To the Graduate Council:

I am submitting herewith a dissertation written by Kevin Kilpatrick entitled "A Preconditioned Krylov Subspace Approach to a Tightly Coupled Aeromechanical System." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Mechanical Engineering.

Allen J. Baker, Major Professor

We have read this dissertation
and recommend its acceptance:

Dr. Carroll E. Peters

Dr. K. C. Reddy

Dr. Roy J. Schulz

Accepted for the Council:

Interim Vice Provost and
Dean of The Graduate School

A Preconditioned Krylov Subspace Approach to a Tightly Coupled Aeromechanical System

A Dissertation Presented
for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Kevin Kilpatrick
August 2001

ABSTRACT

A tightly coupled approach is attempted to compute a modest fluid-structure interaction for high subsonic flow through a converging nozzle with deformable walls. A globally convergent Newton statement and a matrix-free GMRES linear equation solver are used to linearize and solve the coupled system of equations without explicitly forming the left hand side jacobian matrix associated with the Newton method. A variable forcing function term is successfully incorporated into the Newton statement to balance inner (linear) and outer (nonlinear) iterations. The fluid-structure system is solved for comparison purposes using a loosely coupled approach. Residual convergence stagnated in the tightly coupled system approach but converged successfully in the loosely coupled approach using the same coding for domain calculations.

A novel approach using time derivative preconditioning is incorporated to speed convergence of the GMRES linear equation solver. No algebraic preconditioning is used. The fluid flow equations showed significant improvements using the time derivative preconditioning method but the error term generated in the structural equations overwhelmed the physical solution increment.

The Taylor Weak Statement derivation of the finite element form of the fluid flow equations with time derivative preconditioning shows a strong connection to the Streamwise Upwind Petrov Galerkin (SUPG) method. This connection is exploited to develop a theoretical basis for the damping term and the time scale parameter common to the SUPG method.

TABLE OF CONTENTS

1. INTRODUCTION 1

2. APPROACH 6
   - Numerical Solver 7
   - Fluid Flow Solver 8
   - Structural Solver 8
   - Grid Motion 9
   - Coupled Fluid-Structure-Grid 9

3. NUMERICAL SOLVER 11
   - Global Newton Method (Outer Iteration) 12
   - Newton Iterations vs. Pseudo Time Steps 16
   - Preconditioned GMRES Linear Equation Solver (Inner Iteration) 19
   - Inexact Newton-GMRES with Variable Forcing Function 25
   - Termination Criteria 28

4. FLUID FLOW PRECONDITIONING 30
   - Background 30
   - Newton Time Stepping with Preconditioning 32
   - The Time Derivative Preconditioning Matrix 34
   - Discrete Implementation 38

5. DERIVATION OF A TAYLOR WEAK STATEMENT 50
   - One Dimensional Inviscid Fluid Flow with Conservation Variables 50
   - One Dimensional Inviscid Fluid Flow with Primitive Variables 56
   - Two Dimensional Inviscid Fluid Flow with Primitive Variables 62
   - Wave Analysis of Continuum Equation for Inviscid Fluid Flow in Primitive Variables 65
   - Wave Analysis of Discrete Equation for Inviscid Fluid Flow in Primitive Variables 66
   - Derivation of the SUPG τ Parameter 67
   - Summary 74

6. FLUID FLOW ON MOVING OR DEFORMING GRID 76
   - Grid Motion Strategies 76
   - Derivation of a Weighted Arbitrary LaGrange Euler (ALE) Statement 77
   - Geometric Conservation Laws 80

7. STRUCTURAL EQUATIONS 81
   - Solid Equations 82
   - Discrete Implementation 91
   - Locking of Thin Structures 97

8. DYNAMIC RELAXATION IN STRUCTURES 103
   - Background 103
   - Implicit Adaptive Dynamic Relaxation 104
   - Eigenvalue Estimates 116
   - Arnoldi Process for Extreme Eigenvalue Estimates 117

9. FLUID GRID MOTION 119
   - Background 119
   - Discrete Implementation 121

Solution Process 126

10. FLUID-STRUCTURE COUPLING 128
Continuum Interactions 128
Equivalence of Two and Three Field Methods 134
Discrete Interface and Interactions 143
Discrete Time Synchronization 145

11. ANALYSIS OF NUMERICAL SOLVER RESULTS 147
Directional Derivative Step Size Changes 147
Number of Krylov Vectors vs. Restarted GMRES Tests 148
Eisenstat-Walker Forcing Functions 153
GMRES Starting Vectors 155

12. ANALYSIS OF FLUID FLOW TWS/SUPG RESULTS 156
Numerical Solver 156
Residual Scaling 156
Coupon Test Cases 158
Straight Channel Test Cases 160
Nozzle Test Cases 166
Summary 190

13. ANALYSIS OF FLUID FLOW PRECONDITIONING RESULTS 192
TWS/SUPG Terms 192
Newton Time Stepping with Time Derivative Preconditioning 201
Residual Scaling as a Preconditioner 205
Mass Matrix Variations 206
Summary 208

14. ANALYSIS OF STRUCTURAL SOLVER RESULTS 209
Implementation and Validation 209
Summary 213

15. ANALYSIS OF ADAPTIVE DYNAMIC RELAXATION RESULTS 214
Analysis 214
Summary 226

16. ANALYSIS OF COUPLED FLUID-STRUCTURE SOLVER 227
Background 227
Tightly Coupled Fluid-Structure-Grid Solution Results 228
Summary 230

