HIGHER ORDER AND DYNAMIC CFD FOR AEROELASTIC SIMULATIONS

By

CHARLES ROBERT O'NEILL

Bachelor of Science in Aerospace Engineering
Oklahoma State University
Stillwater, OK, USA
2001

Master of Science in Mechanical Engineering
Oklahoma State University
Stillwater, OK, USA
2003

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
December, 2011

HIGHER ORDER AND DYNAMIC CFD
FOR AEROELASTIC SIMULATIONS

Dissertation Approved:

Dr. A. S. Arena, Jr

Dissertation Advisor

Dr. F. W. Chambers

Dr. J. K. Good

Dr. A. H. Johannes

Dr. S. A. Tucker

Dean of the Graduate College

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation develops and analyzes higher order computational fluid dynamics (CFD) simulations for coupled fluid-structure interactions as well as large deflection vehicle motions. The flow regime ranges across subsonic, transonic and supersonic Mach numbers and inviscid and viscid boundary conditions. The primary objective for the CFD solver is to provide time-accurate surface pressures and stresses for coupling into a corresponding structural motion solver.

The author works in the Computational Aero Servo Elasticity Laboratory (CASELab) at Oklahoma State University. Our mission is to provide support to NASA Dryden's flight test programs. The primary task is developing tools for aircraft flutter evaluation, primarily with military aircraft. These vehicles have intrinsic simulation challenges. Operation is within a wide flight envelope including transonic Mach numbers. Vehicle geometries are complex and contain large ratios of scale. Vehicle operation often involves multiple body problems such as store releases, formation flight, and relative motion. Figure 1.1 conceptually shows possible operations.

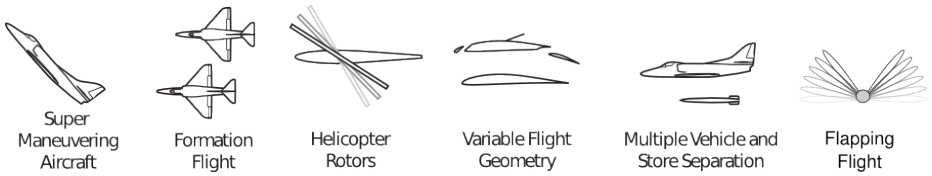


Figure 1.1: Motion Concepts

Nomenclature and abbreviations used in this dissertation are presented in Appendix A.

1.1 Requirements

This section discusses the requirements needed for aeroelastic simulations in the CASELab.

A small discussion of the catagories of unsteady flow will assist in formulating requirements. The lack of unsteadiness in a flow defines a steady flow. Numerically, many steady CFD solvers use a non-time-accurate unsteady time advancement. The next catagory is unsteady flow with steady boundary conditions. This dissertation concerns the last catagory: unsteady flow with unsteady

boundary conditions. The laboratory's current state of the art is unsteady boundary with small deflections (inviscid) and unsteady flow (viscous).

The CASELab specializes in aeroelastic flutter predictions for high performance aircraft. Aeroelasticity in general occurs when stable aerodynamics and stable structural dynamics couple into an unstable system. Flutter usually results in the failure of the offending aerodynamic surface, often leading to loss of the vehicle and to loss of life.

For simulation purposes with a systems approach, the structural stability about an equilibrium point is sufficient for flutter prediction. Possible non-linear responses further away from the equilibrium point are not particularly useful; limit-cycle oscillations limited through aerodynamics are likely just as dangerous as the lead-up to the full limit cycle amplitude.

Thus, the structure is modeled with modeshapes and a linear structural differential equation about the previously mentioned equilibrium point. Modeshapes Φ are determined through an up-front finite element simulation of the structure's free body response much like a ground vibrational test in aircraft certification. Structural motion is modeled with

$$[M]\,\ddot{\Phi} + [C]\,\dot{\Phi} + [K]\,\Phi = (F)$$

where $[M]$, $[C]$, $[K]$ and (F) are the generalized mass, damping, stiffness, and forces. Generalized forces result from the imposition of an external pressure from the flow field. Thus, the generalized force is the integration of the pressure field and the modeshape

$$F_i = \int P \cdot \Phi_i \, dA$$

Figure 1.2 visually shows the analysis procedure. The structural model on the right is the finite

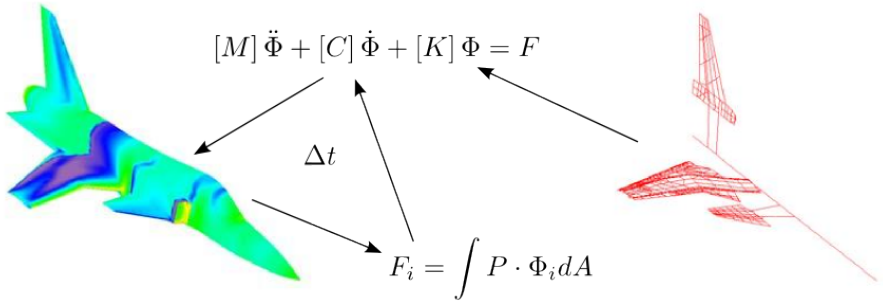


Figure 1.2: Aeroelasticity Simulation Flow with Structural Modeshapes

element representation of the aircraft's structural properties. The model on the left shows the aerodynamic vehicle generating surface pressures. Time advancement occurs within the coupled aerodynamics and generalized motion simulations.

Comparing the structural and aerodynamics simulation indicates that the simulation bottleneck lies with the aerodynamics simulation. For most simulations, the aerodynamics portion takes well into 99% of the total time. Reduced order modeling techniques are available[57] for reducing the simulation cost for particular aerodynamic geometries with a system-identification routine. This technique contains some significant hurdles[8] with respect to training cost. Regardless, improving the aerodynamics simulation speed and accuracy is a priority.

1.1.1 Elastic Motion in Navier-Stokes Flow

In inviscid flows, transpiration efficiently describes elastic motion of boundaries. Transpiration simulates boundary motion with perturbed surface normals. Transpiration is fast and efficient but requires a non-zero boundary velocity. See Fisher[21] for more details. Transpiration constrains boundary flow to a non-normal vector (Fig. 1.3). In Navier-Stokes flows, this strategy fails; non-normal velocity vectors are always identically zero. To date, no effective and fast algorithm is available; one example is Shyy[69].

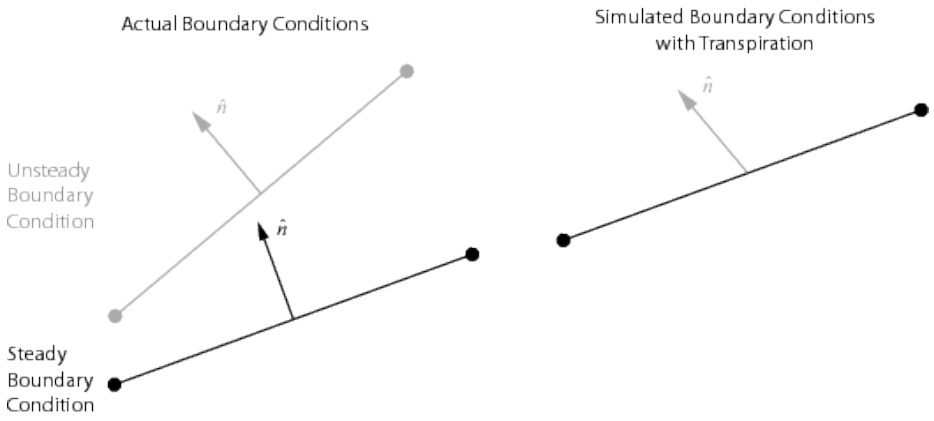


Figure 1.3: Transpiration

There are also coupling effects beween structural dynamics and rigid body dynamics. Large structural deflections can change flight dynamics. For example, the unmanned Helios aircraft was a solar powered flying wing with a large aspect ratio and flexible wing. The NASA Helios accident report[56] stated that "...the rapid divergence of the pitch oscillation when the dihedral reached 40 feet (the third event) was not expected." The chain of events leading to the crash is fascinating: turbulence, flexible geometry change, unstable longitudinal flight dynamics (doubling *every* period[56]), excessive dynamic pressure, local structural failure, and impacting the ocean.

1.1.2 Simulation Turnaround Time

There is a need for fast turnaround time. Spending months simulating a particular case is not particularly desirable or fundable. The objective is to solve a problem with the least amount of input for the most amount of output.

1.2 Current Simulation Tools

The Structural Analysis Routines[25] (STARS) code developed at NASA Dryden supports Dryden's flight tests with a suite of multidisciplinary tools for structural, aerodynamic, thermal, and control system analysis. STARS has aeroelastic simulation capabilities with linear and non-linear aerodynamics. Stability and control simulations and sensor simulations are also available. Recent programs supported by the STARS code include the X–29, X–33, F–18 AAW, and HyperX. The computational fluid dynamics component of STARS solves the Euler and Navier-Stokes governing equations. The recent addition of a non-inertial reference frame CFD formulation extends the capabilities of STARS to include rotational and translational motions[14].

The current aerodynamic simulation process is visually described in Figure 1.4 for the inviscid Euler3d solver. A simulation starts with creating or importing the aerodynamic geometry, then

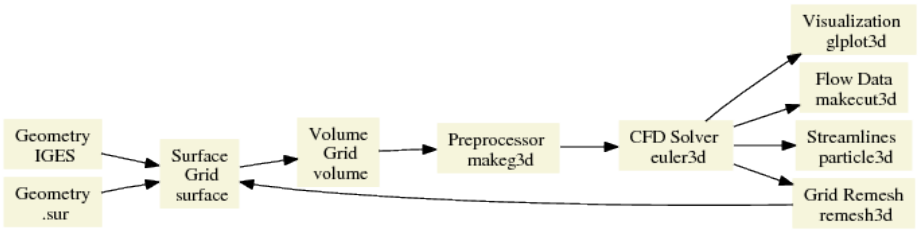


Figure 1.4: STARS Euler3d CFD Procedure

creating a grid, running the CFD solver and finally plotting or refining the solution. With the modularity shown above, this dissertation seeks only to replace the CFD Solver module.

In this section, we will attempt to list the known deficiencies and concerns with the current fluid solver. The euler3d package is robust and has served our lab well, but we would like to improve the following issues.

1.2.1 Test case Size and Solution Quality

The CASELab often struggles to obtain research quality simulations for whole aircraft. The problem is simply a matter of scale; we need to capture local fluid behavior over a large, moving, transonic

aircraft.

A recent attempt to characterize the F18 AAW flutter boundary for a single ASE experiment[24] required months of computational time. Even at over 1 million elements, the F18's unstructured grid still needed vastly more elements. With the current code, modern workstation and desktop computers contain sufficient memory and storage for simulations requiring months or years. The issue is solution speed, not storage capacity.

Non-linear flight dynamics and viscous regimes amplify the grid size problem by requiring accurate solutions throughout a long simulation. To be fair, normal flight dynamics do not require expensive fluid dynamics solutions. Low frequency phugoid motion simulations likely only need a linear aerodynamics approximation. Large amplitude and unusual attitude motions require better quality solutions with more complexs fluid models.

1.2.2 Technology Limitation

Test case size and solution quality is effectively limited by the available computer power. Computation time limits our simulations to about 1 million elements for an unsteady test case requiring 1 month of CPU time. If we need Navier-Stokes solutions, a common heuristic suggests increasing CPU-time by one order of magnitude. Thus, we are looking at solutions around two orders of magnitude slower than what is required for daily flight-test support.

Moore's Law historically shows a doubling of computer power every 1.5 years. Two orders of magnitude via Moore's law will require about 10 years. Even advancing computer technology will not solve our problems and waiting is not an effective solution. Anecdotal evidence suggests that improvements in solution speed are exactly negated by increasing solution complexity.

1.2.3 Solution Methodology and Efficiency

Solution efficiency is a difficult criterion to evaluate; we will always find a faster, but perhaps less useful, methodology in the literature.

Artificial dissipation is an inefficient method in the current solver. Artificial dissipation, necessary for solution stability but not for solving the governing equations, requires 50% of the total CFD solution time. This gives us room to change the complexity/dissipation operating point.

Converting an Euler code to Navier-Stokes, Moffitt[50] found that second derivative specification becomes more difficult, and potentially unstable, with linear elements. Also, the current solution method uses piecewise continuous boundary conditions, which creates slope discontinuities at nodes and zero-order accurate surface stresses.

1.2.4 Adjustable Parameters

All CFD solvers require simulation parameters, but too many parameters cause problems. Euler3d reformed[14] the STARS code by removing several adjustable parameters. However, Euler3d still has multiple adjustable parameters: timestep, inner loop cycles, CFL, and dissipation. Each of these free parameters requires training and experience to avoid computational errors.

A natural tendency is to decrease CPU time at the expense of actual accuracy. We can only measure accuracy by running a still *longer* simulation. We have to fight to keep ourselves in a converged situation. Faced with a month long simulation, we must constantly remind ourselves that a poor simulation takes a week and a good simulation takes a month. Shorter simulations are too tempting and measuring solution convergence is not black and white. A code should steer the user to make conservative choices.

1.2.5 Grid Generation

Grid generation is a key to good solutions. The current surface and volume generation codes are notoriously *picky* about input files. For Navier-Stokes solutions, the size ratio between element sizes varies with Reynolds number and location. The current surface grid code often appears to create a failure condition when element size ratios exceed 10000:1. Thus, if a simulation region is 1000 inches (25 meters), we likely cannot grid below 0.1 inch (2.5 millimeters), which implies a minimum solution scale of about 0.5 inch (13 millimeters). Moffitt[50] found that grid system encounters difficulties with small elements needed for viscous solutions.

1.3 Rationale

This section develops a rationale for investigating higher order CFD methods. It begins with an analysis of current performance and accuracy. Next, a literature review of higher order methods is given. The section ends with two initial higher order solutions of fluid-like governing equations.

1.3.1 Current Performance

This section discusses a single technical point, computer memory bandwidth, with respect to the current performance of the Euler3d CFD code. This couples both computer architecture with numerical methods and fluid dynamics. It is presented as a series of facts and observations that lead to a final statement.

- Fact 1: Fluid dynamics depends on fluid state gradients. In other words, fluid flow simulations require fluid properties at nearby physical locations.
- Fact 2: The Euler3d CFD solver uses an unstructured grid with fluid states stored at segment endpoints. Thus, adjacent physical locations do not, and cannot, map to adjacent memory locations.
- Fact 3: Euler3d uses elemental operations. Thus, a single operation will require variables from potentially nearby to potentially far away memory locations. This is likely a guaranteed RAM cache miss.
- Fact 4: Euler3d is written in Fortran. Fortran stores arrays in a column major format. For example, the next memory location after $a(3,12)$ is $a(4,12)$. By contrast, memory location $a(3,13)$ could be a significant distance from $a(3,12)$.
- Fact 5: Fortran compilers warn users to use column major arrays for maximum performance. Intel claims a 30% or more hit in performance for using row major addressing[31].
- Observation: Reversing the order of arrays did not significantly affect Euler3d's performance. It was originally written in row major format (i.e., the "wrong" way).
- Conclusion: The memory performance benchmark relevant to Euler3d appears to be the random memory recall rate and not the adjacent-memory-sweep bandwidth.

We can get an understanding of this by visually inspecting the mass matrix for a simple test case with a second order expansion. The left image of Figure 1.5 is the discretized grid. The right image of Figure 1.5 shows non-zero components of the mass matrix. A strong diagonal component is evident but no decoupled block diagonal structure exists. Euler3d is categorized as using global operations as every computation requires widely spread memory access.

1.3.2 Current Accuracy with Euler2d

The objective of this section is to determine accuracy with respect to grid convergence and CPU time. The geometry is a cylinder with Mach 0.1 cross flow. The linear element euler2d code is used. The solution domain is a disk of inner radius 1 and outer radius 20.

The initial step generated grids of decreasing element sizes. For reference, Figure 1.6 compares element sizes just ahead of the leading edge; each block is 1 radius by 1 radius. For this 2D grid, we expect and find that the number of elements scales with $(\Delta x)^2$.

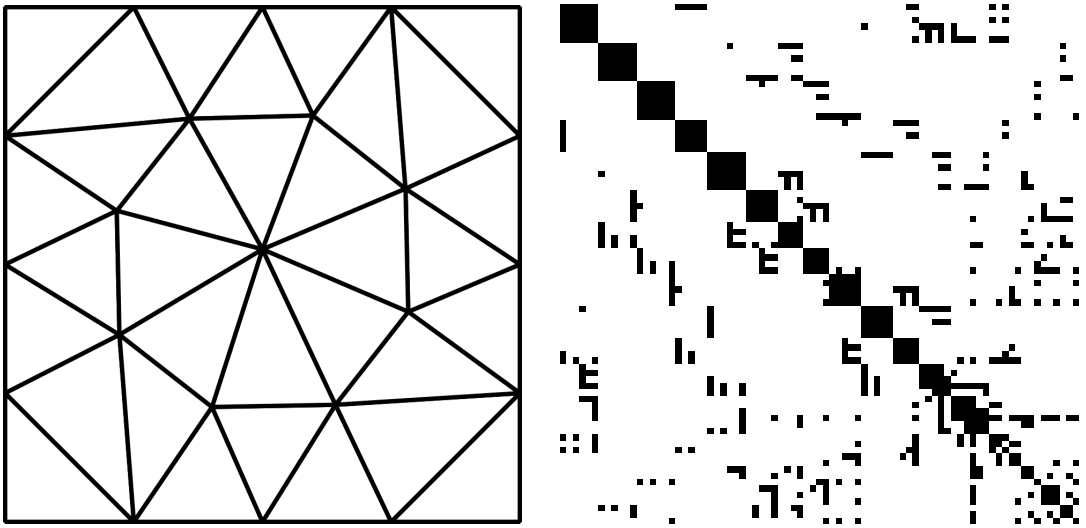


Figure 1.5: Box Grid and Mass Matrix

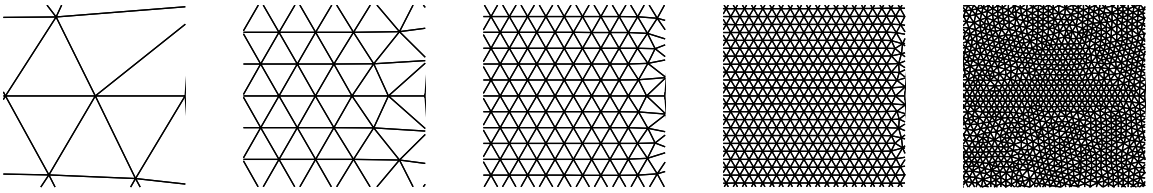
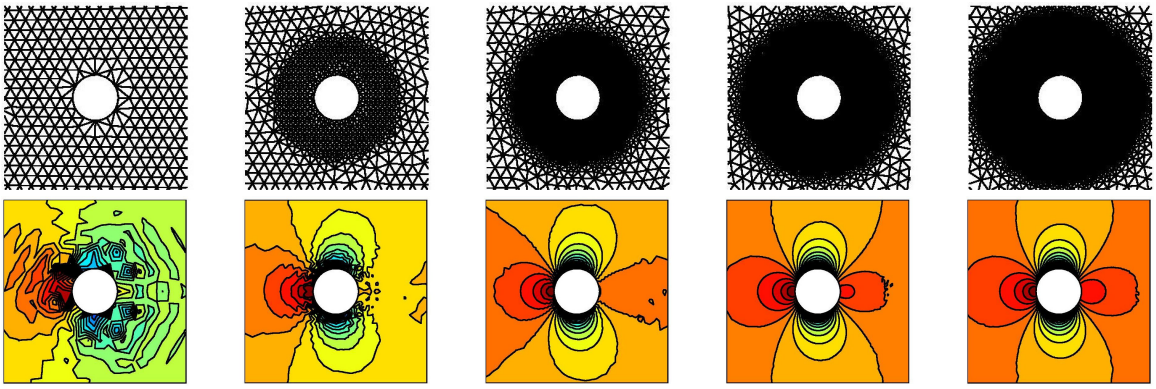


Figure 1.6: Cylinder: Relative Grid Size at Leading Edge

The freestream at Mach 0.1 travels from left to right. Next, the euler2d program was run to enthalpy residual convergence of 10^{-8}. Pressure contours are shown in the lower row of Figure 1.7.

The coarse 0.5 grid really only shows the most basic problem physics, a pressure increase at the leading edge; however, significant aliased 2h waves[10] are seen. Decreasing element sizes to the 0.05 case, the improved grid resolution allows for better aft cylinder pressure recovery. Visual differences becomes ineffective after the 0.05 case. Dissipation is critical in this experiment, but we tried to minimize those effects with low dissipation value (0.15). From the contour plots, we can see a quasi-separation is moving the pressure peaks back along the cylinder. The non-zero Mach number (M=0.1) will also slightly change the solution field. However, for a cylinder geometry, the comparison is likely near optimal. Perhaps numerical dissipation makes the comparison unfair, but the solution method represents how we solve actual problems.

For a more thorough comparison, we use inviscid incompressible ideal aerodynamics. For inviscid



Δx: 0.5 0.2 0.1 0.05 0.03

Figure 1.7: Cylinder: Grid and Solution

cross flow past a cylinder of radius 1, the stream-function is

$$\Psi = V_\infty y \left(1 - \frac{1}{r^2}\right)$$

For a vertical cut-line at $x = -1$, the u and v velocities are

$$u = \frac{d\Psi}{dy} = \frac{y^4 + 3y^2}{(y^2+1)^2}$$

$$v = -\frac{d\Psi}{dx} = \frac{-2y}{(y^2+1)^2}$$

Also, we need the pressure coefficient, which is defined as

$$C_p = \frac{1 - V^2}{q}$$

Now, Figure 1.8 compares the ideal u velocity and pressure coefficient (C_p) with the grid-converging euler2d CFD solutions.

The upper plot shows the velocity converging to the ideal velocity quickly; we'll leave the rate of convergence until later. The pressure coefficient seems to converge more slowly, but it still visually approaches the ideal curve. Zooming into the fore stagnation point along the leading edge shows that the convergence is not smoothly approaching the ideal curve, but has uneven variance, even if the macro-plot looks smooth.

Along the cylinder surface, the pressure coefficient appears to approach the ideal distribution (Fig. 1.9). We also see the non-physical and non-isentropic separation remnants along the aft cylinder ($x > 0.8$). Again, lowering dissipation assists with converging to the ideal solution but hurts the high wavenumber stability.

Finally, we consider the time, error, and grid relationships. This project's primary objective concerns establishing the operating efficiency region. Figure 1.10 (left) presents RMS error for C_p

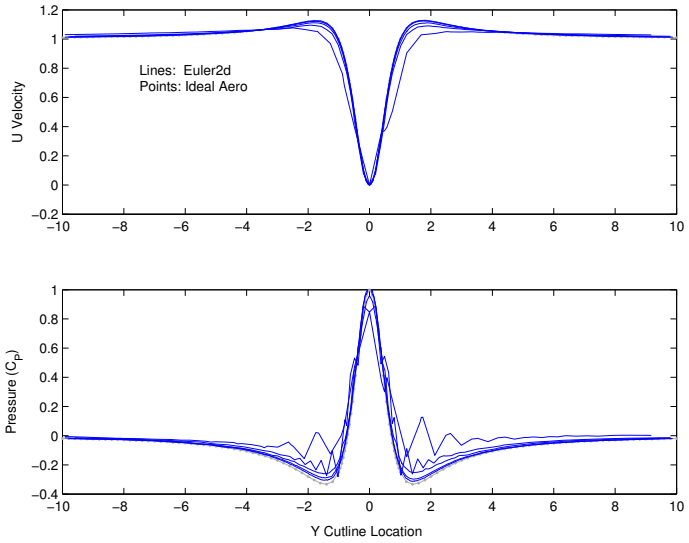


Figure 1.8: Cylinder: Y Cut-line at the Cylinder's Leading Edge

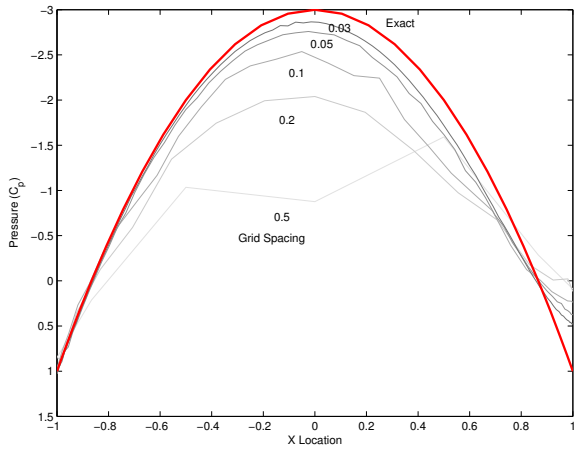


Figure 1.9: Cylinder: Surface Pressure Coefficient, C_p

and u versus grid convergence Δx for the cylinder. From the contour plots and surface pressure plot, a RMS error below 10^{-1} seems preferable. The log-log slope is -0.975 such that an accuracy model is

$$RMS = 1.5\Delta x^{0.975}$$

The euler2d formulation is based on second order accuracy, so the experimental slope is less optimal than the -2 that theory suggests. Accuracy loss contributions likely result from the linearized flux integrals and the dissipation scheme.

More importantly, the right part of Figure 1.10 presents CPU Time versus RMS error. A model of time for a given error is

$$T \propto (RMS)^4$$

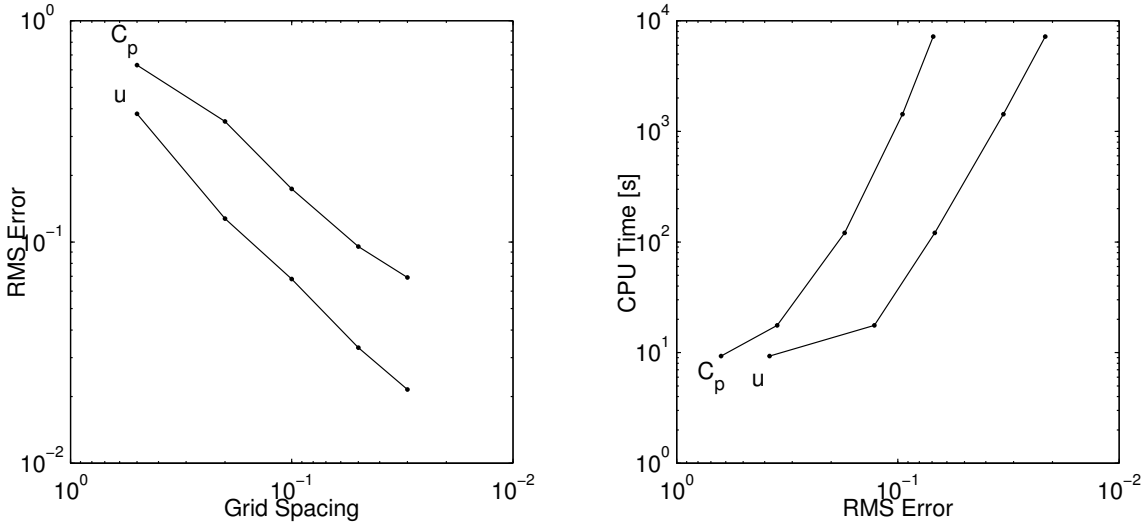


Figure 1.10: Cylinder: Y Cut-line Data Analysis

Matching the soft RMS error limit of 10^{-1} maps to the straight part of the CPU Time curve (CPU Time $\approx 10^2$). Referring to Bar Yoseph[3] places our current code, euler2d, in a likely-to-benefit region, since we are just started into the linear part of the log-log plot. If we expect higher quality solutions ($\Delta x = 0.05$ or 0.03), higher-order becomes significantly more attractive. Comparing only C_p also makes higher-order methods more attractive.

1.3.3 Dynamic Simulations in the Literature

The published literature provides an opportunity to explore and evaluate possible dynamic simulation concepts. Rotational and translational dynamic grids are presented for super maneuvering aircraft[14, 23]. However, their non-rigid-body motions are restricted to small deflections. The overlapping multiple-interacting-grid methods appear popular for store separation simulations. Tomaro[72] and Cenko[12] demonstrate multiple-grid based dynamic meshing for CFD predictions of JDAM store separations with the F–18. Löhner[45] and Prewitt[64] developed overlapping grid Chimera methods for CFD solutions. Grid element selection and assembly appears to be the limiting process for their overlapping-grid methods. Helicopter rotor simulations seem to prefer sliding and overset meshes to take advantage of the harmonic rotational motion[59, 29]. The literature shows that many dynamic meshing concepts are available for exploration and evaluation. Most appear somewhat difficult to implement with complex bookkeeping.

1.3.4 Higher Order Efficiency in the Literature

A primary concern for CFD simulations is the solution efficiency. Solution methods with efficiencies differing by constants are mitigated with parallel processing and other computer science techniques. We are more concerned with methods causing solution efficiencies to differ by orders of magnitude. A preliminary review of the literature suggested that higher order expansions are the latter catagory.

Spectral/higher-order CFD is not new. Weather forecasting continues to use global spectral methods since starting in the 1950's [10], but use in traditional fluid dynamics has only recently become accepted. For spectral methods, Boyd[10] has hundreds of references on global spectral methods. Most differential equations textbooks discuss the Fourier harmonic method, a subset of global spectral methods. We focus on unstructured element grids. However, as Karnidaikis[35] states, "Use of unstructured spectral/hp element methods in computational fluid dynamics has been very limited to date." Disappointing panel method results in the 1980's left skepticism toward higher order CFD simulations. The literature generally agreed that higher order linear panel methods only marginally improved accuracy when given the more complex development and higher CPU effort. Yet, acceptance of higher order methods for CFD has grown in recent years.

The primary rationale for higher-order expansions is efficiency. All methods should approach the exact solution in the limit, but the more efficient method is preferred. By efficiency, we mean: the lowest CPU time for a given solution quality. However, time and solution quality are not easily predicted given a particular scheme and implementation. So, we turn to approximations and order analysis.

Boyd[10] suggests that errors for a spectral expansion of N terms behave as,

$$\epsilon \approx O\left(N^{-N}\right)$$

This approximation shows a strong advantage for higher-order expansions; however, we must remember that runtime increases with order N.

Löhner[44] shows that we should "strive for schemes of higher order as the dimensionality of the problem increases." He claims that most CFD codes are initially formulated and tested in 1D, but generally simulate 3D flows. This suggests that if the minimum operating order for a steady 1D simulation is 2 terms, then a 3D transient simulation should probably use at least 6 terms.

Karniadakis[35] suggests that for "engineering accuracy of 10%" for a linear advection governing equation, the work W required for a solution of length M versus scheme orders of 2, 4, and 6 is approximately

$$W_2 \propto 20M^{1/2} \quad W_4 \propto 14M^{1/4} \quad W_6 \propto 15M^{1/6}$$

For long term simulations, the higher order approximation becomes favored.

Fidkowski[19] claims that runtime T for a given error E is

$$T = O\left(\left(pE^{-1/p}\right)^{wd}/F\right)$$

where p is the expansion order, w is the algorithm complexity, d is the problem dimension, and F is the computer speed. He shows that for "stringent" accuracy requirements, the runtime T depends "exponentially on p, w, and d". For small errors, expansion gives

$$\lim_{E<<1} T \approx E^{-\frac{wd}{p}}$$

Again, we see that larger expansion order p is preferable. Also, Löhner's claim that expansion order should increase with increasing dimensionality appears again. Fidkowski[20] defended a high order discontinuous CFD solver. His method uses multigrid and line smoothing techniques. No unsteady time-accurate solutions are presented or discussed; the multigrid technique assures steady solutions. He claims that "higher order is advantageous over grid refinement when high accuracy is required."

Bonhaus[9] found that higher order SU/PG methods spatially converged as expected. He expressed concern over the accuracy and generation of curved boundaries: "Particularly difficult to generate are meshes that are highly stretched to compute viscous flows - simply moving the boundary control points to match the surface creates overlapping elements which are unacceptable to the scheme."[9] He reports values for solution convergence but not solution timing.

Bassi and Rebay[6] present steady subsonic and supersonic Navier-Stokes solutions for an NACA 0012 airfoil with orders of 1, 2, 3 and 4. Flow field contours and integrated pressures suggest using at least a 4 term expansion. Lomtev[46] presents a supersonic NACA 4420 airfoil (Fig. 1.5) with "P = 1 in front of the shock and P = 6 behind the shock." A detached and recirculating region is captured with only 30 elements around the entire airfoil.

For more insight, we turn to commercial CFD codes. Through market selection, these codes are filtered to provide robust performance and accuracy. Out of 17 commerical external-flow fluid-structure-interation transient CFD solvers sampled, only 1 specifically mentioned a higher order scheme (Argo, P=2). Fluent mentioned a higher order upwinding method with their finite volume code.

1.3.5 Blasius Higher Order

This section's objective is to solve the Blasius Boundary Layer equation[74]

$$f''' + ff' = 0$$

with the boundary conditions

$$f'(0) = f(0) = 0$$

and

$$f'(\infty) = 1$$

So far, no analytical solution to the Blasius equation exists. White[74] gives numerically generated function values. White suggests using a shooting method coupled with a Runge Kutta routine.

We solved a multiple element solution to the Blasius equation. The grid is composed of 5 elements of varying size and covering η from 0 to 10. The far right of Figure 1.11 shows the grid. The solution method will be generated for an arbitrary grid and polynomial order. Inter-element connections become important and require function and derivative matching up to derivative order D. The methodology is to reduce R to zero using Matlab's `fsolve` function. Each of the five elements contains a 20 degree of freedom Chebyschev basis function.

The solution visually matches the known solution. Figure 1.11 gives $f(\eta)$, $f'(\eta)$, $f''(\eta)$ for this numerical simulation (lines) and White's tabulated data (dots). The results match White within his 5 significant figures. The present method has an advantage over Runge-Kutta shooting in that intermediate values are exactly known from the solved coefficients and basis functions.

1.3.6 Burgers Equation

Burgers equation is the 1D simplified advection diffusion equation without the pressure term

$$\frac{du}{dt} + \frac{dF}{dx} - \nu \frac{d^2u}{dx^2} = 0$$

where the advective flux is

$$F = \frac{1}{2}u^2$$

When we apply Galerkin's method, we obtain a symmetric mass matrix, a non-linear advection term, and a linear diffusion term. Thus Burgers equation exercises the fundamental numerical routines for any Euler or Navier-Stokes solver.

$$\int \phi_i \frac{du}{dt} dx + \int \phi_i F dx - \int \phi_i \nu \frac{d^2u}{dx^2} dx = 0$$

Applying integration by parts replaces derivatives with boundary conditions to give

$$\begin{aligned} \int \phi_i \phi dx \frac{da}{dt} &= \int \frac{d\phi_i}{dx}\frac{dF}{dx}dx - \nu \int \frac{d\phi_i}{dx}\frac{du}{dx}dx\, a \\ &\quad -\phi(1) F(1) + \phi(0) F(0) \\ &\quad +\nu\phi(1)\frac{d\phi(1)}{dx}a - \nu\phi(0)\frac{d\phi(0)}{dx}a \end{aligned}$$

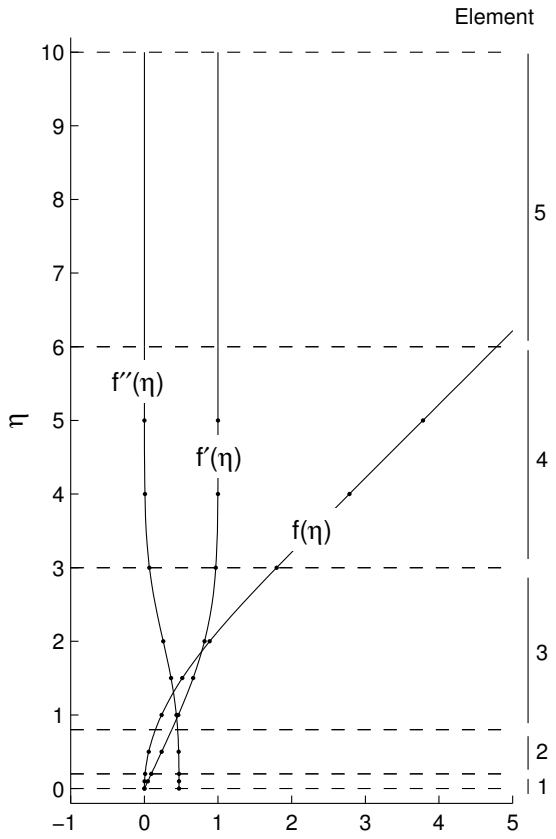


Figure 1.11: Blasius Multiple Element Solution

Conceptually, the parts are

$$M\frac{da}{dt} = RHS(a)$$

The numerical solution is plotted in Figure 1.12 for an 8 term expansion with 10 elements. The initial sine wave advects as expected and forms a shock as expected. As the governing equation is a single state form of the Euler equations with no pressure, the spacial and temporal behavior is encouraging.

1.3.7 1D Euler Compressible

As a final preliminary test, the 1D Euler equations were solved with a variable order solver. For simplification, all flow characteristics $\begin{bmatrix} (u+a) & u & (u-a) \end{bmatrix}$ are positive. This corresponds to supersonic compressible flow. This experiment was performed in Matlab.

First, grid convergence is shown. This is formally called h convergence. Both solutions are 1st order linear. The horizontal axis is a harmonic spacial location; the vertical axis is time. Colors represent the pressure distribution.

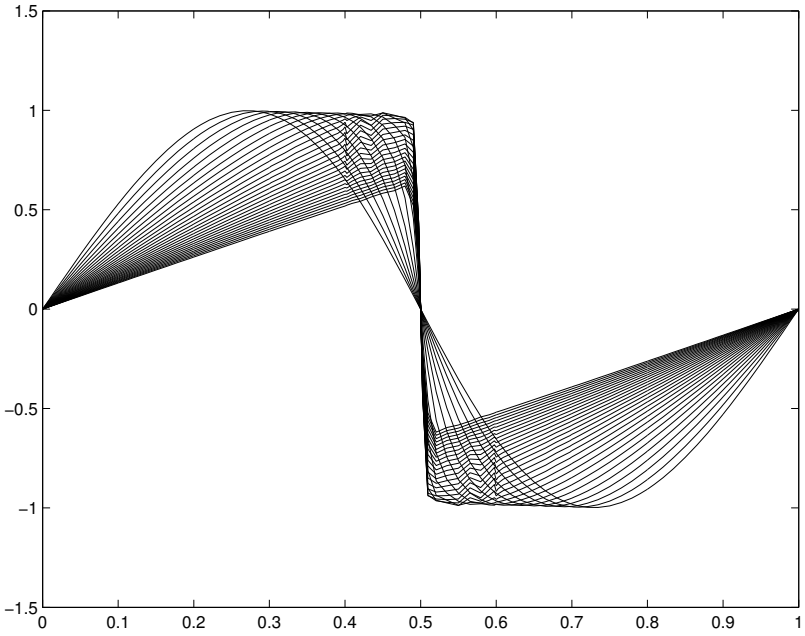


Figure 1.12: Burgers Equation Numerical Solution

Next, order convergence is shown. This is formally called p convergence. Both have the same total number of degrees of freedom. The left solution is 1st order; the right solution is 7th order.

The higher order solution clearly has some issues with solution stabilization eminating from the shocks. This was noted and corrected with a simple stabilization method.

1.4 Hypothesis

In short, this project seeks to research computation fluid dynamics simulations of aeroelasticity and arbitrary boundary motions. The literature indicates that higher order solutions are advantageous with respect to efficiency. A simple comparison with the current CFD solver concurs with simple

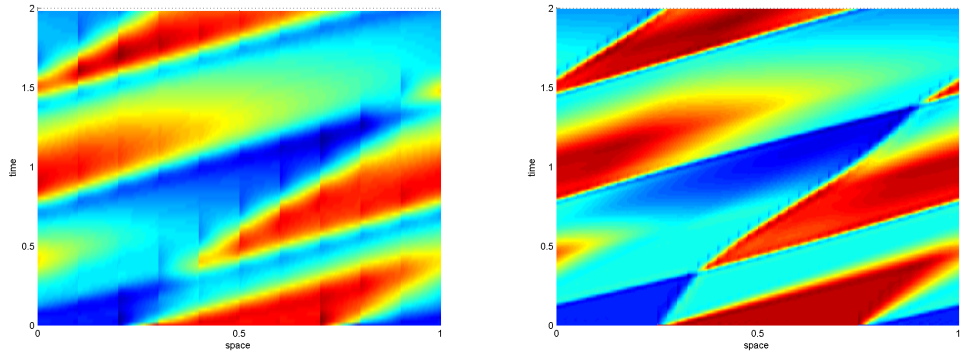


Figure 1.13: 1D Euler: h convergence

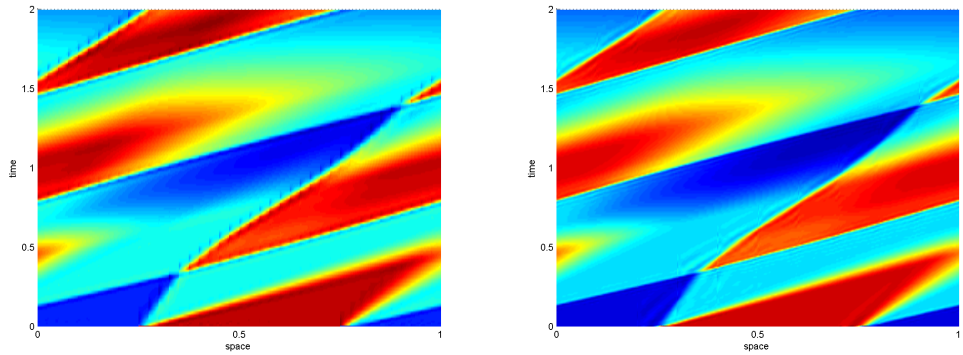


Figure 1.14: 1D Euler: p convergence

higher order tests.

The hypothesis for this higher order CFD project is, *The prediction efficiency of super maneuvering, deforming and constantly remeshing three-dimensional unsteady Navier-Stokes computational fluid dynamics is improved by moving from linear to higher-order simulation methods.*

Chapter 2

Higher Order Basis Function

The basis function ties individual values at specific points to a continuous field of values. In other words, the basis function approximates the solution within an element. The general form for an approximation u and coefficients a is

$$u = \phi_i a_i$$

The goal is to find a robust basis set ϕ_i.

2.1 Literature Review

The basis function maps coefficients to values. As Boyd[10] states, "The general rule is: Geometry chooses the basis set. The engineer never has to make a choice." Boyd's rule is technically correct but historically idealistic. Idealism aside, the current geometry suggests a tri-symmetric basis set for the natural triangular element. Later, we will see that the Belzier B-Spline fits the geometry and thus the rule.

A polynomial "moment"[35] basis $\phi = x^m$ is simple but with an undesirable exponentially growing mass matrix condition number. A general rule is that just because a basis set is orthogonal and can represent any Nth order function does not also indicate that it is numerically identical to a different Nth order function.

The Lagrange basis is common by virtue of being nodally decoupled, a value of one at one node and zero at all others. For greater than 1D, Karniadakis[35] states that "there is not a closed-form expression for the Lagrange polynomial through an arbitrary set of points...."

Boyd strongly recommends a Chebyshev basis as a "moral principle". This is a specially stretched Fourier basis with similar properties. In fact, the Chebyshev basis is defined in terms of a cosine function[10]

$$T_n\left(cos\theta\right) = cos\left(n\theta\right)$$

The advantages of Fourier series are concentrated in computational speed between coefficients and values. Unfortunately, transferring the basic Fourier series to triangles is not trivial.

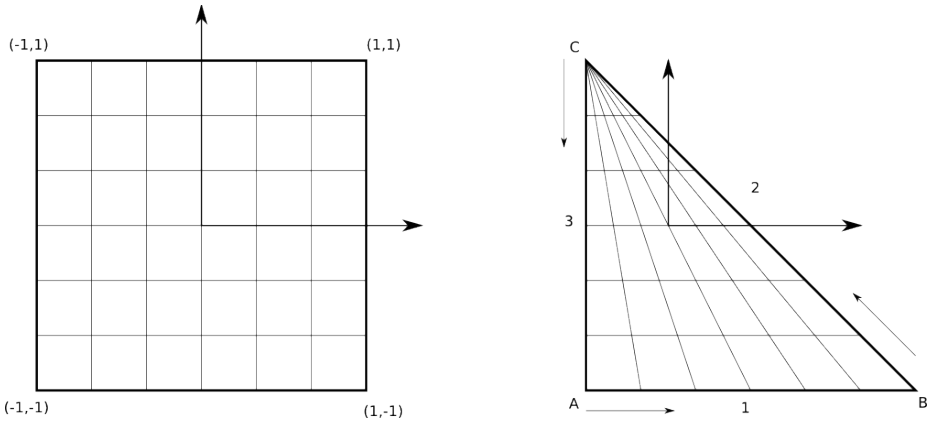


Figure 2.1: Collapsed Coordinate

The literature has volumes of CFD codes where the authors use non-symmetrical mapping functions to transform a particularly interesting basis function onto a different geometry. One compelling reason is sum factorization[35]. These schemes are also full of singularities and non-constant elemental Jacobians. Sum factorization reduces the work required to compute basis operations but increases the work required to tranfer from a local to a global frame. Karniadakis[35] states

> The sum factorization or tensor product technique was first recognized by Orszag and is considered to be the key to the efficiency of spectral methods.... We therefore see that this factorization has reduced the [two-dimensional] cost from an $\mathcal{O}\left(P^4\right)$ operation to an $\mathcal{O}\left(P^3\right)$ operation.

For triangles and tetrahedra, sum factorization requires a modified element geometry seen in Figure 2.1. Point C has a singular Jacobian and multiple values. The Jacobian becomes an extra computation that must be made at each point.

