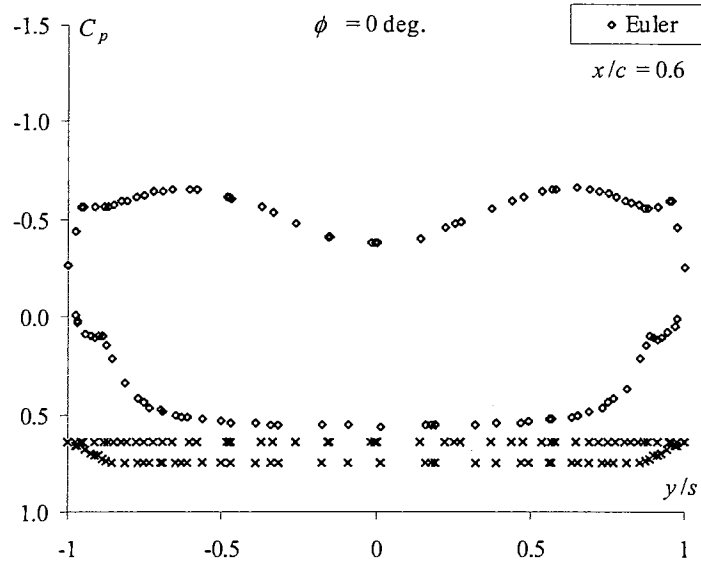
C_p Data for Mach 0.30, Roll Angles of 40 and 50 degrees



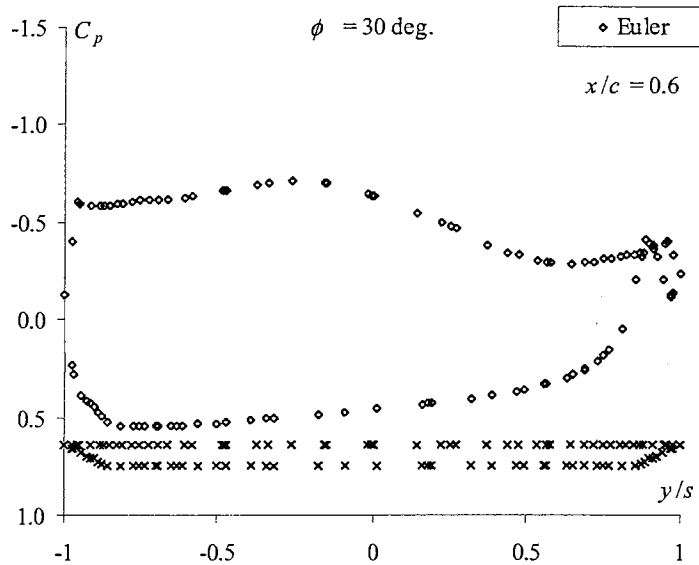
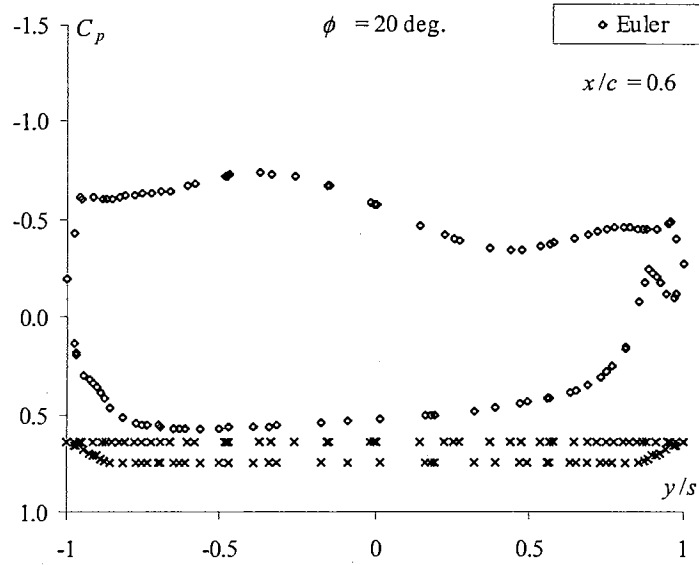
C_p Data for Mach 0.30, Roll Angles of 60 and 70 degrees



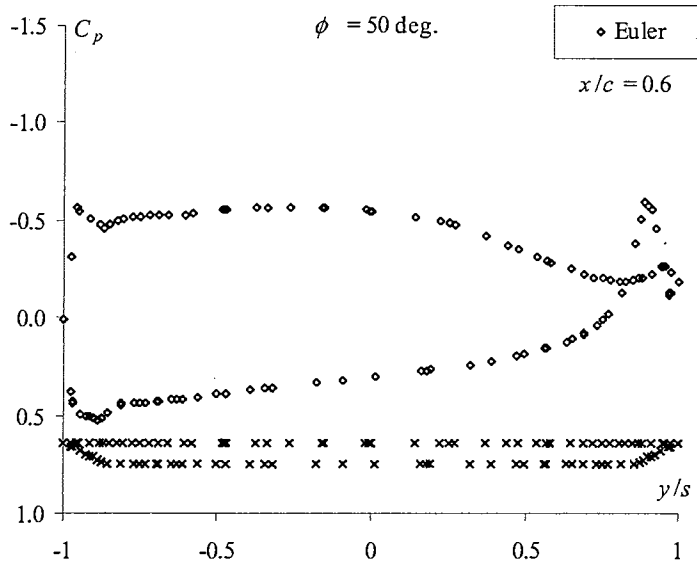
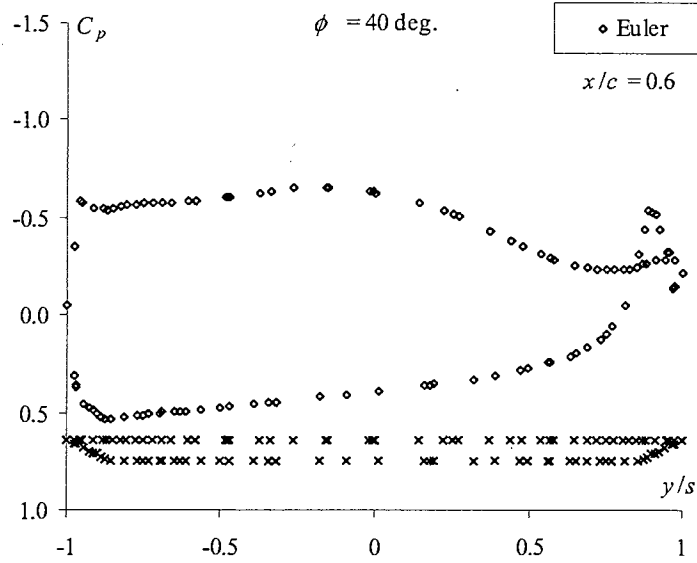
C_p Data for Mach 1.20, Roll Angles of 0 and 10 Degrees