17. CONCLUSIONS 232
Numerical Solver 234
Fluid Flow 234
Structural Solver 235
Grid Motion 236
Coupled Fluid-Structure-Grid 236

BIBLIOGRAPHY 238

APPENDICES 247

A. EIGENVALUES 248
Rayleigh Quotient 249

Rayleigh Quotient Iteration 249
Eigenvalue Bounds 250
Lumped and Consistent Mass Matrices 252
Gershgorin Circle Theorem 253
Condition 253
Condition of Finite Element Basis 257
Condition of Discrete vs. Continuous Problems 258
Perturbed Matrix Inverse 259

B. GRAM SCHMIDT PROCESS 262
Traditional Gram Schmidt 263
Modified Gram Schmidt 264

C. CODE ARCHITECTURE 265

D. NUMERICAL SOLVER ROUTINES 269
KNSOLGMA 272
KFDGMRES 277
KDIRDER 281
KARNOLDI 282
GIVAPP 285
PARAB3P 286

E. FLUID FLOW ROUTINES 287
MASFLO 291
BLOK_RHS 294
BC1_RHS 308
BC2B_RHS 313
BC32_RHS 318
TAU_CALC 320
GAUSS 322
KFLOW1 323
BRIKSHAP 328
TRIASHAP 329
FLOW_LHS 330
BC1_LHS 333

F. SOLID ROUTINES 336
MASTER 340
STR_RHS 343
LFORCE 350
NLFORCE 353
LIN_MAT 354
NLIN_MAT 356
BRIK2ORD 359
STR_EIG 360
STR_LHS 363

G. GRID ROUTINES 367
GOGRID 371
GRID_RHS 373
LIN_GRID 379
GRID_MAT 382
GRID_LHS 384

H. COUPLED MASTER ROUTINES 388
Master Routine Operations 392
Right Hand Side (RHS) Operations 392
Left Hand Side (LHS) Operations 393
MAS_123 393
FSG_MAS 402
FSG_RHS 407
FSG_LHS 411

VITA 416

LIST OF TABLES

Table 11-1. Effect of Residual Scaling on Directional Derivative Step Size 148

Table 11-2. Impact of GMRES Restarts 151

Table 11-3. GMRES Restarts with Krylov Vector Changes 151

Table 11-4. Eisenstat-Walker Forcing Functions 154

Table 11-5. GMRES Starting Vectors 155

Table 12-1. Residual Scaling Test Cases 159

Table 13-1. Physical vs. Preconditioned Mass 193

Table 13-2. Preconditioning Parameters 194

Table 13-3. Preconditioning Parameters for Nozzle Test Case 195

Table 13-4. Convergence Times for Tau Cases 200

Table 13-5. Impact of Time Preconditioning on Nozzle Test Case Convergence Times 202

Table 13-6. Pseudo Time Step Size Studies 203

Table 13-7. Pseudo Time on Right Hand Side 204

Table 13-8. Energy Residual Scaling Effect on Nozzle Test Case Convergence Times 205

Table 13-9. Impact of Amplifying Wall Residuals on Nozzle Test Case Convergence Times 206

Table 13-10. Effect of Simplifying Weight Function for Mass Matrix on Nozzle Convergence Time 207

Table 13-11. Impact of Mass Matrix Lumping on Nozzle Test Case Convergence Times 207

Table 13-12. Impact of Initial Conditions on Nozzle Test Case Convergence Times 208