2.2 1D Bezier Splines

For this project, the selected basis function is the Belzier Spline (B-Spline). First, the mathematical definition of the B-Spline in 1D on a reference segment is

$$p(\zeta_1, \zeta_2) = \sum_{i+j=d} c_{ij} \frac{d!}{i!j!} \zeta_1^i \zeta_2^j$$

where the coordinate directions are dependent with the relation

$$\zeta_2 + \zeta_1 = 1$$

Figure 2.2 shows the 1D B-Spline for $P = 1$ through $P = 9$. Further inspection of the B-Spline's

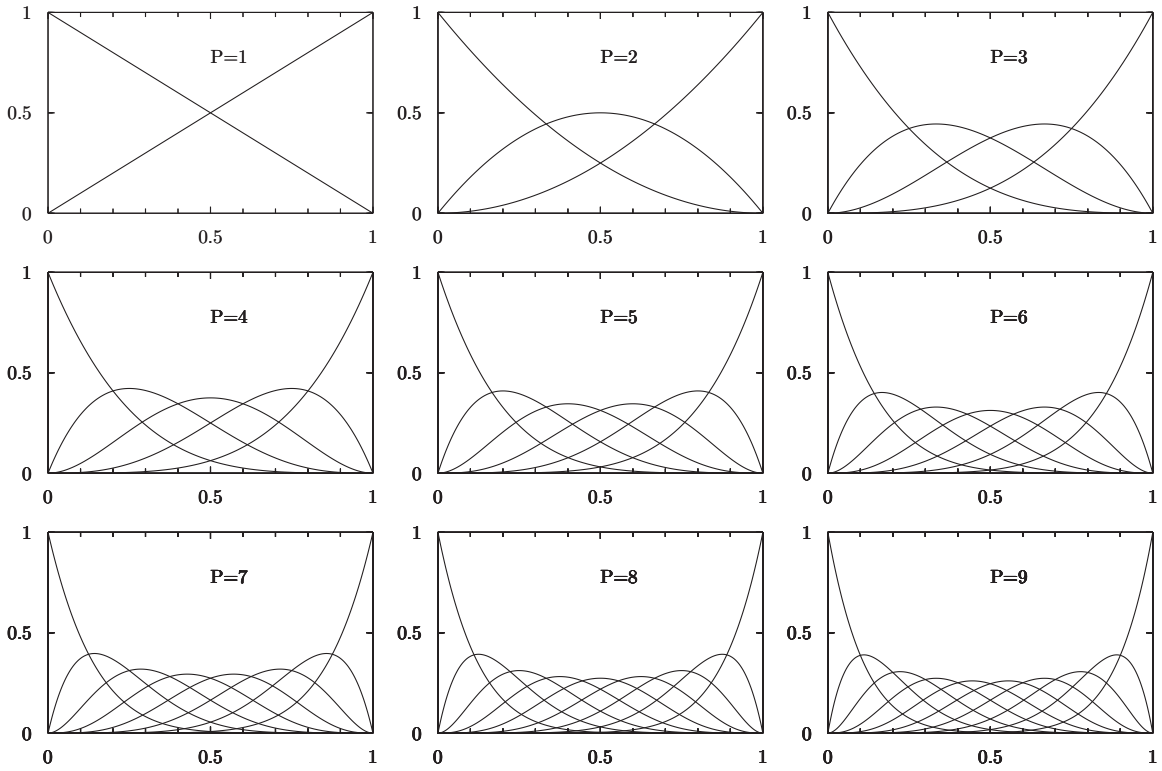


Figure 2.2: B-Spline Basis Functions

definition shows that some useful properties are: convex shell (i.e., values remain inside an envelope of coefficients magnitudes), tri-symmetric about natural coordinates, polynomial basis, unit norm and the maximum is regularly spaced.

2.2.1 Derivatives

The derivative with the directional derivative operator D is

$$\begin{aligned} Dp(P) &= \sum_{i+j=d} c_{ij} \frac{d!}{i!j!} \zeta_1^{i-1} \zeta_2^j i D\zeta_1 \\ &+ \sum_{i+j=d} c_{ij} \frac{d!}{i!j!} \zeta_1^i \zeta_2^{j-1} j D\zeta_2 \end{aligned}$$

Reindexing i and j allows the derivative to reappear as a lower order B-Spline with a derivative coefficient vector $\hat{c}_{ij}$

$$Dp(P) = \sum_{i+j=d-1} \hat{c}_{ij} \frac{(d-1)!}{i!j!} \zeta_1^i \zeta_2^j$$

with

$$\hat{c}_{ij} = d c_{(i+1)j} D\zeta_1 + d c_{i(j+1)} D\zeta_2$$

This simplifies the implementation. A similar derivation is possible for higher derivatives; the coefficient stencil becomes wider.

2.2.2 Integration

The integral over the line element given a polynomial distribution is the identity[42]

$$I = \int_A \zeta_1^l \zeta_2^m dA = \frac{l!m!}{(l+m+1)!}$$

Recall that the B-Spline definition in 1D is

$$p(\zeta_1, \zeta_2) = \sum_{i+j=d} c_{ij} \frac{d!}{i!j!} \zeta_1^i \zeta_2^j$$

Notice that the B-Spline definition and the polynomial integral both contain $l!m!$ and $1/i!j!$. These directly cancel when applying the polynomial integration identity to the B-Spline. Applying gives integrals for a known value coefficient vector.

$$\begin{aligned} I &= \int p \\ &= \sum_{i+j=d} c_{ij} \frac{d!}{i!j!} \frac{l!m!}{(i+j+1)!} \\ &= \sum_{i+j=d} c_{ij} \frac{d!}{(d+1)!} \\ &= \frac{1}{(d+1)} \sum_{i+j=d} c_{ij} \end{aligned}$$

Simply put, B-Spline integration is a simple summation of coefficients divided by the number of coefficients. Compared with Gauss numerical integration, the integration weights are constant. This should simplify implementation.

2.2.3 Multiplication

Multiplication is another critical operation. The B-Spline is

$$p(P) = \sum_{i+j=d} c_{ij} \frac{d!}{i!j!} \zeta_1^i \zeta_2^j$$

Define e as

$$e_{ij} = \frac{d!}{i!j!}$$

Multiplication is

$$p(P)q(P) = \sum_{i+j=d_1} c_{ij} \frac{d!}{i!j!} \zeta_1^i \zeta_2^j \sum_{m+n=d_2} d_{mn} \frac{d!}{m!n!} \zeta_1^m \zeta_2^n$$

compare with p for each value gives

$$p(P)q(P) = \sum_{s+t=d_1+d_2} f_{st} \frac{d!}{s!t!} \zeta_1^s \zeta_2^t$$

Multiplication Patterns

This section attempts to find patterns in the numerical multiplication of two B-Splines with coefficients a and b. Notice that multiplication routines can be pre-calculated for known orders. This gives an equivalent set of coefficients f such that

$$f = M(a, b)$$

For example, multiplication of an order 2 by an order 1 gives the four coefficents of f as

$$f_1 \equiv f_{30} = \frac{c_{20}e_{20}c_{10}e_{10}}{e_{30}}$$

$$f_2 \equiv f_{21} = \frac{c_{11}e_{11}c_{10}e_{10} + c_{20}e_{20}c_{01}e_{01}}{e_{21}}$$

$$f_3 \equiv f_{12} = \frac{c_{11}e_{11}c_{01}e_{01} + c_{02}e_{02}c_{10}e_{10}}{e_{12}}$$

$$f_4 \equiv f_{03} = \frac{c_{02}e_{02}c_{01}e_{01}}{e_{03}}$$

This can be hardcoded for efficiency for each specific multiplication.

$$\begin{bmatrix} d_{20} \\ d_{11} \\ d_{02} \end{bmatrix} = \begin{bmatrix} a_{10} & 0 \\ a_{01} & a_{10} \\ 0 & a_{01} \end{bmatrix} \begin{bmatrix} b_{10} \\ b_{01} \end{bmatrix}$$

$$\begin{bmatrix} d_{40} \\ d_{31} \\ d_{22} \\ d_{13} \\ d_{04} \end{bmatrix} = \begin{bmatrix} a_{20} & 0 & 0 \\ a_{11} & a_{20} & 0 \\ a_{02} & a_{11} & a_{20} \\ 0 & a_{02} & a_{11} \\ 0 & 0 & a_{02} \end{bmatrix} \begin{bmatrix} b_{20} \\ b_{11} \\ b_{02} \end{bmatrix}$$

$$\begin{bmatrix} d_{60} \\ d_{51} \\ d_{42} \\ d_{33} \\ d_{24} \\ d_{15} \\ d_{06} \end{bmatrix} = \begin{bmatrix} a_{40} & 0 & 0 \\ a_{31} & a_{40} & 0 \\ a_{22} & a_{31} & a_{40} \\ a_{13} & a_{22} & a_{31} \\ a_{04} & a_{13} & a_{22} \\ & a_{04} & a_{13} \\ & & a_{04} \end{bmatrix} \begin{bmatrix} b_{20} \\ b_{11} \\ b_{02} \end{bmatrix}$$

Combining gives

$$\begin{bmatrix} ab_{40} \\ ab_{31} \\ ab_{22} \\ ab_{13} \\ ab_{04} \end{bmatrix} = \begin{bmatrix} a_{20} & 0 & 0 \\ a_{11} & a_{20} & 0 \\ a_{02} & a_{11} & a_{20} \\ 0 & a_{02} & a_{11} \\ 0 & 0 & a_{02} \end{bmatrix} \begin{bmatrix} b_{20} \\ b_{11} \\ b_{02} \end{bmatrix} = \begin{bmatrix} a_{20}b_{20} \\ a_{11}b_{20} + a_{20}b_{11} \\ a_{02}b_{20} + a_{11}b_{11} + a_{20}b_{02} \\ a_{02}b_{11} + a_{11}b_{02} \\ a_{02}b_{02} \end{bmatrix}$$

$$\begin{bmatrix} d_{60} \\ d_{51} \\ d_{42} \\ d_{33} \\ d_{24} \\ d_{15} \\ d_{06} \end{bmatrix} = \begin{bmatrix} ab_{40} & & \\ ab_{31} & ab_{40} & \\ ab_{22} & ab_{31} & ab_{40} \\ ab_{13} & ab_{22} & ab_{31} \\ ab_{04} & ab_{13} & ab_{22} \\ & ab_{04} & ab_{13} \\ & & ab_{04} \end{bmatrix} \begin{bmatrix} c_{20} \\ c_{11} \\ c_{02} \end{bmatrix}$$

This operation is convienent and numerically practical.

2.2.4 Conversion between Orders

Here, conversion between different polynomial orders is discussed. Naturally, conversion is only loss-less* when converting to a higher order polynomial. Order conversion does not require symbolic algebra. Multiplication already provides a method to increase order. Replicating the above

*Loss-less means that information is not lost; the original function can be recovered.

conversion from 4 to 6 is possible by multiplying by a unit basis of order 2.

$$\begin{bmatrix} d_{60}e_{60} \\ d_{51}e_{51} \\ d_{42}e_{42} \\ d_{33}e_{33} \\ d_{24}e_{42} \\ d_{15}e_{15} \\ d_{06}e_{06} \end{bmatrix} = \begin{bmatrix} b_{40}e_{40} & & \\ b_{31}e_{31} & b_{40}e_{40} & \\ b_{22}e_{22} & b_{31}e_{31} & b_{40}e_{40} \\ b_{13}e_{13} & b_{22}e_{22} & b_{31}e_{31} \\ b_{04}e_{04} & b_{13}e_{13} & b_{22}e_{22} \\ & b_{04}e_{04} & b_{13}e_{13} \\ & & b_{04}e_{04} \end{bmatrix} \begin{bmatrix} c_{20}e_{20} \\ c_{11}e_{11} \\ c_{02}e_{02} \end{bmatrix}$$

where $c_{ij} = 1$. Remember that all terms are premultiplied by e_{ij}.

$$\begin{bmatrix} d_{60} \\ d_{51} \\ d_{42} \\ d_{33} \\ d_{24} \\ d_{15} \\ d_{06} \end{bmatrix} = \begin{bmatrix} \frac{e_{20}e_{40}}{e_{60}} & & & & & \\ \frac{e_{11}e_{40}}{e_{51}} & \frac{e_{20}b_{31}}{e_{51}} & & & & \\ \frac{e_{02}e_{40}}{e_{42}} & \frac{e_{11}b_{31}}{e_{42}} & \frac{e_{20}b_{22}}{e_{42}} & & & \\ & \frac{e_{02}b_{31}}{e_{33}} & \frac{e_{11}b_{22}}{e_{33}} & \frac{e_{20}b_{13}}{e_{33}} & & \\ & & \frac{e_{02}b_{22}}{e_{42}} & \frac{e_{11}b_{13}}{e_{42}} & \frac{e_{20}b_{04}}{e_{42}} & \\ & & & \frac{e_{02}b_{13}}{e_{15}} & \frac{e_{11}b_{04}}{e_{15}} & \\ & & & & \frac{e_{02}b_{04}}{e_{06}} & \end{bmatrix} \begin{bmatrix} b_{40} \\ b_{31} \\ b_{22} \\ b_{13} \\ b_{04} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & & & & \\ \frac{1}{3} & \frac{2}{3} & & & \\ \frac{1}{15} & \frac{8}{15} & \frac{2}{5} & & \\ & \frac{1}{5} & \frac{3}{5} & \frac{1}{5} & \\ & & \frac{2}{5} & \frac{8}{15} & \frac{1}{15} \\ & & & \frac{2}{3} & \frac{1}{3} \\ & & & & 1 \end{bmatrix} \begin{bmatrix} b_{40} \\ b_{31} \\ b_{22} \\ b_{13} \\ b_{04} \end{bmatrix}$$

2.3 2D Bezier Splines

The B-Spline definition in 2D is

$$p(\zeta_1, \zeta_2, \zeta_3) = \sum_{i+j+k=d} c_{ijk} \frac{d!}{i!j!k!} \zeta_1^i \zeta_2^j \zeta_3^k$$

The 2D form has the same useful characteristics that the 1D form contains. The number of coefficients is

$$N = \frac{(d+1)(d+2)}{2}$$

s	i	j	k	g
1	4	0	0	1
2	3	1	0	2
3	3	0	1	2
4	2	2	0	3
5	2	1	1	3
6	2	0	2	3
7	1	3	0	4
8	1	2	1	4
9	1	1	2	4
10	1	0	3	4
11	0	4	0	5
12	0	3	1	5
13	0	2	2	5
14	0	1	3	5
15	0	0	4	5

Table 2.1: B-Spline Indices and Mapping for $d = 4$

2.3.1 2D Indices

B-Splines are defined with an ijk notation. For efficient computer implementations, we prefer a single index s. So we need a function S to map $\mathbb{R}^3$ to $\mathbb{R}^1$

$$s = S(i, j, k, d)$$

The inverse map is also needed.

$$\{i, j, k\} = S^{-1}(s, d)$$

Understandably, multiple S mapping functions exist; S is not unique. The following develops a mapping corresponding to Table 2.1 for $d = 4$, where i is the most significant index and k is the least significant index. Figure 2.3 graphically shows the triangle geometry, indices, and group number for a $d = 4$ B-Spline expansion.

The mapping function S is

$$s \quad = \quad 1 + k + \sum_{x=1}^{g-1} x$$

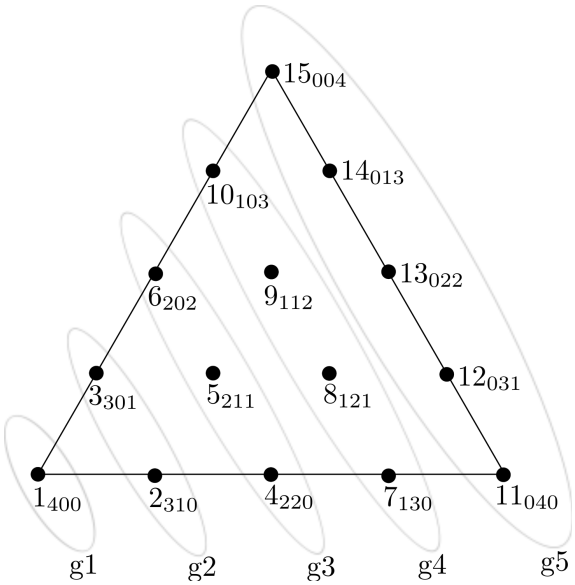


Figure 2.3: B-Spline Geometry and Indices for $d = 4$

Substituting for an arithmetic series identity[75] gives

$$s = 1 + k + \frac{g^2 - g}{2}$$

The group number is

$$g = d - i + 1$$

or

$$g = j + k + 1$$

Simplifying gives

$$s = 1 + k + \frac{d}{2} - \frac{i}{2} - di + \frac{d^2 + i^2}{2}$$

or

$$s = 1 + \frac{j + 3k}{2} + jk + \frac{j^2 + k^2}{2}$$

The inverse mapping function is extracted by noticing that

$$i = d - g + 1$$

$$k = s - \frac{g^2 - g}{2} - 1$$

$$j = d - i - k$$

This still requires finding g. But, s is in group g if

$$s \quad = \quad 1 + k + \frac{g^2 - g}{2}$$

Solving for g gives

$$\begin{aligned} g &= \frac{1}{2} + \frac{\sqrt{1-8(1+k-s)}}{2} \\ &= \frac{1}{2} + \frac{\sqrt{8s-8k-7}}{2} \end{aligned}$$

Unfortunately, we don't know k. However, we do know that changing k does not influence the group number. So, we set $k = 0$ and round-down g to an integer. This gives

$$g = \lfloor \left(\frac{1}{2} + \frac{\sqrt{8s-7}}{2} \right)$$

or in Fortran

$$g = int \left(\frac{1}{2} + \frac{\sqrt{8s-7}}{2} \right)$$

2.3.2 Integration

The integral over the natural triangle given a polynomial distribution is[42]

$$I = \int_A \zeta_1^l \zeta_2^m \zeta_3^n d\Omega = \frac{l!m!n!}{(l+m+n+2)!}$$

Again, the B-Spline function and the integration identity cancel the factorial terms. Substitution for the basis definition and applying the exact integral identity above gives

$$\begin{aligned} I &= \int p \\ &= \sum_{i+j+k=d} c_{ijk} \frac{d!}{i!j!k!} \frac{l!m!n!}{(i+j+k+2)!} \\ &= \sum_{i+j+k=d} c_{ijk} \frac{d!}{(d+2)!} \\ &= \frac{1}{(d+2)(d+1)} \sum_{i+j+k=d} c_{ijk} \end{aligned}$$

Substitution with N, the number of coefficients, reduces integration to

$$I = \frac{1}{2N} \sum c$$

or

$$I = \frac{1}{2N} \sum c$$

Again, B-Spline integration over the local element is a simple summation of coefficients divided by twice the number of coefficients. The B-Spline basis is well suited to coefficient based integration.

2.3.3 Multiplication

The B-Spline is

$$p(\zeta_1, \zeta_2, \zeta_3) = \sum_{i+j+k=d} c_{ijk} \frac{d!}{i!j!k!} \zeta_1^i \zeta_2^j \zeta_3^k$$

Multiplication is

$$p(P)q(P) = \sum_{i+j+k=d_1} a_{ijk} \frac{d!}{i!j!k!} \zeta_1^i \zeta_2^j \zeta_3^k \sum_{m+n+o=d_2} b_{mno} \frac{d!}{m!n!o!} \zeta_1^m \zeta_2^n \zeta_2^o$$

compare with p for each value gives

$$p(P)q(P) = \sum_{s+t+u=d_1+d_2} f_{st} \frac{d!}{s!t!u!} \zeta_1^s \zeta_2^t \zeta_3^u$$

For example, this expands to

$$\begin{bmatrix} f_{200} \\ f_{110} \\ f_{101} \\ f_{020} \\ f_{011} \\ f_{002} \end{bmatrix} = \begin{bmatrix} b_{100} & & \\ b_{010} & b_{100} & \\ b_{001} & & b_{100} \\ & b_{010} & \\ & b_{001} & b_{010} \\ & & b_{001} \end{bmatrix} \begin{bmatrix} a_{100} \\ a_{010} \\ a_{001} \end{bmatrix}$$

$$\begin{bmatrix} f_{300} \\ f_{210} \\ f_{201} \\ f_{120} \\ f_{111} \\ f_{102} \\ f_{030} \\ f_{021} \\ f_{012} \\ f_{003} \end{bmatrix} = \begin{bmatrix} a_{100} & & & & & \\ a_{010} & a_{100} & & & & \\ a_{001} & & a_{100} & & & \\ & a_{010} & & a_{100} & & \\ & a_{001} & a_{010} & & a_{100} & \\ & & a_{001} & & & a_{100} \\ & & & a_{010} & & \\ & & & a_{001} & a_{010} & \\ & & & & a_{001} & a_{010} \\ & & & & & a_{001} \end{bmatrix} \begin{bmatrix} b_{200} \\ b_{110} \\ b_{101} \\ b_{020} \\ b_{011} \\ b_{002} \end{bmatrix}$$

$$\begin{bmatrix} f_{300} \\ f_{210} \\ f_{201} \\ f_{120} \\ f_{111} \\ f_{102} \\ f_{030} \\ f_{021} \\ f_{012} \\ f_{003} \end{bmatrix} = \begin{bmatrix} b_{200} & & \\ b_{110} & b_{200} & \\ b_{101} & & b_{200} \\ b_{020} & b_{110} & \\ b_{011} & b_{101} & b_{110} \\ b_{002} & & b_{101} \\ & b_{020} & \\ & b_{011} & b_{020} \\ & b_{002} & b_{011} \\ & & b_{002} \end{bmatrix} \begin{bmatrix} a_{100} \\ a_{010} \\ a_{001} \end{bmatrix}$$

$$\begin{bmatrix} b_{400} \\ b_{310} \\ b_{301} \\ b_{220} \\ b_{211} \\ b_{202} \\ b_{130} \\ b_{121} \\ b_{112} \\ b_{103} \\ b_{040} \\ b_{031} \\ b_{022} \\ b_{013} \\ b_{004} \end{bmatrix} = \begin{bmatrix} a_{200} & & & & & \\ a_{110} & a_{200} & & & & \\ a_{101} & & a_{200} & & & \\ a_{020} & a_{110} & & a_{200} & & \\ a_{011} & a_{101} & a_{110} & & a_{200} & \\ a_{002} & & a_{101} & & & a_{200} \\ & a_{020} & & a_{101} & & \\ & a_{011} & a_{020} & a_{101} & a_{110} & \\ & a_{002} & a_{011} & & a_{101} & a_{110} \\ & & a_{002} & & & a_{101} \\ & & & a_{020} & & \\ & & & a_{011} & a_{020} & \\ & & & a_{002} & a_{011} & a_{020} \\ & & & & a_{002} & a_{011} \\ & & & & & a_{002} \end{bmatrix} \begin{bmatrix} b_{200} \\ b_{110} \\ b_{101} \\ b_{020} \\ b_{011} \\ b_{002} \end{bmatrix}$$

This form is not as simple as the 1D counterpart. This added complexity dooms direct coefficient multiplication.

2.3.4 Derivatives

The definition of the basis is

$$p(\zeta_1, \zeta_2, \zeta_3) = \sum_{i+j+k=d} c_{ijk} \frac{d!}{i!j!k!} \zeta_1^i \zeta_2^j \zeta_3^k$$

Using a derivative operator D for a generic direction, the basis derivative is

$$Dp(\zeta_1,\zeta_2,\zeta_3) = \sum_{i+j+k=d} c_{ijk}\frac{d!}{i!j!k!}\left(\zeta_1^{i-1}\zeta_2^j\zeta_3^k iD\zeta_1 + \zeta_1^i\zeta_2^{j-1}\zeta_3^k jD\zeta_2 + \zeta_1^i\zeta_2^j\zeta_3^{k-1}kD\zeta_3\right)$$

Renumbering the sum gives

$$Dp(P) \quad = \quad \sum_{i+j+k=d-1} \hat{c}_{ijk}\frac{(d-1)!}{i!j!k!}\zeta_1^i\zeta_2^j\zeta_3^k$$

with

$$\hat{c}_{ijk} = dc_{(i+1)jk}D\zeta_1 + dc_{i(j+1)k}D\zeta_2 + dc_{ij(k+1)}D\zeta_3$$

Notice that the derivative B-Spline is one order lower. In practice, up-converting back to order d is recommended. Conversion is discussed in Section 2.3.5.

Example

Remembering the geometry constraint $\zeta_1 + \zeta_2 + \zeta_3 = 1$ implies $d\zeta_1 = -d\zeta_2 - d\zeta_3$. For example, to pick out the ζ_1 derivative, the operator is

$$\begin{pmatrix} D\zeta_1 \\ D\zeta_2 \\ D\zeta_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$$

so the coefficients are

$$\hat{c}_{ijk} = dc_{(i+1)jk} - dc_{ij(k+1)}$$

These coefficients form the new B-Spline representing the zeta directional derivative.

2.3.5 Conversion

Converting orders is necessary to simplify the spline mathematics. Conversion allows for operations in the lowest necessary order for numerical efficiency[†]. Up-conversion preserves function information; down-conversion loses information and will not be discussed. When using the previously derived multiplication operation, up-conversion from order m to order n requires multiplication by a unity spline of the difference $n - m$.

[†]Compare with Gauss Integration where operations must either be performed optimally on a variable local grid or sub-optimally on a local grid necessary to capture the highest order operation.

For example, to convert a 1st order spline to a second order spline requires multiplying by a 1st order unity spline. In matrix form, the operation is

$$\begin{bmatrix} f_{200} \\ f_{110} \\ f_{101} \\ f_{020} \\ f_{011} \\ f_{002} \end{bmatrix} = \begin{bmatrix} b_{100} & & \\ b_{010} & b_{100} & \\ b_{001} & & b_{100} \\ & b_{010} & \\ & b_{001} & b_{010} \\ & & b_{001} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

As implemented, all operations return a spline corresponding to multiples of the state variable order.

2.4 Value Based Operations

The above B-Spline operations are wonderful from a numerical standpoint but are unusably slow. So the mathematical operations will be revisited with absolute speed as a goal.

2.4.1 Values and Coefficients

In general, converting between coefficients and values takes the form

$$V = [B]\, a$$

A major difference with value based operations is that values must be calculated at off nodal points. The number of nodal points needed for a given order is known beforehand, so the conceptual process is to directly convert values from the coefficients.

2.4.2 Integration

Integration is perfectly efficient for the B-Spline; the integral reduces to a simple summation of coefficients. In general, converting between coefficients and values takes the form

$$V = [B]\, a$$

so that the coefficients are

$$a = \left[B^{-1}\right] V$$

Remembering that the B-Spline integral is of the form

$$I = \frac{1}{C} \sum a$$

allows the integral to be written as

$$I = \frac{1}{C} \sum \left[B^{-1}\right] V$$

Expanding the definition of a sum gives the following integral form

$$I = \frac{1}{C} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \end{bmatrix} \left[B^{-1}\right] \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ \vdots \\ V_N \end{bmatrix}$$

Three terms can be premultiplied and stored to give

$$I = w_i V_i$$

where

$$w = \frac{1}{C} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \end{bmatrix} \left[B^{-1}\right]$$

This form requires an identical number of operations as before but with values rather than coefficients.

2.4.3 Multiplication

The primary slowness comes from multiplication of coefficients. The objective is to convert to value based operations when multiplying. Multiplication is trivial.

$$M(V_i, W_j) = \delta_{ij} V_i W_j$$

The primary constraint is that the pointwise multiplication must include a sufficient number of points to capture the relevant order. Multiplying a c order function with a d order function yields a $c + d$ order function. This requires all values be computed in the highest resulting order. Failure to maintain a sufficient order results in aliasing.

2.4.4 Derivatives

Value based derivatives are more complicated. From the spline section, the directional derivative coefficients are functions of value coefficients

$$a_{d\zeta_i} = \left[D\zeta_i\right] a$$

Converting to coefficients from values is a known process

$$a = \left[B^{-1}\right] V$$

Combining gives

$$a_{d\zeta_i} = [D\zeta_i] \left[B^{-1}\right] V$$

Unfortunately, differentiation reduces the absolute order by one. The resulting derivative values are

$$V_{\zeta_i} = \left[B^{+}\right] a_{d\zeta_i} = \left[B^{+}\right] [D\zeta_i] \left[B^{-1}\right] V$$

The matrix terms are constant and should be premultiplied and stored for each order and direction.

Chapter 3

Higher Order Methodology

This chapter describes the governing equations and forms a higher order numerical method for their solution. The solution domain is decomposed into triangular finite elements. The fluid dynamics governing equations are conservation of mass, momentum, energy, and entropy on a differential fluid element. This section defines the fluid properties and develops the partial differential equations describing the fluid dynamics.

3.1 Nomenclature and Properties of Fluids

This section discusses thermodynamic, fluid, and flow properties and their nomenclature. The section discusses the ideal gas model and flow scales. This project exclusively uses the ideal gas approximation. Remembering that this project's goals are to investigate concepts and methods weakens the need to use better gas models. For comparison, the ideal gas model is designed for standard temperature and pressure gas states in single state gases. For more information regarding the ideal gas's operating limitations refer to Moran[52]. For an ideal gas, the ratio of constant pressure to constant volume specific heat is

$$\gamma = \frac{c_p}{c_v}$$

The gas constant for the ideal gas model is defined as

$$R = c_p - c_v$$

Thermodynamic fluid properties are defined by Moran[52] as "macroscopic characteristics of a system such as mass, volume, energy and temperature to which numerical values can be assigned at a given time without knowledge of the history of the system." Specific properties refer to unit amounts per unit of mass.

The density is defined as

$$\rho$$

The velocity vector is

$$U = V = \begin{pmatrix} u \\ v \end{pmatrix}$$

Total specific energy is

$$e$$

Total Energy is

$$E = \rho e$$

Internal specific energy is

$$\hat{e} = e_{int} = e - \frac{1}{2} V \cdot V$$

Internal Energy is

$$\hat{E} = \rho e - \frac{1}{2} \rho V \cdot V$$

Temperature with the assumption of ideal gas is

$$T = \frac{\hat{e}}{c_v} = \frac{1}{c_v} \left(e - \frac{1}{2} V \cdot V \right)$$

Pressure with the assumption of ideal gas is

$$p = \rho R T = \rho \left(\gamma - 1 \right) \left(e - \frac{1}{2} V \cdot V \right) = \left(\gamma - 1 \right) \left(\hat{E} \right)$$

Specific Enthalpy is

$$h = e + RT = \hat{e} + \frac{p}{\rho} + \frac{1}{2} V \cdot V = \frac{\gamma}{\gamma - 1} \frac{p}{\rho} + \frac{1}{2} V \cdot V$$

Total Enthalpy is

$$H = E + p$$

Specific entropy is

$$s$$

Entropy is

$$S = \rho s$$

Viscosity is

$$\mu$$

Bulk Viscosity is

$$\hat{\lambda} = \lambda \mu$$

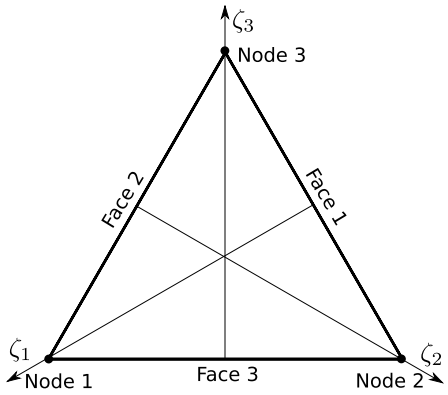


Figure 3.1: Natural Triangle Element

Flow property models are non-dimensional representations of flow characteristic scales. The Mach number assuming an ideal gas is

$$M = \frac{V}{a} = V/\sqrt{\gamma\left(\gamma-1\right)\left(e-\frac{1}{2}V\cdot V\right)}$$

when the acoustic speed is

$$a = \sqrt{\gamma RT} = \sqrt{\gamma\left(\gamma-1\right)\left(e-\frac{1}{2}V\cdot V\right)}$$

The Reynolds number is

$$Re = \frac{\rho V L}{\mu}$$

The Prandtl number is

$$Pr = \frac{\mu c_p}{k}$$

These properties and characteristics are sufficient to describe the inviscid and viscous flows under consideration in this project.

3.2 2D Coordinate System

The solution domain is decomposed into many triangular elements. These elements provide the structure necessary for locating and computing values from a solution state vector. The representation used for this project is a natural coordinate system.

A natural coordinate system for triangles provides 3 fold symmetry of a triangle. Coordinate directions begin at edges with value zero and proceed to the opposing vertex with value one. Figure 3.1 shows a reference triangle with this natural coordinate system. For 2D, the triangle description requires three dependent coordinates, ζ_1,ζ_2,ζ_3, where the summation of these coordinates must be

unity.

$$\zeta_1 + \zeta_2 + \zeta_3 = 1$$

The B-Spline coordinate easily fits into the triangle's natural coordinate system. Later, the need to reduce a 2D triangle to 1D segments will become important with boundary integrals. One dimensional segment descriptions are created by reduction of the 2D descriptions to two dependent coordinates.

3.2.1 Linear Geometry Element

The natural coordinate system described above is a local coordinate system. Potentially, the local coordinates could be mapped to a non-linear curved element. For this project the geometry mapping is restricted to a linear map. There are implications to this linear map, especially when describing and enforcing boundary conditions. These implications will be discussed later.

Returning to the concept of a linear mapping between the local and global coordinates allow for a simple mapping function. The local to global coordinate transform is[26]

$$\begin{pmatrix} 1 \\ x \\ y \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix} \begin{pmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \end{pmatrix}$$

where x and y are nodal locations. The upper row of ones indicates the constraint that within the element, the summation of dependent local coordinates is exactly 1. Substituting the top row reduces the transform to two independent coordinates[14]

$$\begin{aligned} \begin{pmatrix} x \\ y \end{pmatrix} &= \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} + \begin{bmatrix} (x_1 - x_3) & (x_2 - x_3) \\ (y_1 - y_3) & (y_1 - y_3) \end{bmatrix} \begin{pmatrix} \zeta_1 \\ \zeta_2 \end{pmatrix} \\ &= \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} + \begin{bmatrix} x_{13} & x_{23} \\ y_{13} & y_{23} \end{bmatrix} \begin{pmatrix} \zeta_1 \\ \zeta_2 \end{pmatrix} \\ &= \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} + B \begin{pmatrix} \zeta_1 \\ \zeta_2 \end{pmatrix} \end{aligned}$$

Notationally, the coordinate form x_{13} indicates $x_1 - x_3$. This is in a form useful for numerical calculations. It is noted that the reference coordinates for the third node place the element's location; the B transformation matrix indicates the orientation.

For later use, a transformation must be created from the global to the local frame. Specifying

the Jacobian as the transpose of the B gives

$$\mathbf{J} = \begin{bmatrix} x_{13} & y_{13} \\ x_{23} & y_{23} \end{bmatrix} = B^T$$

The Inverse Jacobian is

$$\mathbf{J}^{-1} = \frac{1}{\det J} \begin{bmatrix} y_{23} & -y_{13} \\ -x_{23} & x_{13} \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} y_{23} & -y_{13} \\ -x_{23} & x_{13} \end{bmatrix}$$

Notice that the determinate of the B matrix is twice the element's area.

3.2.2 Derivatives

Transforming derivatives from the global frame to the local frame is a necessary operation. From the chain rule, derivatives in the local frame are related to derivatives in the global frame by

$$\begin{pmatrix} \frac{d}{d\zeta_1} \\ \frac{d}{d\zeta_2} \end{pmatrix} = \begin{bmatrix} \frac{dx_1}{d\eta_1} & \frac{dx_2}{d\eta_1} \\ \frac{dx_1}{d\eta_2} & \frac{dx_2}{d\eta_2} \end{bmatrix} \begin{pmatrix} \frac{d}{dx} \\ \frac{d}{dy} \end{pmatrix}$$

The matrix converting derivatives was defined above as the Jacobian.

$$\mathbf{J} = \begin{bmatrix} \frac{dx_1}{d\eta_1} & \frac{dx_2}{d\eta_1} \\ \frac{dx_1}{d\eta_2} & \frac{dx_2}{d\eta_2} \end{bmatrix}$$

Typically for finite element calculations, the derivatives in the global frame are calculated from local frame derivatives. Inverting the previous transformation gives

$$\begin{pmatrix} \frac{d}{dx} \\ \frac{d}{dy} \end{pmatrix} = J^{-1} \begin{pmatrix} \frac{d}{d\zeta_1} \\ \frac{d}{d\zeta_2} \end{pmatrix}$$

where J^{-1} is referring to the inverse Jacobian. Matrix inversion for this 2 by 2 matrix is exactly

$$\begin{aligned} \mathbf{J}^{-1} &= \frac{1}{\det J} \begin{bmatrix} \frac{dx_2}{d\eta_2} & -\frac{dx_2}{d\eta_1} \\ -\frac{dx_1}{d\eta_2} & \frac{dx_1}{d\eta_1} \end{bmatrix} \\ &= \frac{1}{2A} \begin{bmatrix} \frac{dx_2}{d\eta_2} & -\frac{dx_2}{d\eta_1} \\ -\frac{dx_1}{d\eta_2} & \frac{dx_1}{d\eta_1} \end{bmatrix} \end{aligned}$$

The form is simplified by recalling that the cross product of triangle edge differences, in the guise of a determinate, equals twice the triangle's area. In terms of global coordinate differences and element area, the inverse Jacobian is

$$\mathbf{J}^{-1} = \frac{1}{2A} \begin{bmatrix} y_{23} & -y_{13} \\ -x_{23} & x_{13} \end{bmatrix}$$

Decomposed Jacobian

Later, decomposing the Jacobian and inverse Jacobian matrix will provide an elegant approach to removing an integration term in the governing equations. Notice that the inverse Jacobian contains a permutation of the J terms divided by the determinate of the Jacobian, $|J|$. More importantly, the permuted terms in J^{-1} are well behaved. The modified Jacobian without the$|J|$term is

$$\hat{\mathbf{J}}^{-\mathbf{1}} = |J|\, J^{-1} = \begin{bmatrix} \frac{dx_2}{d\eta_2} & -\frac{dx_2}{d\eta_1} \\ -\frac{dx_1}{d\eta_2} & \frac{dx_1}{d\eta_1} \end{bmatrix} = \begin{bmatrix} \frac{d\hat{\eta}_1}{dx} & \frac{d\hat{\eta}_1}{dy} \\ \frac{d\hat{\eta}_2}{dx} & \frac{d\hat{\eta}_2}{dy} \end{bmatrix}$$

Alternatively, this is

$$\begin{pmatrix} \frac{d}{dx} \\ \frac{d}{dy} \end{pmatrix} = \frac{1}{|J|}\hat{\mathbf{J}}^{-\mathbf{1}} \begin{pmatrix} \frac{d}{d\zeta_1} \\ \frac{d}{d\zeta_2} \end{pmatrix}$$

Directional Derivatives

Transferring the directional derivative between frames also involves the Jacobian transpose

$$\begin{pmatrix} dx \\ dy \end{pmatrix} = \begin{pmatrix} \frac{dx}{d\zeta_1} & \frac{dx}{d\zeta_2} \\ \frac{dy}{d\zeta_1} & \frac{dy}{d\zeta_2} \end{pmatrix} \begin{pmatrix} d\zeta_1 \\ d\zeta_2 \end{pmatrix} = J^T \begin{pmatrix} d\zeta_1 \\ d\zeta_2 \end{pmatrix} = \begin{pmatrix} d\zeta_1 & d\zeta_2 \end{pmatrix} [J]$$

Thus, the local frame directions necessary for unit global frame operations are

$$\begin{pmatrix} d\zeta_1 \\ d\zeta_2 \end{pmatrix} = J^{-T} \begin{pmatrix} dx \\ dy \end{pmatrix} = \frac{1}{2A} \begin{bmatrix} \frac{dx_2}{d\eta_2} & -\frac{dx_1}{d\eta_2} \\ -\frac{dx_2}{d\eta_1} & \frac{dx_1}{d\eta_1} \end{bmatrix} \begin{pmatrix} dx \\ dy \end{pmatrix}$$

So for the x direction global frame, the directional derivative is

$$dx = \begin{pmatrix} dz_1 \\ dz_2 \end{pmatrix} = \frac{1}{2A} \begin{bmatrix} \frac{dx_2}{d\eta_2} & -\frac{dx_1}{d\eta_2} \\ -\frac{dx_2}{d\eta_1} & \frac{dx_1}{d\eta_1} \end{bmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

For the y direction, the global frame directional derivative is

$$dy = \begin{pmatrix} dz_1 \\ dz_2 \end{pmatrix} = \frac{1}{2A} \begin{bmatrix} \frac{dx_2}{d\eta_2} & -\frac{dx_1}{d\eta_2} \\ -\frac{dx_2}{d\eta_1} & \frac{dx_1}{d\eta_1} \end{bmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{|J|} \begin{pmatrix} 0 & 1 \end{pmatrix} \left[\hat{J}^{-1}\right]$$

This allows us to compute global frame derivatives with local frame operators.

3.2.3 Integration

Integration is a critical operation for Galerkin based Finite Element solvers. The governing equations are derived in a global frame; however, element operations are in the local element frame. The area mapping between frames is through the element's Jacobian. For a global domain V with an elemental domain Ω, the following is an identity.

$$\int_V F(X)dV = \int_\Omega F(\eta)\,|J|\,d\Omega$$

When the dimensionality is known (i.e., 1D, 2D, or 3D), integrals may be expressed in local elemental coordinates with the knowledge that not all coordinates are independent. For example, a 2D integral may appear as

$$\int F(\eta)\,|J|\,d\eta_1 d\eta_2 d\eta_3$$

even though the constraint $\eta_1 + \eta_2 + \eta_3 = 1$ holds. In general, the Ω terminology more closely fits the numerical integration process, which operates on the local element as a whole.

Integration by Parts

Integration by parts via Green's theorem is used often in finite element derivations. As confusion often arises with this transformation, the derivation and assumptions are explicitly stated without proof. Green's theorem[71] is

$$\int (\nabla u \cdot w + u\nabla \cdot w)\; d\Omega = \int uv \cdot n\, d\Gamma$$

with the requirements that u and w are continuously differentiable[67], which is often stated as C^1. The derivatives of u and w should be continuous.

For further understanding, the integration by parts identity is derived. First, take two functions u and w. Take the divergence of their product and expand with the chain rule. This requires that u and w be differentiable.

$$\nabla \cdot (uw) = u \cdot \nabla w + \nabla u \cdot w$$

Now take the volume integral

$$\int_\Omega \nabla \cdot (uw)\; d\Omega = \int_\Omega u \cdot \nabla w\, d\Omega + \int_\Omega \nabla u \cdot w\, d\Omega$$

Remembering the divergence integral theorem with its assumption of continuous derivatives as

$$\int_\Omega div(v)\, d\Omega = \int_\Gamma v \cdot n\, d\Gamma$$

allows for substitution into the previous volume integral.

$$\int_\Gamma uw \cdot n\, d\Gamma = \int_\Omega u \cdot \nabla w\, d\Omega + \int_\Omega \nabla u \cdot w\, d\Omega$$

For Galerkin finite elements, the following equivalent form is used

$$\int_\Omega \phi \cdot \nabla F\, d\Omega = -\int_\Omega \nabla\phi \cdot F\, d\Omega + \int_\Gamma \phi F \cdot n\, d\Gamma$$

The assumptions can often be slightly relaxed. For the FE form, let the Flux F be continuous across an elemental boundary. Also let the basis functions be the traditional hat function that is not differentially continuous across an elemental boundary. We need to calculate

$$\int_{\Omega} \phi \cdot \nabla F \, d\Omega$$

which is split into

$$-\int_{\Omega} \nabla\phi \cdot F \, d\Omega$$

and

$$+\int_{\Gamma} \phi F \cdot n \, d\Gamma$$

for all elemental boundaries. Notice that for the first term, $\nabla\phi$ is discontinuous across elements boundaries. For the second term, the interior-element boundary integrals exactly cancel since the basis functions and F are continuous at boundaries. Thus, boundary integral global reduction eliminating internal boundary integrals remains consistent when both F and ϕ are at least C^0.

Numerical Integration

The traditional method of numerical integration is Gauss quadrature. The general form is

$$\int f dA \approx w_i f_i \left(x_i\right)$$

when given optimized locations x_i and weights w_i. This scheme is exact for well-behaved polynomial functions given suffient points. Unfortunately, the derivation of multidimensional higher order Gauss quadrature formulas requires significant computational optimization in a solution space full of local optimums. Gauss integration formulas for triangles and tetrahedra become increasingly scarce for moderate degrees ($P > 5$) and essentially non-existent for truly higher order ($P > 8$). This project attempted to generate higher order Gauss integration formulas via the Jinyun[33] and Keast[37] papers. Low order generation was successful; higher order generation failed.

3.2.4 Element Boundary Normals

Boundary conditions require boundary normals. The normals are applied in two parts of the solver: boundary fluxes and boundary constraints applied through the mass matrix. For an Euler slip boundary condition, normals from the connecting boundary elements must be identical. For a given element, boundary velocities are constrained to a normal velocity of zero

$$V \cdot n = 0$$

From the governing equations, infinite accelerations are not physically consistent. Thus, at a common face, each directional velocity component is C_0 continuous. Removing the normal velocity components simultaneously from both elements reduces the entire velocity to zero. Thus, two adjacent elements must have identical normals along their shared face. A shorter alternative description is: Euler solutions are scale independent. Finite element grids tend to ruin the scale independence.

Past inviscid solvers, such as Euler3d, calculate and use an area weighted normal for all nodes. In equation form for two 1D face elements at a common vertex, this is

$$n_W = A_1 \begin{pmatrix} n_x \\ n_y \end{pmatrix}_1 + A_2 \begin{pmatrix} n_x \\ n_y \end{pmatrix}_2$$

As an interesting aside, this method has a rarely-discussed assumption of boundary element curvature when using a linear grid. That is, an isoparameteric linear basis solver with area weighted normals numerically contains variable boundary normals even though the formulation is based on a constant normal. For flow tangency, the area weighting coupled with nodal integration points is numerically equivalent to assuming a curved boundary. Regardless, most methods use the actual linear element's normal for boundary fluxes and integration. The resulting loss of conservation is likely to be proportional to the difference in adjacent normals. That is, if the edge boundary normals and not parametric with the interior integration, then the integratiom by parts expansion is not exact. Luckily, typical grids require small elements and small changes in normals between adjacent elements. For typical converged grids, the loss should be small.

For many grids, a full nonlinear element and the associated variable Jacobian is both expensive and unnecessary. A notable exception would be for boundary elements. Even then, actual adjacent element normals for geometry converged grids only vary a few degrees.

3.3 Governing Equations

The governing equations in a non-inertial frame are the 2D compressible Navier-Stokes equations. Compressible indicates that density is variable. Navier-Stokes indicates the presence of stress terms resulting from velocity gradients.

With a 2D compressible Navier-Stokes flow, the corresponding conservative states are

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho e \end{pmatrix}$$

A general form of the governing equations contains temporal terms dU/dt, advection flux terms ∇F, body terms B, and non-inertial frame source terms S

$$\frac{dU}{dt} + \nabla F = B + S$$

The flux terms are composed of inviscid and viscous terms,

$$F = F^I - F^V$$

Each of these terms is discussed in detail below.