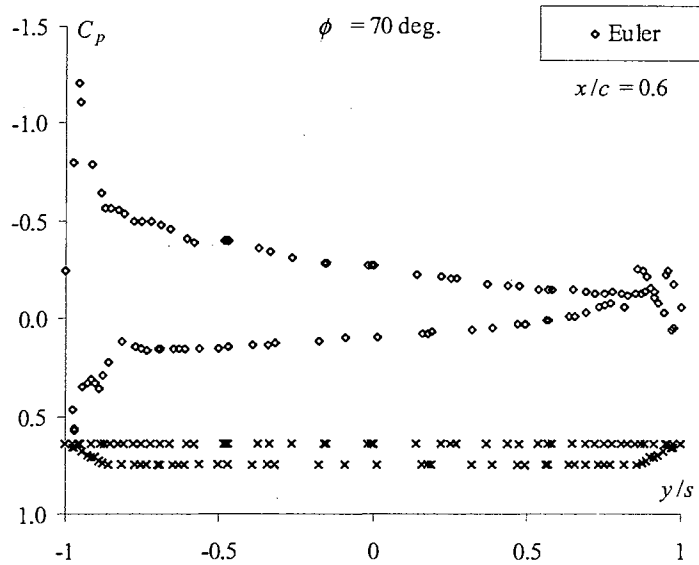
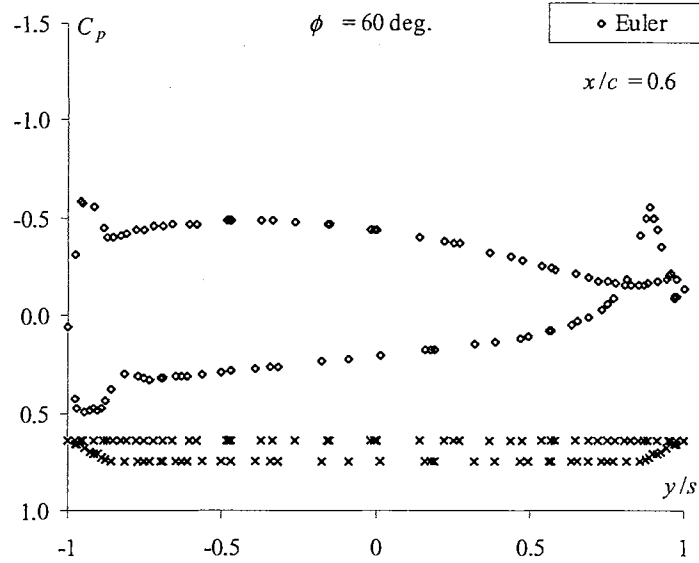
C_p Data for Mach 1.20, Roll Angles of 20 and 30 Degrees



C_p Data for Mach 1.20, Roll Angles of 40 and 50 Degrees



C_p Data for Mach 1.20, Roll Angles of 60 and 70 Degrees



VITA

2 Timothy John Cowan

Candidate for the Degree of

Doctor of Philosophy

Thesis: FINITE ELEMENT CFD ANALYSIS OF SUPER-MANEUVERING AND SPINNING STRUCTURES

Major Field: Mechanical Engineering

Biography:

Personal Data: Born in Tulsa, Oklahoma March 31, 1973. The son of Timothy M. and Marsha L. Cowan. Married Leslie Ann Graham on July 26, 1997. Father of Zachary Andrew born on August 29, 1999, and Nathan Riley born February 1, 2002.

Education: Graduated from Union High School, Tulsa, Oklahoma in May 1991; received a Bachelor of Science degree in Mechanical Engineering from Oklahoma State University in May 1996; received a Master of Science degree with a major in Mechanical and Aerospace Engineering at Oklahoma State University in May, 1998. Completed the requirements for the Doctor of Philosophy degree at Oklahoma State University in August, 2003.

Experience: Graduate Research Assistant, Oklahoma State University School of Mechanical and Aerospace Engineering, 1996-2001; Lecturer, Oklahoma State University Department of Mathematics, 1999-2000; Visiting Researcher, NASA Dryden Flight Research Center, 1998-1999; Software Engineer, Electronic Arts, 2001-2003.

Professional Memberships: American Institute of Aeronautics and Astronautics, Phi Kappa Phi, Tau Beta Pi, Pi Tau Sigma, and Golden Key National Honor Society.