Table 14-1. Structural Coding Validation Test Cases 210

Table 16- 1. Fluid-Structure-Grid Convergence 230

LIST OF FIGURES

Figure 3-1. Sample Parabolic Step Length Reduction 14

Figure 3-2. Sample Eisenstat-Walker Forcing Function 27

Figure 5-1. Minimally Bounding Box 70

Figure 5-2. Wave Number Vector in Direction of Diagonal 73

Figure 5-3. Wave Number Vector in Direction of Velocity Vector 73

Figure 7-1. Structural Models 82

Figure 10-1. Aeroelastic Triangle of Forces 129

Figure 10-2. Fluid-Solid Interface 131

Figure 10-3. Fluid Force Vector 132

Figure 10-4. Signal Paths 133

Figure 10-5. Fluid-Structure Eigenvalue Clusters 135

Figure 10-6. Scaled Fluid-Structure Eigenvalue Clusters 137

Figure 10-7. Fluid-Structure Eigenvalue Cluster Components 139

Figure 10-8. Scaled Fluid-Structure Eigenvalue Cluster Movement 140

Figure 10-9. Mismatched Node Density 143

Figure 10-10. Matched Nodes 145

Figure 10-11. Fluid-Structure Update Strategy 146

Figure 11-1. RHS Residual Convergence Paths with Scaling 149

Figure 11-2. Rescaled Residual Convergence Paths 149

Figure 11-3. Change in Convergence Time with GMRES Restarts 151

Figure 11-4. Nonlinear Iteration Count with GMRES Restarts 152

Figure 11-5. Nonlinear Convergence Histories with GMRES Restarts 152

Figure 12-1. Simple Coupon 159

Figure 12-2. Residual Scaling Results 159

Figure 12-3. Slanted Coupon with Walls 161

Figure 12-4. Slanted Random Variable Coupon Results 161
Figure 12-5. Straight Channel Test Case 161
Figure 12-6. Straight Channel Entropy Profile 163
Figure 12-7. Straight Channel Total Enthalpy Profile 163
Figure 12-8. Enthalpy Surface with Physical Jacobian at Convergence Stagnation 165
Figure 12-9. Enthalpy Plot with Physical Jacobian at Convergence Stagnation 165
Figure 12-10. Nozzle Grid 167
Figure 12-11. % Total Enthalpy Surface for Nozzle Flow Test Case 168
Figure 12-12. % Total Enthalpy Profile for Nozzle Flow Test Case 168
Figure 12-13. Nondimensional Entropy Surface for Nozzle Flow Test Case 169
Figure 12-14. Nondimensional Entropy Profile for Nozzle Flow Test Case 169
Figure 12-15. Pressure Profile for Nozzle Flow Test Case 171
Figure 12-16. Temperature Profile for Nozzle Flow Test Case 172
Figure 12-17. X-Velocity Profile for Nozzle Flow Test Case 172
Figure 12-18. Y-Velocity Profile for Nozzle Flow Test Case 173
Figure 12-19. Velocity Magnitude Profile for Nozzle Flow Test Case 173
Figure 12-20. Velocity Vectors 173
Figure 12-21. Extended In/Outflow Boundaries 175
Figure 12-22. Wall Grid Refinement 175
Figure 12-23. % Enthalpy with Wall Refinement 175
Figure 12-24. Nondimensional Entropy with Wall Refinement 176
Figure 12-25. Full Nozzle Test Case 176
Figure 12-26. Fully Refined Grid Nozzle 176
Figure 12-27. Preconvergence Refined Grid Pressure Profile 178
Figure 12-28. Preconvergence Refined Grid Temperature Profile 178
Figure 12-29. Preconvergence Refined Grid Entropy Profile 179
Figure 12-30. Preconvergence Refined Grid Entropy Surface 179
Figure 12-31. Preconvergence Refined Grid Enthalpy Profile 180

Figure 12-32. Preconvergence Continuity Residual Surface 181
Figure 12-33. Preconvergence X-Momentum Residual Surface 181
Figure 12-34. Preconvergence Y-Momentum Residual Surface 182
Figure 12-35. Preconvergence Y-Momentum Residual Profile 182
Figure 12-36. Preconvergence Energy Residual Surface 183
Figure 12-37. Preconvergence Energy Residual Profile 183
Figure 12-38. Preconvergence Energy Residual Crossflow Profile 184
Figure 12-39. Entropy w/o Damping 187
Figure 12-40. % Enthalpy w/o Damping 187
Figure 12-41. Wall Pressure Change w/o Damping 188
Figure 12-42. Wall Temperature Change w/o Damping 188
Figure 12-43. Wall X-Velocity Change w/o Damping 189
Figure 12-44. Wall Y-Velocity Change w/o Damping 189
Figure 13-1. Tau Forms 199
Figure 13-2. Impact of Pseudo Time Stepping on SRAND 201
Figure 14-1. Finite Element Representation of a Beam in Tension 210
Figure 14-2. Finite Element Representation of a Block in Shear 210
Figure 14-3. Slender beam with point load 212
Figure 14-4. Results of slender beam with point load 212
Figure 14-5. Slender beam with distributed load 212
Figure 14-6. Results of slender beam with distributed load 213
Figure 15-1. Baseline System Convergence 215
Figure 15-2. Baseline w/ Restarted GMRES(40) 216
Figure 15-3. GMRES Restarts 216
Figure 15-4. Comparison of ADR Parameters 218
Figure 15-5. Deflections Near the Tip 221
Figure 15-6. Baseline Deformation 223
Figure 15-7. Optimum ADR Deformation 223

Figure 15-8. Inertial Dominated Deformation 224
Figure 15-9. Damping Dominated Deformation 224
Figure 15-10. Baseline with Limited Krylov Vectors 225
Figure 16-1. Coupled Fluid-Structure Geometry 228
Figure 16-2. Computer Performance 229
Figure C-1. Code Hierarchy 266
Figure C-2. Right Hand Side Residual Hierarchy 268
Figure C-3. Left Hand Side Residual Hierarchy 268
Figure D-1. KNSOLGMA Code Hierarchy 271
Figure D-2. KARNOLDI Code Hierarchy 271
Figure E-1. Fluid Flow Master Hierarchy 288
Figure E-2. Right Hand Side Residual of Fluid Flow 289
Figure E-3. Left Hand Side Residual of Fluid Flow 289
Figure F-1. Structural Solver Master Hierarchy 337
Figure F-2. Right Hand Side Structural Deformation Residual 338
Figure F-3. Left Hand Side Structural Deformation Residual 339
Figure G-1. Fluid Grid Deformation Master Hierarchy 368
Figure G-2. Right Hand Side Fluid Grid Deformation Residual 369
Figure G-3. Left Hand Side Fluid Grid Deformation Residual 370
Figure H-1. Loosely Coupled Solution Process 390
Figure H-2. Tightly Coupled Solution Process 390
Figure H-3. Top Level Module Interactions 391

Volume 1

CHAPTER 1

INTRODUCTION

Fluid-structure interaction problems are pervasive in engineering. They cover a wide range of applications from aeroelasticity and panel flutter to bridges and dams to exhaust stacks and cables to propellers and turbomachinery. The results of small experimental models for complex structures are difficult to scale to full size and the full size failures are often dramatic when they happen. Fortunately, analytic solutions and experiments do exist for some common structural shapes and nondimensional scaling parameters are used to determine design loads.

The fluid and structure communicate through surface loads and shapes. The fluid exerts a load on the structure through pressure forces normal to the surface and shear forces tangent to it. If the fluid is a gas, the shear forces (friction) are often omitted. This is practical but dangerous as the nonlinear effects generated by the viscosity within the fluid may radically change the pressure distributions. Also, the frictional forces can be significant if taken over large areas such as the surfaces of a Boeing 747 aircraft.