3.3.1 Inviscid Flux

The inviscid fluxes are

$$F_x^I = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uh \end{pmatrix} \qquad F_y^I = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vh \end{pmatrix}$$

With the assumption of an ideal gas, the pressure is

$$P = (\gamma - 1)\left(\rho e - \frac{1}{2}\rho V^2\right)$$

The enthalpy is

$$h = e + \frac{p}{\rho}$$

3.3.2 Viscous Flux

The viscous fluxes are

$$F_x^V = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ u\tau_{xx} + v\tau_{xy} - q_x \end{pmatrix} \qquad F_y^V = \begin{pmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ u\tau_{yx} + v\tau_{yy} - q_y \end{pmatrix}$$

The viscous stress tensor τ is

$$\tau = \begin{bmatrix} \tau_{xx} & \tau_{xy} \\ \tau_{yx} & \tau_{yy} \end{bmatrix}$$

With a Newtonian fluid, stresses are linear combinations of velocity gradients. The stress model is

$$\tau_{ij} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) + \delta_{ij}\lambda\nabla u$$

Often, the velocity divergence term is incorporated into the other term with the assumption of a constant bulk viscosity λ. A rationale and discussion is beyond this project's scope. See White[74] for more information.

The surface stress tensor is needed for lift and drag calculations. Stresses and integrated forces are in the global frame. Recalling the Cauchy equations from structural mechanics gives the surface traction on an arbitrary surface as

$$T_x = \tau_{xx} n_x + \tau_{xy} n_y$$

$$T_y = \tau_{xy} n_x + \tau_{yy} n_y$$

The surface area projected in the x direction along a surface of length L is

$$A_x = L n_x$$

Thus, the force in the x direction is

$$F_x = T_x n_x L$$

$$F_x = \tau_{xx} n_x^2 L + \tau_{xy} n_y n_x L$$

The y direction force is

$$F_y = T_x n_x L$$

$$F_y = \tau_{xy} n_x n_y L + \tau_{yy} n_y^2$$

Projecting the stress to the local elemental surface and then re-projecting the resulting force back into a global frame is not necessary.

The heat flux vector is

$$q = -k \nabla T$$

Viscous flows and boundary layers can be sensitive to heat flux. Adding the heat flux term also requires specifying a heat flux boundary condition. As this involves fluid-structure coupling, a more detailed model of the vehicle's thermal characteristics would be required.

3.3.3 Body Forces

Body forces in flight dynamics are typically only gravity induced

$$B = \begin{pmatrix} 0 \\ \rho g_x \\ \rho g_y \\ 0 \end{pmatrix}$$

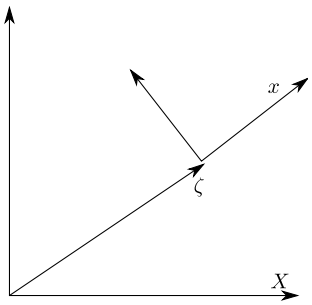


Figure 3.2: ALE frames

The source terms describe non-inertial frame motion (states in body reference frame)

$$S = -\rho \begin{pmatrix} 0 \\ a_t' + \Omega V_r \\ a_t' \cdot (V_t' + V_r) \end{pmatrix}$$

Derivation and use of the non-inertial frame is given in Cowan's dissertation[14].

3.3.4 Arbitrary Lagrangian Eulerian

A critical technology is coupling motion boundary conditions into the CFD solver. Previous work by Cowan[14] developed a non-inertial formulation of the Euler equations to allow arbitrary grid translation and rotation in the body frame. Unfortunately, the body frame motion capability only allows for rigid body motion; relative motion is not modeled. This ALE section introduces local grid motion into the CFD formulation.

Arbitrary Lagrangian Eulerian (ALE) allows local grid motion in the Navier-Stokes governing equations with a grid velocity field appended to the flow states. Conceptually, ALE is a reference frame modification[30, 16] as shown in Figure 3.2. The inertial frame is frame X; the reference frame is x.

Conversion between the frames is through ζ. This gives a frame Jacobian of

$$J = \left| \frac{dx_i}{dX_j} \right|$$

The time derivative of the Jacobian is[16]

$$\frac{dJ}{dt} = J \frac{dW_i}{dx_i}$$

This indicates that the rate of change of the transformation depends on the velocity divergence. As might be expected, we will later see that this term is mirrored in the Lagrangian frame kinematics.

The time derivative of a property f is[16]

$$\left.\frac{\partial f}{\partial t}\right|_X = \left.\frac{\partial f}{\partial t}\right|_x + \frac{\partial f}{\partial x_i}\frac{\partial x_i}{\partial t}$$

But with dx/dt being the x frame velocity w, this is also

$$\left.\frac{\partial f}{\partial t}\right|_X = \left.\frac{\partial f}{\partial t}\right|_x + \frac{\partial f}{\partial x_i}W_i$$

So for the fluid equations, we have[16, 42]

$$\frac{\partial U}{\partial t} + \frac{\partial F^I}{\partial x_j} - \frac{\partial F^V}{\partial x_j} = 0$$

The states are

$$U = \begin{pmatrix} \rho J \\ \rho u_i J \\ \rho e J \end{pmatrix}$$

The inviscid fluxes are

$$F^I = J \begin{pmatrix} \rho\left(u_j - W_j\right) \\ \rho u_i\left(u_j - W_j\right) + \delta_{ij}p \\ \rho e\left(u_j - W_j\right) + p u_j \end{pmatrix}$$

The grid velocity transformation appears in the Jacobian term and the advection term $u_j - W_j$. The viscous fluxes, being frame invariant, are identical to the normal Navier-Stokes fluxes

$$F^V = J \begin{pmatrix} 0 \\ \sigma_{ji} \\ u_k\sigma_{kj} - q_j \end{pmatrix}$$

At this point, the governing equations contain the frame transformation term J. Further simplification is possible by expanding the temporal terms. For example, the momentum term expanded with the chain rule is

$$\frac{d\rho u_i J}{dt} = J\frac{d\rho u_i}{dt} + \rho u_i \frac{dJ}{dt}$$

Remembering the definition of the frame Jacobian gives

$$\frac{d\rho u_i J}{dt} = J\frac{d\rho u_i}{dt} + \rho u_i J\frac{dW_k}{dx_k}$$

Presumably, the frame Jacobian is non-zero, so reducing gives

$$\frac{dU}{dt} + \frac{dF^I}{dx_j} + \frac{dW_j}{dx_j}U = 0$$

where

$$U = \begin{pmatrix} \rho \\ \rho u_i \\ \rho e \end{pmatrix}$$

$$F^I = \begin{pmatrix} \rho\left(u_j - W_j\right) \\ \rho u_i\left(u_j - W_j\right) + \delta_{ij} p \\ \rho e\left(u_j - W_j\right) + p u_j \end{pmatrix}$$

$$F^V = \begin{pmatrix} 0 \\ \sigma_{ji} \\ u_k \sigma_{kj} - q_j \end{pmatrix}$$

Thus, the ALE equations are exactly the inertial equations with the addition of grid divergence terms $U\,dW/dx$ and relative velocity terms $u - W$ in the Euler fluxes.

Adding relative grid motion to an existing Eulerian frame CFD solver is practical and requires few modifications. Furthermore, comparing these ALE terms with Cowan's source terms shows interesting similarities. Conceptually, these two forms should be identical when restricted to rigid body motion.

ALE Compression Verification

At this point, we cautiously check whether or not these equations reflect reality. The objective is to compress a 2D closed control volume in the y direction. From mass conservation, the final density reflects the ratio of initial to final volume. If a unit square is slowly compressed from the bottom edge with a uniform upward velocity w of 1, the density is

$$\rho(t) = \frac{Mass(t)}{Volume(t)} = \frac{\rho_0}{1-t}$$

This is the intuitive control volume approach.

With the ALE equations, the density equation is

$$\frac{d\rho}{dt} = -\rho \frac{dw_j}{dx_j} - \frac{d\left(\rho(u_j - W_j)\right)}{dx_j}$$

Now, the grid motion W is constrained to exactly the fluid velocity u. The uniform compression velocity creates a time-varying velocity gradient

$$\frac{dw_y}{dx} = \frac{1}{L - Wt} = \frac{1}{1-t}$$

The governing equation, being in a pure Lagrangian frame, reduces to

$$\frac{d\rho}{dt} = -\rho \frac{1}{1-t}$$

Integration of this equation gives

$$\rho_f = \frac{\rho_0}{1-t}$$

The governing equation result matches the control volume result.

3.3.5 Entropy

Entropy is a governing equation for solution feasibility. CFD codes require dissipation in some form. In these terms, this project uses entropy not as a governing equation marched forward in time, but as a predictive and corrective dissipation scheme. Most CFD codes use an ad-hoc dissipation scheme. Merriam investigates a common dissipation scheme and makes the following comments.[48]

> The [Tadmor] scheme is nevertheless instructive as written. It shows the one-to-one correspondence between local violations of the second law and unphysical features of the flow (oscillations). This in turn shows the importance of satisfying a local entropy condition in addition to a global one.

Bluntly put, solutions with non-physical oscillations violate the 2nd law.

Boyd[10] views the dissipation/blowup problem as "spectral blocking". The good news is that the spectral expansion allows for a more efficient way to apply dissipation. From turbulence theory, energy cascades governs the gross behavior for perturbations at different wavelengths[62]. The non-linear governing equations disperses the energy bandwidth as time advances. An aliasing problem occurs when energy moving to higher frequencies can not be represented by the finite bandwidth numerical method.

The current Euler3d dissipation routine requires about 50% of the total solution time for a linear dissipation routine. It is strongly recommended to use an entropy formulation, if possible.

Governing Equations

Entropy measures a system's energy spread. For an ideal gas between two states, the differential change in entropy is

$$ds = c_v \frac{dT}{T} + R\frac{dv}{v}$$

or

$$\Delta s = c_v \ln \frac{p_2 \rho_1^\gamma}{p_1 \rho_2^\gamma}$$

As a control volume based transport governing equation, Naterer[53] gives

$$\frac{d\rho s}{dt} + \frac{d}{dx_i}\left(\rho u_i s - \frac{k}{T}\frac{dT}{dx_i}\right) = \frac{k}{T^2}\left(\frac{dT}{dx_i}\right)^2 + \frac{\tau_{ij}}{T}\frac{du_i}{dx_j}$$

Expanding and reducing gives

$$\frac{d\rho s}{dt} + \frac{d}{dx_i}\left(\rho u_i s\right) = \frac{\tau_{ij}}{T}\frac{du_i}{dx_j} + \frac{k}{T}\frac{d^2T}{dx_i^2}$$

Notice that the conduction term is kept in this derivation.

Dissipation Function

The dissipation function models the heat generation by linear viscous dissipation.

$$\Phi = \tau_{ij}\dot{\gamma_{ij}}$$

This is the contraction of the stress tensor and the velocity gradients[54].

$$\Phi = \tau_{ij}\frac{\partial u_i}{\partial x_j}$$

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

First pull out μ so that Φ only contains velocity gradients and defining$\hat{\lambda} = \lambda/\mu$

$$\Phi = \mu\hat{\Phi}$$

$$\hat{\Phi} = \left(\frac{\partial u_i}{\partial x_j}\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\frac{\partial u_i}{\partial x_j}\right) + \delta_{ij}\hat{\lambda}\left(\frac{\partial u_k}{\partial x_k}\frac{\partial u_i}{\partial x_j}\right)$$

$$\hat{\Phi} = \left(2\left(\frac{\partial u}{\partial x}\right)^2 + 2\left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + 2\left(\frac{\partial v}{\partial x}\frac{\partial u}{\partial y}\right)\right) + \hat{\lambda}\left(\frac{\partial u}{\partial x}^2 + \frac{\partial v}{\partial y}^2\right)$$

$$\hat{\Phi} = \left(2\left(\frac{\partial u}{\partial x}\right)^2 + 2\left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)^2\right) + \hat{\lambda}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)^2$$

Dimensionless Derivation

The dimensionless form with reference definitions is

$$\begin{aligned}\frac{U_o}{L_o}\frac{\partial}{\partial t^\star}\left(\rho_o\rho^\star s_o s^\star\right) + \frac{1}{L_o}\frac{d}{dx_i^\star}\left(\rho_o\rho^\star U_o u_i^\star s_o s^\star\right) & \\ -k\frac{1}{L_o^2}\frac{c_p}{\gamma U_o^2}\frac{\gamma U_o^2}{c_p}\frac{1}{T^\star}\frac{d^2T^\star}{dx_i^{\star 2}} & \\ -\frac{c_p}{\gamma U_o^2}\frac{\hat{\mu}\mu_o}{T^\star}\Phi^\star\frac{U_o^2}{L_o^2} & \geq \quad 0\end{aligned}$$

Expanding and dividing by $\rho_o s_o U_o / L_o$

$$\begin{aligned} \frac{\partial}{\partial t^\star}\left(\rho^\star s^\star\right) + \frac{d}{dx_i^\star}\left(\rho^\star u_i^\star s^\star\right) \\ -k\frac{1}{\rho_o s_o}\frac{1}{L_o}\frac{c_p}{\gamma U_o^2}\frac{\gamma U_o}{c_p}\frac{1}{T^\star}\frac{d^2 T^\star}{dx_i^{\star 2}} \\ -\frac{1}{\rho_o s_o}\frac{c_p}{\gamma U_o^2}\frac{\hat{\mu}\mu_o}{T^\star}\Phi^\star\frac{U_o}{L_o} \quad \geq \quad 0 \end{aligned}$$

Divide the viscous dissipative term by Re and the temperature dissipation term by Re and Pr.

$$\begin{aligned} \frac{\partial}{\partial t^\star}\left(\rho^\star s^\star\right) + \frac{d}{dx_i^\star}\left(\rho^\star u_i^\star s^\star\right) \\ -k\frac{1}{\rho_o c_p}\frac{1}{L_o}\frac{c_p}{\gamma U_o^2}\frac{\gamma U_o}{c_p}\frac{\mu_o c_p}{kPr}\frac{\rho_o U_o L_o}{Re\mu_o}\frac{1}{T^\star}\frac{d^2 T^\star}{dx_i^{\star 2}} \\ -\frac{1}{\rho_o c_p}\frac{\rho_o U_o L_o}{Re\mu_o}\frac{c_p}{\gamma U_o^2}\frac{\hat{\mu}\mu_o}{T^\star}\Phi^\star\frac{U_o}{L_o} \quad \geq \quad 0 \end{aligned}$$

This reduces to

$$\begin{aligned} \frac{\partial}{\partial t^\star}\left(\rho^\star s^\star\right) + \frac{d}{dx_i^\star}\left(\rho^\star u_i^\star s^\star\right) \\ -\frac{1}{Pr}\frac{1}{Re}\frac{1}{T^\star}\frac{d^2 T^\star}{dx_i^{\star 2}} \\ -\frac{1}{Re}\frac{1}{\gamma}\frac{\hat{\mu}}{T^\star}\Phi^\star \quad \geq \quad 0 \end{aligned}$$

Finally, substitute $T^\star = e_{int}^\star$ to give

$$\begin{aligned} \frac{\partial}{\partial t^\star}\left(\rho^\star s^\star\right) + \frac{d}{dx_i^\star}\left(\rho^\star u_i^\star s^\star\right) \\ -\frac{1}{Pr}\frac{1}{Re}\frac{1}{e_{int}^\star}\frac{d^2 e_{int}^\star}{dx_i^{\star 2}} \\ -\frac{1}{Re}\frac{1}{\gamma}\frac{\hat{\mu}}{e_{int}^\star}\Phi^\star \quad \geq \quad 0 \end{aligned}$$

The final non-dimensional form is

$$\frac{\partial}{\partial t^\star}\left(\rho^\star s^\star\right) + \frac{d}{dx_i^\star}\left(\rho^\star u_i^\star s^\star\right) = \frac{1}{Pr}\frac{1}{Re}\frac{1}{e_{int}^\star}\frac{d^2 e_{int}^\star}{dx_i^{\star 2}} + \frac{1}{Re}\frac{1}{\gamma}\frac{\hat{\mu}}{e_{int}^\star}\Phi^\star$$

Entropy Final Form

Dimensional form is

$$\frac{d\rho s}{dt} + \frac{d}{dx_i}\left(\rho u_i s\right) = \frac{k}{T}\frac{d^2 T}{dx_i^2} + \frac{\Phi}{T}$$

The non-dimensional form is

$$\frac{\partial}{\partial t^\star}\left(\rho^\star s^\star\right) + \frac{d}{dx_i^\star}\left(\rho^\star u_i^\star s^\star\right) = \frac{1}{Pr}\frac{1}{Re}\frac{1}{e_{int}^\star}\frac{d^2 e_{int}^\star}{dx_i^{\star 2}} + \frac{1}{Re}\frac{1}{\gamma}\frac{\hat{\mu}}{e_{int}^\star}\Phi^\star$$

3.4 Non-Dimensionalization

Non-dimensionalization scales the simulation values to unit computational values. Additionally, non-dimensionalization allows for the congregation of problem-scale terms.

- Density, $\rho^\star = \rho/\rho_o$
- Velocity, $u^\star = u/U_o$
- Pressure, $p^\star = p/\rho_o U_o^2$
- Energy, $e^\star = e/U_o^2$
- Entropy, $s^\star = s/c_p$
- Time, $t^\star = tU_o/L_o$
- Location, $x^\star = x/L_o$
- Temperature, $T^\star = T/T_o = T/\frac{U_o^2}{c_v} = T/\frac{\gamma U_o^2}{c_p}$
- Viscosity, $\mu^\star = \mu/\mu_o$

Subsequent use of a particular variable will imply the non-dimensional version.

3.4.1 Governing Equations

The Euler and Navier-Stokes governing equations are generalized to

$$\frac{dU}{dt} + \nabla F + U\nabla W = B + S$$

The dimensional values are[14]

$$U = \begin{bmatrix} \rho_o & & & \\ & \rho_o U_o & & \\ & & \rho_o U_o & \\ & & & \rho_o U_o^2 \end{bmatrix} U^\star$$

$$F = \begin{bmatrix} \rho_o U_o & & & \\ & \rho_o U_o^2 & & \\ & & \rho_o U_o^2 & \\ & & & \rho_o U_o^3 \end{bmatrix} F^\star$$

$$S = S_o S^\star = \begin{bmatrix} \rho_o \frac{U_o}{L_o} & & & \\ & \rho_o \frac{U_o^2}{L_o} & & \\ & & \rho_o \frac{U_o^2}{L_o} & \\ & & & \rho_o \frac{U_o^2}{L_o} U_o^2 \end{bmatrix} S^\star$$

$$B_o = S_o = [U_o] \frac{U_o}{L_o}$$

Substitution gives

$$\frac{U_o}{L_o} \frac{d}{dt^\star} \left([U_o] U^\star \right) + \frac{1}{L_o} \nabla \left([F_o] F^\star \right) + [U_o] U^\star \frac{1}{L_o} \nabla \left([U_o] W^\star \right) = [B_o] B^\star + [S_o] S^\star$$

Reducing gives

$$\frac{d}{dt^\star} \left(U^\star \right) + \nabla \left(F^\star \right) + U^\star \nabla W^\star = B^\star + S^\star \tag{3.1}$$

3.4.2 Euler Terms

The Euler flux terms non-dimensionalized are

$$F^\star = \frac{F}{F_o}$$

The mass term is

$$\begin{aligned} F_\rho^\star &= \frac{1}{\rho_o U_o} \rho u \\ &= \frac{\rho_o U_o}{\rho_o U_o} \rho^\star u^\star \\ &= \rho^\star u^\star \end{aligned}$$

The momentum terms are

$$\begin{aligned} F_{\rho u}^\star &= \frac{1}{\rho_o U_o^2} \left(\rho u_i u_j + \delta_{ij} p \right) \\ &= \frac{\rho_o U_o^2}{\rho_o U_o^2} \left(\rho^\star u_i^\star u_j^\star - \frac{1}{2} V^\star \cdot V^\star \right) + \frac{\rho_o U_o^2}{\rho_o U_o^2} \left(\delta_{ij} \left(\gamma - 1 \right) \rho^\star e^\star \right) \\ &= \rho^\star u_i^\star u_j^\star + \delta_{ij} p^\star \end{aligned}$$

The energy terms are

$$\begin{aligned} F_{\rho e}^\star &= \frac{1}{\rho_o U_o^3} \rho u_i h \\ &= \frac{\rho_o U_o U_o^2}{\rho_o U_o^3} \rho^\star u_i^\star h^\star \\ &= \rho^\star u_i^\star h^\star \end{aligned}$$

The non-dimensional Euler flux terms do not contain scaling constants. Inviscid Euler fluid dynamics is scale independent.

3.4.3 Viscous Terms

The non-dimensionalization of the viscous stress flux term is

$$F^\star = \frac{F}{F_o}$$

The momentum terms are

$$\begin{aligned} F^\star_{\rho u} &= \frac{\hat{\mu}\mu_o}{\rho_o U_o^2}\left(\frac{U_o}{L_o}\left(\frac{\partial u_i^\star}{\partial x_j^\star}+\frac{\partial u_j^\star}{\partial x_i^\star}\right)+\delta_{ij}\hat{\lambda}\frac{U_o}{L_o}\nabla^\star u^\star\right) \\ &= \frac{\hat{\mu}}{Re}\left(\left(\frac{\partial u_i^\star}{\partial x_j^\star}+\frac{\partial u_j^\star}{\partial x_i^\star}\right)+\delta_{ij}\hat{\lambda}\nabla^\star u^\star\right) \end{aligned}$$

The energy term is

$$\begin{aligned} F^\star_{\rho e} &= \frac{1}{\rho_o U_o^3}\left(U_o u_j^\star \frac{U_o}{L_o}\hat{\mu}\mu_o\tau_{ij}^\star - \frac{k}{c_v}\frac{U_o^2}{L_o}\nabla_i^\star e_{int}^\star\right) \\ &= \frac{\mu_o}{\rho_o U_o L_o}\left(u_j^\star\hat{\mu}\tau_{ij}^\star - \frac{k}{c_v}\frac{1}{\mu_o}\nabla_i^\star e_{int}^\star\right) \\ &= \frac{\mu_o}{\rho_o U_o L_o}\left(u_j^\star\hat{\mu}\tau_{ij}^\star - \frac{k}{c_v}\frac{1}{\mu_o}\frac{\mu_o c_p}{Prk}\frac{c_v\gamma}{c_p}\nabla_i^\star e_{int}^\star\right) \\ &= \frac{1}{Re}\left(u_j^\star\hat{\mu}\tau_{ij}^\star - \frac{\gamma}{Pr}\nabla_i^\star e_{int}^\star\right) \\ &= \frac{1}{Re}u_j^\star\hat{\mu}\tau_{ij}^\star - \frac{\gamma}{Re\,Pr}\nabla_i^\star e_{int}^\star \end{aligned}$$

Unlike the Euler equations, the viscous flux terms contain scaling reference constants: the Reynolds Number for stress terms and the Prandtl Number for heat conduction. Viscous fluid dynamics is scale dependent.

3.5 Numerical Form of Governing Equations

This section discusses how the governing equations are formed into a numerical routine. The raw NS governing equations are not in a form convenient for numerical solutions. Additionally, the numerical iteration requires the equations in a canonical form. Applying the Galerkin method to the Navier-Stokes equations creates a weak formulation where through summation of parts, each element is represented

$$\sum_e \int_V \phi_i\left(-\frac{dU}{dt}-\frac{dF_j^I}{dx_j}+\frac{dF_j^V}{dx_j}-\frac{dW_j}{dx_j}U\right)dV+\int_V \phi_i\left(B+S\right)dV=0$$

For reference, the total Galerkin energy equation is[13]

$$\begin{aligned} L &= T-\Pi \\ &= \frac{1}{2}\dot{Q}^T M\dot{Q}-\frac{1}{2}Q^T KQ-Q^T F \end{aligned}$$

This becomes relevant when considering boundary condition contraint methods.

3.5.1 Mass

The mass term is formed from the terms containing time derivatives. In the traditional Galerkin method, the mass terms are in the form of a mass matrix.

$$M_{ij} = \int \phi_i \phi_j dV$$

Expanded, this is

$$M_{ij} = \int \phi_i \phi_j \left| J\left(\eta_1, \eta_2\right) \right| d\Omega$$

3.5.2 Inviscid Fluxes

The inviscid fluxes F^I are the Euler fluxes. Consistent with tradition, the inviscid fluxes are transformed with Green's theorem

$$-\int_V \phi_i \frac{dF_j^I}{dx_j} dV = \int_V \frac{d\phi_i}{dx_j} F_j^I dV - \oint \phi_i \left(F_j^I \cdot n_j\right) dS$$

Interior

The interior term is

$$\int_V \frac{d\phi_i}{dx_j} F_j^I dV$$

Expanding for the reference element gives

$$\int \frac{d\phi_i}{dx_j} F_j^I \left|J\right| d\Omega$$

The basis and flux terms must be reoriented into the global coordinate directions

$$\frac{d\phi_i}{dx_j} F_j^I = \frac{1}{|J|} \begin{pmatrix} \frac{d\phi_i}{d\hat{x}} & \frac{d\phi_i}{d\hat{y}} \end{pmatrix} \begin{pmatrix} F_x^I \\ F_y^I \end{pmatrix}$$

Substitution gives the Galerkin term

$$\int \begin{pmatrix} \frac{d\phi_i}{d\hat{x}} & \frac{d\phi_i}{d\hat{y}} \end{pmatrix} \begin{pmatrix} F_x^I \\ F_y^I \end{pmatrix} d\Omega$$

Boundary

The boundary term is

$$-\oint \phi_i \left(F_j^I \cdot n_j\right) dS$$

When considering dS_e along an edge e is integrated in zeta space dl from 0 to 1, the edge Jacobian $|J_e|$ is required

$$-\int \phi_i \left(F^I \cdot n_e\right) |J_e| \, dl$$

Calculating the flux in the normal direction expands into

$$F^I \cdot n_e = F_x n_x + F_y n_y$$

Substituting and reducing for the inviscid flux gives

$$\begin{aligned} F^I \cdot n_e &= \begin{pmatrix} \rho\left(u-u_g\right) \\ \rho u\left(u-u_g\right)+p \\ \rho v\left(u-u_g\right) \\ \rho e\left(u-u_g\right)+pu \end{pmatrix} n_x + \begin{pmatrix} \rho\left(v-v_g\right) \\ \rho u\left(v-v_g\right) \\ \rho v\left(v-v_g\right)+p \\ \rho e\left(v-v_g\right)+pv \end{pmatrix} n_y \\ &= \begin{pmatrix} \rho\left(C\cdot\hat{n}\right) \\ \rho u\left(C\cdot\hat{n}\right) \\ \rho v\left(C\cdot\hat{n}\right) \\ \rho e\left(C\cdot\hat{n}\right)+p\left(V\cdot\hat{n}\right) \end{pmatrix} + \begin{pmatrix} 0 \\ pn_x \\ pn_y \\ 0 \end{pmatrix} \end{aligned}$$

where

$$C = \begin{pmatrix} \left(u-u_g\right) \\ \left(v-v_g\right) \end{pmatrix} \quad V = \begin{pmatrix} u \\ v \end{pmatrix} \quad \hat{n} = \begin{pmatrix} n_x \\ n_y \end{pmatrix}$$

and $u_g$and $v_g$are the grid velocities for an ALE formulation. This form simplifies the implementation by only tracking the normal flux components.

3.5.3 Viscous Fluxes

The viscous fluxes before and after applying Green's theorem are

$$\int_V \phi_i \frac{dF_j^V}{dx_j} dV = -\int_V \frac{d\phi_i}{dx_j} F_j^V dV + \oint \phi_i \left(F_j^V \cdot n_j\right) dS$$

Be aware that since Green's theorem requires a continuous flux F, the boundary fluxes at adjoining elements edges do not cancel. Unlike the inviscid case, inviscid boundary terms must be calculated and accumulated for each element.

Interior

The interior term is

$$-\int_V \frac{d\phi_i}{dx_j} F_j^V \, dV$$

Substitution for derivatives and integration area gives

$$-\int \begin{pmatrix} \frac{d\phi_i}{d\hat{x}} & \frac{d\phi_i}{d\hat{y}} \end{pmatrix} \begin{pmatrix} F_x^V \\ F_y^V \end{pmatrix} d\Omega$$

Boundary

The boundary term is

$$\oint \phi_i \left(F_j^V \cdot n_j \right) dS$$

Calculating the flux in the normal direction expands into

$$F^V \cdot n_e = F_x n_x + F_y n_y$$

Substituting and reducing for the inviscid flux gives

$$F^V \cdot n_e = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ u\tau_{xx} + v\tau_{xy} - q_x \end{pmatrix} n_x + \begin{pmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ u\tau_{yx} + v\tau_{yy} - q_y \end{pmatrix} n_y$$

where

$$\hat{n} = \begin{pmatrix} n_x \\ n_y \end{pmatrix}$$

3.5.4 Source Terms

The body force and source terms involve external forces and usually do not contain temporal or spacial derivatives of states.

$$\int_V \phi_i S dV$$

Here, S is used as a generic summation of all source and body terms.

3.5.5 ALE Terms

The ALE terms represent the local grid motion. These terms contain states and grid divergence

$$-\int \phi_i U \frac{dW_j}{dx_j} dV$$

Additionally, the grid velocity vector W is known to be continuous. Gradient continuity of W is not assured. For most internal elements however, W will be zero.

Expanding for the derivative and referential area gives

$$-\int \phi_i U \frac{1}{|J|}\left(\frac{dW_1}{d\hat{x}}+\frac{dW_2}{d\hat{y}}\right)|J|\,d\Omega$$

Reducing gives

$$-\int \phi_i U \left(\frac{dW_1}{d\hat{x}}+\frac{dW_2}{d\hat{y}}\right)d\Omega$$

3.5.6 Entropy Constraint

Here we constrain entropy through increasing the local dissipation. Reordering the non-dimensional entropy transport equation gives

$$e^\star_{int}\frac{\partial}{\partial t^\star}\left(\rho^\star s^\star\right)+e^\star_{int}\frac{d}{dx^\star_i}\left(\rho^\star u^\star_i s^\star\right)=\frac{1}{Re}\left(\frac{1}{Pr}\frac{d^2 e^\star_{int}}{dx^{\star 2}_i}+\frac{1}{\gamma}\hat{\mu}\Phi^\star\right)$$

The Re required to satisfy the entropy equation is

$$\frac{1}{Re}=e^\star_{int}\frac{\frac{\partial}{\partial t^\star}\left(\rho^\star s^\star\right)+\frac{d}{dx^\star_i}\left(\rho^\star u^\star_i s^\star\right)}{\left(\frac{1}{Pr}\frac{d^2 e^\star_{int}}{dx^{\star 2}_i}+\frac{1}{\gamma}\hat{\mu}\Phi^\star\right)}$$

Alternatively, adding a fictitious dissipation Φ_F to the governing equation gives

$$\frac{\partial}{\partial t^\star}\left(\rho^\star s^\star\right)+\frac{d}{dx^\star_i}\left(\rho^\star u^\star_i s^\star\right)=\frac{1}{Pr}\frac{1}{Re}\frac{1}{e^\star_{int}}\frac{d^2 e^\star_{int}}{dx^{\star 2}_i}+\frac{1}{Re}\frac{1}{\gamma}\frac{\hat{\mu}}{e^\star_{int}}\left(\Phi^\star\right)+\frac{1}{e^\star_{int}}\Phi^\star_F$$

or

$$e^\star_{int}\frac{\partial}{\partial t^\star}\left(\rho^\star s^\star\right)+e^\star_{int}\frac{d}{dx^\star_i}\left(\rho^\star u^\star_i s^\star\right)-\frac{1}{Re}\frac{1}{Pr}\frac{d^2 e^\star_{int}}{dx^{\star 2}_i}=\frac{1}{Re}\frac{1}{\gamma}\hat{\mu}\left(\Phi^\star\right)+\Phi^\star_F$$

or

$$\Phi^\star_F=e^\star_{int}\frac{\partial}{\partial t^\star}\left(\rho^\star s^\star\right)+e^\star_{int}\frac{d}{dx^\star_i}\left(\rho^\star u^\star_i s^\star\right)-\frac{1}{Re}\frac{1}{Pr}\frac{d^2 e^\star_{int}}{dx^{\star 2}_i}-\frac{1}{Re}\frac{1}{\gamma}\hat{\mu}\Phi^\star$$

If dissipation is applied only to the divergence of velocity, the additional flux terms are

$$\tau_{ij}=\delta_{ij}\lambda_D\nabla u$$

$$F^V_x=\begin{pmatrix}0\\ \tau_{xx}\\ 0\\ u\tau_{xx}\end{pmatrix}\qquad F^V_y=\begin{pmatrix}0\\ 0\\ \tau_{yy}\\ v\tau_{yy}\end{pmatrix}$$

with a dissipation function of

$$\Phi^\star_D=\lambda_D\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)^2$$

Solving for $\lambda_D$gives

$$\lambda_D = \frac{e^\star_{int}\frac{\partial}{\partial t^\star}\left(\rho^\star s^\star\right) + e^\star_{int}\frac{d}{dx^\star_i}\left(\rho^\star u^\star_i s^\star\right) - \frac{1}{Re}\frac{1}{Pr}\frac{d^2 e^\star_{int}}{dx^{\star 2}_i} - \frac{1}{Re}\frac{1}{\gamma}\hat{\mu}\Phi^\star}{\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)^2}$$

3.5.7 Boundary Conditions

The boundary conditions implemented are No Slip and Freestream. It is important not to let boundary conditions modify the governing equations. Both the boundary conditions and the governing equations must be satisfied simulateously. The approach here is to set the boundary conditions and keep them satisfied by constraining temporal derivatives. Implementation is through a linear addition to the mass matrix. The approach is expandable to linear function constraints as might be given in Euler no-flow boundary conditions.

Multipoint constraints modify the potential energy equation with a Lyaponov-stable energy modification[49, 38]. For a constraint function $f(q) = 0$, the additional potential energy is[13]

$$\Pi_c = \frac{1}{2}C\left(f(q)\right)^2$$

Following through with the Galerkin approach gives a constraint residual addition of

$$R_c = -C\frac{df(q)}{dq}f(q)$$

The constraint is especially efficient to implement when $f(q)$ is a linear function of state q.

Constraining values rather than derivatives significantly increases the solution stiffness by the constraint constant C.

Known State Boundary Condition (No Slip)

A static state boundary condition is one where a particular fluid variable U has a known time variation. The constraint equation is

$$\Pi = \frac{1}{2}C\left(\frac{da_U}{dt} - k\right)^2$$

The residual term is

$$R = -\left(\frac{da_U}{dt} - k\right)$$

Addition of a controller loop allows for boundary condition enforcement of initially non-compliant fields. A single order controller likely is best for stability robustness.

$$k = \lambda\left(a_U - a_{U_o}\right)$$

The gain λ would be a solution dependent parameter. Remember that this strategy is not valid for time accuracy and would only be considered for steady solutions.

Function Boundary Condition (Slip)

A function state boundary condition is one where more than one fluid variable combine in a constraint equation. For the purposes of this paper, only linear functions are useful and will be considered. The constraint equation for the slip condition

$$V \cdot n = 0$$

in a multipoint constraint form is

$$\Pi = \frac{1}{2} C \left(n_x \frac{du}{dt} + n_y \frac{du}{dt} \right)^2$$

$$\Pi = \frac{1}{2} C \left(n_x \sum \phi_i \frac{da_u}{dt} + n_y \sum \phi_i \frac{da_v}{dt} \right)^2$$

The residual is

$$R_{a_{u_i}} = -C \left(n_x \sum \phi_i \frac{da_{u_i}}{dt} + n_y \sum \phi_i \frac{da_{v_i}}{dt} \right) n_x \phi_i$$

Riemann/Roe Solver

The boundary terms are where the far field (freestream) boundary condition is introduced. This method is idential to the far field boundary condition in Euler3d. A formal discussion of the Riemann-Roe solver for use in CFD solvers in given in Toro[73]. The fundamental concept is to correct the boundary flux with flow characteristics. The concepts are quickly introduced here.

For the 1D compressible Euler equation, closed form solutions are available from the method of characteristic lines[41]. For a linear advection equation $du/dt = cdu/dx$, the c term is recognized as the advection velocity. All flow states, including discontinuous states, advect at velocity c. Likewise, for arbitrary governing equations

$$\frac{du}{dt} + \frac{dF}{dx} = 0$$

applying the chain rule gives

$$\frac{du}{dt} + \frac{dF}{du}\frac{du}{dx} = 0$$

Next, define A as the derivative of flux with respect to states

$$A = \frac{dF}{du}$$

The eigenvalues of A give the state velocities. The eigenvectors of A give the proportion of each state corresponding to each state velocity.

In 1D, the method of characteristic lines exactly solves the Euler equations of fluid flow for continuous and discontinuous initial conditions. For the Euler governig equation[41],

$$A = \frac{dF}{du} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{1}{2}(\gamma-3)u^2 & (3-\gamma)u & (\gamma-1) \\ \frac{1}{2}(\gamma-1)u^3 - u\frac{(E+p)}{\rho} & \frac{(E+p)}{\rho} - (\gamma-1)u^2 & \gamma u \end{bmatrix}$$

The three eigenvalues of A for the 1D Euler equation are: $\lambda = u - c,\ u,\ u + c$. That is, velocity u of the contact discontinuity and two waves at u plus and minus sonic velocity c.

For numerical simulations, Roe[66] developed an approximate characteristic lines/Riemann solver. Roe's contribution involved finding a representative average state at a discontinuity. This average for a state s with right s_R and left s_L values is

$$s_{roe} = \frac{s_L\sqrt{\rho_L} + s_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}}$$

This state is substituted into the A matrix to obtain a representative flux at the boundary. The current project uses a Roe averaged Riemann solver[73] for resolving fluxes at the element far-field boundaries.

The boundary flux is augmented as an average of the internal and the desired far field flux. Additionally, changes in state across the boundary models the compressible flow characteristics with a linearized Euler flow model.

$$F = \frac{1}{2}(F_L + F_R) - \frac{1}{2}|A|(U_R - U_L)$$

This allows the model to only pick flow characteristics physically contributing to the actual solution. Outgoing characteristics are ignored.

Constraints

We also recognize that using Green's theorem in the Galerkin equation requires special continuity constraints. The inviscid fluxes are functions of state values, so states must be C^0 continuous. The viscous fluxes are functions of state derivatives, so the states must also be C^1 continuous.

Maintaining state continuity is simple with the B-Spline basis function; state coefficients along inter-element boundaries must be identical. Traditionally, identical coefficients are specified by condensing edge coefficients together with the element and node connectivity. Alternatively, a routine could be generated to ensure values match on either side of an inter-elemental boundary. Thus, a more generic code is available at the expense of extra dependent values requiring numerical constraints.

Maintaining equal state gradients is more complex. Although a geometrical constraint exists for gradient continuity of the B-Spline basis, the constrain must eventually be expressed numerically and without the benefit of simple condensation of dependent coefficients as used for state continuity.

Multipoint Constraint Theory From before, the Galerkin energy equation is

$$\begin{aligned} L &= T - \Pi \\ &= \frac{1}{2}\dot{Q}^T M \dot{Q} - \frac{1}{2} Q^T K Q - Q^T F \end{aligned}$$

Multipoint constraints modify the potential energy equation with a Lyaponov-stable energy modification[38]. For a constraint function $f(s) = 0$, the additional potential energy is[13]

$$\Pi_c = \frac{1}{2} C \left(f(s)\right)^2$$

Following through with the Galerkin approach gives a constraint residual addition of

$$R_c = -C \frac{df(s)}{ds} f(s)$$

The constraint is especially efficient to implement when $f(s)$ is a linear function of state q.

Residual Boundary Constraints In practice, the most effecitve boundary constraint method is to simply specify the value and then disallow modification. When computing updates, a residual is formed from the governing equations

$$RSD = B - As$$

This residual is then used to compute a state vector update

$$s^{new} = s^{old} + f(RSD)$$

The residual boundary constraint simply applies zero change to the particular boundary coefficients. This method is used extensively in Euler2d with a more difficult slip condition. For no-slip boundaries, the residual boundary constrain is even more simplified.

Curved Elements and Boundaries This section attempts to evaluate the qualitative and quantitative effects of modeling the boundary element and boundary representation with a curved representation.

In one dimension, or along a 2D edge, the edge is parametrically defined as

$$\begin{aligned} x(\zeta) &= \phi_i x_i \\ y(\zeta) &= \phi_i y_i \end{aligned}$$

The Jacobian would be

$$|J| = \sqrt{\left(\frac{dx}{d\zeta}\right)^2 + \left(\frac{dy}{d\zeta}\right)^2}$$

or

$$|J| = \sqrt{\left(x_i \frac{d\phi_i}{d\zeta}\right)^2 + \left(y_i \frac{d\phi_i}{d\zeta}\right)^2}$$

For a simple linear element where $0 \leq \zeta \leq 1$, shape function derivatives are constants.

$$\frac{d\phi_i}{d\zeta} = \pm 1$$

This would be

$$\begin{aligned} |J| &= \sqrt{\left(x_i \frac{\Delta\phi_i}{\Delta\zeta}\right)^2 + \left(y_i \frac{\Delta\phi_i}{\Delta\zeta}\right)^2} \\ &= \sqrt{(-x_0 + x_1)^2 + (-y_0 + y_1)^2} \\ &= \sqrt{(\Delta x)^2 + (\Delta y)^2} \end{aligned}$$

The Jacobian for a linear element segment is constant.

For a quadratic element where $0 \leq \zeta \leq 1$, shape function are

$$\phi = \left\{ \begin{array}{ccc} \left(2\zeta^2 - 3\zeta + 1\right) & \left(2\zeta^2 - \zeta\right) & \left(4\zeta - 4\zeta^2\right) \end{array} \right\}$$

Thus, the derivatives are

$$\frac{d\phi_i}{d\zeta} = \left\{ \begin{array}{ccc} (4\zeta - 3) & (4\zeta - 1) & (4 - 8\zeta) \end{array} \right\}$$

This would be

$$\begin{aligned} |J| &= \sqrt{\left(x_i \frac{\Delta\phi_i}{\Delta\zeta}\right)^2 + \left(y_i \frac{\Delta\phi_i}{\Delta\zeta}\right)^2} \\ &= \sqrt{\left(x_0 (4\zeta - 3) + x_1 (4\zeta - 1) + x_2 (4 - 8\zeta)\right)^2 + \left(y_0 (4\zeta - 3) + y_1 (4\zeta - 1) + y_2 (4 - 8\zeta)\right)^2} \end{aligned}$$

The Jacobian is not constant. This increases the computational work required per element.

3.6 Decoupled States

The mass term is formed from the terms containing time derivatives. In the traditional Galerkin method, the mass terms are in the form of a mass matrix.

$$M_{ij}^{trad} = \int \phi_i \phi_j dV$$

Beyond this point, mass matrix refers to the following derivation and not the traditional form given above.

For the derivation in this dissertation, the traditional mass matrix is extended to obtain useful computational advantages. This section discusses decoupling states in the temporal terms and forming an efficient numerical routine with the extended mass matrix.

3.6.1 Decoupled Temporal Derivative

The temporal derivative given in the compressible Navier-Stokes equations contains coupled conservative-form thermodynamic properties (e.g., $d(\rho u)/dt$). Because decoupled properties are advantageous numerically, this section seeks to decouple the temporal derivatives. There are two major advantages to decoupled raw properties, both of which concern calculating fluxes. One, raw properties avoid rational polynomial calculations in calculating inviscid fluxes; non-rational polynomials have significant numerical advantages in integration and avoiding floating point division-by-zero. Numerical routines are bounded in the zero density limit. Two, calculating derivatives for the viscous flux is considerably simplified with raw properties (e.g., u) rather than the compressible-conservation properties (e.g., ρu). Applying the chain rule to derivatives of rational polynomials quickly becomes error prone, denominator sensitive, and computationally expensive.

Rather, the chain rule is used to decompose the conservative state derivatives into primative state derivatives.

$$\frac{dU}{dt} = \begin{pmatrix} \frac{d\rho}{dt} \\ \frac{d\rho u}{dt} \\ \frac{d\rho v}{dt} \\ \frac{d\rho e}{dt} \end{pmatrix} = \begin{pmatrix} 0 \\ \rho\frac{du}{dt} \\ \rho\frac{dv}{dt} \\ \rho\frac{de}{dt} \end{pmatrix} + \begin{pmatrix} \frac{d\rho}{dt} \\ u\frac{d\rho}{dt} \\ v\frac{d\rho}{dt} \\ e\frac{d\rho}{dt} \end{pmatrix}$$

This form is further reduced with the observation that states are described by basis functions and coefficients. This gives the following form

$$\begin{aligned} \frac{dU}{dt} &= \begin{bmatrix} 1 & & & \\ u & \rho & & \\ v & & \rho & \\ e & & & \rho \end{bmatrix} \begin{pmatrix} \frac{d\rho}{dt} \\ \frac{du}{dt} \\ \frac{dv}{dt} \\ \frac{de}{dt} \end{pmatrix} = \begin{bmatrix} \phi & & & \\ u\phi & \rho\phi & & \\ v\phi & & \rho\phi & \\ e\phi & & & \rho\phi \end{bmatrix} \begin{pmatrix} \frac{da_\rho}{dt} \\ \frac{da_u}{dt} \\ \frac{da_v}{dt} \\ \frac{da_e}{dt} \end{pmatrix} \\ &= \begin{bmatrix} \phi & & & \\ \phi a_u \phi & \phi a_\rho \phi & & \\ \phi a_v \phi & & \phi a_\rho \phi & \\ \phi a_e \phi & & & \phi a_\rho \phi \end{bmatrix} \begin{pmatrix} \frac{da_\rho}{dt} \\ \frac{da_u}{dt} \\ \frac{da_v}{dt} \\ \frac{da_e}{dt} \end{pmatrix} \end{aligned}$$

While this form is implementable, decoupling the basis function from the basis coefficients simplifies the expression to

$$\frac{DU}{dt} = \phi \begin{bmatrix} \{1\} & & & \\ a_u & a_\rho & & \\ a_v & & a_\rho & \\ a_e & & & a_\rho \end{bmatrix} \phi \begin{pmatrix} \frac{da_\rho}{dt} \\ \frac{da_u}{dt} \\ \frac{da_v}{dt} \\ \frac{da_e}{dt} \end{pmatrix}$$

Notice that the matrix's top-left $(1,1)$ location contains a vector of ones that effectively eliminates the "extra" ϕ for the first row since the basis is by design normalized (i.e $\sum \phi = 1$).