The solid communicates with the fluid through its shape or geometry. The fluid typically flows around a solid shape or through channels in its body designed for the purpose. A change in the load distribution on the body may cause it to change shapes, thus altering the load distribution, which again drives a different shape. This process can be cyclic and unstable.

As structural shapes and their systems get more complicated, more sophisticated tools are needed to predict the outcome of fluid induced motions. For example, the response of a high aspect ratio solid block to a distributed side load may be predictable, but if the block is a skyscraper of modern construction with active sway dampers, the response can be quite complicated. Likewise, classic aeroelastic problems such as wing design are still at the forefront of research. The large aircraft manufacturers use sophisticated codes to optimize wing profiles but these are immediately coupled with structural models to estimate weight, wing twist and lateral deformation, internal volume available for fuel, etc.

Many computer codes have been developed over the last 50 years to assist in predicting the performance of fluid-structure systems. Typically, an initial shape is selected based on design criteria and

objectives. Then, fluid calculations about the structure are used to predict the loads on a given surface and the total load. Structural designers will either design a system capable of handling the loads or assess a predetermined system's ability to handle the fluid loads in combination with any other loads that may be present. The loads may lead to unacceptable structural response either as steady state deformations or as transient deformations, or both. The deformed system creates a new interaction with the fluid and the process may cycle repeatedly. Transient responses also add new dimensions of inertia, structural damping, unsteady aerodynamics and control system response.

Engineers have historically developed computer codes to assist in analyzing the fluid or structure in isolation. However, demands for a faster design process and the desire to understand nonlinear fluid-structure interactions drove demand for coupled codes. Structural engineers typically added simple fluid models to their complicated structural codes and fluid dynamicists typically added simple structural models to their complicated fluid codes. These approaches are still prevalent today and the technical literature is full of examples.

Codes that contain full fidelity of both the fluid flow dynamics and structural deformation modes simply don't exist. There have been significant efforts towards developing these in the past but the efforts quickly became bogged down by complexity, inflexibility, and an insatiable need for more computational horsepower.

Fortunately, a very good and practical alternative takes advantage of the best of the fluid and structural codes without adding complexity or limiting flexibility. This is the (obvious) partitioning of the problem into its various disciplines and iterating between the existing codes to a converged solution. This approach follows the same template as the design process and has been proven to work quite well.

However, the intent of this dissertation research is to revisit the fully coupled (monolithic) approach to fluid-structure modeling to determine if there is a way to use computer code that has high fidelity for both fluids and structures in a tightly coupled fashion without adding complexity or special requirements. Modern partial differential equation solvers such as the GMRES (Generalized Minimum RESidual) method of [Saad and Schultz, 1986] do not restrict the problem to symmetric, positive definite systems so, theoretically, any combination of problem statements is possible. In addition, matrix-free

implementation of the "left hand side" removes the need to develop specialized cross coupling terms between the systems and eliminates the need to form and store the numerical (linearization) matrix. This also eliminates issues associated with node numbering, storage strategies, and matrix bandwidth.

In this research project, I attempted a monolithic approach to solving a coupled fluid-structure-grid system to steady state using a matrix-free GMRES solver and a globally convergent quasi-Newton method. Additive preconditioners were implemented as artificial mass matrices and stepped forward in pseudo time to speed convergence. The primary fluid-structure system simulated was representative of the nozzle section in large flexible-wall wind tunnels used by the United States government and aerospace industry. The physical system dimensions included a nozzle 8 feet long at the centerline with a 2 foot (from centerline) inlet and a 1 foot exit (area ratio = 2). The system was modeled as two-dimensional plane flow of inviscid air flowing through a simple $24x8$ node subsonic nozzle with a deformable wall and symmetry along the centerline. The physical wall was 1.5-inch thick mild steel with rigid boundaries at the inlet and exit and an external pressure load equal to the nozzle exit pressure. The wall was represented by continuum solid equations on a $24x2$ grid using bilinear quadrilateral elements. The fluid flow wall nodes were coincident with the structural nodes so no interpolation was needed at the fluid-structure interface to align nodal values. The inviscid fluid flow acted on the structure through the calculated wall pressure distribution. In turn, the structure deformed under the load causing similar deformations in the fluid flow grid. The fluid grid was treated as a continuum solid with Dirichlet conditions on wall node displacements. The deformed wall nodes created displacements for the nodes internal to the fluid flow grid through a discrete structural calculation. The fluid flow was then recalculated on the deformed fluid grid and the process iterated to convergence.

During the course of the investigation I advanced the state of the art, demonstrated techniques or discredited approaches in numerical methods, computational fluid dynamics, structural dynamics and coupled fluid-structure interactions:

- Proved the benefits of using an additive preconditioner to speed convergence of a globally convergent Newton iteration using a GMRES solver for the inner solution iterations