It is important to notice that although the expression is decoupled, the expression is still the compressible-conservative form. Keeping the compressible conservative form is necessary to keep the flux terms in a conservative form to keep the weak solution consistent with the governing equation. Failure to do so would result in incorrect wave characteristic speeds[41].

3.6.2 Galerkin Mass Term

The Mass matrix is defined as

$$M_{ij}(a) = \int_A \phi_i \phi_j \begin{bmatrix} \{1\} & & & \\ a_u & a_\rho & & \\ a_v & & a_\rho & \\ a_e & & & a_\rho \end{bmatrix} \phi_j \left| J\left(\eta_1, \eta_2\right) \right| d\Omega$$

Notice that the mass matrix is now a function of property coefficients a but is otherwise decoupled from the time derivatives.

As derived above, the mass term is conceptually expensive to calculate. So a modification is made where a generalized mass tensor is defined as

$$\hat{M}_{ijk} = \int \phi_i \phi_j \phi_k dV$$

Now, the intermediate mass terms are computed as

$$\begin{aligned} M^1_{ij} &= \hat{M}_{ijk} 1 \\ M^\rho_{ij} &= \hat{M}_{ijk} a_\rho(k) \\ M^u_{ij} &= \hat{M}_{ijk} a_u(k) \\ M^v_{ij} &= \hat{M}_{ijk} a_v(k) \\ M^e_{ij} &= \hat{M}_{ijk} a_e(k) \end{aligned}$$

where the free indicies imply summation: $v_i = m_{ij}a_j = \sum_j m_{ij}a_j$. The full mass matrix is now

$$M_{ij} = \begin{bmatrix} M^1 & & & \\ M^u & M^\rho & & \\ M^v & & M^\rho & \\ M^e & & & M^\rho \end{bmatrix}$$

When implemented in software, the actual mass matrix is rarely used. Rather, the typical requirement is for the matrix multiplication by a vector.

$$Q = [M]\, V$$

With the expensive integrations cached in $\hat{M}_{ijk}$, the decoupled mass terms is now numerically feasible to implement.

3.6.3 Time Updates

A critical routine is the time advancement and update. Most iteration methods use a form conceptually like

$$x_{t+1} = x_t + \lambda q$$

Now there are many methods available, but interestingly enough, the governing equations as given above have a nasty pitfall. A typical temporal derivative expansion is the finite difference.

$$\frac{da}{dt} \approx \frac{a(t+\Delta t) - a(t)}{\Delta t}$$

Multiplication by the mass term gives q as a linear function of a

$$q = M\frac{da}{dt} = M\frac{a(t+\Delta t)}{\Delta t} - M\frac{a(t)}{\Delta t}$$

The Jacobi iteration is stable for direct computations of a.

However, when the mass terms are a function of states, the mass terms of the Galerkin equation become nonlinear when using a finite difference expansion for time. Thus q is

$$q = M(a)\frac{a(t+\Delta t)}{\Delta t} - M(a)\frac{a(t)}{\Delta t}$$

This non-linearity effectively *destroys* any Jacobi iteration techniques and initially indicated an expensive and non-robust nonlinear iterative solver.

We *can* use Jacobi iteration provided the Galerkin mass matrix does not change between iterations. One possibility is to solve for the time derivatives inside the Jacobi iteration for a fixed mass state.

$$M(a)\frac{da}{dt} \Rightarrow a$$

This requires a continuous time integration scheme such as Runga-Kutta.

3.6.4 Operations Count

The decoupled state method scales poorly. Timing an actual computer code shows this method to be about an order of magnitude slower than the traditional conservative states method. Assembling and iterating the effective mass matrix does not tradeoff favorably with the simplified state variables. In short, decoupled state methods are interesting but not compelling.

3.7 Discontinuous Galerkin

Discontinuous Galerkin methods trade left-hand-side mass matrix connectivity complexity for complexity in right-hand-side flux connectivity. Derivation is identical to the regular Galerkin method except that nodes (i.e., coefficients) are not shared along interior element edges. Thus, inter-element boundary integrals do not cancel, rather, a connective flux is calculated from the discontinuous flow states. The form appears as

$$M_{ele}\frac{da_{ele}}{dt} = F_{ele} + B(ele, neighbors)$$

Inter-element boundary fluxes are typically computed with a linearized characteristics corrections given by the Roe or Riemann invariant forms. Details are available in Li[42]. As part of this project, a finite element discontinuous Galerkin solver was prototyped. Observations are noted below.

3.7.1 Advantages

The discontinuous Galerkin (DG) scheme naturally has several competitive advantages over the regular Galerkin scheme.

The primary advantage is that assembly operations are local element based with the exception of boundary fluxes*. The inverse mass matrix operations only rely on an inverse mass matrix computed once upfront.

The presence of local indexes strongly assists with both computer memory bandwidth and parallelization. From the hardware performance aspect, memory bandwidth is improved when requested values are contiguous and lumped in large blocks; the DG scheme exactly fits this requirement. A similar analysis applies to parallelization since the left-hand-side terms are exactly block diagonal. A trial parallelization of the Fortran code required adding exactly one line (!$OMP PARALLEL DO) and resulted in a 1.8 speed-up on 2 processors.

*Boundary calculations require the neighboring value.

```
!$OMP PARALLEL DO
do iel=1,NumberElements
   call ApplyBoundaryConditions(Ele(iel),Time0)
   call ComputeDaDtGalerkin(Ele(iel),Time0)
enddo
```

In essence, the flow complexities are decoupled into the boundary terms rather than the mass matrix connectivity.

The literature claims that the DG with Riemann boundary fluxes can often dispense with a formal dissipation model (the dissipation is supposed to be inherent in the flux connectivity). This project was not able to substantiate that claim. Details are contained in the disadvantages section below.

DG methods for linear advection are theoretically optimal. The governing equation for linear advection has the form

$$\frac{du}{dt} + c\frac{du}{dx} = 0$$

This optimality does not extend to non-linear advection inherent in the Navier-Stokes equations of motion.

For unsteady boundaries and remeshing refined grids, the DG method conceptually contains significant advantages. Because the element connectivity is no longer tied to nodes shared between connected elements, so-called non-conformal grids are possible. Non-conformal means that edge nodes for a particular element are not necessarily coincident with the neighboring element's nodes. This is particularly valuable for remeshing and variable order elements since a original conformal elements can easily be subdivided without being constrained by element connectivity. Again, Li[42] and Karnadaikis[35] provide examples of non-conformal DG solvers. This project originally considered the DG method to provide significant advantages for moving mesh and unsteady wake problems for similar reasons.

3.7.2 Disadvantages

The DG method also exhibits disadvantages causing this project to abandon further work on the topic. The decoupled nature of DG fields also created significant challenges.

A critically important component, visualization, becomes considerably more difficult for discontinuous elements. Traditional visualization packages were not designed for discontinuous elements; grid data structures and field operators were not applicable. In particular, the multi-valued but

coincident nodes of neighboring elements became a significant issue. Additionally, the higher-order aspect of the DG solutions required in-visualization remeshing that slowed the visualization refresh rate.

Explicit time advancement of a DG formulation is essentially not feasible. The lack of mass matrix coupling requires iteration of current and past states for boundary conditions and inter-element coupling. Additionally, updating all coefficients simultaneously is numerically superior to element by element updates. Element-by-element update methods make iteration between elements slow and possibly unstable. Discontinuous methods appear to have a reduced iterative radius of convergence when compared to traditional Galerkin methods.

Discontinuous elements dictate a more complicated flux connection scheme because of the jump condition on interfaces. To develop a more consistent flux connection we turn to the physics of fluid flow. The only connectivity is through the boundary flux terms. These fluxes require some averaging scheme, usually Riemann invariant based. Unfortunately, the lack of formal connectivity complicates the solid wall boundary conditions. Elements with a face on a particular boundary are easily specified, but elements with only a node on the boundary are not. Worse, the nodal mass has a small influence within the governing equations. Thus, DG methods could be described as elastic plates connected by small springs with smaller spring constants at the vertices. This behavior is commonly seen as solution fields diverging at the element corners. It is not realistic to expect coincident nodes in a DG method to naturally converge to the same value.

As an simple and small example, Figure 3.3 shows the characteristics of DG difficulties. First, the multivalued field issue is apparent at the back step's corner. Pressures and velocities in the element #2 are clearly not constrained to the slip condition boundary condition. Constraining the flux is neither sufficient nor effective; constraining the flow state is required. In this example, the solution blow-up is located at a boundary; however, the situation could occur at any element connection in the domain. DG is a feasible method but requires more attention to fluxes and boundary conditions.

3.8 Derivatives

Use of the Navier-Stokes equations of motion requires gradients of the flow field properties and functions of properties. In particular, the Newtonian stress terms and the heat flux are functions of velocity and internal energy derivatives. The numerical formulation required the use of Green's theorem which assumes continuous flux functions in space. A method is needed to convert the

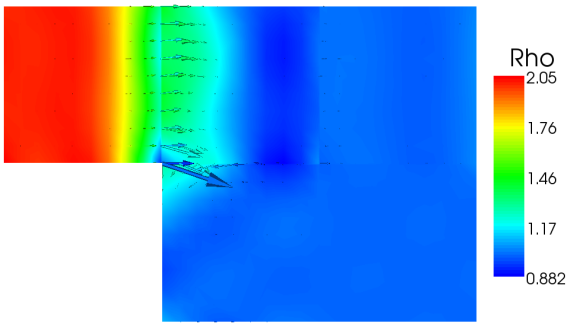


Figure 3.3: Discontinuous Galerkin Back-Step Density and Velocity Field

discontinuous derivatives to continuous fields.

3.8.1 Basis Derivatives

The trivial method for computing derivatives is to take the raw derivative of the basis function and correct with the element Jacobian. This method has a significant disadvantage as it results in discontinuous derivatives across neighboring elements. For linear basis functions, the derivative is constant.

3.8.2 Auxiliary State Galerkin

The approach used for this section is to define auxiliary states that represent a continuous derivative field.

$$s_{ux} = \frac{du}{dx}$$

These auxiliary states are computed by applying the traditional Galerkin approach

$$\int \phi_i \left(s_{ux} - \frac{du}{dx} \right) = 0$$

This reduces to

$$M_G a_{sux} = \int \phi_i \frac{du}{dx}$$

Unfortunately, the velocity derivatives are not continuous. Applying Green's theorem reforms the equations into

$$M_G a_{sux} = -\int \frac{d\phi_i}{dx} u \, d\Omega + \int \phi_i u \cdot n_x \, d\Gamma$$

This has the advantage of not needing to explicitly calculate the derivatives. The disadvantage is that the integrated velocities should be continuous for the boundary integral to extend to the domain boundary. Another disadvantage is that the inverse global mass matrix is needed via iteration.

3.8.3 Per-Element Galerkin (Local Operations)

Global mass matrix operations are expensive, so the use of per-element operations is considered (c.f. Li[42]). Retaining the per-element boundary integral gives

$$M_L a_{sux} = -\int \frac{d\phi_i}{dx} \hat{u}\, dA + \int \phi_i \hat{u}\, dL$$

Now, we need a continuous velocity applied to both integrals. Li[42] suggests an average velocity

$$\hat{u} = avg(u_{ele_1}, u_{ele_2})$$

This form does not result in the desired output; the results are equivalent to the inputs. No smoothing is possible unless the underlying derivatives are already pre-smoothed.

3.8.4 Stencil Derivatives

The approach used for this section is to define auxiliary states that represent a continuous derivative field.

$$s_{fx} = \frac{df}{dx}$$

These auxiliary state errors are minimized with the traditional Galerkin approach

$$\int \phi_i \left(s_{fx} - \frac{df}{dx} \right) = 0$$

This reduces to

$$M_S a_{sfx} = \int \phi_i \frac{df}{dx}$$

Applying Green's theorem transforms the equation into

$$M_S a_{sfx} = -\int \frac{d\phi_i}{dx} f\, d\Omega + \int \phi_i f \cdot n_x\, d\Gamma$$

Rather than the entire grid, only a subset (the nearest elements) are used. Figure 3.4 shows the stencil and element orientations.

The basis functions are given in Figure 3.5. This scheme has a significant advantage in that operations are local to the element and its immediate neighbors.

The complete 6 point, 4 element mass matrix is

$$M = \begin{bmatrix} 2A_1 + 2A_3 + 2A_4 & A_1 + A_4 & A_1 + A_3 & & A_3 & A_4 \\ A_1 + A_4 & 2A_1 + 2A_2 + 2A_4 & A_1 + A_2 & A_2 & & A_4 \\ A_1 + A_3 & A_1 + A_2 & 2A_1 + 2A_2 + 2A_3 & A_2 & A_3 & \\ & A_2 & A_2 & 2A_2 & & \\ A_3 & & A_3 & & 2A_3 & \\ A_4 & A_4 & & & & 2A_4 \end{bmatrix}$$

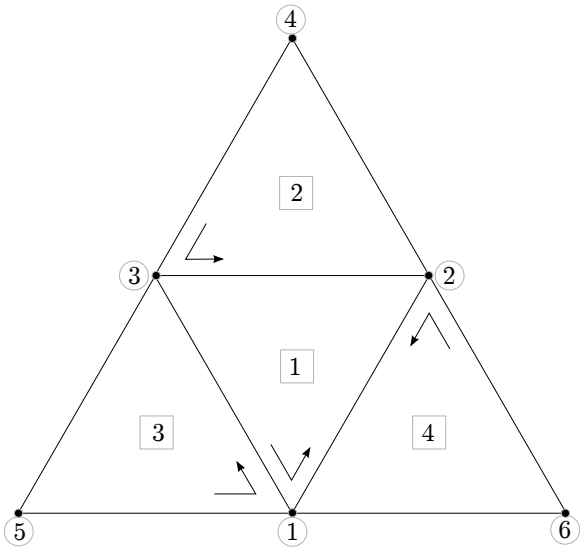


Figure 3.4: Stencil Triangle Basis Functions

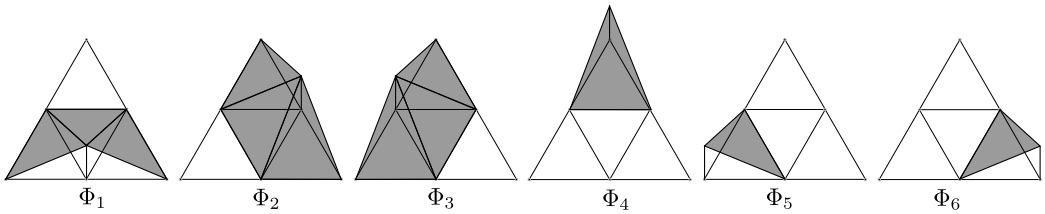


Figure 3.5: Stencil Triangle Basis Functions

The stencil's force vector is composed of the following boundary terms

$$I_1 = \frac{1}{6} L_{e3,3} \left(2u_1 + u_5\right) \left(n_{e3,3} \cdot n_D\right) + \frac{1}{6} L_{e4,2} \left(2u_1 + u_6\right) \left(n_{e4,2} \cdot n_D\right)$$

$$I_2 = \frac{1}{6} L_{e4,3} \left(2u_2 + u_6\right) \left(n_{e4,3} \cdot n_D\right) + \frac{1}{6} L_{e2,2} \left(2u_2 + u_4\right) \left(n_{e2,2} \cdot n_D\right)$$

$$I_3 = \frac{1}{6} L_{e3,2} \left(2u_3 + u_5\right) \left(n_{e3,2} \cdot n_D\right) + \frac{1}{6} L_{e2,3} \left(2u_3 + u_4\right) \left(n_{e2,3} \cdot n_D\right)$$

$$I_4 = \frac{1}{6} L_{e2,2} \left(2u_4 + u_2\right) \left(n_{e2,2} \cdot n_D\right) + \frac{1}{6} L_{e2,3} \left(2u_4 + u_3\right) \left(n_{e2,3} \cdot n_D\right)$$

$$I_5 = \frac{1}{6} L_{e3,2} \left(2u_5 + u_3\right) \left(n_{e3,2} \cdot n_D\right) + \frac{1}{6} L_{e3,3} \left(2u_5 + u_1\right) \left(n_{e3,3} \cdot n_D\right)$$

$$I_6 = \frac{1}{6} L_{e4,2} \left(2u_6 + u_1\right) \left(n_{e4,2} \cdot n_D\right) + \frac{1}{6} L_{e4,3} \left(2u_6 + u_2\right) \left(n_{e4,3} \cdot n_D\right)$$

where $L_{ei,j}$ and $n_{ei,j}$ are the length and normal respectively of element i on face j.

The internal force terms are

$$I_1 = \frac{B_{D1}|_{e1}}{6} \left(u_1 + u_2 + u_3\right) + \frac{B_{D1}|_{e3}}{6} \left(u_1 + u_3 + u_5\right) + \frac{B_{D2}|_{e4}}{6} \left(u_1 + u_2 + u_6\right)$$

$$I_2 = \frac{B_{D2}|_{e1}}{6} \left(u_1 + u_2 + u_3\right) + \frac{B_{D2}|_{e2}}{6} \left(u_2 + u_3 + u_4\right) + \frac{B_{D1}|_{e4}}{6} \left(u_1 + u_2 + u_6\right)$$

$$I_3 = \frac{(-B_{D1} - B_{D2})|_{e1}}{6} \left(u_1 + u_2 + u_3\right) + \frac{B_{D1}|_{e2}}{6} \left(u_2 + u_3 + u_4\right) + \frac{B_{D2}|_{e3}}{6} \left(u_1 + u_3 + u_5\right)$$

$$I_4 = \frac{(-B_{D1} - B_{D2})|_{e2}}{6} \left(u_2 + u_3 + u_4\right)$$

$$I_5 = \frac{(-B_{D1} - B_{D2})|_{e3}}{6} \left(u_1 + u_3 + u_5\right)$$

$$I_6 = \frac{(-B_{D1} - B_{D2})|_{e4}}{6} \left(u_1 + u_2 + u_6\right)$$

The stencil requires boundary conditions and partial stencil conditions. The previously derived 6pt stencil element is not tile-able to boundaries. Fitting the stencil to boundaries requires fundamental shapes. Interior points use nodes of neighboring elements. Freestream points are collapsed by removing the exiting triangle. Solid wall boundaries also remove the exiting triangle.

Conceptually, the stencil is reduced by removing triangles. This is equivalent to reducing the area and redirecting the boundary. The mass matrix is

$$M = \begin{bmatrix} 2A_1+2A_3+2A_4 & A_1+A_4 & A_1+A_3 & & A_3 & A_4 \\ A_1+A_4 & 2A_1+2A_2+2A_4 & A_1+A_2 & A_2 & & A_4 \\ A_1+A_3 & A_1+A_2 & 2A_1+2A_2+2A_3 & A_2 & A_3 & \\ & A_2 & A_2 & 2A_2 & & \\ A_3 & & A_3 & & 2A_3 & \\ A_4 & A_4 & & & & 2A_4 \end{bmatrix}$$

When one element is collapsed, that element's area becomes zero, causing a rank deficient mass matrix. As this node is no longer relevant, adding any non-zero diagonal term for that node in the mass matrix is sufficient to establish determinacy.

The stencil scheme is local, fast, and accurate inside the center element. Disadvantages are that the derivatives are not smooth across different stencils. Also, the derivative quality is poor on outside triangles near the edges. Finally, as derived, the stencil derivative is only applicable to linear basis functions.

In short, the finite element based stencil derivative is akin to a 6 point finite difference derivative.

3.8.5 Taylor Series Galerkin

Analysis of the Galerkin auxiliary state method for determining derivatives indicated that the force vector integrals with linear basis interpolation was the dominate error contributor. Given that the scheme contains derivatives in the solution vector, adding curvature information to the interpolation seemed prudent and possible.

The 2D Taylor series is

$$f(X) = f(a) + (X-a)^T \begin{bmatrix} \frac{df}{dx} \\ \frac{df}{dy} \end{bmatrix} + \frac{1}{2}(X-a)^T \begin{bmatrix} \frac{d^2f}{dx^2} & \frac{d^2f}{dxdy} \\ \frac{d^2f}{dxdy} & \frac{d^2f}{dy^2} \end{bmatrix} (X-a)$$

Yet, we prefer to work in the local coordinate system ζ. After some mathematics, the Taylor series in a local frame is

$$T(\zeta) = T(z) + (\zeta - z)^T [J] \begin{bmatrix} \phi b_x \\ \phi b_y \end{bmatrix} + \frac{1}{2}(\zeta - z)^T \begin{bmatrix} \frac{d\phi}{d\zeta_1} b_x & \frac{d\phi}{d\zeta_1} b_y \\ \frac{d\phi}{d\zeta_2} b_x & \frac{d\phi}{d\zeta_2} b_y \end{bmatrix} J^T (\zeta - z)$$

where J is the familiar elemental Jacobian.

Looking ahead, the maximum order required is 2nd order. Thus, the 6 point triangle with points along the edges is sufficient. The location of point A is $\zeta^T = (1, 0, 0)$. The location of point B is

$\zeta^T = (0,1,0)$. The location of point C is $\zeta^T = (0,1,0)$. These are the segment endpoints. The halfway points are AB as $\zeta^T = \left(\frac{1}{2},\frac{1}{2},0\right)$, BC as $\zeta^T = \left(0,\frac{1}{2},\frac{1}{2}\right)$ and CA as $\zeta^T = \left(\frac{1}{2},0,\frac{1}{2}\right)$.

Each midpoint is interpolated from both directions, so each nodal point needs two increments in coordinates ($AB - A$ for the segment from A to the midpoint of A and B). For node A,

$$\begin{aligned}(\zeta - z)^T_{AB-A} &= \left(\frac{1}{2},\frac{1}{2},0\right) - (1,0,0) = \left(-\frac{1}{2},\frac{1}{2},0\right)\\ (\zeta - z)^T_{CA-A} &= \left(\frac{1}{2},0,\frac{1}{2}\right) - (1,0,0) = \left(-\frac{1}{2},0,\frac{1}{2}\right)\end{aligned}$$

For node B,

$$\begin{aligned}(\zeta - z)^T_{AB-B} &= \left(\frac{1}{2},\frac{1}{2},0\right) - (0,1,0) = \left(\frac{1}{2},-\frac{1}{2},0\right)\\ (\zeta - z)^T_{BC-B} &= \left(0,\frac{1}{2},\frac{1}{2}\right) - (0,1,0) = \left(0,-\frac{1}{2},\frac{1}{2}\right)\end{aligned}$$

For node C,

$$\begin{aligned}(\zeta - z)^T_{BC-C} &= \left(0,\frac{1}{2},\frac{1}{2}\right) - (0,0,1) = \left(0,\frac{1}{2},-\frac{1}{2}\right)\\ (\zeta - z)^T_{CA-C} &= \left(\frac{1}{2},0,\frac{1}{2}\right) - (0,0,1) = \left(\frac{1}{2},0,-\frac{1}{2}\right)\end{aligned}$$

The third ζ term is dependent via $\zeta_1 + \zeta_1 + \zeta_1 = 1$ and can be ignored. Notice that the third term *must not* be ignored when using basis expansion coefficients.

Linear Basis

For the linear basis

$$\begin{aligned}\phi &= \begin{pmatrix} \zeta_1 & \zeta_2 & \zeta_3 \end{pmatrix}\\ &= \begin{pmatrix} \zeta_1 & \zeta_2 & 1-\zeta_1-\zeta_2 \end{pmatrix}\end{aligned}$$

At AB,

$$\phi = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix}$$

At BC,

$$\phi = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

At CA,

$$\phi = \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}$$

The derivatives in the local frame are

$$\frac{d\phi_{j=1,2,3}}{d\zeta_{i=1,2}} = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix}$$

or

$$\frac{d\phi}{d\zeta_1} = \begin{pmatrix} 1 & 0 & -1 \end{pmatrix}$$

$$\frac{d\phi}{d\zeta_2} = \begin{pmatrix} 0 & 1 & -1 \end{pmatrix}$$

Simplify the basis derivatives to

$$\begin{aligned} T(\zeta) &= T(z) + (\zeta - z)^T \begin{bmatrix} x_{13} & y_{13} \\ x_{23} & y_{23} \end{bmatrix} \begin{bmatrix} \phi b_x \\ \phi b_y \end{bmatrix} \\ &\quad + \frac{1}{2}(\zeta - z)^T \begin{bmatrix} \begin{pmatrix} 1 & 0 & -1 \end{pmatrix} b_x & \begin{pmatrix} 1 & 0 & -1 \end{pmatrix} b_y \\ \begin{pmatrix} 0 & 1 & -1 \end{pmatrix} b_x & \begin{pmatrix} 0 & 1 & -1 \end{pmatrix} b_y \end{bmatrix} \begin{bmatrix} x_{13} & x_{23} \\ y_{13} & y_{23} \end{bmatrix} (\zeta - z) \end{aligned}$$

After more mathematics and reduction, the elemental contribution is

$$\begin{aligned} |J|\, Mb_x &= -\left\{ \begin{array}{c} y_{23} \\ -y_{13} \\ -y_{23} + y_{13} \end{array} \right\}_{1,2,3} \frac{1}{6} S \\ &+ n_x L \left\{ \begin{array}{c} \frac{1}{6}T(A) + \frac{1}{3}T(AB) \\ \frac{1}{3}T(AB) + \frac{1}{6}T(B) \end{array} \right\}_{edge1} \\ &+ n_x L \left\{ \begin{array}{c} \frac{1}{6}T(B) + \frac{1}{3}T(BC) \\ \frac{1}{3}T(BC) + \frac{1}{6}T(C) \end{array} \right\}_{edge2} \\ &+ n_x L \left\{ \begin{array}{c} \frac{1}{6}T(C) + \frac{1}{3}T(CA) \\ \frac{1}{3}T(CA) + \frac{1}{6}T(A) \end{array} \right\}_{edge3} \end{aligned}$$

and

$$\begin{aligned} |J|\, Mb_y &= -\left\{ \begin{array}{c} -x_{23} \\ x_{13} \\ x_{23} - x_{13} \end{array} \right\}_{1,2,3} \frac{1}{6} S \\ &+ n_y L \left\{ \begin{array}{c} \frac{1}{6}T(A) + \frac{1}{3}T(AB) \\ \frac{1}{3}T(AB) + \frac{1}{6}T(B) \end{array} \right\}_{edge1} \\ &+ n_y L \left\{ \begin{array}{c} \frac{1}{6}T(B) + \frac{1}{3}T(BC) \\ \frac{1}{3}T(BC) + \frac{1}{6}T(C) \end{array} \right\}_{edge2} \\ &+ n_y L \left\{ \begin{array}{c} \frac{1}{6}T(C) + \frac{1}{3}T(CA) \\ \frac{1}{3}T(CA) + \frac{1}{6}T(A) \end{array} \right\}_{edge3} \end{aligned}$$

with

$$\begin{aligned} S &= T(A)+T(B)+T(C) \\ &+ \frac{1}{8}(x_{13})(-2b_{x_1}+b_{x_2}+b_{x_3}) \\ &+ \frac{1}{8}(x_{23})(b_{x_1}-2b_{x_2}+b_{x_3}) \\ &+ \frac{1}{8}(y_{13})(-2b_{y_1}+b_{y_2}+b_{y_3}) \\ &+ \frac{1}{8}(y_{23})(b_{y_1}-2b_{y_2}+b_{y_3}) \end{aligned}$$

and

$$\begin{aligned} T(AB) &= \frac{1}{2}T(AB)_A + \frac{1}{2}T(AB)_B \\ &= \frac{1}{2}(T(A)+T(B)) \\ &+ \frac{1}{8}(x_{23}-x_{13})(b_{x_1}-b_{x_2}) \\ &+ \frac{1}{8}(y_{23}-y_{13})(b_{y_1}-b_{y_2}) \end{aligned}$$

Likewise

$$\begin{aligned} T(BC) &= \frac{1}{2}(T(B)+T(C)) \\ &+ \frac{1}{8}(x_{23})(b_{x_3}-b_{x_2}) \\ &+ \frac{1}{8}(y_{23})(b_{y_3}-b_{y_2}) \end{aligned}$$

And

$$\begin{aligned} T(CA) &= \frac{1}{2}(T(C)+T(A)) \\ &+ \frac{1}{8}(x_{13})(b_{x_3}-b_{x_1}) \\ &+ \frac{1}{8}(y_{13})(b_{y_3}-b_{y_1}) \end{aligned}$$

This appears to be a beautiful and elegant result. Unfortunately, it also does not work. The method converges slowly and has variable accuracy depending on the derivative direction and location. It also explains why traditional hybrid Galerkin methods for calculating derivatives often give noisy results along the domain edges.

A Singular Experiment in 1D

Reducing the dimensionality to 1D provides some enlightenment to the issue. Given a linear basis function and three elements of unit length, the above methodology reduces to

$$\frac{1}{6}\begin{bmatrix} 2 & 1 & & \\ 1 & 4 & 1 & \\ & 1 & 4 & 1 \\ & & 1 & 2 \end{bmatrix} b = -\frac{d\phi}{d\zeta} T$$

where b is the derivative state vector and T is the field state vector. When substituting for the Taylor series expansion, this form further reduces

$$\begin{bmatrix} 4 & 2 & & \\ 2 & 8 & 2 & \\ & 2 & 8 & 2 \\ & & 2 & 4 \end{bmatrix} b + \begin{bmatrix} -1 & 1 & & \\ 1 & -2 & 1 & \\ & 1 & -2 & 1 \\ & & 1 & -1 \end{bmatrix} b = \begin{bmatrix} 0 & 6 & & \\ -6 & 0 & 6 & \\ & -6 & 0 & 6 \\ & & -6 & 0 \end{bmatrix} T$$

Now, consolidating the derivative state vector and the field state vector gives

$$\begin{bmatrix} 3 & 3 & & \\ 3 & 6 & 3 & \\ & 3 & 6 & 3 \\ & & 3 & 3 \end{bmatrix} b = \begin{bmatrix} 0 & 6 & & \\ -6 & 0 & 6 & \\ & -6 & 0 & 6 \\ & & -6 & 0 \end{bmatrix} T$$

The mass matrix is *singular*. Further, *any* consistent trial function or *any* consistent integration scheme still gives a singular mass matrix.

The traditional hybrid Galerkin method does not use the Taylor series for interpolation, so its convergence is better than the current method. Yet, the same near singular behavior still exists. The approximate method solves better than the perfect method. This also explains the poor convergence around the domain edges for the hybrid method.

Stiffness

The solution to the singular mass matrix is to add a physical constrain to the governing equations. From physics, the nearest analog is the strain energy of a thin plate

$$I = \int\int \left(f_{xx}^2 + 2f_{xy}^2 + f_{yy}^2\right) dxdy$$

We know that the finite element Galerkin method has an equivalent T and U energy form. Thus, strain energy is added to the Galerkin equations. The augmented residual is expected to be dI/db.

$$R = -C\frac{\partial}{\partial b}I$$

For 1D

$$\begin{aligned} I &= \int\int \left(\frac{dq_x}{dx}\right)^2 dxdy \\ &= \int\int \left(\frac{d\phi}{dx}b_x\right)^2 dxdy \\ &= \int\int \left(\begin{pmatrix} -1 & 1 \end{pmatrix} b_x\right)^2 dxdy \end{aligned}$$

$$-\frac{d}{db}\int \left(\begin{pmatrix} -1 & 1 \end{pmatrix} b\right)^2 dx = -C \begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix} b$$

Add this onto the original equations.

$$\begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix} b = \begin{bmatrix} -6 & 6 \\ -6 & 6 \end{bmatrix} T - C \begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix} b$$

$$\begin{bmatrix} 3+2C & 3-2C \\ 3-2C & 3+2C \end{bmatrix} b = \begin{bmatrix} -6 & 6 \\ -6 & 6 \end{bmatrix} T$$

The mass matrix is no longer singular.

In 2D after some mathematics, the additional residual due to strain energy is

$$R = -C\,[A]\left(\begin{pmatrix} b_{x_1} \\ b_{x_2} \\ b_{x_3} \end{pmatrix} + \begin{pmatrix} b_{y_1} \\ b_{y_2} \\ b_{y_3} \end{pmatrix}\right)$$

where

$$A = \begin{pmatrix} \left(y_{23}^2 + x_{23}^2\right) & \left(-y_{13}y_{23} - x_{23}x_{13}\right) & \left(-y_{23}^2 + y_{13}y_{23} - x_{23}^2 + x_{13}x_{23}\right) \\ \left(-y_{13}y_{23} - x_{13}x_{23}\right) & \left(y_{13}^2 + x_{13}^2\right) & \left(y_{13}y_{23} - y_{13}^2 + x_{13}x_{23} - x_{13}^2\right) \\ \left(-y_{23}^2 + y_{13}y_{23}\right) + \left(-x_{23}^2 + x_{13}x_{23}\right) & \left(y_{13}y_{23} - y_{13}^2\right) + \left(x_{13}x_{23} - x_{13}^2\right) & \left(-y_{23} + y_{13}\right)^2 + \left(x_{23} - x_{13}\right)^2 \end{pmatrix}$$

This form was implemented and gave smooth derivatives. One disadvantage is that this method is a global method poorly parallizable. Another disadvantage is that the stiffness must be specified; if the stiffness is too large, the solution is excessively smooth.

3.9 Numerical Methods

Now that the govering equations are in a numerical form, there are two major numerical methods required: time integration and matrix inversion.

3.9.1 Time Integration

Time integration involves integrating a temporal differential equation forward in time. The general governing equation is

$$\frac{dy}{dt} = f(y)$$

with initial conditions

$$y(0) = y_0$$

Order Analysis

We have two types of governing equations: convection and diffusion. Given an arbitrary solution in Fourier space

$$y = ae^{ikx}$$

The convection equation expands to

$$\frac{d\left(\frac{(\rho u)^2}{\rho}\right)}{dx} = \frac{d}{dx}\left(abe^{2ikx}/ce^{ikx}\right) = ike^{ikx}$$

This gives an eigenratio of

$$\lambda = k$$

The dissipation equation expands to

$$\frac{1}{Re}\frac{d^2(\rho e)}{dx^2} = \frac{d}{dx}\left(ae^{ikx}\right) = -\frac{k^2}{Re}e^{ikx}$$

for an eigenratio of

$$\lambda = \frac{k^2}{Re}$$

This indicates that when the Reynolds number is greater than k, then the limiting timestep is convection. Boyd[10] states that "There is little advantage to treating the nonlinear terms implicitly because a timestep longer than the explicit advective stability limit would be too inaccurate to be acceptable."

Literature

Time integration advances a solution when temporal derivatives are known. In general, we are interested in 1st order ordinary differential equations of the form

$$\frac{dy}{dt} = f(t, y)$$

with

$$y(t_0) = y_0$$

We must evaluate accuracy and efficiency of various integration schemes for our particular CFD method. The field of differential equation integration has evolved tremendously in the last few decades, so the well-known methods commonly seen in engineering textbooks[63] must not be prematurely selected. The state of the art in the late 20th century is represented by the *Solving Ordinary Differential Equations* books in two volumes [28, 27]. For reasons to be discussed, volume two[27] is a primary reference for this paper.

The Galerkin formulation of the Navier-Stokes equations are "very stiff" according to Boyd[10]. This implies either an implicit solver or an explicit solver with small timesteps. Increasing the spatial solution order increases the solution stiffness. We should seriously consider an implicit scheme of approximately the same temporal order as spatial order.

Implicit solvers have advantages for boundary conditions and solution assurance. First, the implicit solvers have known residuals. These residuals are easy to watch. Explicit solvers usually do not have as simple of a visual quality indicator. However, too small of a timestep reduces higher order implicit schemes to an equivalent backwards Euler scheme (with all of the disadvantages of such).

For implicit iteration, Hairer and Wanner[27] say:

> For a general nonlinear differential equation the system... has to be solved iteratively. In the stone-age of stiff computation (i.e., before 1967) people were usually thinking of fixed-point iteration. But this transforms the algorithm into an explicit method and destroys the good stability properties.

Traditional numerical method for implicit ode solution is based on Newton's method which requires a Jacobian matrix. Our FE equations are not easily decomposed into an explicit Jacobian, nor is a numerical approximation of the Jacobian appropriate with array sizes in the millions. The most difficult part of an implicit FE solver is not the time advancement scheme but solving the linear equation resulting from the scheme. State of the art for implicit ODE solutions does not yet match

the complexity of FE solvers. The issue is that while the mass matrix is linear, the force vector is not.

$$M\frac{da}{dt} = F$$

F is neither trivial to calculate nor trivial to decompose into linear components necessary for the Jacobian dF/da. Of course, Newton's method is preferred over an iterative Krylov or Jacobi method simply for the convergence rate. Press[63] states

> Even when Newton-Raphson is rejected for the early stages of convergence..., it is very common to "polish up" a root with one or two steps of Newton-Raphson, which can multiply by two or four its number of significant figures!

Predictor Corrector

Predictor corrector (PC) methods are a traditional[63] integration method with the form

$$y_j = hb_i f\left(t_i, y_i\right)$$

Expanded for order $p+1$, this is

$$y_{n+1} = y_n + h\beta_0 f\left(t_{n+1}, y_{n+1}\right) + h\beta_1 f\left(t_n, y_n\right) + \cdots + h\beta_p f\left(t_{n-p+1}, y_{n-p+1}\right)$$

Predictor methods omit the implicit β_0 term to get started; corrector methods include the β_0 term for higher accuracy. Increasing order is obtained by adding more past derivatives. Thus, changing step size h either requires restarting with a lower order approximation or deriving a series of special β terms for the step size propagation.

Press[63] states "We suspect that predictor-corrector integrators have had their day, and that they are no longer the method of choice for most problems in ODEs.... There is one exceptional case: high-precision solutions of very smooth equations with very complicated right-hand sides...." Even worse, the predictor corrector's stability domain shrinks as the integration order increases[27]. Since the time integrator should roughly match the domain expansion order, a decreasing stability domain is certainly not wanted. Innately, the predictor corrector requires that the mapping $f(t, y)$ does not change. In a CFD code, a constant mapping indicates a fixed computational grid.

Runge Kutta

Runge Kutta methods refer to both implicit and explicit multi-stage time integration. Iserles's[32] book provides a valuabe reference for Runge Kutta schemes.

The general form of a Runge-Kutta type integrator is

$$y_{n+1} = y_n + hb_i k_i$$

$$k_i = f\left(t_n + hc_i,\, y_n + ha_{ij}k_j\right)$$

$$t_i = t_o + hc_i$$

Increasing order is obtained by adding more k terms. Changing step size h is possible at each step. This form is often displayed as a tableau

$$\begin{array}{cccc} c_1 & a_{11} & \cdots & a_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ c_p & a_{p1} & \cdots & a_{pp} \\ \square & b_1 & \cdots & b_p \end{array}$$

Explicit methods consist of a lower triangular a where all $a_{ij} = 0$ when $j \geq i$; implicit methods have at least one non-zero a_{ij} term where $j \geq i$. Implicit RK requires iteration. RK properties include: $c_i = \sum_j a_{ij}$ and $\sum_j b_j = 1$. For the typical application, these properties are usually confined to transcription error identification.

Each RK step is completely independent of previous steps. More importantly, the mapping $f(t, y)$ can change space. For CFD applications, RK allows for a completely different computational grid at each timestep.

We will describe some of the common RK integrators below.

Forward and Backward Euler The forward Euler, a 1st order method, is a simple integrator.

$$y_{n+1} = y_n + hf\left(t_n, y_n\right)$$

The tableau is

$$\begin{array}{cc} 0 & 0 \\ & 1 \end{array}$$

By comparison, the backward Euler is an implicit 1st order method

$$y_{n+1} = y_n + hk$$

$$k = f\left(t_n + h, y_n + hk\right)$$

Notice that y_n does not form a closure; iterations and stopping criteria are required. Its tableau is

$$\begin{array}{cc} 1 & 1 \\ & 1 \end{array}$$

Crank Nicholson Crank Nicholson (C-N) is an implicit second order method often seen in finite difference codes. The tableau is

$$\begin{array}{ccc} 0 & 0 & 0 \\ 1 & \frac{1}{2} & \frac{1}{2} \\ & \frac{1}{2} & \frac{1}{2} \end{array}$$

IRK2 One possible second order implicit RK2 method is

$$\begin{array}{ccc} 0 & \frac{1}{4} & -\frac{1}{4} \\ \frac{2}{3} & \frac{1}{4} & \frac{5}{12} \\ & \frac{1}{4} & \frac{3}{4} \end{array}$$

Expanded, this is

$$\begin{aligned} t_1 &= t_o \\ t_2 &= t_o + \frac{2}{3}\Delta t \\ z_1 &= y_n + \Delta t \left(\frac{1}{4} F(z_1, t_1) - \frac{1}{4} F(z_2, t_2) \right) \\ z_2 &= y_n + \Delta t \left(\frac{1}{4} F(z_1, t_1) + \frac{5}{12} F(z_2, t_2) \right) \\ y_{n+1} &= y_n + \Delta t \left(\frac{1}{4} F(z_1, t_1) + \frac{3}{4} F(z_2, t_2) \right) \end{aligned}$$

This looks ripe for iteration; however, this is exactly the situation Haier warns about using fixed point iteration rather than fully implicit inversion.

Hammer-Hollingsworth This is another implicit RK2 method.

$$\begin{array}{ccc} \frac{3-\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4} - \frac{\sqrt{3}}{6} \\ \frac{3+\sqrt{3}}{6} & \frac{1}{4} + \frac{\sqrt{3}}{6} & \frac{1}{4} \\ & \frac{1}{2} & \frac{1}{2} \end{array}$$

RK4 The canonical Runge-Kutta integrator is the explicit RK4. The tableau is

$$\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

Adaptive RK Adaptive RK typically indicates two RK methods where one lower-order method is a subset of the higher-order method. Adaptive methods allow for fast error estimates for calculating step sizes. Famous methods are Cash-Karp, RKF, etc[63].

Efficient Implicit Formation

Directly implementing one of the above RK routines is not especially efficient. For the implicit RK routines, there are stability and major efficiency issues with stagewise iteration. The conceptual issue is that the mass matrix only contains the temporal information about the governing equations. From theory, this restricts both the rate of convergence and the timestep, since the product of timestep and right-hand-side eigenvalues must be small. We need Jacobian information.

A generic form of a RK stage is

$$z_i = y_o + \Delta t\, a_{ij} \left(\frac{dz}{dt}\right)_j$$

This implies a matrix inverse operation

$$z_i = y_o + \Delta t\, a_{ij} \left(M^{-1}B\right)_j$$

where

$$M\frac{dz}{dt} = B$$

contains the fluid governing equations. However, this form is not especially efficient.

Instead, the equation is premultiplied by the conceptual mass matrix to form

$$Mz_i = My_o + \Delta t\, a_{ij} B_j$$

Even further simplification occurs when expanding z_i

$$z = y_o + \Delta z$$

Also, B is expanded into a 1st order Taylor series as

$$B\left(z_i\right) = B\left(y_0\right) + \frac{dB\left(y_0\right)}{dz}\left(z_i - y_0\right) = B\left(y_0\right) + \frac{dB\left(y_0\right)}{dz}\Delta z$$

Combining gives

$$M\Delta z_i = \Delta t\, a_{ij} \left(B\left(y_0\right) + \frac{dB\left(y_0\right)}{dz}\Delta z\right)$$

Now, forcing function information is available for stage iterations

$$\left(M - \frac{dB}{dz}\right)\Delta z_i = \Delta t\, a_{ij} B_o$$

This appears to be a distinct disadvantage since dB/dz, the Jacobian term, is complicated. This is where a non-direct matrix inversion method allows a simplification for an already multiplied Jacobian and vector term. Following Gear and Saad[22], a numerical Jacobian approximation is typically sufficient

$$Jv \approx \frac{1}{\epsilon}\left(B\left(z+\epsilon v\right)-B\left(z\right)\right)$$

The time integration equations are now in a canonical form for numerical inversion.

As expected, the multi-stage RK routines are now coupled where previously they were block independent. A two stage RK routine ready for numerical inversion has the form

$$\begin{bmatrix} [M-\Delta t\, a_{11}J_1] & [-\Delta t\, a_{12}J_2] \\ [-\Delta t\, a_{21}J_1] & [M-\Delta t\, a_{22}J_2] \end{bmatrix} \begin{pmatrix} \Delta z_1 \\ \Delta z_2 \end{pmatrix} = \Delta t \begin{bmatrix} a_{11}B(y_0)+a_{12}B(y_0) \\ a_{21}B(y_0)+a_{22}B(y_0) \end{bmatrix}$$

whereas the non-Jacobian form was

$$\begin{bmatrix} [M] & \\ & [M] \end{bmatrix} \begin{pmatrix} \Delta z_1 \\ \Delta z_2 \end{pmatrix} = \Delta t \begin{bmatrix} a_{11}B(z_1)+a_{12}B(z_2) \\ a_{21}B(z_1)+a_{22}B(z_2) \end{bmatrix}$$

This reflects the change to a true implicit iterative scheme that satisfies the Hairer and Wanner[27] fixed-point iteration stability comment. The disadvantage is a *tremendous* increase in the computational requirement (i.e., computing $J\Delta z$ at each stage and step).

Expansion Point Expanding around a different point is instructive.

$$z = \bar{z} + \Delta z$$

so that

$$\Delta z = z - \bar{z}$$

The 1st order Taylor series is

$$B\left(z_i\right) = B\left(\bar{z}_i\right) + \frac{dB\left(\bar{z}_i\right)}{dz}\left(z_i-\bar{z}\right) = B\left(\bar{z}_i\right) + \frac{dB\left(\bar{z}_i\right)}{dz}\Delta z_i$$

Combining as before gives

$$M\left(z_i-y_0\right) = \Delta t\, a_{ij}\left(B\left(\bar{z}_i\right) + \frac{dB\left(\bar{z}_i\right)}{dz}\Delta z_i\right)$$

This needs one more step

$$\left(z_i-y_0\right) = \left(z_i-y_0+\bar{z}-\bar{z}\right) = \Delta z_i + \left(\bar{z}-y_0\right)$$

Combining gives

$$M\Delta z_i = \Delta t\, a_{ij}\left(B\left(\bar{z}_i\right) + \frac{dB\left(\bar{z}_i\right)}{dz}\Delta z_i\right) - M\left(\bar{z}-y_0\right)$$

Nicely, the initial iteration residual is

$$R = \Delta t \, a_{ij} B\left(z_i\right)$$

and the linear term is

$$Ax = M \Delta z_i$$

This form should be more robust when the Jacobian dB/dz is not exact. The objective would be to reduce the converged Δz to zero. Otherwise, this form reduces to the previous form.