- Provided direct comparison, hence beneficial assessments, of pseudo time stepping versus pure Newton iterations for both fluid flow and structural equations
- Showed the correct form for additive preconditioners, implemented as a pseudo time step rather than a multiplicative preconditioner in fluid flow calculations
- Developed, documented and implemented various residual scaling strategies to overcome numerical stiffness created by residual orders of magnitude for fluid flow equations, structural equations, and coupled fluid-structure systems
- Implemented and documented the impact and influence of the number of Krylov vectors and restarts allowed in GMRES for both fluid flow and structural problems
- Implemented and documented the benefits of halting GMRES prior to full convergence of the inner iteration to speed the total solution process
- Implemented and documented the benefits of the Eisenstat-Walker forcing function for nonlinear iterations and developed an optimum starting condition for both fluid flow and structural equations
- Developed and implemented a Taylor Weak Statement/Streamwise Upwind Petrov Galerkin (TWS/SUPG) formulation for a finite element approach to inviscid fluid flow using (dimensional) primitive variables
- Developed and documented the theoretical basis for the TWS/SUPG time scale parameter, τ, and demonstrated significant performance improvements over the current state of the art
- Developed the correct form for a TWS/SUPG damping term for finite element fluid flow codes and demonstrated its impact on both stability and consistency
- Developed and implemented the correct form of the weight function to use in a finite element weak statement formulation of the inviscid fluid flow equations in primitive variables
- Documented the choices for free parameters and incorporated time derivative preconditioning into TWS/SUPG fluid flow equation with pseudo time stepping
- Developed both monolithic and partitioned approaches to solving coupled fluid-structure problems and implemented a matrix-free solution process while providing a direct comparison of convergence results

- Documented computational shortfalls of the monolithic solution process for coupled fluid-structure problems associated with the continued solving (oversolving) of converged domains while trying to get difficult domains to converge
- Developed the theoretical basis and implemented preconditioned structural equations using an implicit form of the Adaptive Dynamic Relaxation technique
- Proved the benefits of additive preconditioning by dramatically speeding convergence of the inner solution of a thin structure, diagnosed active terms, and documented adverse impact on the outer iteration

The GMRES method worked well but uncertainty over the number of Krylov vectors to include and the impact of restarts drove the need for a robust preconditioner to guarantee convergence. The additive preconditioner used in the fluid flow calculations proved to have significant benefits but still required restarts in GMRES. The TWS/SUPG form of the finite element fluid flow equations worked well and the theoretical development of the time scale parameter yielded significant improvements over the current state of the art.

The structural preconditioner eliminated the need to restart GMRES but the errors created were unacceptable to the nonlinear (outer) iteration.

The monolithic approach to the coupled fluid-structure problem proved to have significant flaws even with the matrix-free implementation. Common code was used to generate left and right hand side vectors for both the monolithic and partitioned approaches and time-derivative preconditioning enhanced convergence. However, the monolithic solution process was not competitive with a partitioned approach in terms of its rate of residual convergence due to excessive read/write operations and oversolving in converged domains. The partitioned approach was successful with accuracy equal to that of calculations on the individual domains in isolation.

CHAPTER 2

APPROACH

The fluid-structure-grid problem has many facets and draws on several different engineering and mathematical disciplines. The overarching strategy is outlined below but the background on each of the different areas is saved for those specific chapters.

At the top level, there are three sets of field equations: fluid flow on a fixed or moving grid, structural deformation under external load, and a description of fluid grid motion driven by the structural deformations. All the field equations have the common and familiar form

$$Ax = b \tag{2-1}$$

The "left hand side" is an incremental form of conservation laws linearized about the current state where A is a matrix of partial derivatives and x is an incremental change to the current state. The "right hand side" is a nonlinear form of the conservation laws where b is the residual vector generated by the current state vector.

The fluid grid and structural grid nodes are coincident at the fluid-structure interface. Otherwise a fourth set of field equations would be needed to describe the relationship between fluid and structure grid nodes. The intent is to add pseudo time terms to the left hand side of a Newton statement for the coupled systems of equations to act as a preconditioner and speed numerical convergence of the linearized system of equations. The preconditioner is developed for each of the computational domains (fluid, structure, and grid) separately and tested for effect. The well conditioned individual systems are then tightly coupled into a single, monolithic set of equations and tested for convergence and accuracy. If appropriate, the preconditioning terms are modified or a new one added to control convergence of the coupled system.

Unlike the multiplicative preconditioners commonly used today such as incomplete factorizations [Axelsson, 1994][Barrett, et.al., 1993], the preconditioning applied in this effort is additive. Incomplete factorization preconditioners require a left hand side matrix to be formed at least once at the start of the solution process before so it can be factored into the preconditioner. An additive preconditioner has no such requirement and matrix-free implementations of the left hand side linearization matrix may be used.

The nonlinear equations are assembled in strict conservation law form on the right hand side of the Newton statement but a linearized form is used on the left hand side. The left hand side equations are fully integrated to the same order as the right hand side. The individual linearized element integration matrices are formed at the same time as the right hand side residual and stored unassembled in a compact form for faster calculation of the left hand side linear residuals.

All codes used in the dissertation were written and tested in the MATLAB computational environment and run on a 200MHz, Pentium II personal computer.

Numerical Solver

Details and references for each aspect of the numerical solver are described in Chapter 3. The numerical solver puts to test some of the latest developments in partial differential equation solvers. The restarted Generalized Minimum Residual, or GMRES, method is implemented to solve iteratively the linear system of equations within a globally convergent Newton statement of the coupled field equations. The Newton statement forms a directional derivative for each new search direction, enabling a matrix-free implementation of the left hand side. The Eisenstat-Walker algorithm generates a variable sequence of forcing functions that determines the exit criteria for each linear solution. The implementation relies heavily on the code and theoretical developments in [Kelly, 1995]. The major steps in the numerical solution are:

1. form a linearized set of discrete equations and a globally convergent Newton method
2. solve the linear set of equations using the GMRES iterative solution technique
3. control memory requirements by creating a matrix free implementation using forward difference directional derivatives to simulate the Jacobian matrix-vector product
4. control memory requirements for Krylov subspace vectors using restarted GMRES
5. accelerate convergence by using physics based (rather than algebraic) preconditioning
6. balance inner and outer iterations using the Eisenstat-Walker approach to create variable forcing functions in the outer iteration

Fluid Flow

The discretized fluid flow equations are developed in detail in Chapters 4 through 6. The fluid flow solver implements a two dimensional finite element formulation of the inviscid, compressible Euler equations using linear triangle and bilinear quadrilateral elements. The equations are developed around the primitive flow variables: pressure, temperature and the velocity vector.

The time derivative preconditioning terms are based on the approach outlined in [Merkle, 1995]. The intent is to precondition the eigenvalues of the fluid flow continuum description and adjust for any ill conditioning created by the discretization. A Taylor Weak Statement analysis [Baker and Kim, 1987] is used to develop the appropriate damping terms for the primitive variable formulation given the preconditioning strategy.

The structural response to the fluid loads is expected to change the geometry of the flow domain. Rather than developing a new grid for the new geometry, the grid shape from the original geometry is warped to match the new boundaries. The motion of the grid nodes is taken into account in the fluid code using the Arbitrary Lagrange Eulerian (ALE) approach.

Structural Solver

The details of the structural solver are provided in Chapters 7 and 8. The structural solver implements continuum finite elements based on the Navier equations rather than specialized elements such as beams, plates and shells. The code follows the Total Lagrangian approach and is based heavily on the methods outlined in [Zienkiewicz and Taylor, 1989]. Both linear and nonlinear stiffness matrices are incorporated using linear triangle, bilinear quadrilateral, and biquadratic quadrilateral elements.

The structural solver also implements a time derivative preconditioning matrix to remove numerical stiffness. The approach is based on an implicit implementation of the Adaptive Dynamic Relaxation algorithm outlined in [Oakley and Knight, 1995].

Grid Motion

Chapter 9 provides the details for generating a moving fluid grid that conforms to the deformed structural shape. Rather than using algebraic methods or generating a completely new grid, the initial fluid grid is treated as if it were a continuum solid. The motion of the physical structure then deforms the fluid region to update the location of the fluid nodes. The field equations for the grid motion are the structural equations implemented over the geometry of the fluid grid. Boundary conditions are Dirichlet statements for wall node locations rather than applied loads. The Dirichlet boundaries are determined by the deformation of the wall as produced by the structural solver. The grid does not contribute to the stiffness of the wall. The grid "pseudo structure" may have artificial material properties if needed to provide a smooth grid in the interior of the deformed region. The solid equations satisfy the "geometric conservation laws" outlined in [Thomas and Lombard, 1978].

Coupled Fluid-Structure-Grid

Chapter 10 provides the details for the coupled fluid-structure-grid system. The coupled fluid-structure-grid problem is formed using an element by element approach. The appropriate field equations are used for each element and the combination of elements is assembled into a single vector reflecting the nodal residuals for the various conservation laws. Once assembled, the normalized 2-norm of the right hand side residual vector is checked against the convergence criteria and initiates a linear GMRES solution process.

The linearized left hand side residual vectors for the coupled system are formed using the same strategy as the right hand side. Since the logic in the GMRES algorithm simply requires the output of a matrix vector product (a vector), the product can be created using a finite difference approximation to the directional derivative. Each vector in the finite difference approximation is just a fully assembled residual vector for the coupled system. The vectors may be created using exactly the same nonlinear algorithms used in to form the right hand side residual or a simplified version (for speed). The coupled system vectors are formed element by element and assembled. This eliminates the need for special coding requirements,

as the residual vector for a particular set of field equations is the same (prior to assembly) whether the domain is solved independently or as part of a coupled system

CHAPTER 3

NUMERICAL SOLVER

Some form of numerical solver is at the heart of every computational engineering code. The success of the code depends critically on the match between the physical set of equations, the particular solution method, and the computing hardware. The solver may be customized to the characteristics of a particular set or class of procedures such as "skyline solvers" and conjugate gradient methods for symmetric structural equations, or Thomas algorithm tri-diagonal solvers for fluid flow calculations using Alternating Direction Implicit (ADI) schemes. Computer architecture and hardware is a primary consideration as the processor may be sequential, vector, or parallel, and core memory may be centralized or distributed and is sometimes shared between multiple processors.

The approach in this research project is to use a general-purpose non-linear partial differential equation solver designed for a single sequential processor with limited memory. The intent is to compare monolithic and partitioned solution approaches for fluid/structure interactions without changing the solver. The strategy is to:

1. form a linearized set of equations using a globally convergent Newton method
2. solve the linear set of equations using the GMRES iterative solution technique
3. control memory requirements by creating a matrix free implementation using forward difference directional derivatives to simulate the Jacobian matrix-vector product
4. control memory requirements for Krylov subspace vectors using restarted GMRES
5. accelerate convergence by using physics based (rather than algebraic) preconditioning
6. balance inner and outer iterations using the Eisenstat-Walker approach to create variable forcing functions in the outer iteration

The various steps in this numerical approach follow very closely those described in [Kelly, 1995] and the MATLAB code described in [Kelly, 1995] forms the bulk of the numerical solver code used.