Time Integration Experiment The concepts introduced above are tested for a known solution. The differential equation is

$$\frac{dy}{dt} = -1 - Cy$$

with the initial condition

$$y(0) = 1$$

The solution is

$$y(t) = -\frac{1}{C} + \left(1 + \frac{1}{C}\right) e^{-Ct}$$

The objective is to compare simple one-stage time integration methods for explicit (Forward Euler), fixed-point implicit (Backwards Euler), and Jacobian-coupled implicit (Backwards Euler) routines. Figure 3.6 plots the one-step prediction for an increasing timestep and increasing time constant C. Repeated, this experiment only considers one step with a varying timestep Δt from 0 to 2. Forward Euler behaves as expected with a linear prediction based on the solution derivative at $y(0)$. Fixed point iteration of the backwards Euler method converges for small timesteps and diverges for larger timesteps; this situation is what Hairer and Wanner mean by fixed-point iteration stability. The fixed point scheme becomes unstable for timesteps larger than approximately $1/C$. This is consistent with the previous assertion that timestep multiplied by eigenvalues must be small. The Backwards Euler method with Jacobian information converges for all timesteps.

The point to take away is that just because a scheme is iterative does not mean that it is guaranteed to converge. Nor does an arbitrary iterative scheme always allow larger timesteps than an explicit scheme. Fixed point iteration does indeed destroy the stability advantages of an implicit scheme. As Boyd illustrates[10], implicit methods track the slow manifold.

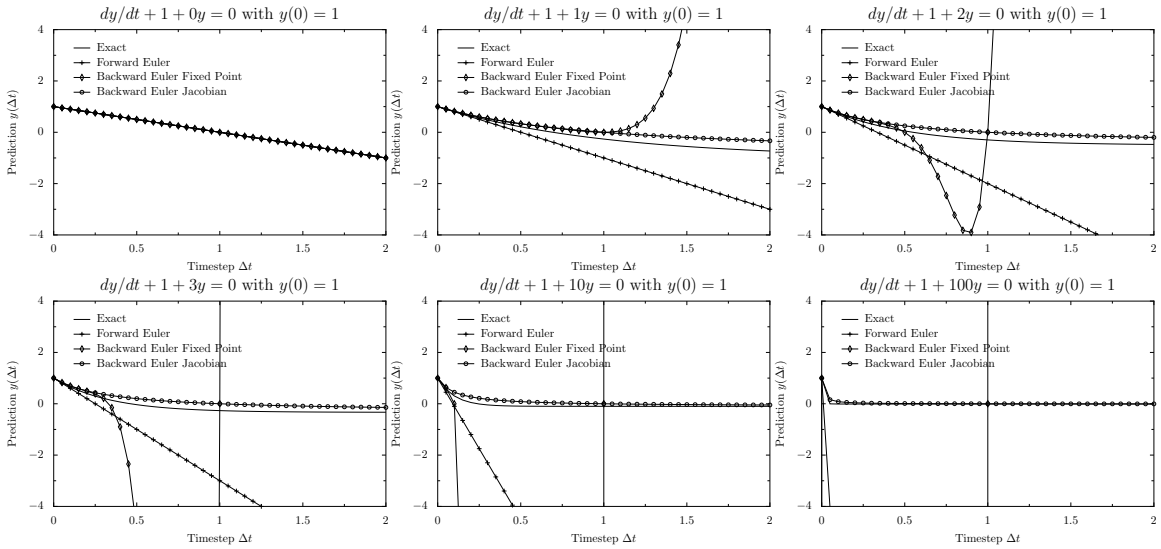


Figure 3.6: Time Integration Experiment

3.9.2 Numerical Matrix Inversion

Matrix inversion is a critical operation for effective Galerkin solver design. The canonical form for matrix inversion is

$$Ax = b$$

In residual form for iteration, the canonical form is

$$r = b - Ax$$

The objective is to reduce the residual r to zero.

The numerical matrix inversion literature is large and continuously evolving. The *Templates* book[4], Boyd's book[10], and [67] are useful starting points for investigating iterative methods. Preconditioning and other advanced routines[15] are known to improve convergence rates.

Generic Jacobi and Krylov Iteration

Jacobi iteration updates the state vector with the residual scaled for stability.

$$x_{i+1} = x_i + \alpha_i r_i$$

where α is chosen based on an approximation to A's eigenvalues[71].

Richardson Residual Minimization

The Richardson Residual Minimization method[10] uses Jacobi iteration with

$$\alpha_i = \frac{\sum r_i q_i}{\sum q_i q_i} = \frac{\sum r_i r_i}{\sum r_i q_i}$$

where q_i is calculated as

$$q_i = A r_i$$

This method is a linear equation residual minimization along the steepest descent direction. A should be positive semi definite but not necessarily symmetric[71].

Conjugate Gradient

The conjugate gradient (CG) method is popular with nice convergence properties at the expense of more storage. CG methods also require a positive semi-definite symmetric A. Shewchuk[68] provides an excellent foundation to the various CG methods. Press[63] shows Polak-Ribiere correction for β as

$$\beta_i = \frac{(r_{i+1} - r_i) \cdot (r_{i+1})}{r_i \cdot r_i}$$

This correction gracefully adapts the CG to a soft restart. Computing the correction is not quite as graceful.

Preconditioning

Preconditioning the inversion improves the iterative process. The general idea is that the inverse of A is difficult, but an approximation to A is easy to invert. So premultiply by the approximation P^{-1}

$$P^{-1} A x = P^{-1} B$$

Naturally, if $P^{-1}A = I$, then there is no need to iteratively invert A. Yet, when P contains some fundamental portions of A's eigenvectors, then $P^{-1}A$ becomes more diagonal. The tradeoff is finding a sufficiently complex but invertible approximation to A.

3.10 Initial Conditions

Initial conditions are required. The 2D Eulerian flow states are

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho e \end{pmatrix}$$

The objective is to determine the initial non-dimensionalized states needed as freestream initial conditions. Initial velocity conditions are set to the reference of 1. With a specified Mach number and sonic velocity, the energy is constrained. Freestream values are used as the dimensional reference values. Density is simple.

$$\rho^\star = \frac{\rho}{\rho_o}$$

Momentum is also unity

$$\rho V^\star = \frac{\rho V}{\rho_o V_o}$$

Energy initial conditions requires recognition that it is dependent on Mach number and sonic velocity

$$a = \sqrt{\gamma RT} = \sqrt{\frac{\gamma p}{\rho}}$$

so that

$$\rho e = \frac{p}{\gamma - 1} + \frac{1}{2}\rho V^2$$

substitute to give

$$\rho e = \frac{\rho a^2}{\gamma\left(\gamma - 1\right)} + \frac{1}{2}\rho V^2$$

When using non-dimensional reference values

$$\rho e^\star = \frac{\rho e}{\rho U_o^2} = \frac{a^2}{\gamma\left(\gamma - 1\right) a^2 M^2} + \frac{1}{2}\frac{\rho V^2}{\rho a^2 M^2}$$

simplifies to

$$\rho e^\star = \frac{1}{\gamma\left(\gamma - 1\right) M^2} + \frac{1}{2}$$

Thus, the non-dimensional reference pressure is

$$p^\star = \frac{p}{\rho_o U_o^2} = \left(\gamma - 1\right)\left(\rho e - \frac{1}{2}\rho V^2\right)\frac{1}{\rho_o U_o^2} = \left(\gamma - 1\right)\left(\rho e^\star - \frac{1}{2}\right)$$

or

$$p^\star = \frac{1}{\gamma M^2}$$

Boundary conditions on velocity are also required. For no-slip conditions, the boundary velocity is zero and is trivial to set.

$$V = 0$$

For the slip condition, the boundary velocity V should have no component in the wall normal direction n.

$$V \cdot n = 0$$

So an initial condition for a slip boundary condition should adjust the boundary velocity by removing the normal component

$$V_{IC} = V - (V \cdot n)\, n$$

Expanded, this is

$$\begin{pmatrix} u_{IC} \\ v_{IC} \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} - (un_x + vn_y) \begin{pmatrix} n_x \\ n_y \end{pmatrix}$$

This form does not conserve the kinetic energy of the non-consistent initial condition. Conservation is a primary issue for constraining the boundary conditions in the mass matrix time derivatives rather than directly through multiple applications of a velocity adjustment routine.

There is a fundamental question of how to implement the above adjustment equation with the expansion of basis functions. Remember that $u = \phi a_u$ so that for the u velocity

$$\phi a_u = \phi a_u - \phi a_u n_x^2 + \phi a_v n_x n_y$$

Expanding for all locations x_i gives

$$\begin{aligned} [\phi_j(x_i)] \begin{pmatrix} \\ (a_u)_j \\ \\ \end{pmatrix} &= [\phi_j(x_i)] \begin{pmatrix} \\ (a_u)_j \\ \\ \end{pmatrix} \\ &- \begin{bmatrix} (n_x^2)_1 & & \\ & \ddots & \\ & & (n_x^2)_L \end{bmatrix} [\phi_j(x_i)] \begin{pmatrix} \\ (a_u)_j \\ \\ \end{pmatrix} \\ &+ \begin{bmatrix} (n_x n_y)_1 & & \\ & \ddots & \\ & & (n_x n_y)_L \end{bmatrix} [\phi_j(x_i)] \begin{pmatrix} \\ (a_v)_j \\ \\ \end{pmatrix} \end{aligned}$$

Now the critical question concerns the invertability of the matrix $\phi_j(x_i)$. Rephrased, the basis set is uniquely determined, so that operations are possible on the raw coefficients only when the normals

are constant.

$$a_u = a_u - a_u n_x^2 + a_v n_x n_y$$

This form is ready for implementation.

Chapter 4

Higher Order Analysis and Results

This chapter discusses implementation, analysis, and results of a higher order Navier-Stokes CFD solver.

4.1 Implementation

Implementing a higher order solver is complicated. This section explains the fundamental routines and choices. An in-depth analysis would require significant amounts of code and is not performed.

The governing equations are the ALE Navier-Stokes equations discretized with a Galerkin approach. Almost all code is written in the Fortran 2003 format. With one exception (g2d grid geometry input and output from Tim Cowan), all code was written by the author. There are two parts to this analysis: numerical operations and data structures.

Numerical operations are primarily composed of B-Spline operations, Galerkin operations, and time-advancement operations. The B-Spline operations in Math2DOps.f90 and Math1DOps.f90, being especially large and complex as basis order increases, are generated into Fortran automatically with a Python computer code. As order increases, the number of characters per line-of-operation increases. Thus, the Python code was programmed to both generate the code and properly format the code (i.e., splitting a line of code requires both mathematical and Fortran consistency). As much as possible, these generated codes use parameters and precalculated values to assist the compiler with optimization. For example, the code that generates values from coefficients is

```
def genFortranValues(n,m,p):
        listA=[]
        v =PyValues(n*p,m*p)
name = "ValP%iP%i" % (n,m)
        string ="""
                ! Calculate Values for grid size Pn and Coefficients Pm
                pure function %s(a)
                real(WP) :: %s(p%i)
                real(WP),intent(in) :: a(p%i)""" % (name,name,n,m)
        for j in BsplineRange(n*p):
                listA=[]
                index = 1
```

```
                for i in v[j-1]:
                        listA.append( str(round(i,14))+"_wp*a(%i)" % index )
                        index += 1
                string += """
                        %s(%i)= """ %(name,j)
                string += ' + '.join(listA)
                string += """
                        end function %s \n""" %(name)
        return string
```

The upfront cost to generate the Python code became negligible compared to the avoided cost of generating and then debugging these operations by hand. Galerkin operations are primarily composed of determining values and then integrating. The mass matrix exemplifies the structure of all Galerkin operations.

```
! Mass
write(*,*) "Generating Mass Matrix"
MassRaw = 0
do i=1,p1
        ! Set basis i
        call setOrtho(Icoeffs,i,p1)
        ! Values of basis i
        Ivals = ValP2P1(Icoeffs)
                do j=1,p1
                        ! Set basis j
                        call setOrtho(Jcoeffs,j,p1)
                        ! Values of basis j
                        Jvals = ValP2P1(Jcoeffs)
                        ! Multiply Values of basis i and j
                        T = Ivals*Jvals
                        ! Integrate and store in mass matrix
                        MassRaw(j,i) = Integrate2dValuesP2(T)
                enddo
enddo
```

The Galerkin force calculations are *considerably* more complicated and lengthy. Time operations are composed of determining the solution residuals and then correcting the solution vector. For a backwards Euler time advancement method, the code necessary is given below

```
do rkiter=1,IterationMax
        ! Derivatives
        call ComputeDerivatives(Snapshot)

        ! Galerkin Force Residual
        call FluxResidual(Snapshot%Coeffs1,rhs1)

        ! Mass Temporal Residual
        call Aprod(n,Snapshot%Coeffs1,rhs3)
```

```
            call Aprod(n,Snapshot%Coeffs0,rhs2)

            ! Total Residual
            rhs1 = DeltaT*rhs1 + rhs2 - rhs3

            ! Constrain Residual
            call BoundaryConstraint(rhs1)

            ! Create Scalar Representation of Residual
            DeltaNew = sqrt(sum(rhs1*rhs1))

            ! Update States
            Snapshot%Coeffs1 = Snapshot%Coeffs1 + ml*rhs1

            ! Inform user
            call writeIteration("+")

            ! Exit if Residual are small enough
            if(DeltaNew < ResidualTolerance) exit
        end do
```

These are the major components of the CFD solver.

The second major component of the CFD solver is the data structure. The data structures are based on the Euler3d data structures. In particular, the most used data structure is the element nodal map, a vector describing the global indices for a given element. A significant task of the data structures is implementing the higher order grid from a generated linear grid. This pre-processing task was a challenge and was moved to a separate routine, makea2d. Ironically, one justification for attempting higher order was to simplify the grid. The major difficulty is that each element now has an order-dependent number of coefficients. For the continuous Galerkin method to work, values along the element edges must be equal; this requires shared coefficients. Shared coefficients must be identified and placed as a pre-processing step.

A final implementation requirement was visualization. The existing VTK library was used by creating a Paraview plugin that reads and converts the Ale2d output data file and geometry to the VTK data structure. The interpolation order for visualization is only 1st order; the higher order solutions are sub-sampled to the element vertices.

4.2 ALE Results

As this project progressed, the emphasis turned from a strict development and verification task to a more general efficiency analysis task. Additionally, a poorly functioning dissipation routine

restricts solutions to low Reynolds numbers. This section therefore presents Ale2D solutions in a more qualitative manner than the typical formal verification and validation process.

A cross-flow cylinder provides an excellent viscous validation test case. Figure 4.1 gives the streaklines and Mach number distribution for the transient buildup of trailing vortices at a Reynolds number of 500. There are several issues that need to be discussed. First, the grid when compared

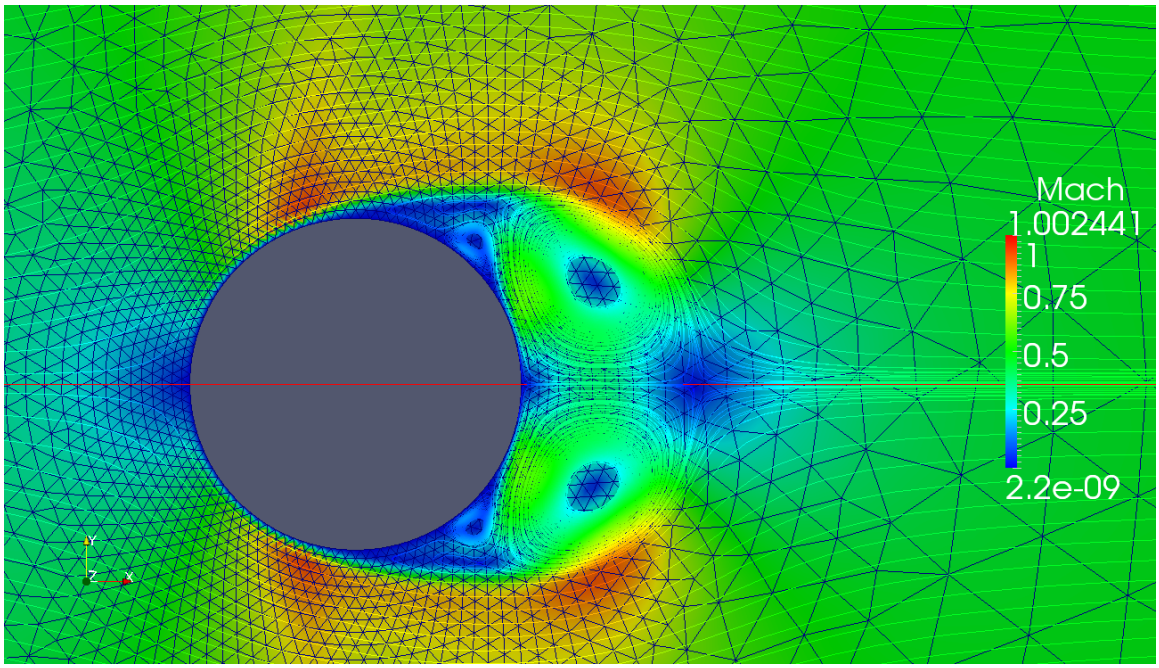


Figure 4.1: Unsteady Cylinder: Viscous $Re = 500$ with ALE P=2

to a linear solution is significantly coarser. Secondary vortices are resolved within only 5 elements. Second, solution instabilities eventually occur at the 90 degree upper and lower regions where the velocity is largest; these instabilities eventually corrupt and end the simulation. The simulation never progressed far enough to measure the Strouhal number of shed vortices.

A Sod shock tube test case demonstrates compressible flow simulations. The flow solution is expected to show three waves moving at $u + a$, u, and $u - a$. Figure 4.2 shows densities, velocities, and pressures along a cutline. The wave speeds and values match theory. However, the strong compression wave generates noise (especially between locations of 0.6 and 0.85). In general, compression waves for all testcases are generating unphysical noisy solutions. Computing entropy indicates that the 2nd law is being locally violated; the weak nature of the Galerkin method is allowing incorrect solutions.

An NACA 0008 airfoil was simulated at a Reynolds number of 2000 and Mach 0.3. The flow velocity field and streaklines are plotted in Figure 4.7. Timesteps are on the order of 0.0001 to

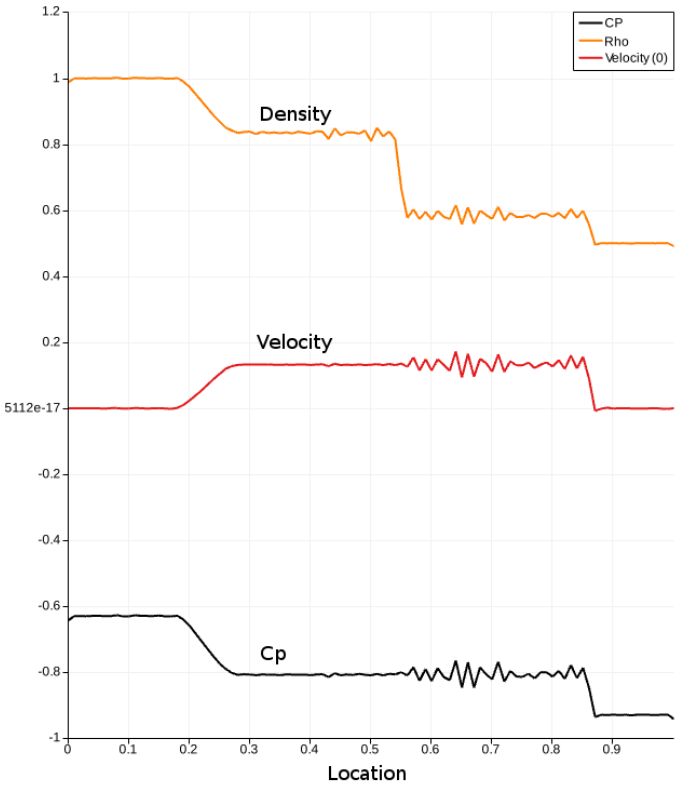


Figure 4.2: Sod Shock Tube P=2

0.00001 with a minimum element size of about 0.01. Experimental data for this ultra low Reynolds number is not prevalent. Rather, the XFOIL[17] prediction program is used for comparison. A sweep of angle of attack is shown in Figure 4.3 for C_L, Figure 4.4 for C_D, and Figure 4.5 for the polar plot. Lift coefficients for angles of attack at and below 4 degrees match XFOIL's predictions within a few percentage points. As the angle increases, XFOIL's accuracy degrades as the trailing edge separation begins. Ale2d poorly captures the separation and thus overpredicts the lift coefficient. Drag is overpredicted by at least 10% to 20% across the entire angle of attack range.

At 8 degrees angle of attack, the airfoil should show trailing edge separation. This is not occuring. Figure 4.6 shows the velocity field at 8 degrees. Unfortunately, the Ale2d solver is sensitive to separation. Simulations consistently failed to iteratively converge as soon as any separation started.

In fact, the most stalled simulation produced by Ale2d is shown in Figure 4.7 for an NACA 0012. The Ale2d simulation failed to converge as the separation began. It is unclear how the Ale2d boundary layer would form when given more simulation time. However, the overly restricted short simulations times are a significant failure. It is likely that the lack of dissipation is allowing this particular failure.

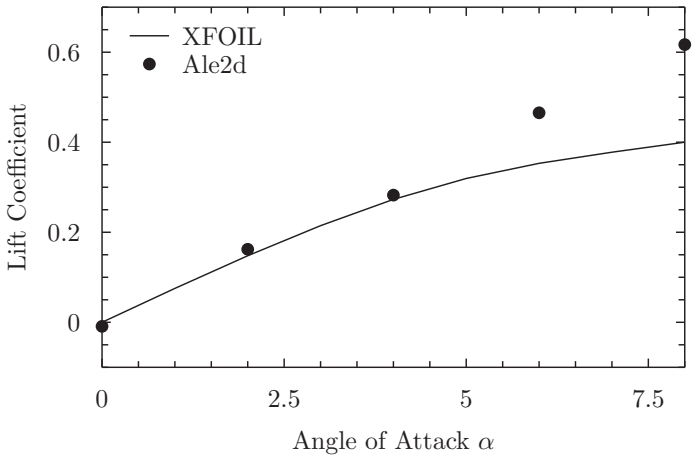


Figure 4.3: NACA 0008 C_L vs α

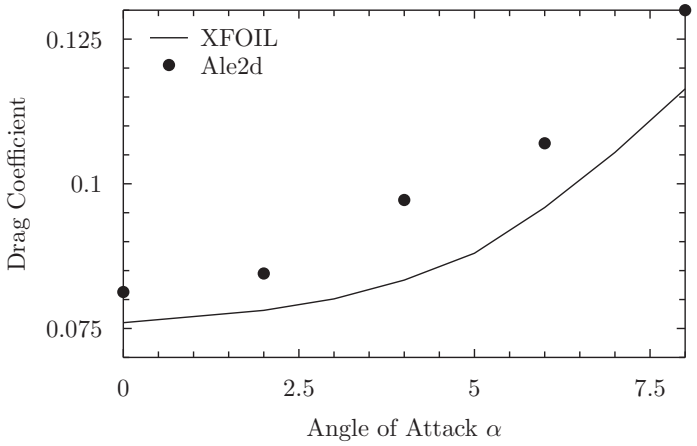


Figure 4.4: NACA 0008 C_D vs α

The following observations were made when comparing low order and higher order simulations. These observations are frustrating when encountered and make lower order solvers (such as euler2d or ns2d) appear more robust regardless of efficiency.

Using a residual based iteration is more robust than computing d/dt of coefficients and the applying these temporal derivatives to an RK routine. Not only does the RK routine spend time iterating for an intermediate sub-step, but the stability also appears to be significantly worse. It is strongly suggested that residual iterations be performed on coefficent values and not temporal derivatives.

Solution divergence (i.e., blowup time) is shorter with higher order. There are fewer warning signs when compared to the linear solvers. These higher order methods also are significantly more timestep sensitive. Small increases (say +10%) in timestep often causes an immediate failure with diverging iterations.

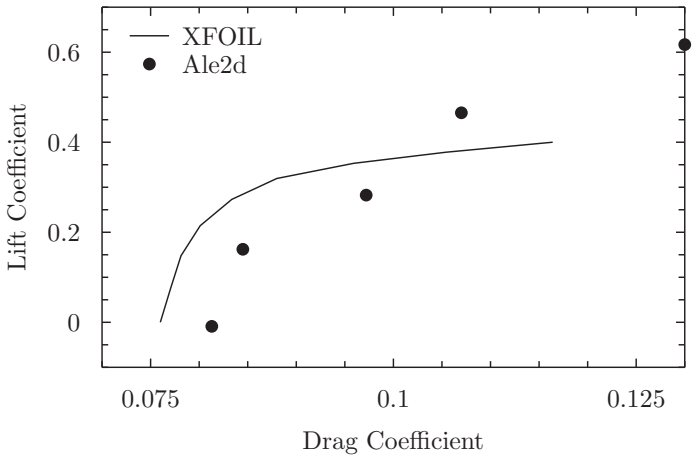


Figure 4.5: NACA 0008 Drag Polar Plot

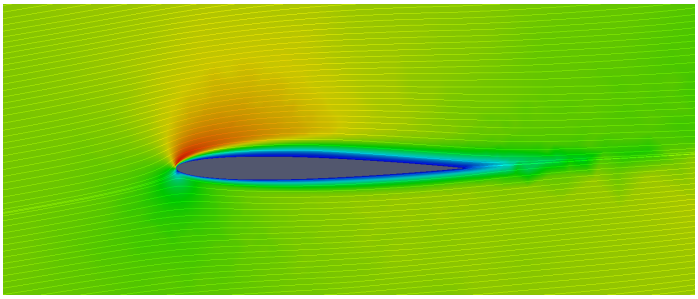


Figure 4.6: NACA 0008 Unsteady Velocity: $Re = 2000$ (P=3)

Another solution failure mode is under-resolution. This failure tends to appear as either upstream moving waves generated in high velocity areas or noise in pressure in the local flow direction with a wavelength equal to twice the grid spacing. When these regions are identified, remeshing is essentially required.

Galerkin computed derivatives (Section 3.8) are expensive. Directly computing derivatives on every iteration requires significantly more time than the actual iteration. Faster approximations with poorer quality appear more efficient.

4.3 Model Based Solution Timing

This experiment investigates total solution time with respect to basis order. From earlier, the solution efficiency tradeoff is accuracy versus CPU work. The solution accuracy depends on the solution properties (e.g., smooth or discontinuous field). The concept of work depends on the solution type (e.g., steady versus moving boundaries). For this experiment, these combinations of the solution field and solution type are tested. For the numerical code, a simulation to a given

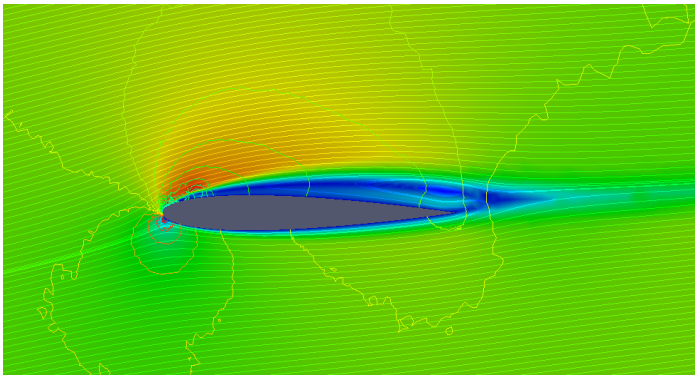


Figure 4.7: NACA 0012 Unsteady Velocity: $Re = 2000$

real time requires a solution time containing grid generation, integration of the Galerkin forces, the solution vector iteration, and time integration.

The objective is to create timing models of the CFD components with respect to accuracy and basis order. This approach should give insight into the work and accuracy contribution from individual components.

4.3.1 Grid Timing

Generating the finite element grid requires work. For this experiment, the grid is one of three 2D geometries but with varying triangle element sizes: a rectangular box, flow past a cylinder, and a 1:1 cavity. As element size h decreases, the number of elements increases as h^2; the experimental results for the rectangular box with varying spacing match this theoretical exponent (Fig. 4.8). The 2.2 coefficient occurs because the grid generator prefers equilateral triangles. Thus, the per-element average area is only $\frac{\sqrt{3}}{4}h^2$ rather than $\frac{1}{2}h^2$ as might be expected. Figure 4.9 plots the grid generation time versus number of elements. A model of time for a given number of elements is

$$T_{grid} = 3.7 \times 10^{-10} \cdot N^{2.32}$$

The log-log plot indicates a start-up overhead time of about 0.01 seconds.

4.3.2 Solution Accuracy

Solution accuracy primarily depends on the solution field characteristics and the element spacing. Frequency and polynomial content reflect in the p convergence rate. The element spacing reflects in the h convergence rate. The objective is to estimate the global accuracy across order and spacing. As we will see, convergence theory given in the finite element literature contains a subtle assumption and is not applicable to the specific task of comparing accuracy for variable order.

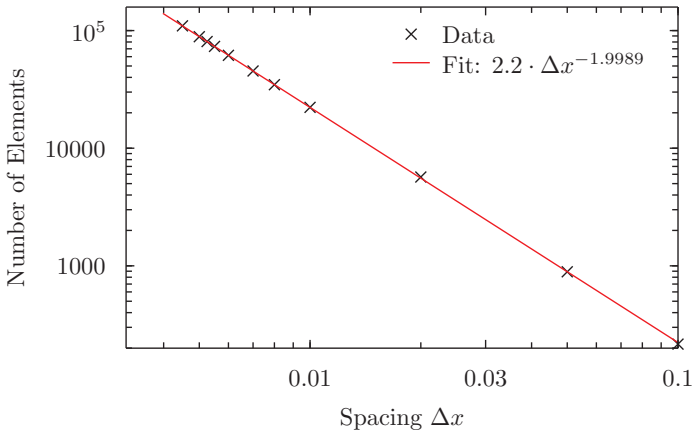
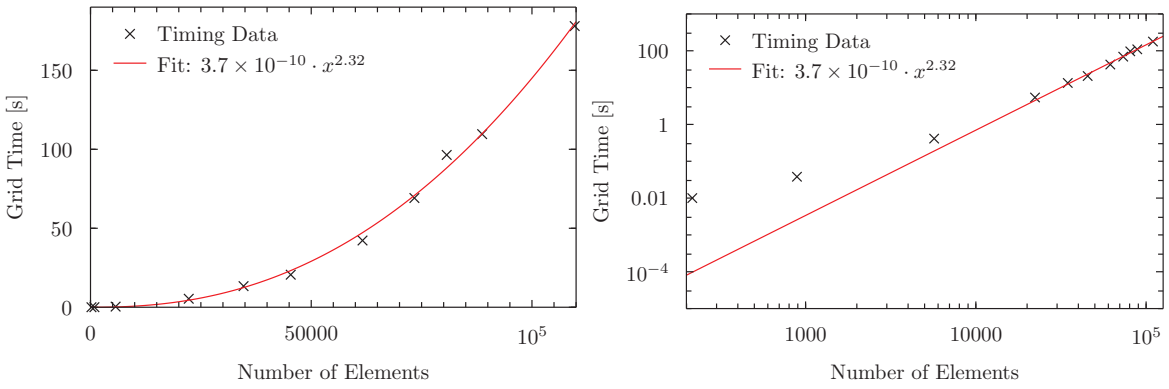


Figure 4.8: Number of Elements vs. Spacing



(a) Lin-Lin

(b) Log-Log

Figure 4.9: Grid Timing

From theory for smooth fields, one form of the solution accuracy depends on the element spacing to the power of the basis order[34]

$$e \approx C\Delta x^P$$

For non-smooth fields such as a shock, the solution accuracy depends on element spacing to the first power

$$e \approx C\Delta x$$

Theory[34, 35] suggests that the L_2 error scales with

$$e = Ch^{p+1} \|u\|_2$$

For an actual field, the solution error is between these two curves. It should be noted that this behavior is not merely theoretical; a conference paper[51] discussing grid spacing and accuracy for

supersonic flows implied requiring tens to hundreds of coefficients per shock depending on the Mach number and shock angle.

Unfortunately, the constant C is not independent of the basis order and is thus only valid for that particular order. This model is meant for comparing same-order convergence curves versus element spacing. It is not meant for comparing different order solutions. For more information on the derivation's assumptions, refer to Johnson[34]. At this point, pure theory must be rejected for forming an accuracy versus spacing and order model.

Rather, actual experimental accuracy data is used. Forming an accuracy model from experimental results requires a careful setup and attention to sensitivities. First, the definition of accuracy must be precisely defined and must correspond to the useable in the actual CFD solution. Second, the experimental field must continuously excite the basis (e.g., a quadratic field poorly excites higher than 2nd order polynomial terms). Finally, we must be diligent to avoid computing accuracy with the same numerical routines being measured.

The traditional measure of accuracy for CFD codes is the L_2 norm of a residual $r(x,y) = f(x,y) - \phi(x,y)a$ defined as

$$L_2 = \sqrt{\int r(x,y)\, dA}$$

where $f(x,y)$ is the desired field and ϕa is the finite element approximation. This allows for a comparison of not just the nodal values but the entire solution field. Again, for further details regarding the mathematics of L_2 norms in CFD formulation, refer to Johnson[34].

The first task is to calculate the solution coefficients for a given field. Perhaps the easiest method is to apply the Galerkin method to the desired solution field.

$$Ma = \int \phi_i f(x,y)\, dA$$

This is iterated to floating point accuracy (approximately 15 digits of accuracy). Fundamentally, this process is identical to an actual CFD solver with the exception of a drastically simplified governing equation.

It is *not* appropriate to use the CFD solver's integration routines to compute the L_2 error. Those integration routines were already used to compute the nodal coefficients and field values. Additionally, this integral needs an error bound that the solver's routines are just not capable of providing.

Monte-Carlo integration was selected as a completely independent numerical integration routine. This integration method[63] uses randomly selection locations for valuation divided by the total

number of sampled locations N

$$\int f \, dA \approx \frac{A}{N} \sum f$$

For this application, the number of samples is increased until the integral converges within a specified tolerance. The tradeoff is that the standard deviation of errors only drops as $\sqrt{N}$.

After some experimentation, an exponential function field was selected for the smooth field.

$$u = e^{-x^2-y^2}$$

From a Taylor series view, this field exercises all polynomial terms; this was a particular problem with using a sine or cosine fields. For the non-smooth (shock) field, a unit step circle was selected.

For each basis order from 1 to 9, the L_2 accuracy was computed for a range of grid spacings. Figure 4.10 shows the L_2 data. As theory suggests, the curves are linear in a log-log plot. The

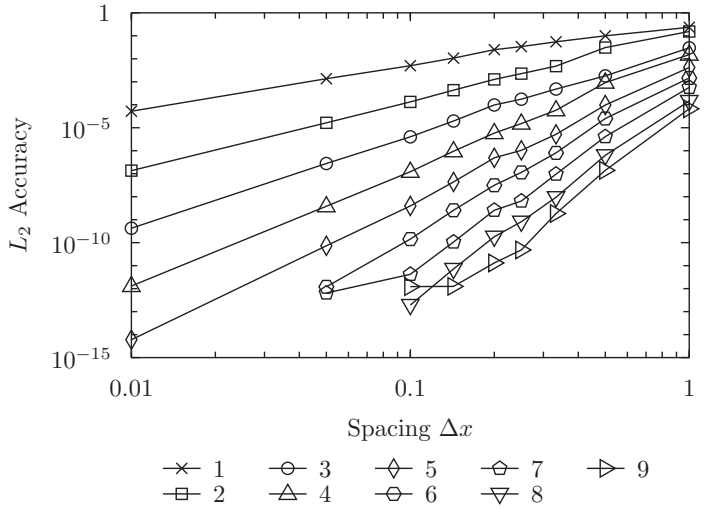


Figure 4.10: Grid L_2 Accuracy: Smooth

minimum odd basis order accuracy is 10^{-12}. Even order basis functions are exactly represented in the numerical operations and minimums extend to lower error values. Interestingly, the basis order lines converge at a point when the grid spacing is $2\sqrt{2} \approx 2.5$. This is the diagonal grid distance.

For the shock field, Figure 4.11 plots the accuracy versus grid spacing for order 1 through 9. The curves are essentially identical. This is expected from convergence theory.

L_2 errors along specific basis orders are converted into a model using theory. The error is expected to be proportional to a power of grid spacing.

$$L_2 = a \, (\Delta x)^b$$

A line fit routine was used to compute the best fit coefficients with the previous power law. Table 4.1 gives the best fit equations. This confirms the suggestion by theory that the power coefficient

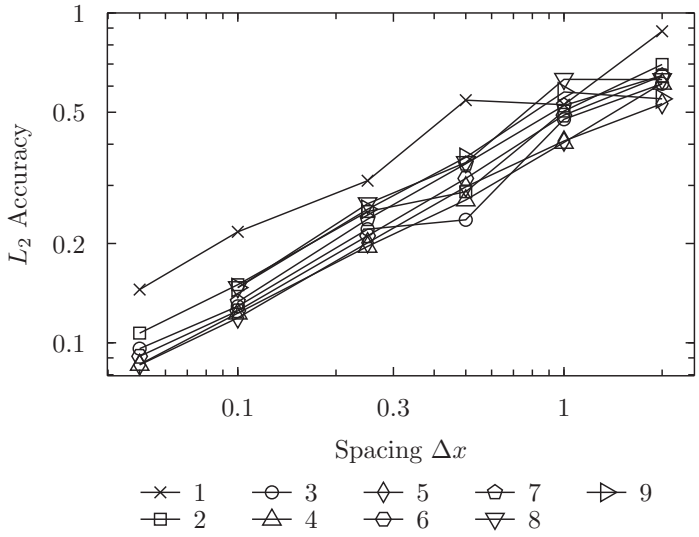


Figure 4.11: Grid L_2 Accuracy: Shock

is approximately the basis order plus 1. Figure 4.12 plots these curves for a visual representation of the model. It should be noted that the model does not automatically contain any convergence

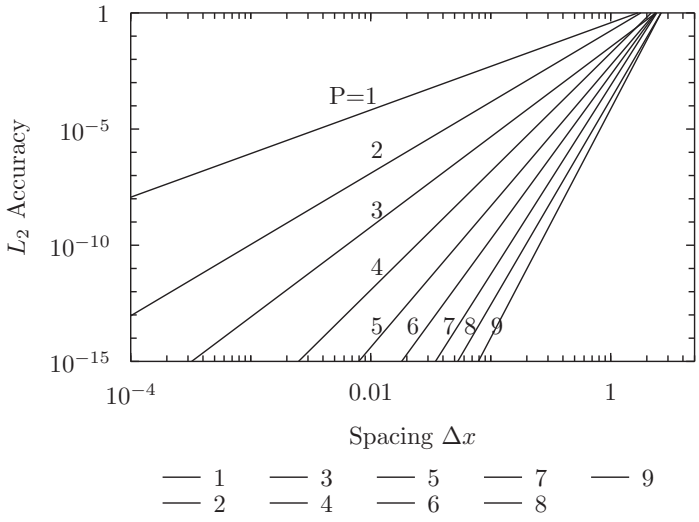


Figure 4.12: Grid L_2 Accuracy Model: Smooth

round-off plateau. For the shock field, the model is

$$L_2 = 0.62\,(\Delta x)^{0.47}$$

Order	Best Fit
1	$L_2 = 3.7 \cdot 10^{-1} (\Delta x)^{1.9}$
2	$L_2 = 1.7 \cdot 10^{-1} (\Delta x)^{3.1}$
3	$L_2 = 3.4 \cdot 10^{-2} (\Delta x)^{3.9}$
4	$L_2 = 1.9 \cdot 10^{-2} (\Delta x)^{5.1}$
5	$L_2 = 5.6 \cdot 10^{-3} (\Delta x)^{6.1}$
6	$L_2 = 2.2 \cdot 10^{-3} (\Delta x)^{7.1}$
7	$L_2 = 7.6 \cdot 10^{-4} (\Delta x)^{8.1}$
8	$L_2 = 1.9 \cdot 10^{-4} (\Delta x)^{8.9}$
9	$L_2 = 7.3 \cdot 10^{-5} (\Delta x)^{9.8}$

Table 4.1: Grid L_2 Model: Smooth

4.3.3 Galerkin Integration

Galerkin methods require integration of the force (i.e., non-temporal terms). The force vector results from integration of the basis derivative and the governing equation's flux term and a boundary term.

$$R_{ij} = -\int \frac{d\phi_j}{dx_i} F_i \left|J\right| \, dA + \int \phi F \, dL$$

For the purpose of this experiment, the lower dimension boundary terms are not considered; the interior term dominates the calculation time by at least a factor of P, basis order. From the governing equations, Navier-Stokes fluxes require 1 multiple of basis order for the density $F = \rho u$ to 3 multiples of basis order for the energy $F = \rho u e$. Worse still, these multiples of basis order are non-rational. For instance, calculating pressure from the states and the ideal gas approximation requires the following conversion of states

$$\begin{aligned} P_{ig} &= (\gamma - 1)\left(\rho e - \frac{1}{2}\rho u^2 - \frac{1}{2}\rho v^2\right) \\ &= (\gamma - 1)\left(\rho e - \frac{1}{2}\frac{(\rho u)^2}{\rho} - \frac{1}{2}\frac{(\rho v)^2}{\rho}\right) \end{aligned}$$

Realistically, the convergence of numerical integration is no longer independent of the density values. Boyd[10] covers this issue with his "Witch-of-Agnesi" rule of thumb. Mitigation naturally occurs with increased grid resolution in high-gradient flow regions imposing only small changes in density across an individual element. As the stated goal of higher order basis functions is to reduce the number of elements, this directly conflicts with the reduced accuracy of numerical integration during large density changes. Thus, we expect the numerical *over-integration* requirement to increase as

order increases during the same time as increasing the order also increases the number of numerical integration points required. A lack of availability of very high order Gauss numerical integration weights and points relegated this project to the non-optimal scheme described in the methodology chapter. The mathematical operations required a large code base. For a comparison of lines of code and compilation time refer to Figure 4.13. Compiling the tenth order operations file required nearly an hour. This was an unexpected complication that earlier prevented use of basis orders greater than about 6 or 7. In short, the integration method for an N coefficient triangle requires N integration

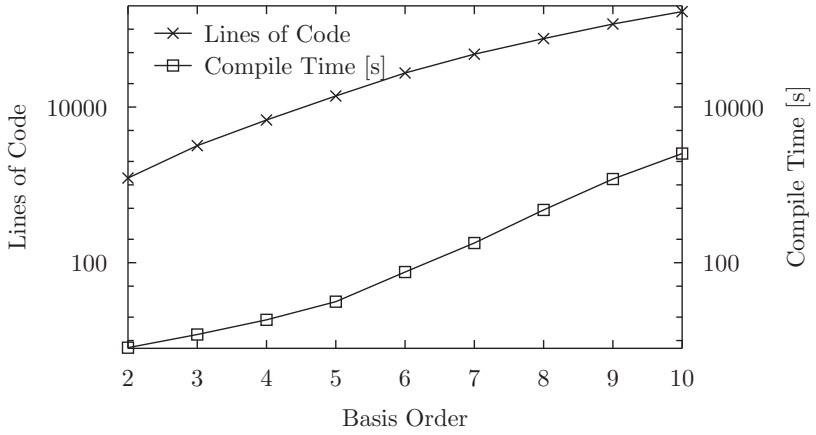


Figure 4.13: Mathematical Operations Size and Compile Time

points.

From counting the required order for effective integration, a single analytical model of integration complexity versus basis order is constructed. The energy equation requires at least 3 multiples of basis order P. Forming the Galerkin forces requires another $P-1$ orders; the derivative of order P gives order $P-1$. This is approximately a total order of $4P$. In two dimensions, each triangle of order P has $(P+1)(P+2)/2$ points. So the number of integration points for the energy equation is

$$\begin{aligned} N &= \frac{((4P)+1)((4P)+2)}{2} \\ &= 8P^2+6P+1 \end{aligned}$$

When performing this estimate over all 4 governing equations, the implementer must choose to either vastly over-integrate the density and momentum equations with the points from energy or recalculate values of state for each governing equation. Regardless, obtaining the values at these N points also contributes to the total work. For the first choice, using the same values, an estimate of required operations to calculate the values is $8N \times P$. The constant is 4 for calculating values and

		Time [s] for Order P								
Grid	Ele	2	3	4	5	6	7	8	9	10
Tiny Box	32	0.056	0.136	0.276	0.608	1.304	3.092	7.54	14.445	28.76
Cavity	2693	2.72	7.32	38.5	62.8	124	269.6	631.6	1194.9	2355.2
Cylinder	56314	167	405	738	1360.5	2501.4	5561.5	12930.8	24894	49397
Huge Cavity	82984	268	461	1098	1869	3704	8203	19156	36163	MEMORY

Table 4.2: Galerkin Force Integration Timing Raw Values

4 for the integration. This would indicate a work complexity of

$$W_{pts} \approx 64P^3 + 48P^2 + 8P$$

Additionally, there is a force contribution from each basis function represented by the j index in the Galerkin equation given above. Remembering that there are N basis functions gives another

$$\begin{aligned} N_\phi &= \frac{(P+1)(P+2)}{2} \\ &= \frac{1}{2}P^2 + \frac{3}{2}P + 1 \end{aligned}$$

operations across the entire number of integration points. So the work complexity is now

$$\begin{aligned} W_{total} &= W_{pts} N_\phi \\ &\approx 32P^5 \end{aligned}$$

There are strategies for reducing the dominating powers of order. This will be discussed elsewhere.

The task is to experimentally obtain a work complexity via timing the actual CFD code. Four cases are considered: a rectangular box, a 1:1 cavity, a cross-flow cylinder, and a huge over-resolved 1:1 cavity. Table 4.2 shows the raw timings in seconds over a range of P. The tiny box case timing ranges from 0.056 seconds for second order to 28 seconds for tenth order. The larger cases required tens of thousands of seconds for the orders above 8. The huge cavity case failed for $P = 10$ by exceeding the computer's 2GB of memory. Analysis is complicated by the reality that the number of elements for a given grid is fixed while the resolved accuracy increases as order increases. We truly want the integration time expressed per element. Luckily, the time on a per-element basis is somewhat consistent across different cases, so dividing the time by the number of elements gives Table 4.3 and Figures 4.14 and 4.15 for a log-lin and log-log plot. A function model for T_{integ} is not used; rather, the average per-element data will be used directly.