Global Newton Method (Outer Iteration)

The Newton method linearizes a set of equations through a truncated Taylor series expansion of the original set of equations.

$$f(x+\Delta x) = f(x) + f'(x)\Delta x + \vartheta(\Delta x^2) \tag{3-1}$$

If the set of equations is such that $f(x+\Delta x) = 0$, then one can ignore the higher order terms and rearrange to get

$$f'(x)\Delta x = -f(x) \tag{3-2}$$

where $f'(x)$ is a matrix associated with the derivative of $f(x)$ at x [Conte and de Boor, p217]. Thus, the solution of the linearized set of equations is given by

$$\Delta x = -f'(x)^{-1} f(x) \tag{3-3}$$

where Δx is a vector and $x + \Delta x$ points in the direction of the solution of the original set of equations $f(x) = 0$. The magnitude of the Δx vector is not as clear as the directing and $x + \Delta x$ may overshoot the actual solution if x is not within the basin of attraction.

The globally convergent Newton algorithm essentially acknowledges that there is a better solution than the current state vector and treats the direction of Δx as a search direction. If $f(x+\Delta x)$ is greater in magnitude than $f(x)$, the magnitude of Δx is reduced but the direction is maintained. The scaling applied to the search direction should incorporate logic to enforce a "sufficient" decrease in the residual norm to avoid oscillation around a fixed point. The method implemented in the code is referred to as the Armijo rule [Kelly, p137] and declares that if

$$d = -f'(x)^{-1} f(x) \tag{3-4}$$

then the search direction must produce a sufficient decrease defined by

$$\left|f(x+\lambda d)\right| < (1-\alpha\lambda)\left|f(x)\right| \tag{3-5}$$

where $0 < \lambda < 1$ and $\alpha = 10^{-4}$ [Kelly, p136]. Since the goal is to find a state vector, x^*, such that the residual vector $f(x^*) \approx 0$, new state vectors, $x + \lambda d$, must produce a residual, $|f(x + \lambda d)|$, that's not just smaller but sufficiently smaller than the current residual, $|f(x)|$.

In the coded algorithm, the actual value for λ is variable and is selected based on a three-point parabolic interpolation model. In the model, $f(x)$ is known initially and $f(x + \Delta x)$ is tested for sufficient decrease (so it is also known). If the decrease is not sufficient, λ is set to ½ and $f(x + \Delta x/2)$ is tested for sufficient decrease. If it is sufficient, $x^{n+1} = x^n + \frac{\Delta x}{2}$. If not, a new λ_j is selected based on a second order interpolating polynomial using the three known points: $f(x)$, $f(x + \Delta x)$, and $f(x + \Delta x/2)$. The polynomial comes from a second order equation in λ where $p(\lambda) = a\lambda^2 + b\lambda + c$. Solving for the unknown coefficients leads to

$$p(\lambda) = f(x) + \frac{\lambda}{\lambda_c - \lambda_-}\left(\frac{(\lambda - \lambda_-)(f(x + \lambda_c d) - f(x))}{\lambda_c} + \frac{(\lambda_c - \lambda)(f(x + \lambda_- d) - f(x))}{\lambda_-}\right) \quad \textbf{(3-6)}$$

where λ_c is the largest step of the three points and λ_- is the middle step. Only three points are used so the additional λ_- becomes λ_c in subsequent iterations. If $p''(0) > 0$, then the parabola has a minimum

$$\lambda = \frac{\left[(f(0) - f(\lambda_-))\lambda_c^2 - (f(0) - f(\lambda_c))\lambda_-^2\right]}{2\left[(f(0) - f(\lambda_-))\lambda_c - (f(0) - f(\lambda_c))\lambda_-\right]} \quad \textbf{(3-7)}$$

An example case using a parabola to anticipate the correct step length reduction is graphically illustrated in Figure 3-1.

A safeguarding step is included to ensure progress in reducing $f(x + \lambda_+ d)$. This is given by

$$\lambda_+ = \begin{cases} \sigma_0\lambda_c \text{ if } \lambda < \sigma_0\lambda_c \\ \sigma_1\lambda_c \text{ if } \lambda > \sigma_1\lambda_c \text{ where } \sigma_0 = .1 \text{ and } \sigma_1 = .5 \\ \lambda \text{ otherwise} \end{cases} \quad \textbf{(3-8)}$$

If $p''(0) \le 0$, simply set $\lambda_+ = \sigma_1\lambda_c$ [Kelly, pp 142-144].

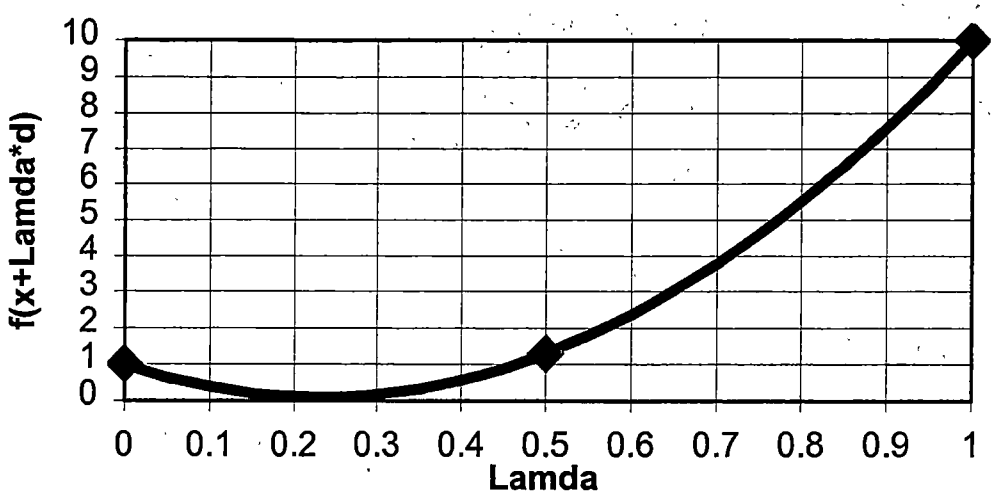


Figure 3-1: Sample Parabolic Step Length Reduction

In the Newton method, many techniques have been developed to ease the computational burden associated with solving $f'(x)\Delta x = -f(x)$. The Jacobian, $f'(x)$, is typically a banded matrix with significant off diagonal elements and may be extremely complicated to form. In fact, for structural dynamics problems, the correct form of the Jacobian for dynamic, large displacement problems was contested until nearly 1971 [McNamara and Marcal, 1973] and not entirely settled until 1978 [Cescotto, Frey, and Fonder, 1979].