		Time [s] for Order P								
Grid	Ele	2	3	4	5	6	7	8	9	10
Tiny Box	32	1.7×10^{-3}	4.2×10^{-3}	8.6×10^{-3}	1.9×10^{-2}	4.1×10^{-2}	9.7×10^{-2}	2.4×10^{-1}	4.5×10^{-1}	9.0×10^{-1}
Cavity	2693	1.0×10^{-3}	2.7×10^{-3}	1.4×10^{-2}	2.3×10^{-2}	4.6×10^{-2}	1.0×10^{-1}	2.4×10^{-1}	4.4×10^{-1}	8.8×10^{-1}
Cylinder	56314	3.0×10^{-3}	7.2×10^{-3}	1.3×10^{-2}	2.4×10^{-2}	4.4×10^{-2}	9.8×10^{-2}	2.3×10^{-1}	4.4×10^{-1}	8.8×10^{-1}
Huge Cavity	82984	2.4×10^{-3}	5.5×10^{-3}	1.3×10^{-2}	2.3×10^{-2}	4.5×10^{-2}	9.9×10^{-2}	2.3×10^{-1}	4.4×10^{-1}	
	Average	2.2×10^{-3}	4.9×10^{-3}	1.2×10^{-2}	2.2×10^{-2}	4.4×10^{-2}	9.8×10^{-2}	2.3×10^{-1}	4.4×10^{-1}	8.8×10^{-1}

Table 4.3: Galerkin Force Integration Timing Per Element

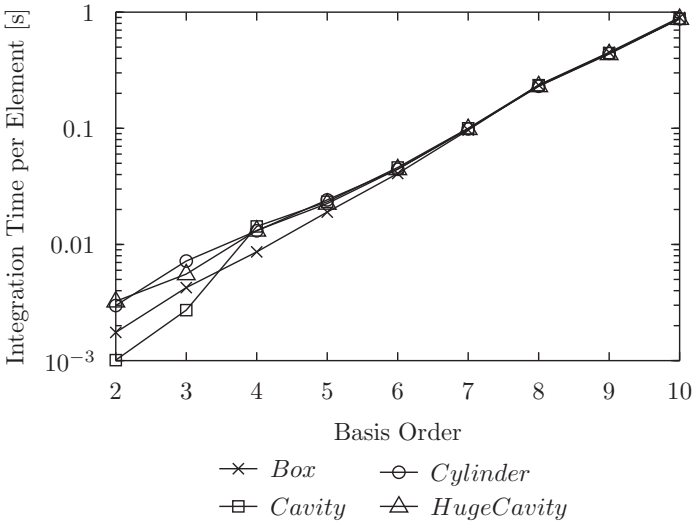


Figure 4.14: Galerkin Force Timing per Element (Semilog)

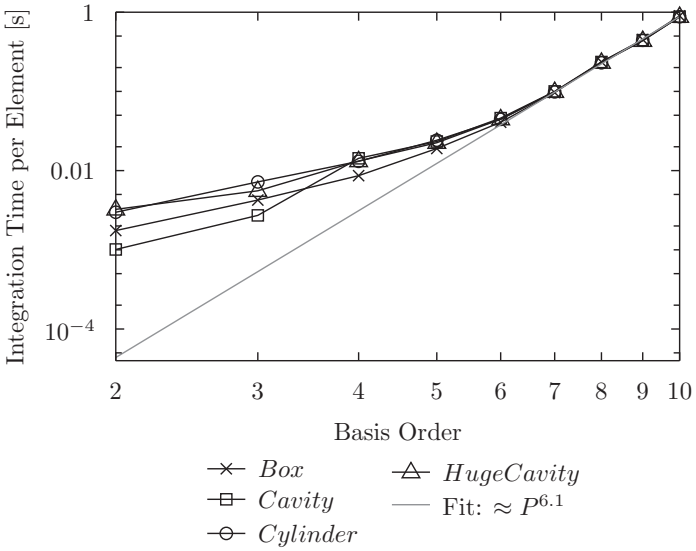


Figure 4.15: Galerkin Force Timing per Element (Log-Log)

Unfortunately, the average per-element time appears to approximately double for each increment in order which implies an unforeseen and undesirable exponential process. So what explains this result and the poor estimate? The basis order range is fairly small, so that the observed exponential process is likely a polynomial process with decreasing overhead costs. The overhead likely results from the non-contiguous memory access inherent in finite element nodal numbering; a higher order element contains relatively more contiguous or near contiguous memory locations especially for interior nodes. This overhead principle is conceptually an important advantage altogether dominated by the P^6 asymptotic behavior. Regardless, the integration time result reflects reality.

This timing order is perhaps the most significant issue hindering the implementation of higher order CFD. Techniques[47, 35, 10] are available that reduce the work from P^6 to P^4 with a modification of the basis function. The effect is investigated in Section 4.3.7.

4.3.4 Mass Matrix Iteration

The Galerkin method creates the Galerkin mass matrix M from the temporal terms

$$M\frac{da}{dt} = \int \phi\phi \, |J| \, dA \frac{da}{dt}$$

such that

$$M\frac{da}{dt} = F$$

Previously, the Galerkin force matrix timing was considered. Now, determining the temporal coefficient vector da/dt is considered.

As previously discussed in the methodology section, direct inversion of M is not feasible for even simple cases. The primary concern is how much work or time is required to determine the left hand side vector. Three grids were used for computing the iteration time versus basis order curves: a 39670 element rectangular *box*, a 56314 element cross flow *cylinder,* and a 82984 element *cavity.* The iterative solve used is a Jacobi lumped mass iterative scheme.

$$x_{i+1} = x_i + \alpha_i r_i$$

Several iterative methods were compared; the Jacobi was selected based on performance and timing. Refer to 3.9.2 for more information on iterative methods. Additionally, the Jacobi iterative scheme is used in the laboratory's current Euler2d and Euler3d CFD solvers. The L_2 error is computed as the root mean square of the Jacobi residual vector r. Figures 4.16, 4.17, and 4.18 plot the iteration time versus basis order with lines of constant L_2 inversion error. The iterative convergence is characterized by three regions: an initial convergence with $T \approx P^{2.5}$, a slow middle convergence region of an order

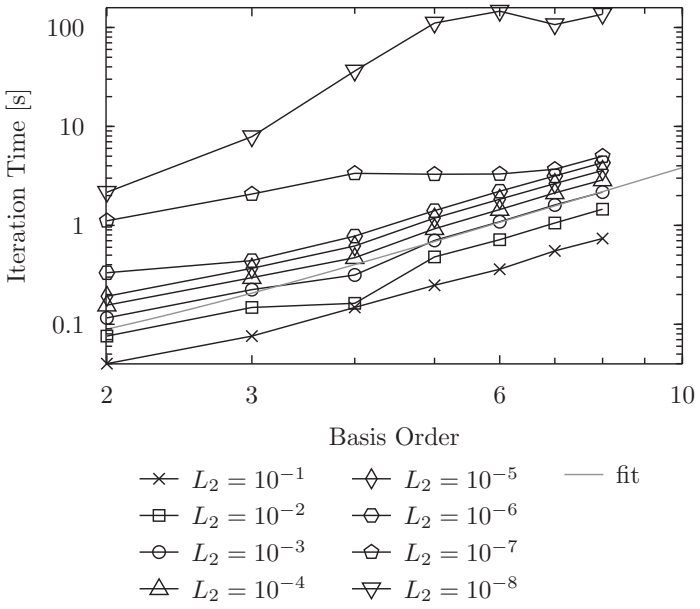


Figure 4.16: Galerkin Mass Timing (Box)

of magnitude of time with $T \approx P^0$, and an upper convergence region with $T \approx P^{4.5}$. The middle region of slow convergence increases in size as the order increases.

Forming a model of iteration time for a given basis order requires selecting a convergence criterion. A time-per-element model based on the convergence bounding the initial and middle convergence regions appears to collapse the time-order curves for the three cases. The convergence criterion is $L_2 = 10^{-4}$. Figure 4.19 plots the selected time-order curves for the selected convergence. A fit of the averages is

$$T_{iter} = 1.4 \times 10^{-7} \cdot P^{3.1}$$

Again, we notice a slight overhead component for the lower orders and an approximate polynomial increase in time with order.

4.3.5 Temporal Integration

Unsteady solutions require temporal integration (i.e., solving a differential equation forward in time). From the methodology section, the are many competing methods for performing a step in time

$$a_{t+\Delta t} = a_t + \Delta t \frac{da}{dt}$$

Regardless of the scheme, the CFL condition applies

$$CFL \equiv \frac{u\Delta t}{\Delta x} < 1$$

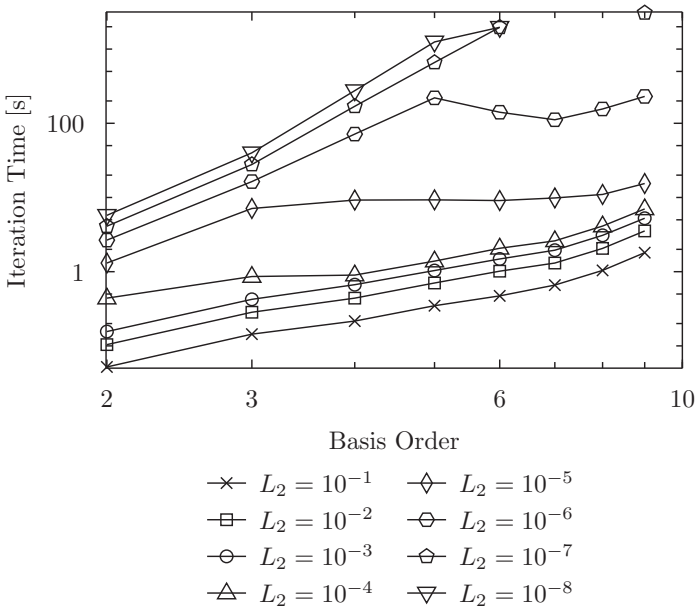


Figure 4.17: Galerkin Mass Timing (Cylinder)

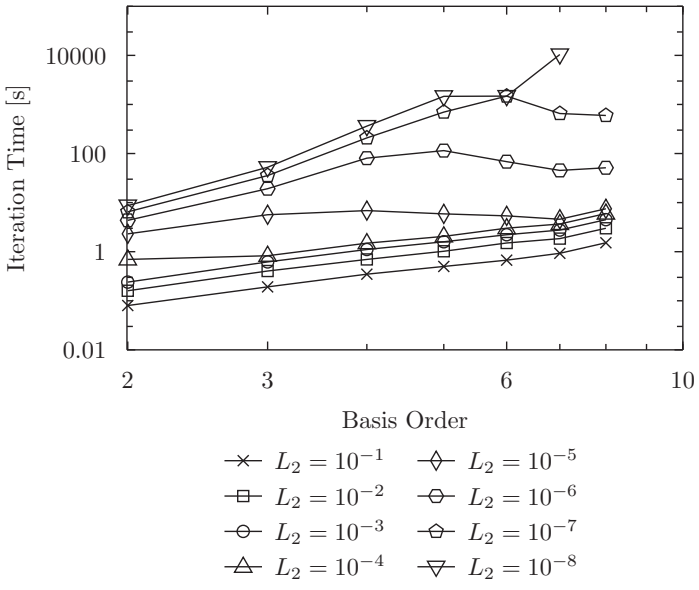


Figure 4.18: Galerkin Mass Timing (Huge Cavity)

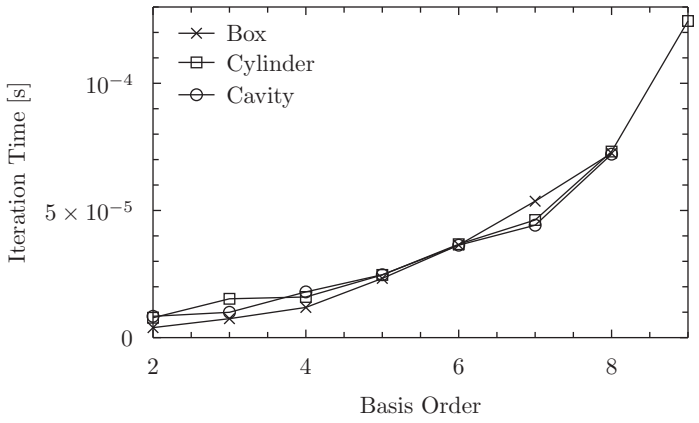


Figure 4.19: Galerkin Mass Timing Convergence

The CFL condition ensures that advected information is properly propagated. In general, the constant c depends on the numerical scheme, the governing equations and for the Galerkin method, properties of the mass matrix. Karniadakis[35] indicates that the maximum timestep for higher order advection equations is proportional to order squared

$$P^2 \frac{u\Delta t}{\Delta x} < 1$$

This restricts the timestep to

$$\Delta t < \frac{\Delta x}{uP^2}$$

The number of steps to advance to a specified time increases with a decrease in timestep

$$\begin{aligned} N_{dt} &= \frac{T_{sim}}{\Delta t} \\ &= \frac{T_{sim} u P^2}{\Delta x} \end{aligned}$$

4.3.6 Residual Minimization

A critical operation is minimizing the Galerkin residual. This is mathematically minimizing

$$R = B - Ax$$

where B contains the Galerkin forces and Ax contains the temporal terms. Computing both of these terms requires all of the operations timed so far.

Ideally, we want each iteration to reduce the residual of the final result and not a fixed point of the converging solution vector. Changing the iteration controller is critical to the final timings. Even better, there is no need to precisely compute the solution to intermediate convergence of right hand side. Bonhaus[9] states

An unexpected observation of particular importance can be seen in the convergence histories presented in figure 22. Note that contrary to conventional wisdom, the nonlinear system converges more quickly as the accuracy of the scheme is increased. Not reflected in the figure, however, is the fact that the linear system does become more difficult to solve.

4.3.7 Total Timing

The total simulation time is composed of grid, Galerkin force, mass matrix inversion, and temporal integration. The above sections composed experimental and theoretical models for time versus basis order. Now, the data is combined for a total simulation cost.

The input variable is basis order. The output variable is time. Two free variables exist: solution error and solution field continuity (i.e., presence of shocks). The total time is the summation of grid time, integration time, and iteration time for a number of timesteps.

$$T = N_{dt} \left(T_{grid} + T_{iter} + T_{integ} \right)$$

For a steady or a non-moving boundary unsteady simulation, the total time is

$$T = T_{grid} + N_{dt} \left(T_{iter} + T_{integ} \right)$$

The solution error free variable is fixed to five values: 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5} and 10^{-6}. We will soon see that this range is sufficient to describe the quality of all simulations performed in the Caselab. The second free variable, solution continuity, is described as either smooth or fully discontinuous (i.e., shock).

Unsteady Smooth

Unsteady smooth simulations refer to constantly regridding domains with a shock-free solution field. Figure 4.20 plots total time versus basis order for an unsteady simulation. For a linear solver (i.e., $P = 1$), the solution times are approximately 200 seconds for $L_2 = 10^{-2}$, 3 days for $L_2 = 10^{-4}$, and a million years for $L_2 = 10^{-6}$. In these terms, the Caselab currently performs most of our linear unsteady simulations in the days to weeks timeframe. The curves are also labeled in terms of *current* solution quality: qualitative, research, paper, benchmark, and galactic.

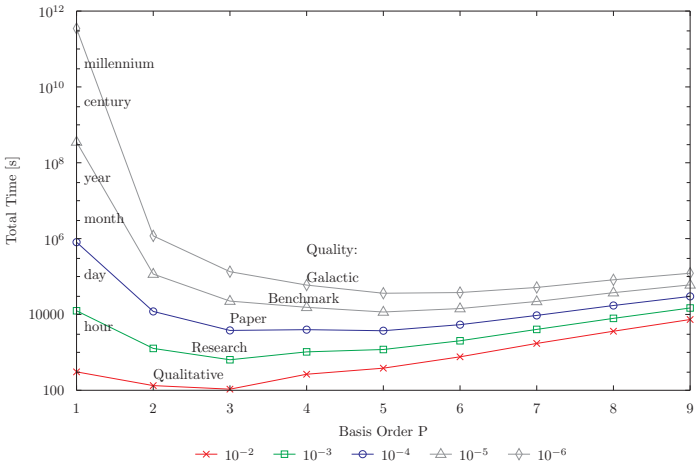


Figure 4.20: Total Time versus Order: Unsteady Smooth

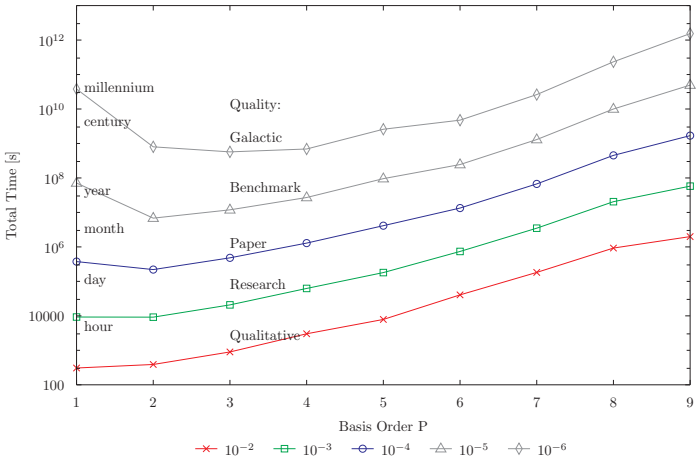


Figure 4.21: Total Time versus Order: Unsteady Shock

Unsteady Shock

Unsteady shock refers to moving boundary and transient simulations with the presence of shocks. Figure 4.21 plots total time versus basis order for the same set of L_2 errors given above. The behavior is primarily characterized by an increase in time as basis order increases. As solution accuracy improves (i.e., smaller L_2 error), the linear solver's marginal efficiency decreases.

Steady Smooth

Steady smooth refers to CFD solutions of time-averaged and smooth fields. Figure 4.22 plots total time versus basis order. The behavior is dominated by a decrease in time when basis order increases.

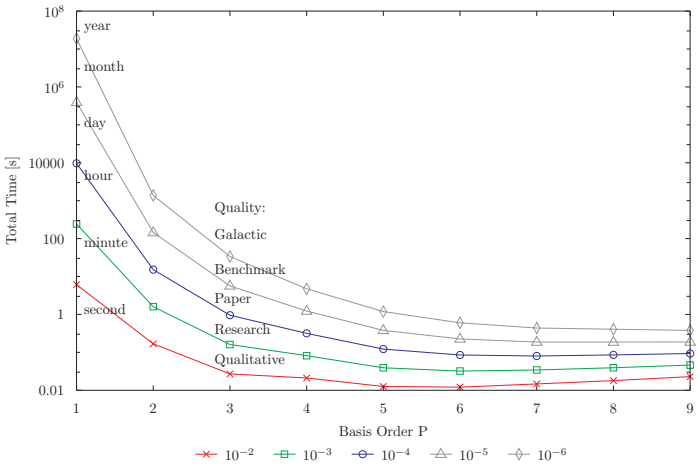


Figure 4.22: Total Time versus Order: Steady Smooth

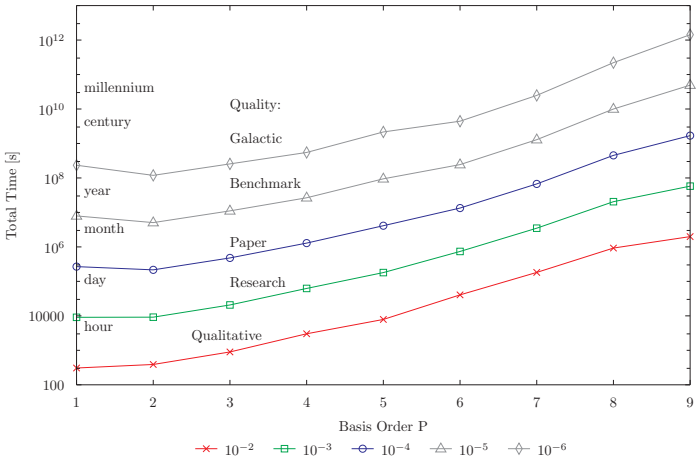


Figure 4.23: Total Time versus Order: Steady Shock

Steady Shock

The steady shock concerns a time-averaged solution with the presence of shocks. Figure 4.23 plots time versus order. The behavior is again characterized by an increase in time with an increase in order.

Advanced Techniques

The literature suggests applying advanced techniques for improved higher order efficiency. Sum-factorization[10, 35] and similar techniques[47] are suggested. From theory and experience these techniques reduce the integration costs by P^2 in 2D. Figure 4.25 compares time versus order for the original technique and this advanced technique. As expected, the marginal change becomes larger as order increases. However, the optimal order for our typical research quality $L_2 = 10^{-4}$ solutions

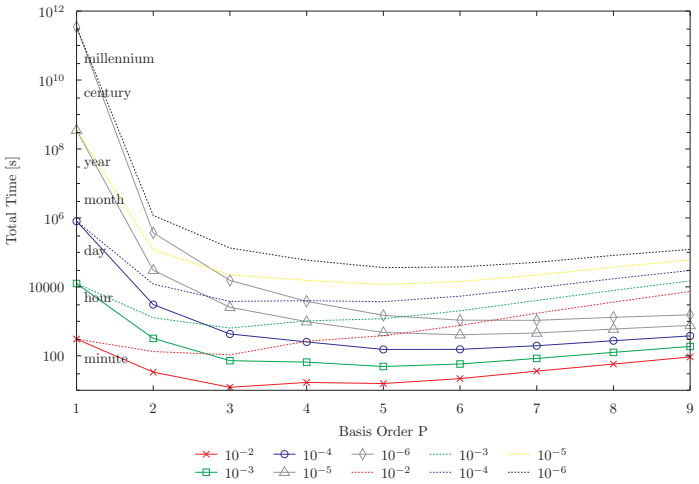


Figure 4.24: Total Time versus Order: Unsteady Smooth Advanced

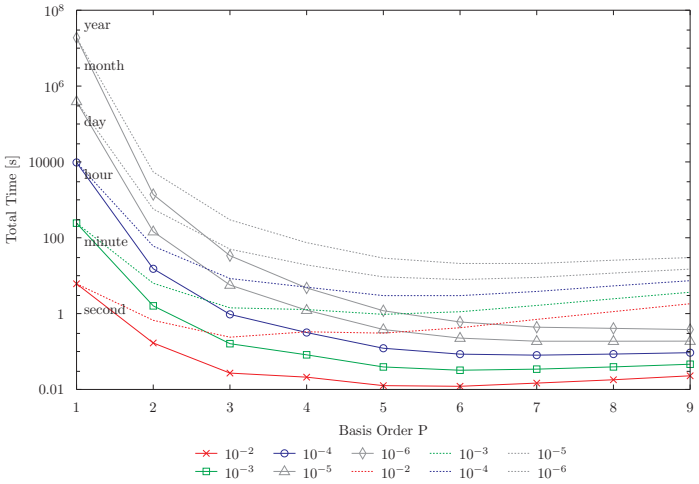


Figure 4.25: Total Time versus Order: Steady Smooth Advanced

is not significantly improved.

For the steady smooth case, the sum-factorization methods reduce the integration cost by supposedly 2 basis orders. Figure 4.25 shows the time versus order when assuming these advanced methods are used. In reality, these methods are not likely to provide as much benefit as expected since the elemental Jacobian is no longer constant. Non-constant Jacobians require at least an extra order for the integration cost.

4.4 Experimental Solution Timing

The previous experiment using a build-up approach is convenient for conceptual understanding, but it begs a critical question. That question is considered in this section. Wall clock timing of the full

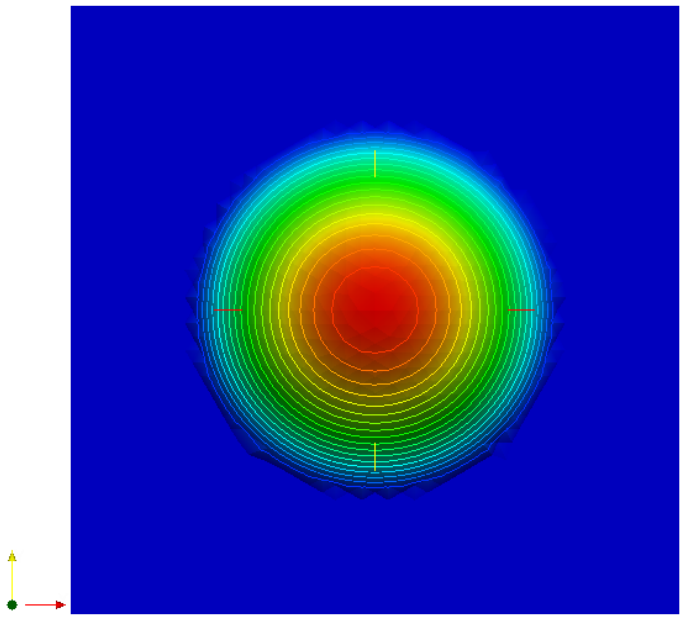


Figure 4.26: Experimental Timing: Velocity Initial Condition

CFD solver running an unsteady problem for various basis orders is performed.

Immediately, there is a snag. Most unsteady problems have no analytical solution. Furthermore, the test case must not require months, as there is a tremendous number of simulations required. Another constraint is that the test case must allow a wide range of grid spacings (i.e., integer number of elements; non-integer spacings do not exist.) while fitting within the *surface* grid generator's restrictions. The chosen case is a square domain with a velocity "puff" in the center. The four walls have the no-slip velocity condition applied. Figure 4.26 shows the initial conditions for velocity magnitude. After $\Delta t^\star = 0.1$, the velocity vectors are shown in Figure 4.27. Two vortices are generated off to either side of the puff. Most importantly, a right-traveling velocity front is generated via advection.

The Ale2d code was used exclusively for this test. The grid generation time is not considered.

The first task is to generate grid spacings corresponding to desired levels of accuracy. Recalling the previous experiment, error of 10^{-2}, 10^{-4} and 10^{-6} are suggested. Testing revealed that 10^{-2} poorly fit the grid constraints previously discussed; rather, 10^{-3} was substituted as the *qualitative* accuracy level. For each basis, a timestep study tested total solution time and convergence stability. Only stable timesteps are plotted. Total times and timesteps are mapped to a log-log plot.

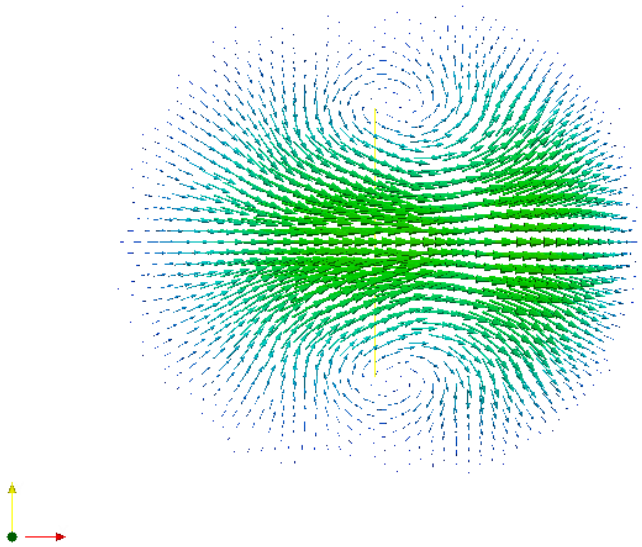


Figure 4.27: Experimental Timing: Velocity Vectors at $t^\star = 0.1$

Figure 4.28 gives the 10^{-3} accuracy curves. The linear basis (P=1) has a maximum timestep of 0.008 for a total time of 0.588 seconds. The minimum time for the solution occurs at the largest stable timestep. Second order (P=2) has a minimum timestep of 0.01 with a total time of 0.06 seconds. The minimum time occurs at a timestep smaller than the stability boundary. Interestingly, the trend continues; higher order solutions have a minimum total time within the stable timestep range. The global minimum solution time occurs for 3rd order solutions. Increasing the order higher than 3rd order requires more time for the same accuracy.

Figure 4.29 gives the 10^{-4} accuracy curves (i.e., research to paper quality). These exhibit a similar behavior as seen in lower accuracy experiment. Again, the global minimum time is for the 3rd order solution.

Finally, Figure 4.29 gives the 10^{-6} accuracy curves (i.e., paper to galactic quality). This exhibits behavior not seen in the lower accuracy solutions. The 1st order solution failed from a lack of sufficient memory (i.e., required more than 6GB of virtual memory with a 7 million element grid). Again, 3rd order is fastest. The behavior differs because of the poor convergence of the Jacobi iteration routine (cf. Fig. 4.16). For high accuracy solutions, this convergence issue must be investigated and eliminated.

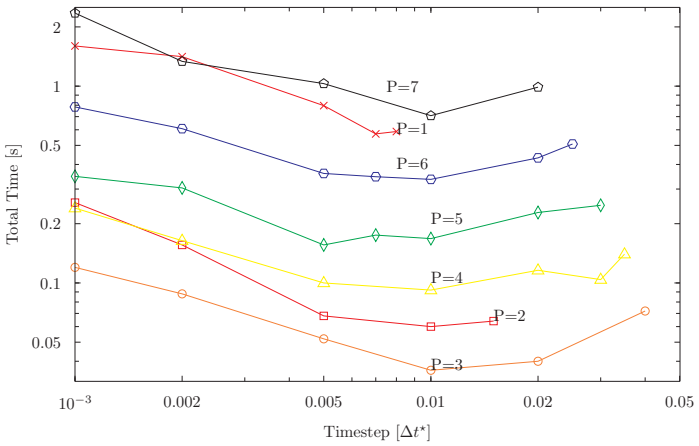


Figure 4.28: Experimental Timing: Total Time versus Order 10^{-3}

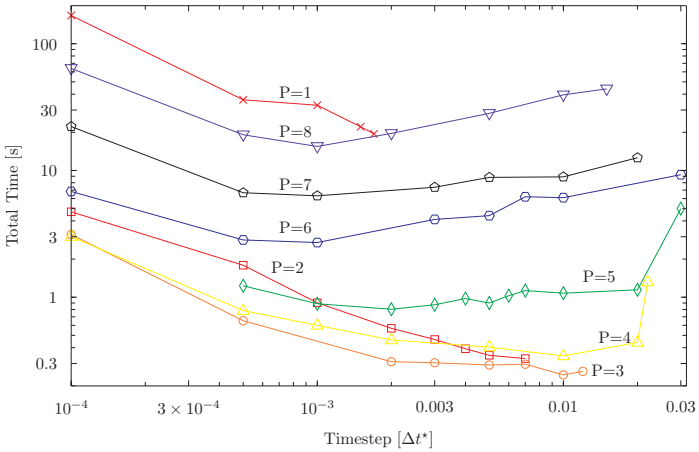


Figure 4.29: Experimental Timing: Total Time versus Order 10^{-4}

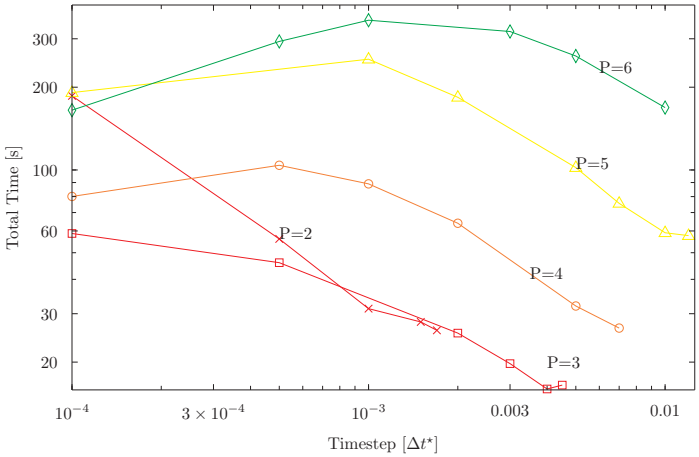


Figure 4.30: Experimental Timing: Total Time versus Order 10^{-6}

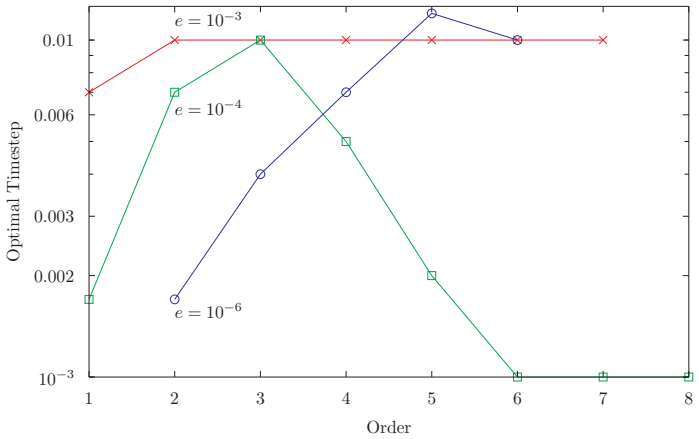


Figure 4.31: Experimental Timing: Optimal Timestep versus Order

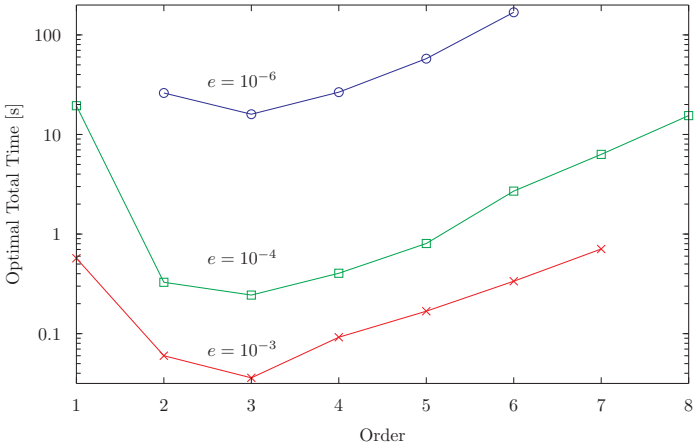


Figure 4.32: Experimental Timing: Total Time versus Order

Optimal solutions exist at the minimum time for a given accuracy. Figure 4.31 shows the optimal timestep for the tested basis orders. Unlike the previous experiment's model, the actual timestep appears to have a maximum value somewhat independent (for the tested range) of the basis order at 0.01; the previously expected power law of basis order appears not capture the relevant dynamics. More instructive is the optimal total time versus order (Fig. 4.32). This figure strongly resembles the model based figure (Fig. 4.20) given in the previous experiment. For the two lowest accuracy levels, the marginal utility of moving from a 1st order to 2nd order solver is at least an order of magnitude in time. The marginal utility of moving from a 2nd order to 3rd order solver is still significant but less than one order of magnitude. Finally, successively moving to orders higher than 3rd each results in a negative marginal utility.

From these experiments, the 3rd order solver appears best for unsteady smooth solutions. That said, several caveats should be discussed. First, solutions with shocks are expected to reduce the

local accuracy to 1st order regardless of the solver's accuracy. In fields with a significant number of shocks, both linear and higher order solutions approach the actual solution at the same convergence rate. Second, the above experiments illustrate the spectacular convergence rate of higher order methods. There is a downside; the solution sensitivity to grid size become more pronounced as order increases. The relatively wide range of grid sizes allowed in the 1st order solver is *not* allowed in the 8th order solver. Effectively this causes solution failure when the grid size is vastly too large (i.e., under-resolved field causing unstable convergence) or vastly too small (i.e., tiny timesteps trying human patience). The present experiment assumed a perfect grid for the solution field at the desired accuracy level. The experiment used an unsteady field with a decreasing maximum velocity. Aerodynamics of vehicles usually involves a pressure differential that requires a velocity increase. Effectively, expect grid remeshing for all but the most simple testcases.

Timing experiments presented above found a previously unknown behavior of higher order unsteady CFD solvers. The marginal utility of increasing order found in these experiments is only positive at and below 3rd order. Increasing order beyond 3rd order is not productive. The objective of this part is to briefly explain these results from a conceptual standpoint.

Figure 4.33 presents timing curves for the various components of an unsteady CFD simulation. For low orders, the grid and element operational cost dominate the time. For high orders, the Galerkin integration costs dominate the time. Larger elements allow a larger CFL based timestep. Not being time accurate, steady simulations move information according to the element size. In a way, the steady higher order solver behaves similar to a multigrid solver. Also, for low orders, the grid time is a larger percentage of the time in steady.

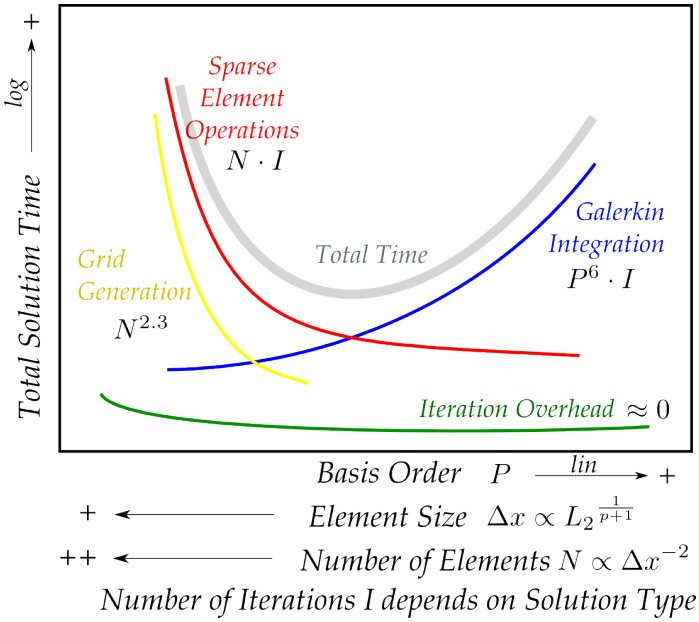


Figure 4.33: Conceptual Time vs. Order

Chapter 5

Higher Order Challenges

This chapter lists challenges that must be solved for development of a successful higher order CFD. These challenges are based on experiences from this project and are ordered from most critical to least critical.

5.1 Solution Stabilization

The most critical challenge to higher order solvers is solution stabilization. Of particular concern is the synthesis of a stabilization scheme that preserves the high order solver's inherent accuracy. A poor stabilization scheme renders the solver unstable and unusable.

Traditionally, stabilization in a linear solver is achieved with artificial dissipation. Many stabilization schemes are available from the CFD literature. Most are ad-hoc formulations[35, 39, 10] based on smoothing and variation minimization rather than physical governing equations.

For this project, dissipation was generated with an entropy measuring scheme. Unfortunately, the dissipation was introduced through the Reynolds number and stresses and was not sufficient for solution stabilization. Additionally, it had difficulties because of low quality derivative calculations. Further work would use a physically consistent transfer from mechanical energy to internal energy.

The stabilization challenge for higher order solvers is significant and necessary. Qualitatively, more degrees of freedom in higher order solvers will appear to only give more modes of divergence. In short, no dissipation means entropy violation. Entropy violation means diverging and unphysical solutions.* *This challenge must be solved first.*

5.2 Numerical Efficiency

The second critical challenge concerns numerical efficiency and timing. The challenge is to select a scheme that minimizes the time required to achieve a desired accuracy. As might be expected, this

*The magnitude and general behavior of the unphysical results are not quantifiable in a strict sense. The numerical method and non-linear functions of states will determine the solution. "How much" is determined through the magnitude of entropy destruction.

challenge is composed of a multitude of partial challenges. Regardless, the final tally involves only maximizing accuracy per time.

The limiting rate for a Galerkin CFD solver is calculating the force vector. The analysis section showed that this calculation depends on order to the 6th power. This is expensive. Previous challengers conditionally reduced this complexity to order to the 4th power or unconditionally to the 5th power. The challenge is to incorporate and show these savings in an actual code.

Maintaining numerical accuracy is a challenge. This project used double precision floating point numbers for all computations (i.e., about 15 digits of accuracy), yet the resulting actual precision is diminished for several reasons. First, floating point operations tend to accumulate errors from purely numerical reasons[31]. Second, the generation of numerical constants was performed in double precision. Generating double precision operations and constants will require more than double precision pre-processing. Although the double precision number stores about 15 digits of accuracy, this project saw an effective convergence limit of 12 digits of accuracy.

It should be noted that linear basis functions have a significant numerical efficiency advantage in that a human can usually reduce mathematical derivations further than a computer's compiler can find the equivalent optimization. As an example, consider the global derivatives computed on a particular element. For the linear basis, the global derivative is

$$\frac{d\phi}{dx} = \frac{1}{2A} \begin{pmatrix} B_{11} \\ B_{12} \\ -B_{11} - B_{12} \end{pmatrix}$$

The equivalent higher order derivative depends on basis functions, locations, local derivatives in both directions, and the element Jacobian. In general, the global derivative is reduced at most to

$$\frac{d\phi}{dx} = \frac{d\zeta_1}{dx}\frac{d\phi}{d\zeta_1} + \frac{d\zeta_2}{dx}\frac{d\phi}{d\zeta_2} + \frac{d\zeta_3}{dx}\frac{d\phi}{d\zeta_3}$$

This project's ALE solver was written completely generically for an arbitrary basis order. That arbitrary nature strongly contributed to the long compile times and large generated code base. It is likely that 2nd or 3rd order is the maximum complexity a normal human can mathematically reduce a CFD derivation. This is a challenge where computerized symbolic mathematical derivations could assist.

It is interesting that complex CFD solvers spend most of their time solving what becomes a linear algebra equation. Traditional Galerkin methods involve a mass matrix, a solutions vector and a force vector. The form is

$$M\frac{da}{dt} = F(a)$$

Admittedly, the force vector is neither constant nor linear with respect to the state vector. This fact enormously complicates the solution of the governing equations. In particular, the expensive force vector must be recomputed at each iteration. Additionally, as order increases, the mass matrix can become nearly indeterminate. Furthermore, adding strict constraints can destroy the symmetric mass matrix. Thus, the use of a Jacobi, Richardson, or CG iterative method is either not strictly residual minimizing or completely unsuitable.

For unsteady simulations, determining a stable yet useful timestep is a significant challenge. A constant timestep works sufficiently for the Euler3d code. Yet, for bursts of unsteady flow, a variable timestep would be a considerable advantage. Worse, higher order solvers appear more sensitive to timestep than the existing linear solver. The disadvantage is that a variable timestep requires a timestep controller. Several methods were tried for this project. None were robust. Time control is a persistent challenge.

5.3 Complexity

A significant challenge to the implementation of higher order solvers is the complexity in numerical routines and especially the boundary conditions.

The higher order solver developed in this project, Ale2D, required generating a significant number of core mathematical operations. The Fortran mathematical operations were vastly too complex and tedious to generate and verify by hand. Additionally, these operations depend and scale on order. The solution was to generate these Fortran source files with a separate Python program. A related complexity involved the difficulty of generating or obtaining Gauss integration points for high accuracy high orders. This challenge is mitigated somewhat by fixing the basis order.

5.4 Finite Element Grid

Another set of challenges involves the generation and convergence of the finite element grid. This finite element grid gives the method incredible computational advantages and disadvantages.

Increasing the basis order by design increases the grid convergence rate. The good news is that fewer grid elements are required, which reduces the grid generation time. The bad news is that the solution is now more sensitive to small changes in grid spacing. In effect, the convergence rate in space also works as the divergence rate in space. To be effective, the gridder would need to automatically regrid *even for steady solutions.*

As dynamic simulations become more prevalent, these significant grid generation and modifica-

tion challenges will continue to arise. Furthermore with ALE, the grid velocity must be specified. There must be a variation from Lagrangian boundary velocity as a grid velocity to zero grid velocity in the freestream.

The challenge to future grid generation is contained in four parts: Robust and Autonomous, Small Deflection with same topology, Boundary Curvature, and Remeshing in the loop. Also interpolation from one grid to another will be critical. Visually, these are shown in Figure 5.1.

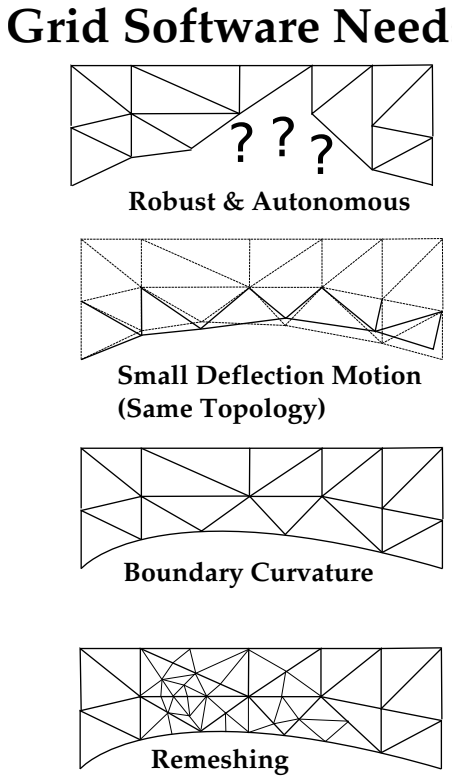


Figure 5.1: Grid Needs

Boundary curvature brings up another point with higher order CFD solvers and grid spacing. The grid must not only be sufficiently refined for the relevant flow field but also be refined for the relevant local geometry. The grid spacing for a linear grid should be at least 5% of the local radius of curvature[11]. For a higher order solver with linear elements, the limitation on grid spacing is in the geometry convergence, not the flow convergence.

5.4.1 Corner Vertex

For the Euler and Navier-Stokes equations, density changes depend on the gradient of momentum.

$$\frac{d\rho}{dt} + \frac{d\rho u}{dx} = 0$$

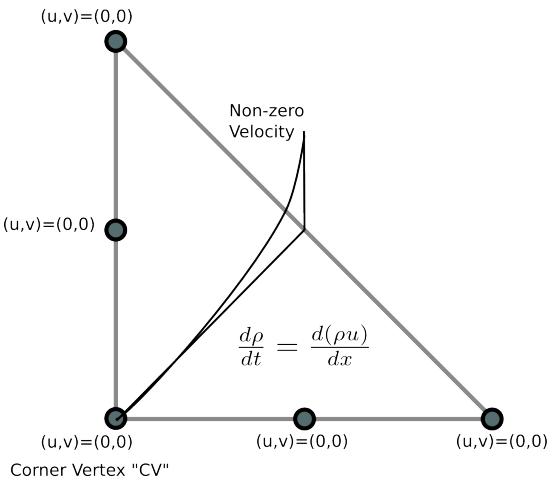


Figure 5.2: Corner Vertex

At a no-slip corner, all edge velocities are zero. The non-zero velocity component decays to zero at the corner vertex. More importantly, the derivative of the velocity also decays to zero at CV. Thus, there is no change in density at the corner vertex for all time. This is fundamentally incorrect and numerically generates spurious solution fields.

There are two remedies. First, ensure every element has only one face with an applied no-slip boundary condition. This allows for gradients at the corner. The second remedy is to apply a basis order greater than 2. This again allows for gradients at the corner.

Unfortunately, this issue was shown to occur with the existing ns2d code. Solving this issue most effectively requires a change in the grid software to prevent corner elements. In 2D, a work-around is to pre-process the grid and split these corner elements. This issue was not found in the literature, but should be a widespread issue with viscous triangle and tetrahedron CFD solvers.

5.4.2 Boundary Conditions

Boundary conditions for a higher order solver are certainly significant challenges. For the best use of a higher order CFD solver, surface elements should contain surface curvature information. Viscous solutions are possible without surface curvature; however, inviscid flow with faceted surface curvature *does not work* because of the dual tangent constraints given at a single node.

Another non-intuitive challenge is correctly applying boundary conditions for sharp edges and for elements straddling on changing boundary conditions. Galerkin Navier-Stokes solutions tend to have difficulties when incompatible boundary conditions are specified at the same points. This could occur when an in/out flow boundary meets a no-slip boundary condition.

Additionally, tracking the boundary conditions is not strictly a per-element issue. Unfortunately,

moving to higher order CFD tends to concentrate operations to element based rather than nodal based. This causes a small challenge with storing which nodes are considered boundary conditions.

5.5 Derivatives

There is a constant challenge to balance accuracy with speed when computing derivatives. In general, higher order basis functions have discontinuous derivatives at element edges. The challenge is to form smooth enough derivatives without requiring a rate limiting computation time. This is especially true when solving the Navier-Stokes equations for computing surface stresses for drag predictions[20].

5.6 Visualization

Precise visualization of higher order fields is a challenge. After spending a significant portion of time learning and then adapting a visualization tool, VTK, to higher-order elements, the visualization tool performed poorly. The tradeoff is visualization speed and accuracy. In the end, this project finally resorted to interpolating all fields to a simple linear visualization at the element nodes. Additionally, unsteady visualization of higher order fields with varying grids is currently (2011) state of the art.

Chapter 6

Lagrangian Methodology

This chapter discusses the development of a Lagrangian frame CFD solver for dynamic simulations. The Lagrangian frame differs from the previous Euler frame by tracking field states at particles rather than fixed locations. As this is quite different and not widely used for CFD, this chapter starts with a rationale and literature review before deriving the governing equations and then presenting results.

6.1 Preliminary Rationale

As previously mentioned, the Lagrangian frame is not commonly used for CFD solvers. This section discusses the preliminary discussion and rationale for this switch as resulting from concerns encountered in the previous higher-order Navier-Stokes project.

The primary reason for considering a Lagrangian frame is the need to simulate boundary motion. A study of the Euler frame indicated that any relative grid motion involved non-physical operations. Of particular concern was the maintenance of boundary layers with large normal velocities. Reducing the timestep to a fraction of the boundary layer height divided by the normal velocity seemed unnecessarily restrictive and unphysical. At this point, the Arbitrary Lagrangian Euler (ALE) form discussed in a previous chapter was introduced. The ALE form requires a smooth transition between the fully Eulerian frame and the fully Lagrangian frame. In essence, this requires two CFD solvers formed into one and requires a grid modification routine to generate that smooth transition. Rather than this route, reducing the solver to only the Lagrangian form shows the fundamental concept with fewer complexities.

A secondary reason concerns the numerical form of the Navier Stokes equations for two reasons. Both result from the Galerkin method applied to the governing equations. First, the Galerkin method requires expensive numerical integration of composed functions as discussed in Section 4.3.3. Although the Galerkin method forms numerical ODEs from previous PDEs and assures a global error bound on conserved variable, as the method degrades, it allows for un-physical valuation of these flow properties. For example, a numerically necessary section of euler3d modifies the Galerkin suggested density to prevent negative values of density. The Lagrangian governing equations appear to create

a loss of accuracy before a loss of stability. Second, the Galerkin numerical operations become global operations as discussed in Section 1.3.1. Given that the creation of any *new* CFD solver should give considerable thought into parallelization, a scheme formed from local operations should be preferred for performance reasons. Since the Lagrangian governing equations track particles, applying the Galerkin method is not necessary since these equations are already in an ODE form coupled with velocity and pressure deriviatives.

A third reason concerns the work required for a given solution as order increases. Lagrangian work is composed of interpolation, computing derivatives, and time advancement. These depend on the total number of coefficients. It is expected that the Lagrangian framework is a P^2 process rather than the P^6 process for the Galerkin method.

A final reason is the simplified nature of the governing equations. The frame tracks the conceptual particle and thus hides advective terms within a separate motion differential equation. This eliminates the traditional CFL stability restriction on timestep seen in the Euler frame. The Eulerian maximum timestep is restricted by velocity and grid spacing. Rather, the Lagrangian timestep is restricted by the maximum velocity divergence. From a preliminary point of view, the Lagrangian frame appears to be dependent on the flow properties and not on grid properties.

For these reasons, further study of the Lagrangian frame was started. Several concerns were found during this preliminary investigation. Two primary concerns are tracking particles and deriving the compressible governing equations.

6.2 Literature Review

Using the Lagrangian frame for CFD is not new. Many of the oldest CFD solvers were Lagrangian based. However, the Euler frame methods are considerably more numerous. However, its cousin, Lattice Boltzman, is a popular method for viscous low Mach flows. Lagrangian methods are however common[10, 40] in meteorology "where the use of a large time-step is essential for efficiency" as Karniadakis[35] states. Meteorology in particular is concerned with advection* of multi-species with complex boundary conditions; Aerodynamics is primarily concerned with single-species compressible flow of more simplified boundary conditions. Pironneau's 1982 paper[61] is seen as a reintroduction to the properties of Lagrangian solvers. A useful reference is Bennett's *Lagrangian Fluid Dynamics*[7].

This project uses a semi-Lagrangian form that resets the particle labels at every timestep. Boyd provides a discussion of the mathematical reasoning for using a semi-Lagrangian form rather than

*Compressible meteorology occurs primarily in altitude; sonic fronts are traditionally ignored.

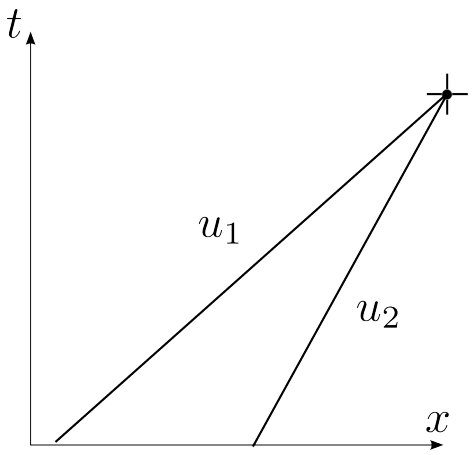


Figure 6.1: Lagrangian Space-Time Intersection

a Lagrangian form. Boyd[10] states the following:

> However, exclusive use of a Lagrangian coordinate system is usually a disaster-wrapped-in-a-catastrophe for numerical schemes because the particle trajectories become chaotic and wildly mixed in a short period of time.... Semi-Lagrangian (SL) algorithms avoid this problem (and earn the modifier "semi") by reinitializing the Lagrangian coordinate system after each time step.

The Lagrangian equations have the following timestep limitation

$$\Delta t \left| \frac{du}{dx} \right| < 1$$

This is visually verified by plotting a space time diagram of two flow particles intersecting when differing by a given velocity and spacing. Flow particles must not intersect. Figure 6.1 shows two intersecting paths with velocity $u_1$and u_2. The time of intersection Δt is when

$$u_1 - u_2 = \frac{\Delta x}{\Delta t}$$

The spacial velocity divergence is

$$\left| \frac{du}{dx} \right| = \frac{u_2 - u_1}{\Delta x}$$

"Hence in semi-Lagrangian models the time step can be chosen for accuracy and not for stability "[40]. The Lagrangian frame timestep depends on the flow field, (i.e., proportional to the flow field's velocity divergence); the Eulerian frame timestep limitation depends on the grid (i.e., proportional to the grid spacing and velocity). For semi-Lagrangian methods, Karniadakis[35] states

> semi-Lagrangian methods require more computational cost than the Eulerian counterpart on a per-time-step consideration.... However, with much larger allowable CFL numbers,

the total CPU time required for the SLSE method to reach a certain time-level is significantly less than that of the Eulerian method.

Bartello found that an atmospheric Lagrangian turbulent Navier Stokes CFD solver required 5 to 10 times more work than the equivalent Euler solver[5].

6.3 Terminology

We need to distinguish between the Eulerian and Lagrangian frames in a systematic manner as described in Bennett[7]. States in the Eulerian frame are given as

$$u[x,t]$$

Lagrangian states are given as

$$u(x,t)$$

When derivatives are involved, the Eulerian derivative is

$$\frac{\partial q}{\partial t}$$

and the Lagrangian derivative is

$$\frac{dq}{dt}$$

These are related by

$$\frac{dq}{dt} = \frac{\partial q}{\partial t} + u\frac{\partial q}{\partial x}$$

For clarity, the difference between the Eulerian and Lagrangian frames must be shown. The substantial derivative or particle derivative is

$$\frac{D\,()}{Dt} = \frac{\partial\,()}{\partial t} + v \cdot \nabla\,()$$

The Euler frame has

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u_i) = 0$$

The Lagrangian frame has

$$\frac{D\rho}{Dt} + \rho\frac{du_i}{dx_i} = 0$$

Substituting the Lagrangian frame equation into the substantial derivative equation gives

$$\frac{\partial\,()}{\partial t} + v \cdot \nabla\,() + \rho\frac{du_i}{dx_i} = 0$$

We notice that there two terms that combine to form

$$\frac{\partial\,()}{\partial t} + \nabla \cdot (\rho u_i) = 0$$

This is identically equal to the Euler frame.

6.4 Governing Equations

Lagrangian governing equations operate in a particle tracking frame, so the temporal and advective terms are different, yet the other terms should appear relatively unchanged. One significant difference is that the Lagrangian equations require an extra particle tracking equation of motion. The fundamental concept is visually described in Figure 6.2. Conceptually, the governing equations

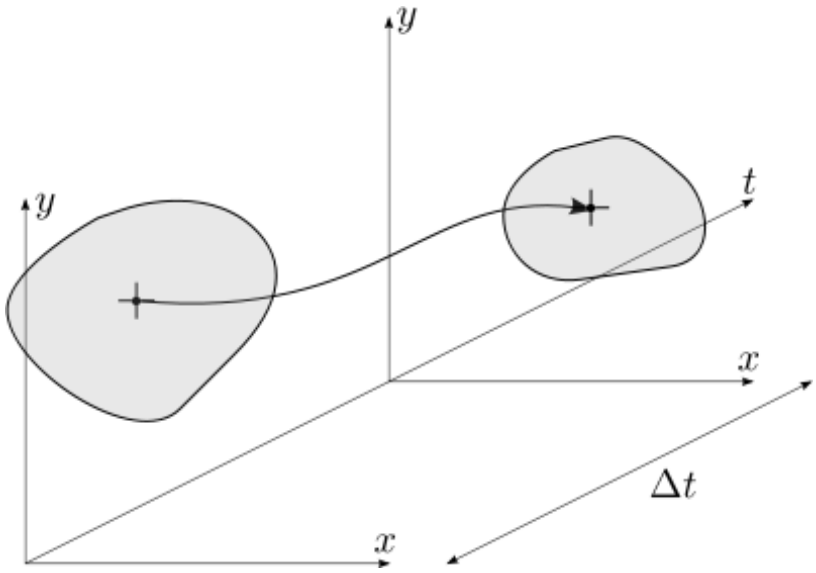


Figure 6.2: Lagrangian Frame Conceptual Particle

track a single control volume being stretched and morphed by velocity gradients. The following derivations are inspired by Bennett[7].

6.4.1 Labeling and Location

A primary concept for any derivation in the Lagrangian frame is the labeling transformation J_s. This term is identical to the term previously seen in the ALE derivation. It relates the spacial transformation for the particle from the initial labeling location and time. Bennett shows that

$$\frac{dJ}{dt} = J\, divU$$

So conceptually, J tracks the volume for a set particle.

Lagrangian governing equations require tracking conceptual particles in time along the velocity vector. The particle equation tracks the particle location with a known flow velocity field.

$$\frac{dx}{dt} = U$$

This appears trivial, but we will later see the complications that arise from particle tracking with known grid locations.

6.4.2 Mass Derivation

For the mass conservation equations, consider a moving particle W so that conservation of mass gives

$$\frac{d}{dt}\int_W \rho dV = 0$$

The volume changes with time but total mass within the volume does not.

$$\frac{d}{dt}\int_W \rho J_s dW = 0$$

As the temporal derivative is independent of the volume, the integration and derivative are independent.

$$\int_W \frac{d}{dt}\left(\rho J_s\right) dW = 0$$

With an arbitrary particle, the governing equation for mass conservation is

$$\frac{d}{dt}\left(\rho J_s\right) = 0$$

We prefer that the governing equation not contain a transformation

$$\rho\frac{d}{dt}\left(J_s\right) + \frac{d\rho}{dt}J_s = 0$$

With the previously derived identity, the mass equation becomes

$$\rho J_s divU + \frac{d\rho}{dt}J_s = 0$$

Assuming that density is finite and non-zero requires that J_s is finite and non-zero. This reduces the previous equation to

$$\frac{d\rho}{dt} = -\rho divU$$

This is the implementable form of the mass conservation equation.

6.4.3 Momentum Derivation

As with mass conservation, consider a moving particle W so that conservation of momentum gives

$$\frac{d}{dt}\int_W \rho u J_s dW = -\int_W \frac{\partial p}{\partial x}J_s dW$$

or

$$\int_W \frac{d}{dt}\left(\rho u J_s\right) dW = -\int_W \frac{\partial p}{\partial x}J_s dW$$

Again, the particle is arbitrary, so one form of the momentum governing equation is

$$\frac{d}{dt}(\rho u J_s) = -\frac{\partial p}{\partial x} J_s$$

Expanding and simplifying gives a final governing equation of

$$\frac{d}{dt}(\rho u) + \rho u \, div U = -\frac{\partial p}{\partial x}$$

6.4.4 Momentum Derived From the Euler Frame

Alternatively, transforming the well known Euler frame equations of motion should result in an identical governing equation with considerably less effort.

The Euler frame equation for momentum is

$$\frac{\partial(\rho u)}{\partial t} + \frac{\partial}{\partial x}\left(\rho u^2 + p - \tau_{xx}\right) + \frac{\partial}{\partial y}(\rho v u - \tau_{yx}) = 0$$

The material derivative is

$$\frac{D(\rho u)}{Dt} = \frac{\partial(\rho u)}{\partial t} + U \cdot \nabla(\rho u)$$

Substituting the above Navier-Stokes form into the above material derivative form gives

$$\begin{aligned} \frac{D(\rho u)}{Dt} &= -\frac{\partial}{\partial x}(\rho u u) - \frac{\partial}{\partial x}(p - \tau_{xx}) \\ &\quad -\frac{\partial}{\partial y}(\rho v u) - \frac{\partial}{\partial y}(-\tau_{yx}) \\ &\quad +U \cdot \nabla(\rho u) \end{aligned}$$

reducing to

$$\begin{aligned} \frac{D(\rho u)}{Dt} &= -u\frac{\partial}{\partial x}(\rho u) - \rho u \frac{\partial}{\partial x}(u) - \frac{\partial}{\partial x}(p - \tau_{xx}) \\ &\quad -v\frac{\partial}{\partial y}(\rho u) - \rho u \frac{\partial}{\partial y}(v) - \frac{\partial}{\partial y}(-\tau_{yx}) \\ &\quad +U \cdot \nabla(\rho u) \end{aligned}$$

which is

$$\frac{D(\rho u)}{Dt} = -\rho u \frac{\partial u}{\partial x} - \frac{\partial}{\partial x}(p - \tau_{xx}) - \frac{\partial}{\partial y}(-\tau_{yx})$$

A similar governing equation results from expanding the temporal derivative and substituting continuity

$$\frac{D(u)}{Dt} = \rho u \tfrac{\partial u}{\partial x} \quad -\rho u \frac{\partial u}{\partial x} - \frac{\partial}{\partial x}(p - \tau_{xx}) - \frac{\partial}{\partial y}(-\tau_{yx})$$

which is

$$\frac{D(u)}{Dt} = -\frac{\partial}{\partial x}(p - \tau_{xx}) - \frac{\partial}{\partial y}(-\tau_{yx})$$

6.4.5 Energy From the Euler Frame

The Euler frame equation for energy is

$$\frac{\partial(\rho e)}{\partial t}+\frac{\partial}{\partial x}\left(\rho u\left(e+\frac{p}{\rho}\right)\right)+\frac{\partial}{\partial y}\left(\rho v\left(e+\frac{p}{\rho}\right)\right)-\frac{\partial}{\partial x}\left(u\tau_{xx}+v\tau_{xy}-q_x\right)-\frac{\partial}{\partial y}\left(u\tau_{yx}+v\tau_{yy}-q_y\right)=0$$

with the substantial derivative

$$\begin{aligned}\frac{D(\rho e)}{Dt} &= u\frac{\partial}{\partial x}(\rho e)+v\frac{\partial}{\partial y}(\rho e)\\ &\quad -u\frac{\partial}{\partial x}(\rho e)-\rho e\frac{\partial}{\partial x}(u)-\frac{\partial}{\partial x}(up)\\ &\quad -v\frac{\partial}{\partial y}(\rho e)-\rho e\frac{\partial}{\partial y}(v)-\frac{\partial}{\partial y}(vp)\\ &\quad \frac{\partial}{\partial x}\left(u\tau_{xx}+v\tau_{xy}-q_x\right)\\ &\quad \frac{\partial}{\partial y}\left(u\tau_{yx}+v\tau_{yy}-q_y\right)\end{aligned}$$

which is

$$\begin{aligned}\frac{D(\rho e)}{Dt} &= -\rho e\, divU\\ &\quad -\frac{\partial}{\partial x}(up)-\frac{\partial}{\partial y}(vp)\\ &\quad \frac{\partial}{\partial x}\left(u\tau_{xx}+v\tau_{xy}-q_x\right)\\ &\quad \frac{\partial}{\partial y}\left(u\tau_{yx}+v\tau_{yy}-q_y\right)\end{aligned}$$

6.4.6 Entropy

The entropy equality is

$$\frac{D(\rho s)}{Dt}=\frac{\Phi}{T}+\frac{d}{dx}\left(k\frac{dT}{dx}\right)-T\rho s\, divU$$

6.4.7 Non-dimensionalization

This section derives a non-dimensional form of the governing equations. Location is

$$\begin{aligned}L_o\frac{U_o}{L_o}\frac{dr^\star}{dt^\star} &= u^\star U_o\\ \frac{dr^\star}{dt^\star} &= u^\star\end{aligned}$$

This is self similar.

The density governing equation becomes

$$\rho_o\frac{U_o}{L_o}\frac{d\rho^\star}{dt^\star}+\rho_o\rho^\star\frac{U_o}{L_o}\frac{du_k^\star}{dx_k^\star}=0$$

$$\frac{d\rho^\star}{dt^\star} + \rho^\star \frac{du_k^\star}{dx_k^\star} = 0$$

Again, it is self similar.

The velocity governing equation is

$$\rho_o U_o \frac{U_o}{L_o} \frac{d\left(\rho u_i\right)^\star}{dt^\star} + \rho_o U_o \left(\rho u_i\right)^\star \frac{U_o}{L_o} \frac{du^\star}{dx^\star} + \rho_o U_o^2 \frac{1}{L_o} \frac{dp^\star}{dx_i^\star} = \frac{1}{L_o} \frac{\mu_o \hat{\mu} U_o}{L_o} \frac{d\tau_{ji}^\star}{dx_i^\star}$$

$$\frac{d\left(\rho u_i\right)^\star}{dt^\star} + \left(\rho u_i\right)^\star \frac{du^\star}{dx^\star} + \frac{dp^\star}{dx_i^\star} = \frac{\hat{\mu}}{Re} \frac{d\tau_{ji}^\star}{dx_i^\star}$$

The inviscid portion is self similar, while the viscous portion is scaled by Reynolds number.

The energy governing equation is

$$\begin{aligned} \rho_o U_o^2 \frac{U_o}{L_o} \frac{dE^\star}{dt^\star} + \rho_o U_o^2 E^\star U_o \frac{1}{L_o} \frac{du_k^\star}{dx_k^\star} + \rho_o U_o^2 U_o \frac{1}{L_o} \frac{d\left(u_k^\star p^\star\right)}{dx_k^\star} &= -\frac{1}{L_o^2} k \frac{\gamma U_o^2}{c_p} \frac{d^2\left(T^\star\right)}{dx_j^{\star 2}} \\ &\quad + U_o \frac{1}{L_o} \frac{\mu_o \hat{\mu} U_o}{L_o} \frac{d\left(u_k^\star \tau^\star\right)}{dx_k^\star} \end{aligned}$$

Dividing and gathering terms gives

$$\frac{dE^\star}{dt^\star} + E^\star \frac{du_k^\star}{dx_k^\star} + = -\frac{1}{Re} \frac{\gamma}{Pr} \frac{d^2\left(T^\star\right)}{dx_j^{\star 2}} + \frac{\hat{\mu}}{Re} \frac{d\left(u_k^\star \tau^\star\right)}{dx_k^\star}$$

6.5 Galerkin Method Applied to Lagrangian Equations

Applying Galerkin method and integration by parts does not eliminate the spacial derivative but only shifts it from one state to another. For example,

$$\frac{d\rho}{dt} + \rho \frac{du}{dx} = 0$$

Applying the Galerkin method gives

$$\int \phi \frac{d\rho}{dt} + \int \phi \rho \frac{du}{dx} = 0$$

The objective is to eliminate the velocity divergence term. Grouping ϕ and ρ together and applying the Green's Theorem gives

$$\int \phi \frac{d\rho}{dt} - \int \frac{d\phi\rho}{dx} u + \int \phi \rho u \cdot n_x = 0$$

This form is not an improvement since the derivative is now on density.

Worse, applying Galerkin causes the governing equations to be strongly coupled. Also remember that a *major* advantage of applying the Galerkin method for the Euler frame equations was that the integrals became integrals of fluxes. The Galerkin method is rejected for the Lagrangian equations.

6.6 Element Operations

Unlike the Eulerian frame CFD solver, the Lagrangian frame solver requires several element operations. Primarily these operations are needed for determining advected locations within a known grid and known flow field.

6.6.1 Linear Geometry Element

For a linear element, the local to global coordinate transform is[26]

$$\begin{pmatrix} 1 \\ x \\ y \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix} \begin{pmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \end{pmatrix}$$

where x and y are nodal locations. Substituting the top row reduces the transform to two independent coordinates[14]

$$\begin{aligned} \begin{pmatrix} x \\ y \end{pmatrix} &= \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} + \begin{bmatrix} (x_1 - x_3) & (x_2 - x_3) \\ (y_1 - y_3) & (y_1 - y_3) \end{bmatrix} \begin{pmatrix} \zeta_1 \\ \zeta_2 \end{pmatrix} \\ &= \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} + \begin{bmatrix} x_{13} & x_{23} \\ y_{13} & y_{23} \end{bmatrix} \begin{pmatrix} \zeta_1 \\ \zeta_2 \end{pmatrix} \\ &= \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} + B \begin{pmatrix} \zeta_1 \\ \zeta_2 \end{pmatrix} \end{aligned}$$

The Jacobian is

$$\mathbf{J} = \begin{bmatrix} x_{13} & y_{13} \\ x_{23} & y_{23} \end{bmatrix} = B^T$$

Notice that the Jacobian matrix is the transpose of the local to global transform matrix. The Inverse Jacobian is

$$\mathbf{J}^{-1} = \frac{1}{\det J} \begin{bmatrix} y_{23} & -y_{13} \\ -x_{23} & x_{13} \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} y_{23} & -y_{13} \\ -x_{23} & x_{13} \end{bmatrix}$$

6.6.2 Within

A primary operation is to test if a coordinate is within a particular element. Given the global location $X = (x, y)$ with barycentric coordinate $\Xi = (\zeta_1, \zeta_2, \zeta_3)$, the location is within element E when each individual coordinate ranges from zero to one

$$0 \leq \zeta_i \leq 1$$

and the sum is exactly one

$$\zeta_i = 1$$

The global to local transformation is

$$\begin{aligned}
\begin{pmatrix} \zeta_1 \\ \zeta_2 \end{pmatrix} &= J^{-T} \begin{pmatrix} x - x_3 \\ y - y_3 \end{pmatrix} = \begin{bmatrix} (x - x_3) & (y - y_3) \end{bmatrix} J^{-1} \\
&= \frac{1}{2A} \begin{bmatrix} y_{23} & -x_{23} \\ -y_{13} & x_{13} \end{bmatrix} \begin{pmatrix} x - x_3 \\ y - y_3 \end{pmatrix} \\
&= \frac{1}{2A} \begin{bmatrix} (x - x_3) & (y - y_3) \end{bmatrix} \begin{bmatrix} y_{23} & -y_{13} \\ -x_{23} & x_{13} \end{bmatrix}
\end{aligned}$$

Since the third coordinate is calculated as

$$\zeta_3 = 1 - \zeta_1 - \zeta_2$$

this implies that the summation is implicitly satisfied. Thus, only using the range test is sufficient.

Knowledge of the barycentric coordinates also allows for a search direction into the adjacent element. When coordinate ζ_i violates the test, search the element adjacent to edge i. The range test is

$$edge = \begin{cases} 1 & if\, \zeta_1 < 0 \\ 2 & if\, \zeta_2 < 0 \\ 3 & if\, \zeta_1 + \zeta_2 > 0 \\ & otherwise \end{cases}$$

The degenerate case of finding an element on the other side of a bisected domain is not valid; fluid particles must be continuously transportable within a domain.

6.6.3 Interpolation

Interpolation is a primary operation. Given the value coefficients of an element, c_s, and a local location Ξ, an interpolation routine determines a function value

$$v = f(c_s, \Xi)$$

With a basis function expansion the function value is

$$v = c_s \phi_s (\Xi)$$

6.7 Numerical Methods

A significant requirement for Lagrangian methods is numerical time integration.

6.7.1 Time Integration

Lagrangian frame time integration is an interesting task for a single reason: the fundamental labeling is particle based. A general rule is that for a particular particle, the path is tracked from the reference node location to unknown locations. So, the routine solves the question "Where did the particle at this node come from?" rather than the question "Where will the particle at this node go?".

In 1D, the one element visual representation is shown in Figure 6.3. Grid points are known for

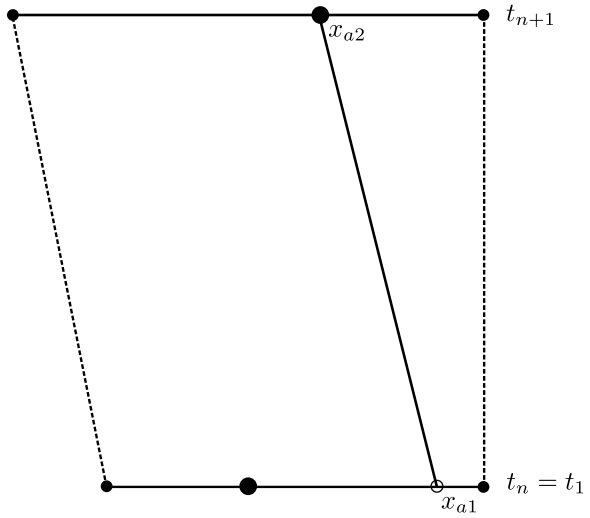


Figure 6.3: Lagrangian Time Integration

two different grids. The velocity for a given particle is known from the velocity field.

For prototyping, backwards Euler time integration may seem ideal. The traditional BE form is

$$y_{n+1} = y_n + h\, f(t_{n+1}, y_{n+1})$$

For Lagrangian methods, backwards Euler methods are not sufficient; no-flow regions tend to get value-locked.

Explicit projection forward is essential. Yet, this forward projection is numerically complicated since particles are not guaranteed to land on future grid nodes. This requires at least an initial interpolation forward to the future timestep grid.

6.8 Results and Analysis

A 1st order (P=1) Lagrangian solver, lagr2d, was implemented. A 1st order B-Spline (i.e., traditional hat function) provides the interpolation. The numerical operations and data structures are from the previously discussed Ale2d solver with one exception. A particle tracking routine was added based on the derivation given above.

The time integration method is a 2nd order RK routine. Unexpectedly, the maximum timestep did not follow the theoretical estimate. In fact, the timestep is of the same magnitude as the corresponding Euler timestep. There is a conceptual reason for this discrepancy. Compressible flows have wave propagation velocities. The timestep must capture both the flow velocity advection and the sonic characteristic velocity. Thus, the traditional Lagrangian timestep limit is sufficient only for incompressible flows.

An unsteady viscous cylinder was simulated with the lagr2d code. Figure 6.4 shows the velocity distribution. The cylinder test case was tested past 70 clearing times. Unlike the ale2d code, the

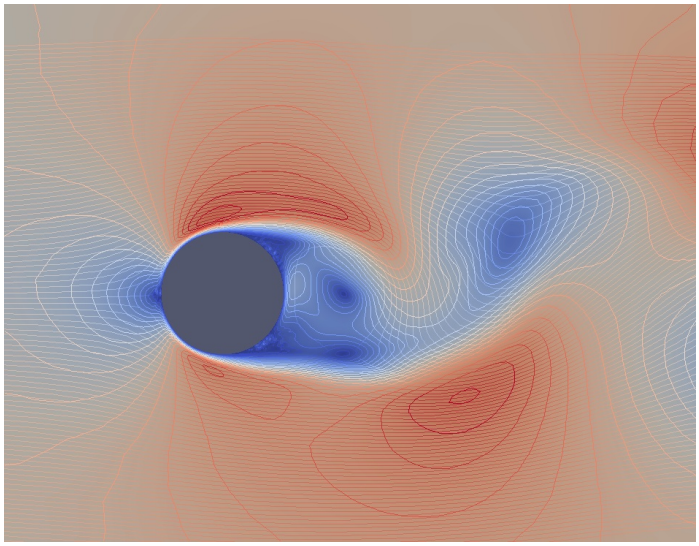


Figure 6.4: Cylinder: Velocity for viscous $Re = 500$ flow with lagr2d (P=1)

lagr2d code appears to have fewer stability restrictions. The current lagr2d code also appears to have a weakly unstable exiting far-field condition. The cylinder presented this issue as an increasing backpressure.

For comparison, the cylinder's wake Strouhal number is computed. The Strouhal number is defined as

$$St = \frac{fD}{U}$$

For a circular cylinder, the Strouhal number depends on Reynolds number. From Fey[18], at a

Reynolds number of 500, the Strouhal number is 0.21. From lagr with $P = 1$, the numerical shedding frequency is

$$St = 0.218 \pm 0.008$$

A comparison with ns2d slightly depends on dissipation. For dissipation of 0.2, the shedding frequency is

$$St = 0.228 \pm 0.004$$

Comparing solution clock-time with ns2d (an in-house Navier-Stokes linear Galerkin solver) is interesting. The lagr2d code is approximately half as fast as ns2d. The lagr2d solver struggled with small timesteps caused by large velocity gradients allowed by, yet again, a poor dissipation scheme. Yet the local-operation lagr2d code is likely inherently easier to parallelize. This could be a significant advantage.

The Lagrangian frame appears to have merit for CFD solutions. Primary concerns are particle tracking, timesteps, and boundary conditions.

Chapter 7

Lagrangian Challenges

A successful Lagrangian frame CFD solver has several challenges that must be met.

7.1 Particle Tracking

Particle tracking in a Lagrangian solver is absolutely essential. Particle tracking appears simple: given a location and a time, find state variables corresponding to that location and time. In practice, there are complications. Also remember that this tracking must be performed for each particle at every timestep.

A first complication is finding an arbitrary point within a complex domain. Conceptually, a nearest-point map between past and current grids is sufficient. Computer science routines are available for this situation; some require significant setup costs.

A second complication arises at boundary conditions. As a particle attempts to leave the domain, the boundary condition must be applied. One particular difficulty is encountered with the far-field condition. At some level, the Lagrangian frame governing equations are similar to the far field flux boundary condition computed with the Roe/Riemann equations. Yet, simply applying the far-field boundary condition specification allows for unphysical phenomena including inlet velocity lock and outlet density build-up.

Where a low quality routine in the Eulerian frame merely degrades the solution accuracy, a low quality particle tracking routine will eventually fail to find an appropriate starting or ending location. This immediately ends the simulation.

7.2 Time Integration

A second essential operation is time integration. Again, this seems to be a solved problem. Yet, fully explicit solvers have a fundamental implementation issue. Explicit time advancement requires expensive interpolation onto the future grid. A discussion of interpolation relevant to this issue is discussed in Boyd[10]. Fully implicit solvers get stuck for zero velocities. The implicit point tracking solver working backwards in time with zero velocity never changes the interpolation point at the

previous timestep. Thus, velocity becomes stuck. Solving this requires forward projection (explicit) time integration.

Lauritzen[40] states that "There is a need to incorporate trajectories in a more consistent manner in semi-Lagrangian models instead of treating trajectory determination and the solution to the remaining equations of motion as two separate tasks."

The Lagrangian timestep limit is only valid for incompressible flows. For compressible flows, only the density characteristic moves at the flow velocity. The $u + a$ and $u - a$ characteristics significantly constrain the maximum timestep, especially for low Mach number flows. This issue potentially mitigates the Lagrangian timestep advantage. Bartello[5] states that "wave propagation implies identical time scales in Eulerian and Lagrangian frames".

7.3 Turbulence Model

The author is unclear about the ramifications of the Lagrangian frame regarding turbulence models. It seems likely that the regular Eulerian frame turbulence models could easily be reformed with a frame transformation. Lagrangian turbulence modeling in the literature seems to be a primitive field. This project does not address this issue.

7.4 Grid

Grid generation and remeshing will become a critical challenge for the Lagrangian solver. Two particular concerns stand out: coupling the boundary motions with updated grid generation and interpolating from different grids. Overgridding and variable element sizes are not expected to be a significant contribution to the challenge.

A second grid related challenge is interpolation from one solution field to another solution field with a different element topology. This challenge becomes necessary when the domain is regridded.

Chapter 8

Rigid Body Dynamics

Dynamic CFD is defined as computationally simulating flow fields with the boundary conditions of a arbitrarily translating, rotating, and deforming vehicle. Rigid body dynamics simulates the constantly moving case. This requires tracking body motion with position and orientation. This section of the project was developed as an upgrade to the existing Euler3d CFD code, a linear element 3D inviscid code with non-inertial based motion capability.

This section's objective is to develop a six degree of freedom rigid body solver and then to couple the rigid body solver into a computational fluid dynamics (CFD) solver. Coupling into a CFD solver requires determining boundary conditions from the previous aerodynamically and inertial generated forces. This paper concentrates on the flight dynamics applications of the rigid body dynamics in an inviscid but compressible flow.

Rigid body dynamics are governed by two sets of equations: attitude representation and body-frame motion. Attitude representation uses an inertial fixed reference frame for translations and directions. The body-frame kinematics uses translational and rotational forms of Newton's classical law.

8.1 Literature Review

Simulating aerospace vehicle dynamics is common; CFD based flight dynamics is rarer. Nelson[55] derives a traditional approach to applying rigid body dynamics to aircraft. Phillips[60] reviews Euler angles, direction cosines, and quaternions for aircraft motion specification. Phillips suggests using quaternions to avoid the computational expenses and singularities inherent in the Euler angle representation. Stevens and Lewis[70] derive a quaternion approach to rigid body dynamics. Visually, Kato[36] discusses large amplitude maneuvers and their effect on motion descriptions. Store separations with multiple body dynamic simulations is a related and active field[72, 12, 43]. Rizk[65] implemented a 6 degree-of-freedom store separation dynamics simulation.

Primary references for this paper are *Aircraft Control and Simulation*[70], *Flight Stability and Automatic Control*[55], and *Finite Element CFD Analysis of Super-Maneuvering and Spinning Struc-*

tures[14].

8.2 Attitude Representation

Attitude representation involves specifying the aircraft's position and orientation and converting between inertial and non-inertial frames. For the scope of this paper, the inertial frame is Earth fixed, and the non-inertial frame is aircraft body-fixed.

8.2.1 Inertial and Non-Inertial Frames

The crux of attitude representation is converting between body fixed and inertial reference frames. Figure 8.1 shows a two-dimensional representation of an inertial frame (X,Y) and a non-inertial frame (x,y) connected by vector R. A point at vector r_b in the non-inertial frame transforms to a vector q_i in the inertial frame. The relationship is

$$q_i = R_i + Br_b$$

B is a transformation operator between the body b and inertial i frames. So, B^{-1} transforms from the inertial frame to the body frame. Intuition suggests that B and B^{-1} must be similar. In fact, transformation matrices are orthogonal so that $B^{-1} = B^T$.

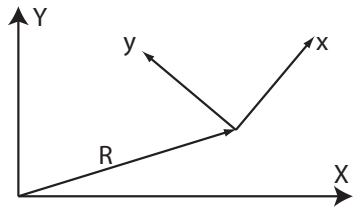


Figure 8.1: Coordinate Systems

8.2.2 Orientation

Orientation concerns the directionality of the body-fixed frame with respect to an inertial frame. For the scope of this project, a flat Earth is a sufficient inertial frame. This paper uses quaternions for orientation. The objective is to convert body frame rotations to inertial frame attitudes.

Euler angles were rejected for the well-known pitch singularity. Preliminary testing also showed that while the Euler angle singularity is at $\theta = \pm 90°$, attitude errors become noticeable earlier. For a generic motion simulations, Euler angles are unwelcome. Technically, Euler angles can be made to work, but doing so requires repeated coordinate transforms and a switching algorithm to prevent gimble lock. Quaternions are just less troublesome to implement.

Quaternions have no such singularity, but this comes at the expense of an extra parameter. The quaternion consists a scalar (q_0), and a vector (q_1, q_2, q_3). An extra constraint is required, $q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$. For a quaternion based representation system, the transformation matrix B is[70]

$$B = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

Quaternion updates are via four 1st order differential equation. The *quasi-linear* quaternion differential equation is[70]

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = -\frac{1}{2} \begin{bmatrix} 0 & p & q & r \\ -p & 0 & -r & q \\ -q & r & 0 & -p \\ -r & -q & p & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

Solely for human visualization, traditional Euler angles are needed. The quaternion to Euler angle conversion is[70]

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \arctan\left(2(q_0 q_1 + q_2 q_3)/\left(q_0^2 - q_1^2 - q_2^2 + q_3^2\right)\right) \\ \arcsin\left(2(q_0 q_2 - q_1 q_3)\right) \\ \arctan\left(2(q_0 q_3 + q_1 q_2)/\left(q_0^2 + q_1^2 - q_2^2 - q_3^2\right)\right) \end{bmatrix} \quad \text{with the ranges} \quad \begin{array}{c} -\pi \le \phi \le \pi \\ -\pi/2 \le \theta \le \pi/2 \\ -\pi \le \psi \le \pi \end{array}$$

Inspection of the Quaternion to Euler angle conversion shows that unity magnitude quaternions are needed to remain in the real valued arcsin() and arctan() domains. Re-normalizing the quaternion appears necessary before converting to Euler angles.

8.2.3 Position

Inertial frame positions are calculated from body frame velocities and inertial orientation. Position updates use the B transformation matrix developed above. The translational equation is

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = B \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

The result is three 1st order differential equations for inertial position. Integration is simple when no rotations occur.

8.3 Body Frame Kinematics

Aircraft velocity kinematics are calculated in the non-inertial (body) frame. Nelson[55] derives a set of aircraft equations of motion. Inertia are referenced to the body fixed frame. Except for certain body forces such as gravity, the body fixed equations of motion are independent of attitude.

8.3.1 Translation

For the translational rigid body modes, the equations of motion are[55]

$$X - mgS_\theta = m(\dot{u} + qw - rv)$$
$$Y + mgC_\theta S_\phi = m(\dot{v} + ru - pw)$$
$$Z + mgC_\theta C_\phi = m(\dot{w} + pv - qu)$$

The translation equations become nonlinear when the rotation axis is not along the translation velocity axis. Solving for the translational derivative terms yields

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \frac{1}{m} \begin{bmatrix} X - mgS_\theta \\ Y + mgC_\theta S_\phi \\ Y + mgC_\theta C_\phi \end{bmatrix}$$

The translational equation of motion consists of three 1st order nonlinear differential equations.

8.3.2 Rotation

The rotational equations of motion in the body frame for typical symmetrical aircraft are[55]:

$$L = I_x\dot{p} - I_{xz}\dot{r} + (I_z - I_y)qr - I_{xy}pq$$
$$M = I_y\dot{q} + (I_x - I_z)rp + I_{xz}(p^2 - r^2)$$
$$N = -I_{xz}\dot{p} + I_z\dot{r} + (I_y - I_x)pq + I_{xz}qr$$

The equations are nonlinear when certain translations and rotations occur. In contrast to the single translational mass, 6 rotational inertias are possible. Solving for the rotational equations or motion for the rotational derivatives yields

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = I^{-1} \begin{bmatrix} L - (I_z - I_y)qr + I_{xz}pq \\ M - (I_x - I_z)rp - I_{xz}(p^2 - r^2) \\ N - (I_y - I_x)pq - I_{xz}qr \end{bmatrix}$$

The general inertia matrix is

$$I = \begin{bmatrix} I_x & I_{xy} & I_{xz} \\ I_{xy} & I_y & I_{yz} \\ I_{xz} & I_{yz} & I_z \end{bmatrix}$$

The general inverse inertia is

$$I^{-1} = \frac{1}{\Gamma} \begin{bmatrix} \left(I_y I_z - I_{yz}^2\right) & \left(I_{xz} I_{yz} - I_{xy} I_z\right) & \left(I_{xy} I_{yz} - I_{xz} I_y\right) \\ \left(I_{yz} I_{xz} - I_{xy} I_z\right) & \left(I_x I_z - I_{xz}^2\right) & \left(I_{xz} I_{xy} - I_x I_{yz}\right) \\ \left(I_{xy} I_{yz} - I_y I_{xz}\right) & \left(I_{xy} I_{xz} - I_x I_{yz}\right) & \left(I_x I_y - I_{xy}^2\right) \end{bmatrix}$$

$$\Gamma = I_x I_y I_z - I_x I_{yz} I_{yz} - I_{xy} I_{xy} I_z + I_{xy} I_{yz} I_{xz} + I_{xz} I_{xy} I_{yz} - I_{xz} I_y I_{xz}$$

The rotational equation of motion consists of three 1st order nonlinear differential equations.

8.4 Coupled Rigid Body Equations of Motion

The objective of this section is to show the total 6 degree of freedom rigid body dynamics equations of motion. The state vector is: $S = [x\ y\ z\ u\ v\ w\ p\ q\ r\ q_0\ q_1\ q_2\ q_3]^T$. Combining the above orientation and kinematic equations yields thirteen 1st order nonlinear differential equations. From Stevens and Lewis[70] the complete system with lumped coupling terms (Ω_B, Ω_q, I) is:

$$\dot{S} = \begin{bmatrix} 0 & B & 0 & 0 \\ 0 & -\Omega_B & 0 & 0 \\ 0 & 0 & -I^{-1}\Omega_B I & 0 \\ 0 & 0 & 0 & -\frac{1}{2}\Omega_q \end{bmatrix} S + \begin{pmatrix} 0 \\ m^{-1}F_B \\ J^{-1}T_B \\ 0 \end{pmatrix} \tag{8.1}$$

On first inspection, the system appears linear, but this is not the case since the state variable is contained inside the gradient matrix.

8.4.1 Numerical Methods

Appropriate numerical methods are required for the quaternion governing equation since the quaternion magnitude must remain unity. Phillips[60] discusses this topic and suggests at least a 4th order ODE numerical solution. This paper uses a 4th order Adams Moulton finite difference ODE numerical integration method. The update is discrete in time based on continuous derivatives ($\dot{y}$) at four discrete timesteps.

$$y(t+1) = y(t) + \frac{dt}{24}\left(55\dot{y}(t) - 59\dot{y}(t-1) + 37\dot{y}(t-2) - 9\dot{y}(t-3)\right)$$

8.5 Rigid Body Dynamics Results

A 6 degree of freedom rigid body dynamics solver was designed and implemented. This section verifies and validates the solver's behavior, representation, and integration.

8.5.1 Verification

The objective of this section is to establish that the rigid body equations of motion are being solved correctly.

Energy Conservation

This case tests for rigid body dynamics energy conservation. The concept is to give a body an initial motion and then track the total energy. Conserved energy verifies that the rigid body dynamics solver is solving the correct equations. Additionally, this test case evaluates timestep sensitivities of the rigid body dynamics solver.

The governing energy equation for translations and rotations is

$$E(t) = \sum_{i=x,y,z} \frac{1}{2} m_i v_i(t)^2 + \sum_{i=p,q,r} \frac{1}{2} I_i \omega_i(t)^2$$

For this particular test case, the masses and inertias are

$$M = 1, \quad I_x = 1, \quad I_y = 2, \quad I_z = 3$$

For an initial body-fixed translation vector of (1,2,3) and a rotation vector of (4π, 2π, π) radians per second, the theoretical kinetic energy is 140. A time history plot (Fig. 8.2) shows the non-linear behavior. Figure 8.3 shows the kinetic energy content for timesteps varying from 0.01 to 0.0001, equivalent to 50 to 5000 points per highest frequency at roll rate of 720 degrees per second. Above 100 points per cycle seems appropriate; fewer points per cycle tend to artificially dampen the dynamics solution. Timestep sizes for accurate dynamics responses appear to be larger than the corresponding CFD timestep sizes.

Translational and Rotational Forced

This case directly specified the translational forces inside the CFD solver. The objective is to verify the constant force displacement motion. The non-dimensional forces in each coordinate direction are $F_x = 0.25$, $F_y = 0.5$, and $F_z = 1.0$. Since the forces are uncoupled when viewed in each orthogonal coordinate system, the displacement motion's form in each coordinate direction is

$$d_i(t) = \frac{1}{2} \frac{F_i}{M} t^2$$

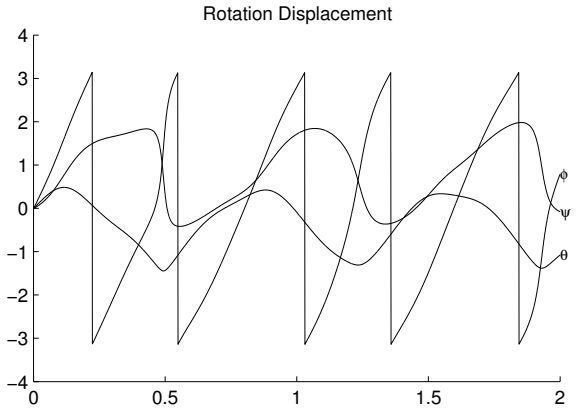


Figure 8.2: Rotational Displacement Time History

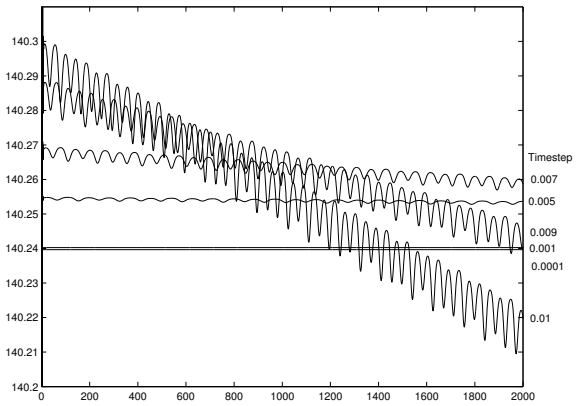


Figure 8.3: Energy Sensitivity: Retained Energy over Time for Various Timesteps

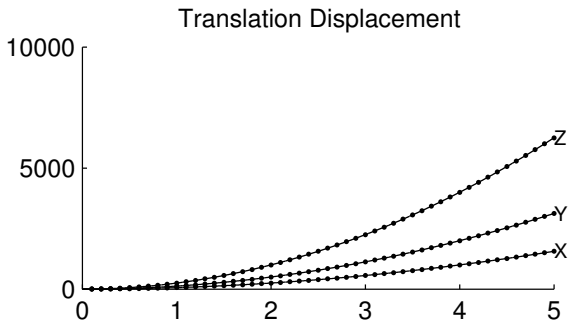


Figure 8.4: Translational Displacement

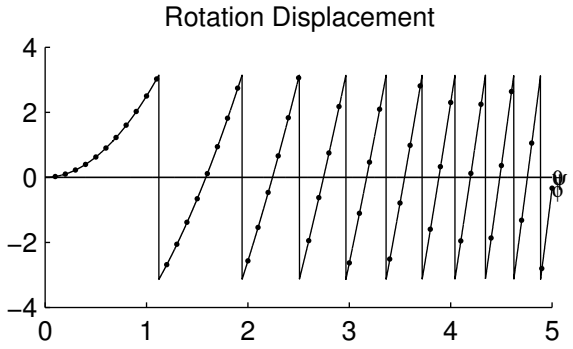


Figure 8.5: Rotational Displacement

The CFD solver was hard-coded to represent the above forces. Figure 8.4 shows the translational displacements from 0 to 5 seconds with a mass of 1/500.* The solution for z(t) is

$$z(t) = 250\, t^2$$

At time 5, the error between theory and the solver is 1.5 out of 6250 (approximately 0.02%). The dynamics output matches theory.

Likewise, the rotational degree of freedom is tested. Figure 8.5 shows the dynamic solver Euler angles versus theory for an accelerating roll. The roll Euler angle, Φ, maps between $\pm\pi$ regardless of the total rolled angle. Thus, the solution for Φ is

$$\Phi(t) = 2.5\, t^2 - 2n\pi$$

The error at time 5 is 0.04 out of 62.5 total radians of rotation (0.06%). Again, the dynamic solver matches theory.

*The 1/500 mass ratio occurs because the CFD forces are scaled by dynamic pressure.

Simple Pressure Field Motion

The objective of this section is to verify that the CFD solver's pressure integration is input correctly into the rigid body simulation. This case will test the CFD pressure to rigid body coupling. A simple pressure field was specified

$$p^*(x,y,z,t) = \begin{cases} 1 & \text{if } z > \epsilon \\ -1 & \text{if } z < \epsilon \\ 0 & \text{otherwise} \end{cases}$$

Since the trailing edge does not *exactly* lie at $z = 0$, a deadband parameter ϵ is used to prevent pressure wrap-over on the trailing edge node.

The pressure field is shown in Figure 8.6. Conceptually, the pressure above the zero waterline is greater than below. For reference, the dimensional pressure is 1000 psf (47.9 kpa). With a rotation point at the leading edge, the theoretical pitch moment is 20 ft-lb (27 N-m) or 0.04 (CFD non-dimensional). The time and moment response is given in Figure 8.7. The integrated CFD moment gives 0.0392; the error occurs in the finite length of the trailing edge element. This test concludes the verification process.

Figure 8.6: Specified Pressure Field

8.5.2 Validation

The objective of this section is to validate the dynamics solver with quasi-steady aerodynamics solutions. Both the translational and rotational frames are tested.

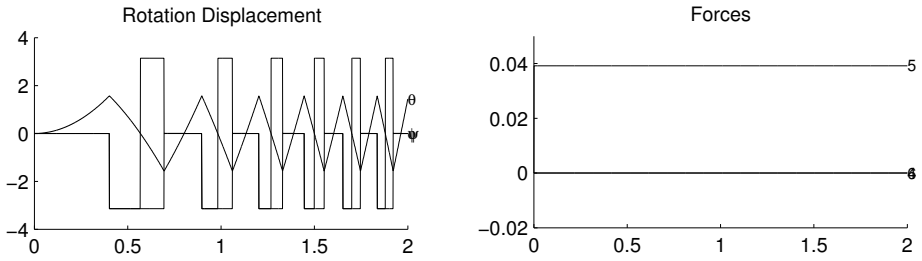


Figure 8.7: Specified Pressure Motion and Forces

Translational Rate Damping

This test case verifies the z-axis translational motion with airfoil rate damping. The concept (Fig. 8.8) is to allow an airfoil to reach a steady state upward velocity —via the airfoil's lift— when starting from an initial angle of attack, α_0. Intuition indicates that the final upward translational velocity will be such to provide an effective angle of attack of zero.

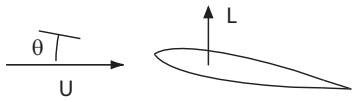


Figure 8.8: Translation Rate: Geometry

Assuming quasi steady aerodynamics, the lift is a linear function of the instantaneous angle of attack. For this problem, the effective angle of attack comes from an initial angle of attack, α_0, and a plunging velocity to freestream ratio, $\dot{x}(t)/V$. Thus, the governing equation is

$$m\ddot{x}(t) = qC_{L_\alpha}S\left(-\arctan\frac{\dot{x}(t)}{V} + \alpha_0\right)$$

When assuming small angles, $\arctan\frac{x}{V}$ is approximately $\frac{x}{V}$. A solution to the above differential equation is

$$x(t) = \alpha_0 Vt - \frac{\alpha_0 V^2 m}{qC_{L_\alpha}S} + \frac{\alpha_0 V^2 m}{qC_{L_\alpha}S}\exp\left(-\frac{qC_{L_\alpha}St}{mV}\right)$$

The solver (dots) and theory (lines) predictions are shown in Figure 8.9. As expected, a steady state velocity is reached. The dynamics solver matches theory, which appears to suggest that the dynamic solver's translational displacement and velocity are properly passed to the CFD solver's boundary conditions.

Rotational Rate Damping

This testcases's objective is to validate a rotational degree of freedom. An airfoil *slit* is rotated axially about an inboard axis. A rotation rate, ω, creates an effective angle of attack at the airfoil

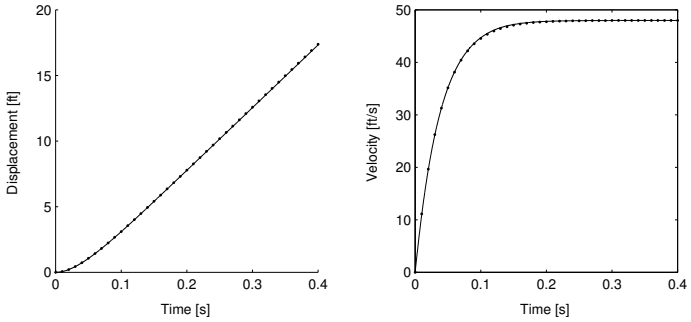


Figure 8.9: Translation Rate: Lift

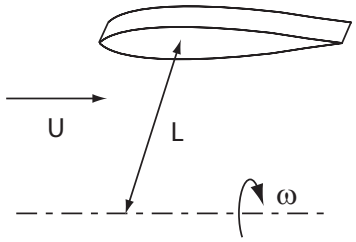


Figure 8.10: Rotating Rate Damping Geometry

of $\alpha = \arctan(\omega L/U)$. Figure 8.10 shows the geometry. The governing equation of motion when assuming quasi-steady aerodynamics is

$$\ddot{\phi}(t) = -I^{-1} L q c S C_{L_\alpha} \arctan \frac{\dot{\phi} L}{U}$$

The case has an initial rotation velocity of 90 degrees per second, a velocity of 500, and an axis offset L of 16. The rotational rate damping response is shown in Figure 8.11 for theory (lines) and the dynamic solver output (dots). The rotational degree of freedom validation matches the theoretical response.

8.5.3 Flight Dynamics

Flight dynamics concerns the interaction between aerodynamics and a rigid body with respect to aircraft motions. An aircraft undergoing common maneuvers and a rotating wedge are presented.

Simplified General Aviation

A Navion general aviation aircraft was approximated with simplified geometry. The aircraft is modeled with a wing, a horizontal stabilizer, and a vertical stabilizer. Figure 8.12 shows the geometry. Figure 8.13 shows the surface Mach distribution at $\alpha = 0$ and $\beta = 0$ for 174 ft/s (53 m/s) at SSL. Mass and sizing information comes from Nelson[55].

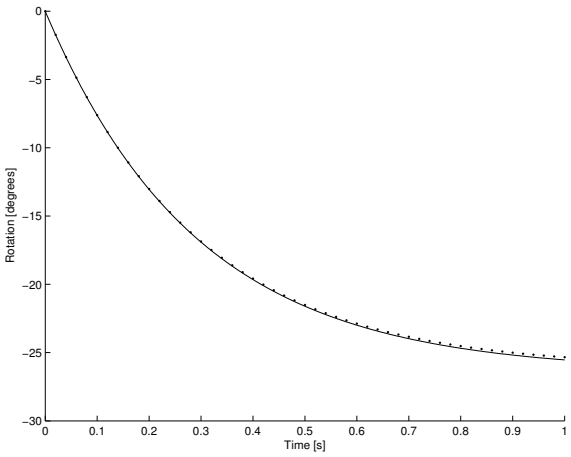


Figure 8.11: Rotating Rate Damping Response

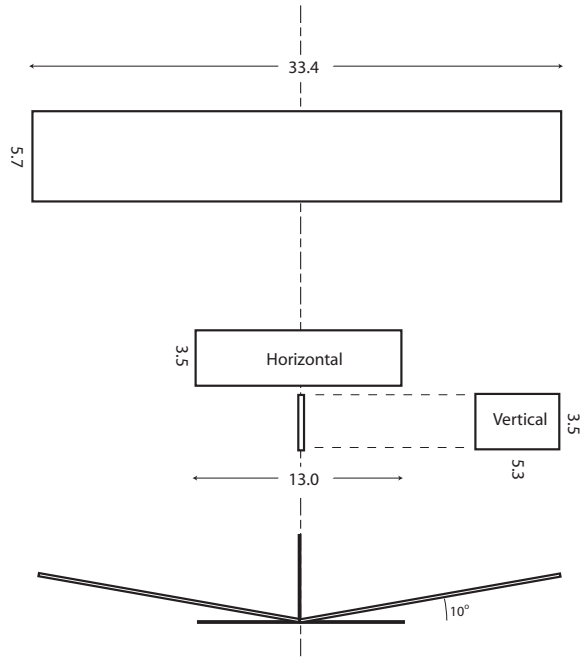


Figure 8.12: Navion Geometry

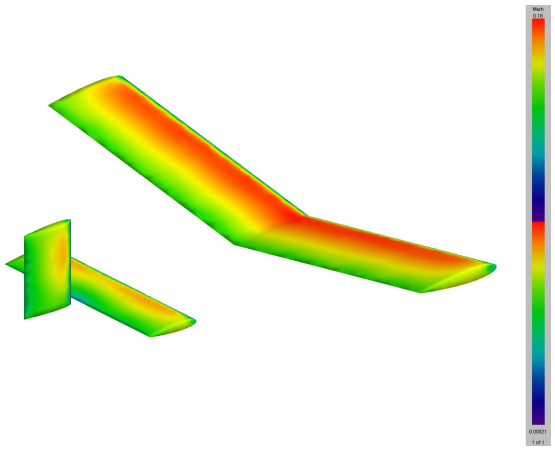


Figure 8.13: Navion 174 ft/s Mach Distribution

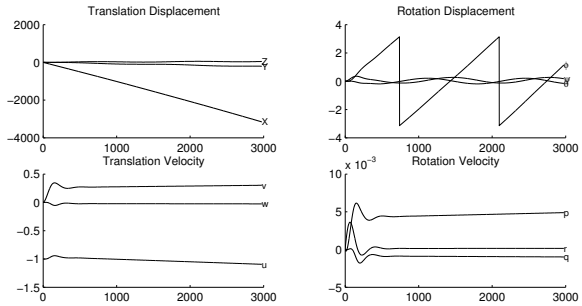


Figure 8.14: Navion Rudder/Dihedral Response Time History

Rudder/Dihedral Response The dihedral roll moment effect can be created with a rudder input. This case considers the stick-fixed free response for an initial left rudder input of 20 degrees. Again, the Navion's initial states are a velocity of 174 ft/s, stick-fixed controls, and zero bank, pitch, and yaw angles. For easier visualization, gravity is removed. Intuition suggests that the *nearly* constant yaw angle will cause a roll moment that over time creates a steady roll rate.

Figure 8.14 shows the translational and rotational motions. The deflected rudder rolls the aircraft in the expected barrel-roll maneuver in about 8 seconds. Figure 8.15 shows a visual representation of the aircraft's trajectory. The yaw angle is visible.

Loop This case's objective is to loop the Navion with a constant elevator deflection. To ensure sufficient energy to complete the maneuver, the aircraft begins inverted at the top of the loop. Initial conditions are 174 ft/s and a 20 degree elevator deflection.

The trajectory for the first case with the CG at 30% MAC (Fig. 8.16) shows a successful loop with the expected tightening at the top and an overall loss of altitude. Aircraft attitudes during the loop are visually consistent with reality.

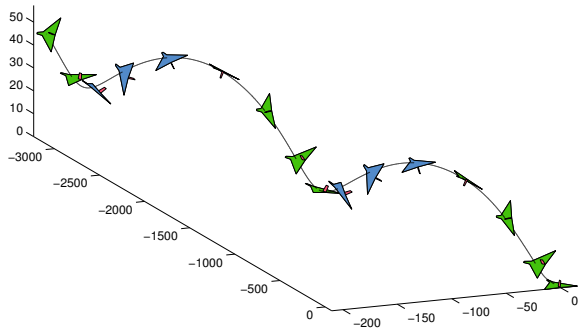


Figure 8.15: Navion Rudder Response Trajectory

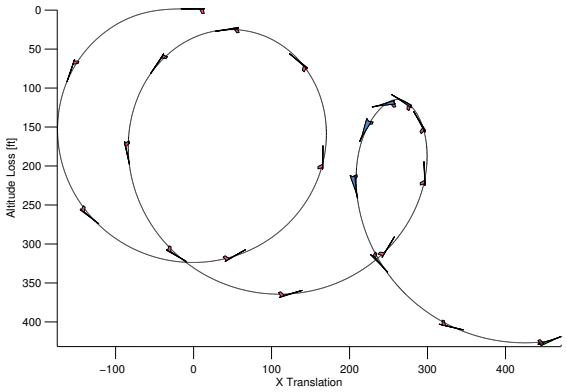


Figure 8.16: Navion Loop

Case 2 considers the same Navion aircraft and initial conditions but with the CG at a vastly tail-heavy and statically unstable 88% MAC. Figure 8.17 shows the trajectory with a stall/fall-out coming through the loop's bottom. Visually, the aircraft is pitch unstable.

Spin Spinning aircraft exhibit non-linear behavior. This spin case considers the Navion with 20 degrees up-elevator and 20 degrees left-rudder. Initial conditions are a negative vertical velocity with an initial yaw rate. Spin entry is not considered. Also, the inviscid Euler solution is likely to attenuate the separation for this viscous dominated *stalled* airfoil flow. Figure 8.18 shows the translational and rotational motions. From the low, non-increasing forward velocity and the harmonic rotational motion, the maneuver appears to be a spin and not a spiral motion. Altitude loss is approximately 500 feet per turn —interestingly consistent with reality when considering the inviscid Euler solution. The spin trajectory (Fig. 8.19) also appears consistent with an actual spin's behavior.

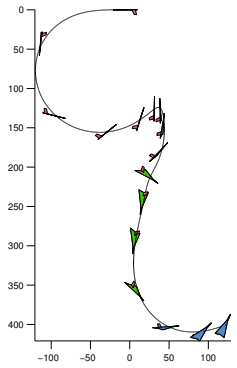


Figure 8.17: Navion Loop with Fall Out

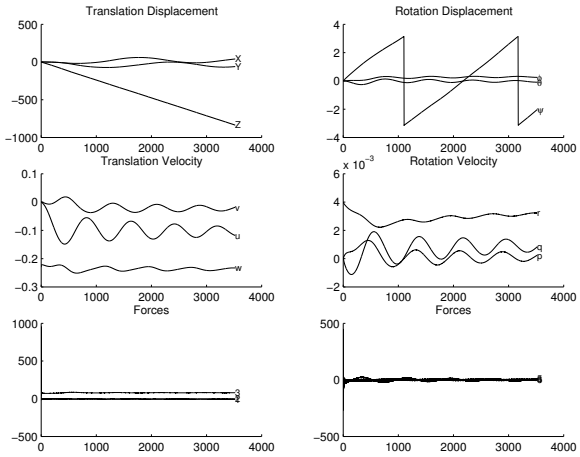


Figure 8.18: Navion Spin Motion

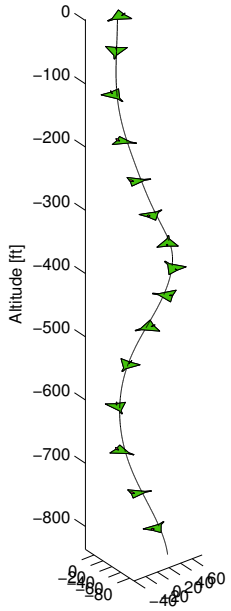


Figure 8.19: Navion Spin Trajectory

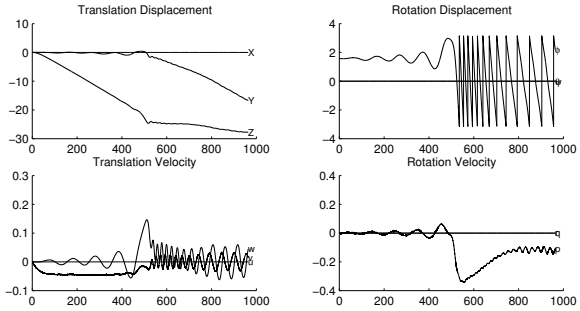


Figure 8.20: Wedge Time History

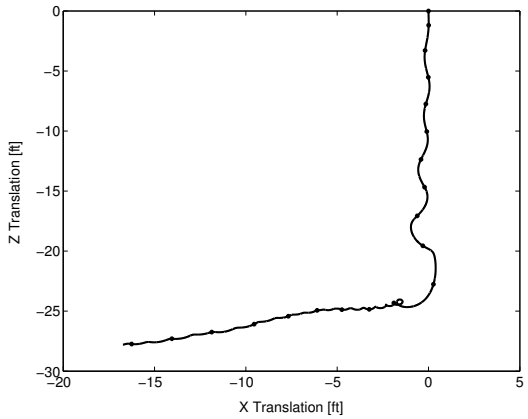


Figure 8.21: Wedge Trajectory

Wedge Drop

This case simulates a 10% thick wedge, free to translate and rotate, being dropped in air. The concept is to release the wedge flat from rest. A pair of vortices form off the sharp wedge edges, which eventually degenerate into a vortex street with an asymmetrical pressure distribution. The translational and rotational motions are shown in Figure 8.20. The wedge's CG trajectory is shown in Figure 8.21 with tick marks at each 1/2 second.

Initially, alternating vortices appear to form causing a tumbling leaf motion. Interestingly, the wedge transitions to a *lift generating* Magnus motion with an apparent L/D of near 4. This experiment qualitatively matches the tumbling behavior of thin, light strips.

8.6 Rigid Body Dynamics Summary

A quaternion-based rigid body dynamics solver was successfully applied to an inviscid CFD code. This gave the laboratory the capability to simulate non-linear flight dynamics.

Chapter 9

Conclusions

This chapter discusses this project's conclusions. The objective was to investigate methods and their efficiency for simulating coupled fluid-structure interactions and large deflection vehicle motions. An arbitrary order Galerkin Navier Stokes CFD solver was constructed and tested. A 1st order Lagrangian viscous CFD solver was constructed. A rigid body dynamics solver was constructed and tested. The project's hypothesis as presented in Section 1.4 was

> The prediction efficiency of super maneuvering, deforming and constantly remeshing three-dimensional unsteady Navier-Stokes computational fluid dynamics is improved by moving from linear to higher-order simulation methods.

This hypothesis needs qualification regarding the subjective term *higher-order*. The hypothesis is accepted as

> The prediction efficiency of super maneuvering, deforming and constantly remeshing unsteady Navier-Stokes computational fluid dynamics is improved by moving from linear to 2nd and 3rd order simulation methods.

9.1 Conclusions and Recommendations

This section gives specific conclusions and recommendations.

- The basic Euler frame equations of motion alone are not sufficient for simulating boundary motions. A non-inertial frame is sufficient for the specific case of rigid body motion. The Arbitrary Lagrangian Euler frame and the Lagrangian frame are both practical for arbitrary moving and deforming boundary simulations.

- For rigid body dynamics simulations, quaternion tracked orientation is robust. This project strongly suggests simulating rigid body dynamics using quaternions rather than Euler angles.

- The Galerkin method is expensive for finite element based CFD. The rate limiting computation is the Galerkin force integration. Furthermore, Galerkin methods require global operations,

making parallelization more difficult.

- Increasing order reduces the grid element count at the expense of grid sensitivity.
- For unsteady solutions, the total solution time appears to have a minimum at 3rd order.
- Higher order CFD increases the numerical and geometrical complexities. Stabilization and boundary conditions are especially affected. This project recommends developing higher order stabilization as a prerequisite to a higher order CFD solver.
- There is no maximum-timestep advantage for *compressible* Lagrangian CFD. The Lagrangian frame's (and Euler frame's) maximum timestep is also restricted by wave propagation.

9.2 Addressing Cowan's *Future Challenges*

Previously, Tim Cowan developed the laboratory's current CFD solver, Euler3d[14]. His dissertation's conclusions contained a *Future Challenges* section to discuss his concerns. Since the present project is based on Euler3d, Cowan's discussion points must be addressed.

Parallel Processing Not directly addressed. The higher order analysis showed a potential for order of magnitude efficiency improvement equivalent to workstation level parallel processing. The Lagrangian code was specifically designed for future parallelization.

Convergence Requirements The context that Cowan discusses convergence requirements is an implicit solution of a global mass matrix. In general however, solution convergence remains an open problem. The current solver provides a solution quality calculation to adjust the time integration routines. A poorly specified solution quality hinders temporal solution convergence.

Time Step Requirements Satisfied. The current adaptive time step routine allows for changing the timestep based on solution quality. Capturing variable timescales becomes easier without requiring *a priori* knowledge. Runge Kutta allows full precision restarts with arbitrary timesteps.

Grid Generation Partially satisfied. Higher order elements allows for coarser grids. Fewer required elements opens the possibility (and honestly, the constraint) of frequent remeshing.

Improved Dissipation Model Unsuccessfully addressed. Cowan's discussion implies Galerkin residual based dissipation. Yet, dissipation models for numerical solutions are an open topic. Entropy based dissipation is theoretically optimal. This project had implementation issues with an entropy governing equation dissipation model.

Rigid-Body Dynamics Satisfied. A fully 6 degree of freedom rigid n-body dynamics solver was implemented[58] in both euler3d and the current solver.

Aerodynamic Modeling Not addressed here. Aerodynamic modeling refers to system identification of aerodynamic motions and forces. This topic is addressed in O'Neill[57] and Babcock[1].

9.3 Future Work

9.3.1 Grid Generation

Our current grid generation capability absolutely must be improved. While the grid generation tools are effective, they are not maintainable or extendable; the last known maintenance performed on surface and volume was conversion to double precision in 2003. To the best knowledge of the author, the technical methodology of *surface* and *volume* are excellent and should be revised rather than rejected.

With the capability to simulate dynamic solutions, the grid generation constitutes a major constraint. For viscous boundary layers, triangular and tetrahedral element include too many lateral degrees of freedom and not enough transverse degrees of freedom (i.e., boundary layers have steep velocity gradients in only one direction). Future inclusion of different element types could alleviate this.

9.3.2 ALE grid motion

This project did not formally test the ALE boundary motion component of the governing equations. Arbitrary boundary motion requires a term only provided by ALE and the Lagrangian frame. The ALE governing equations requires a specified grid motion. This in turn requires a grid motion distribution algorithm. Poorly specified algorithms create kinked, needled, or reversed elements. At this point, the best approach might involve solving a Poisson type equation for grid velocity. The key will be to smoothly describe surface motions without discontinuities.

9.3.3 Parallel Processing

Parallel processing is just not going away. As CPU frequency has stagnated for physical limitations, adding multiple CPUs is now an established trend. For example, Fused Multiply Add (FMA) will be available in consumer level CPUs in 2012. FMA is a fundamental operation for CFD with most algorithms containing a $y = ax + b$ term. This will allow current vector operations to be more

efficiently compiled. Additionally, this could make parallel processing even more critical. The key likely lies with decoupled algorithms.

9.3.4 Structural Dynamics

As we trend away from static grids, non-linear structural dynamics becomes more critical. The current elasticity models are typically linearized about a given state. At some point, it becomes easier to simulate the whole structural system rather than to create a reduced order model, as typically done through frequencies and mode shapes. This becomes especially noticeable in hypersonic and thermal systems because the mean shape changes.

9.3.5 Visualization

Dynamic solution capabilities strain the present visualization tools. As our simulations become more dynamic, we expect solution grids to become more time dependent.

Moving to a visualization library freed us from developing a copy of already existing functionality. We have the capability now at zero price; the cost is that we develop the translator between our CFD data formats and the library.

Currently limited to quadratic visualization, VTK plots higher order fields with subdivision to linear elements. Our brain has to do the actual curvature interpolation. Repeated half-subdivision is possible, but it's not implemented in VTK. Even VTK's newest "generic" dataset tools use subdivision to linear elements; it runs a grid generator to form the subdivided grid.

9.3.6 Iteration Techniques

Investigating the iteration algorithms might yield productive results. Theoretically, Jacobi iterative solvers prefer preconditioned matrices. Trading iteration speed with preconditioning computations likely produces a non-trivial optimum. Preconditioner integration with parallel processing presents an additional constraint. Alternative algorithms might also contain unknown advantages.

9.3.7 Geometry Parametric Description

Surface geometry is currently limited to linear elements. Linear element restrict the use of an Euler solver without accepting incorrect surface fluxes (See Cowan for more information on the development of an Euler solver with linear elements). The current surface gridding software *surface* already contains the necessary curvature descriptions for nonlinear geometry elements.

Work on this topic will be simplified because adding variable Jacobian elements will only require creating and using a function that creates the appropriate Jacobian as coefficients and then multiplying this set of coefficients by the existing integral calculations.

9.3.8 PETSc

PETSc is a library parallelization library. Integrate PETSc into euler3d would allow for parallelization. This would allow arbitrary non-shared memory calculations. Dropping the PETSc code into certain locations could give parallel processing capability with a couple months of work. A parallelization library such as PETSc[2] could provide a significant non-SMP (shared memory) speedup. It is suggested to use PETSc rather than continuing the OpenMP routine described and tested in 3.7.1. The future of computing is likely non-SMP and NUMA memory architectures. In general, canned routines for parallelization should be preferred over an in-house attempt.

9.3.9 Time Integration in euler3d code

Implementing the time integration methods described could potentially improve the efficiency and performance of the Euler3d series of solvers. Of particular interest is that the euler3d code currently uses a backwards difference time integration routine with a constant timestep. This constant timestep hinders timely solutions of variable unsteady simulations.

Bibliography

[1] Deric A. Babcock. Aircraft Stability Derivative Estimation from Finite Element Analysis. Master's thesis, Oklahoma State University, 2004.

[2] Satish Balay, Kris Buscelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.0.0, Argonne National Laboratory, 2008.

[3] Pinhas Z. Bar-Yoseph and Eduard Moses. Space-Time spectral element methods for unsteady convection-diffusion problems. *International Journal of Numerical Methods for Heat & Fluid Flow*, 7(2/3):215–235, 1997.

[4] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[5] Peter Bartello and Stephen J. Thomas. The cost-effectiveness of Semi-Lagrangian advection. Technical report, Atmospheric Environment Service, Quebec, 2007.

[6] F. Bassi and S. Rebay. A high-order discontinous finite element method for the numerical solution of the compressible Navier-Stokes equations. *Journal of Computational Physics*, 131: 267–279, 1997.

[7] Andrew Bennett. *Lagrangian Fluid Dynamics*. Cambridge University Press, Cambridge, UK, 2006. ISBN 978-0-521-85310-1.

[8] Anthony A. Boeckman. Accelerating Computational Fluid Dynamics Based Aeroelastic Analysis Using Distributed Processing. Master's thesis, Oklahoma State University, 2003.

[9] Daryl L. Bonhaus. *A Higher Order Accurate Finite Element Method for Viscous Compressible Flows*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, November 1998.

[10] John P. Boyd. *Chebyschev and Fourier Spectral Methods*. Dover, Mineola, NY, 2 edition, 2000.

[11] Colin W. Brown. Evaluating integration methods and examining sensitivity to temporal and spacial variations in euler2d. Master's thesis, Oklahoma State University, Stillwater, OK, May 2010.

[12] Alex Cenko. Experience in the use of computational aerodynamics to predict store release characteristics. *Progress in Aerospace Sciences*, 37(5):477–495, July 2001.

[13] T. R. Chandrupatla and A. D. Belegundu. *Introduction to Finite Elements in Engineering.* Prentice-Hall, New Jersey, 3rd edition, 2002. ISBN 0-13-061591-9.

[14] Timothy John Cowan. *Finite Element CFD Analysis of Super-Maneuvering and Spinning Structures.* PhD thesis, Oklahoma State University, Stillwater, OK, May 2003.

[15] W. La Cruz and M. Raydan. Residual iterative schemes for large-scale nonsymmetric positive definite linear systems. *Comput. Appl. Math.*, 27(2), 2008.

[16] J. Donea, S. Giuliani, and J. P. Halleux. An Arbitrary Lagrangian-Eulerian finite element method for transient dynamic fluid-structure interactions. *Computer Methods in Applied Mechanics and Engineering*, 33:689–723, 1982.

[17] Mark Drela. XFOIL: An analysis and design system for low Reynolds Number aerofoils. *Lecture Notes in Engineering*, 54, 1989.

[18] Uwe Fey, Michael König, and Helmut Eckelmann. A new Strouhal-Reynolds-number relationship for the circular cylinder in the range $47 < Re < 2 \times 10^5$. *Phys. Fluids*, 10(7):1547–1549, July 1998.

[19] K. Fidkowski. Development of a Higher-Order Solver for Aerodynamic Applications. 42nd AIAA Aerospace Sciences Meeting and Exhibit, Jan 2004.

[20] Krzysztof J. Fidkowski. *A High-Order Discontinous Galerkin Multigrid Solver for Aerodynamic Applications.* PhD thesis, MIT, Cambridge, MA, June 2004.

[21] C. C. Fisher and A. S. Arena. On The Transpiration Method For Efficient Aeroelastic Analysis Using An Euler Solver. In *AIAA Atmospheric Flight Mechanics Conference*, AIAA Paper 96-3436, 2005.

[22] C. W. Gear and Y. Saad. Iterative solution of linear equations in ODE codes. *SIAM J. Sci. Stat. Comput.*, 4(4):583–601, December 1983.

[23] P. Geuzaine, G. Brown, C. Harris, and C. Farhat. Aeroelastic dynamic analysis of a full F–16 configuration for various flight conditions. *AIAA Journal*, 41(3):363–371, March 2003.

[24] K. Gupta, T. Doyle, and E. Hahn. AE/Flutter Simulation and Flight Test Correlation of the F/A-18 Active Aeroelastic Wing (AAW). In *43rd AIAA Aerospace Sciences Meeting and Exhibit*, AIAA Paper 233–2005, 2005.

[25] K. K. Gupta. Development of a finite element aeroelastic analysis capability. *Journal of Aircraft*, 33(5):995–1002, September–October 1996.

[26] Kajal K. Gupta and John L. Meek. *Finite Element Multidisciplinary Analysis*. AIAA, Virginia, 2000. ISBN 1-56347-393-3.

[27] Ernst Hairer and Gerhard Wanner. *Solving ordinary differential equations II: Stiff and differential-algebraic problems*, volume 2. Springer-Verlag, Berlin, 2nd edition, 2010. ISBN 978-3-642-05220-0.

[28] Ernst Hairer, Syvert Paul Nørsett, and Gerhard Wanner. *Solving ordinary differential equations I: Nonstiff problems*, volume 1. Springer-Verlag, Berlin, 2nd edition, 2008. ISBN 978-3-540-56670-0.

[29] Nathan Hariharan and Lakshmi N. Sankar. Unsteady Overset Simulation of Rotor-Airframe Interaction. *Journal of Aircraft*, 40(4):662–674, July-August 2003.

[30] A. Huerta. Progress in Arbitrary Lagrangian-Eulerian Analysis of Fluid and Solid Problems. *12th International Conference on Structural Mechanics in Reactor Technology*, B01/1, 1993.

[31] *Intel Fortran Compiler XE 12.0 User and Reference Guides*. Intel, 2011.

[32] Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, Cambridge UK, 1996. ISBN 0521556554.

[33] Yu Jinyun. Symmetric Gaussian Quadrature Formulae for Tetrahedronal Regions. *Computer Methods in Applied Mechanics and Engineering*, 43:349–353, 1984.

[34] Claes Johnson. *Numerical solution of partial differential equations by the finite element method*. Studentlitteratur, Lund, Sweden, 1987. ISBN 91-44-25241-2.

[35] George Em Karniadakis and Spencer J. Sherwin. *Spectral/hp Element Methods for Computational Fluid Dynamics*. Oxford, New York, 2 edition, 2005. ISBN 019852869.

[36] Osamu Kato. Some basic considerations on angles describing airplane flight maneuvers. *Journal of Guidance, Control, and Dynamics*, 17(2):378–384, March-April 1994.

[37] Patrick Keast. Moderate-degree Tetrahedral Quadrature Formulas. *Computer Methods in Applied Mechanics and Engineering*, 55:339–348, 1986.

[38] H. K. Khalil. *Nonlinear Systems*. Prentice-Hall, New Jersey, 3rd edition, 2002.

[39] Robert M. Kirby and Spencer J. Sherwin. Stabilisation of spectral/*hp* element methods through spectral vanishing viscosity: Application to fluid mechanics modelling. *Computer methods in applied mechanics and engineering*, 195:3128–3144, 2006.

[40] Peter Hjort Lauritzen. *An inherently mass-conservative semi-implicit semi-Lagrangian model*. PhD thesis, University of Copenhagen, Copenhagen, Denmark, September 2005.

[41] Randall J. LeVeque. *Numerical methods for Conservation Laws*. Birkhäuser Verlag, Basel, 1 edition, 1992.

[42] Ben Q. Li. *Discontinuous Finite Elements in Fluid Dynamics and Heat Transfer*. Springer, 1 edition, 2006.

[43] L. Lijewski and N. Suhs. Time-accurate computational fluid dynamics approach to transonic store-separation trajectory prediction. *Journal of Aircraft*, 31(4):886–891, 1994.

[44] R. Löhner. *Applied Computational Fluid Dynamics Techniques: an introduction based on finite element methods*. John Wiley, New York, 1 edition, 2001.

[45] Rainald Löhner, Dmitri Sharov, Hong Luo, and Ravi Ramamurti. Overlapping unstructured grids. AIAA-01-0439, 2001.

[46] I. Lomtev, C.B. Quillen, and G.E. Karniadakis. Spectral/hp algorithms on unstructured meshes for compressible viscous flows. *Journal of Computational Physics*, 144(2):325–357, August 1998.

[47] Magda Martins-Wagner. On efficient numerical integration of p-version finite element stiffness matrices. Technical Report Internal Report No. 4, Institut für Mathematik and Bauinformatik, Neubiberg, Germany, November 2003.

[48] Marshal L. Merriam. An entropy-based approach to nonlinear stability. Technical Report TM 101086, NASA Ames, March 1989.

[49] E. Misawa. Nonlinear system analysis and control MAE 5463 class notes, Fall 2003.

[50] N. J. Moffitt. First Stages of a Viscous Finite Element Solver for Non-Inertial and Aeroelastic Problems. Master's thesis, Oklahoma State University, Stillwater, OK, December 2004.

[51] Nicholas J Moffitt, Cody W. Pinkerman, and Andrew S. Arena. Modeling Hypersonic Propulsion Using Finite Element CFD. AIAA Paper 2004-5174, AIAA Atmospheric Flight Mechanics Conference and Exhibit, Providence, RI.

[52] M. J. Moran and Howard N. Shapiro. *Fundamentals of engineering thermodynamics.* John Wiley & Sons, New York, 3 edition, 1996.

[53] G.F. Naterer and J.A. Camberos. *Entropy-based design and analysis of fluids engineering systems.* CRC, 2008.

[54] Greg. F. Naterer. *Heat transfer in single and multiphase systems.* CRC Press, Boca Raton, FL, 2003. ISBN 0849310326.

[55] Robert C. Nelson. *Flight Stability and Automatic Control.* McGraw-Hill, Boston, MA, 2 edition, 1998.

[56] Thomas E. Noll, John M. Brown, Marla E. Perez-Davis, Stephen D. Ishmael, Geary C. TIffany, and Matthew Gaier. Investigation of the Helios Prototype Aircraft Mishap Volume I Mishap Report. Technical report, NASA, January 2004.

[57] C.R. O'Neill. Improved system identification for aeroservoelastic predictions. Master's thesis, Oklahoma State University, 2003.

[58] C.R. O'Neill and A. S. Arena. Aircraft Flight Dynamics with a Non-Inertial CFD Code. 43rd AIAA ASM, January 2005.

[59] D.Z. Pan, M.J. Chao, and S.K. Chien. Euler Computations of Rotor Flows on Unstructured Rotating Meshes. *Journal of Aircraft*, 38(4):672–679, July–August 2001.

[60] W. F. Phillips, C. E. Hailey, and G. A. Gebert. Review of attitude representations used for aircraft kinematics. *Journal of Aircraft*, 38(4):718–737, July–August 2001.

[61] O. Pironneau. On the transport-diffusion algorithm and its applications to the navier-stokes equations. *Numer. Math*, 38:309–332, 1982.

[62] Stephan B. Pope. *Turbulent Flows.* Cambridge University Press, Cambridge, UK, 1 edition, 2000.

[63] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, Cambridge, 2 edition, 1992.

[64] Nathan Prewitt, Davy Belk, and Wei Shyy. Parallel computing of overset grids for aerodynamic problems with moving objects. *Progress in Aerospace Sciences*, 36(2):117–172, Feb 2000.

[65] Magdi Rizk and Jae M. Lee. Beggar code implementation of the (6+)DOF capability for stores with moving components. AIAA Paper 2004-1251, 42nd AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, January 2004.

[66] P. L. Roe. Approximate Riemann solvers, parameter vectors and difference schemes. *Journal of Computational Physics*, 43:357–372, 1981.

[67] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA, 2nd edition, 2003. ISBN 0-89871-534-2.

[68] J.R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[69] Wei Shyy et al. *Computational fluid dynamics with moving boundaries*. Taylor & Francis, Washington, DC, 1996. ISBN 1-56032-458-9.

[70] Brian L. Stevens and Frank L. Lewis. *Aircraft Control and Simulation*. Wiley, New York, NY, 1st edition, 1992.

[71] Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge, Wellesley, MA, 1986.

[72] R. F. Tomaro, F. C. Witzeman, and W. Z. Strang. Simulation of store separation for the F/A–18C using Cobalt60. *Journal of Aircraft*, 37(3):361–367, May-June 2000.

[73] Eleuterio F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer-Verlag, Berlin, 2 edition, 1997.

[74] Frank M. White. *Viscous Fluid Flow*. McGraw-Hill, Boston, 2nd edition, 1991. ISBN 007069712.

[75] Daniel Zwillinger. *CRC Standard Mathematical Tables and Formulae*. CRC Press, Boca Raton, FL, 30 edition, 1996.

Appendix A

Nomenclature and Abbreviations

The following is a list of general nomenclature and abbreviations used in this dissertation. Nomenclature associated with specific sections are defined in the appropriate section.

Abbreviations

1D One Dimensional

2D Two Dimensional

3D Three Dimensional

AAW Active Aeroelastic Wing

ALE Arbitrary Lagrangian Eulerian

Ale2D 2D ALE code developed for this dissertation

ASE Aeroservoelastic

CASELab Computational Aeroservoelasticity Laboratory

CFD Computational Fluid Dynamics

CFL Courant–Friedrichs–Lewy

CG Conjugate Gradient

CG Center of Gravity

CPU Central Processing Unit

DG Discontinuous Galerkin

euler2d/euler3d Existing 2D and 3D Eulerian CFD Codes

FE Finite Element

FMA Fused Multiply Add

GB Gigabytes

JDAM Joint Direct Attack Munition

Lagr2d Lagrangian Frame CFD code developed for this dissertation

L/D Lift to Drag ratio

MAC Mean Aerodynamic Chord

NACA National Advisory Committee for Aeronautics

NASA National Aeronautics and Space Administration

ns2d 2D Viscous Galerkin CFD solver

NUMA Non-Uniform Memory Architecture

ODE Ordinary Differential Equation

PDE Partial Differential Equation

PC Predictor Corrector

RAM Random Access Memory

RK Runge Kutta

RMS Root Mean Square

RSD Residual

SLSE Semi-Lagrangian Spectral Element

SMP Symmetric Multiprocessing

SSL Standard Sea Level

STARS Structural Analysis Routines

SU/PG Streamwise Upwind / Petrov Galerkin

VTK Visualization Toolkit

Fluid Nomenclature

α Angle of Attack

β Sideslip Angle

ρ Density

γ Ideal Gas Specific Heat Ratio

μ Viscosity

λ Bulk Viscosity

τ Viscous Stress Tensor

Φ Viscous Dissipation Function

$()^\star$ Non-dimensional Variable

$()_o$ Non-dimensional Reference

a Acoustic Speed

c_p Ideal Gas Constant Pressure Specific Heat

c_v Ideal Gas Constant Volume Specific Heat

C_D Coefficient of Drag

C_L Coefficient of Lift

C_P Coefficient of Pressure

div Divergence

e Total Specific Energy

E Total Energy

$\hat{e}$ Internal Specific Energy

$\hat{E}$ Internal Energy

F Flux

h Specific Enthalpy

H Total Enthalpy

M Mach Number

n Normal Vector

p Pressure

Pr Prandtl Number

q Dynamic Pressure

q Heat Flux Vector

R Ideal Gas Constant

Re Reynolds Number

s Specific Entropy

S Entropy

St Strouhal Number

t Time

T Temperature

u Velocity in x direction

v Velocity in y direction

U Velocity Vector

V Velocity Vector

CFD Nomenclature

ζ Barycentric Coordinate

Φ Structural Modeshapes

ϕ Basis Function

Π Potential Energy

Ω Element Area

Ξ Local Location

a Solution State Vector

d 1D basis order

F Galerkin Force

h Step Size

K Galerkin Stiffness

L_2 Root Mean Square Error Norm

M Galerkin Mass Matrix

N Number of Elements

$\mathbb{R}$ Dimension

P Basis Order

$\mathcal{O}$ Order

C^0 Continuous Values

C^1 Continuous Values and 1st Derivative

J Jacobian Matrix

$|J|$ Jacobian Determinate

$\lfloor$ Floor Function

VITA

Charles Robert O'Neill

Candidate for the Degree of

Doctor of Philosophy

Dissertation: HIGHER ORDER AND DYNAMIC CFD
FOR AEROELASTIC SIMULATIONS

Major Field: Mechanical and Aerospace Engineering

Biographical:

Education:

Private Pilot ASEL.
Received Aerospace Engineering B.S. degree from Oklahoma State University in 2001
Received Mechanical Engineering M.S. from Oklahoma State University in 2003
Completed the requirements for the degree of Doctor of Philosophy with a major in Mechanical and Aerospace Engineering from Oklahoma State University in December, 2011.

Name: Charles O'Neill Date of Degree: December, 2011

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: HIGHER ORDER AND DYNAMIC CFD FOR AEROELASTIC SIMULATIONS

Pages in Study: 175 Candidate for the Degree of Doctor of Philosophy

Major Field: Mechanical Engineering

This dissertation develops and analyzes higher order computational fluid dynamics simulations for coupled fluid-structure interactions as well as large deflection vehicle motions. The flow regime ranges across subsonic, transonic and supersonic Mach numbers and inviscid and viscid boundary conditions.

The basic Euler frame equations of motion alone are not sufficient for simulating boundary motions. For rigid body dynamics simulations, quaternion tracked orientation is robust. The Galerkin method is expensive and rate-limits finite element based CFD. Increasing order reduces the grid element count at the expense of grid sensitivity. Higher order CFD increases the numerical and geometrical complexities; stabilization and boundary conditions are especially affected. There is no maximum-timestep advantage for compressible Lagrangian CFD over Euler frame CFD. The prediction efficiency of super maneuvering, deforming and constantly remeshing unsteady Navier-Stokes computational fluid dynamics was found to improve by moving from linear to 2nd and 3rd order simulation methods.

ADVISOR'S APPROVAL: _____