Many attempts have been made to simplify the Jacobian matrix or reduce the effort required in its formation and inversion [Venkatakrishnan, 1989]. The simplification techniques typically eliminate or lump insignificant terms or "freeze" the Jacobian for reuse in later steps. More aggressive techniques may replace the inverse Jacobian altogether with a new matrix, $B(x)$ where the action of $B(x)$ on a vector is less expensive to compute than an inversion of $f'(x)$. Here, $\Delta x = -f'(x)^{-1} f(x)$ is replaced by $\Delta x = -B(x) f(x)$. This is referred to as an "approximate Newton." If B is independent of x, then the iteration can be viewed as

$$x_+ = x_c - Bf(x) \tag{3-9}$$

a preconditioned nonlinear Richardson iteration [Kelly, p. 77]. Many of these attempts have been practically successful even though they degrade the theoretical convergence possibilities associated with using a true, up to date Jacobian.

Direct inversion methods for $f'(x)$ typically use some sort of factorization step or matrix preconditioning to ease the computational burden. These tend to fill the zeros in the (relatively) sparse Jacobian and may significantly increase storage requirements. As a workaround, techniques such as the "minimum degree" and "reverse Cuthill-McKee" algorithms reorder the rows and columns to reduce the bandwidth of the matrix and produce less fill during matrix factorization [Venkatakrishnan, 1989] [Pueyo and Zingg, 1997], [Blanco and Zingg, 1997], [MathWorks, pp 505-509].

A matrix free difference approximation to $f'(x)\Delta x$ avoids the problems associated with the corrected form of the Jacobian, row/column order, memory requirements and fill during factorization. Note that a difference approximation to the Jacobian, $\nabla_h f(x)$, multiplied by the solution vector, Δx, is not the same as a difference approximation to the directional derivative, $D_h\left(f'(x)\Delta x\right)$, formed by the matrix-vector product. For ill-conditioned problems, the difference approximation to the directional derivative is usually more accurate [Kelly, pp 70-82] and is implemented in the computer code used in this research effort.

The directional derivative concept is clever but subtle. The Jacobian is a matrix describing the gradient of the function $f(x_c)$ in all directions at the point x_c. However, the only direction desired is the solution direction. The matrix is inverted to find the direction, Δx, that has the greatest overall slope. The directional derivative method works by checking a sequence of guesses, Δx_i, and selecting the one (or a combination of them) that provides the "best" answer given by minimizing

$$\left\| D_h\left(f'(x)\Delta x\right) + f(x) \right\| \tag{3-10}$$

Thus, the problem shifts from one of matrix inversion to one of selecting a good sequence of search vectors. If $f'(x)$ is $n \times n$, the solution can always be found with n or less independent search directions [Kelly, 1995].

The concept of the forward difference approximation to the directional derivative is given by

$$\frac{f(x+h\Delta x)-f(x)}{h} \tag{3-11}$$

The difference increment, $h \approx 10^{-7}$, comes from minimizing the roundoff error in a computer with finite precision of approximately 10^{-15}. The error is minimized when $h = \sqrt{precision}$. In the practical implementation, the difference increment is scaled by $\|x\| / \|\Delta x\|$ to accommodate x and Δx not being the same size. This addresses the concerns identified in [Tidriri, 1995] for selecting step size. In MATLAB, the smallest increment on a Pentium II class personal computer is $2^{-52} \approx 2.2204e-016$ based on the IEEE arithmetic used in the computer. This is the distance from 1.0 to the next largest floating point number [MATLAB User's Guide, pp 2-6] and has a square root leading to $h_{ideal} = 1.4901e-008$. Practical implementation is straightforward using the default value $h = 1 \times 10^{-7}$ and the algorithm described in [Kelly, p 81]:

$$D_h(f'(x)\Delta x) = \begin{cases} 0 & \Delta x = 0 \\ \|\Delta x\| \dfrac{f\left(x + h\|x\| \Delta x / \|\Delta x\|\right) - f(x)}{h\|x\|} & x,\ \Delta x \neq 0 \\ \|\Delta x\| \dfrac{f\left(h \Delta x / \|\Delta x\|\right) - f(x)}{h} & x = 0,\ \Delta x \neq 0 \end{cases} \tag{3-12}$$

Newton Iterations vs. Pseudo Time Steps

Pseudo time methods are standard practice in most fluid flow solvers where an artificial transient is used to solve the system of equations to an asymptotic steady state. They have also been used in structural calculations and typically fall under the name of "dynamic relaxation" [Underwood, 1983]. The transient has no physical meaning and is discarded after convergence. Also, it is assumed any path theoretically leads to the same converged state. This allows innovation in the convergence path and techniques such as local time stepping often lead to faster solutions. Some [Merkle, 1995] [Buelow,

Venkateswaran and Merkle, 1995] [Godfrey, Walters, and van Leer, 1993] [Turkel and Arnone, 1993] [Shuen, Chenand Choi, 1993] have developed special mass matrices for pseudo time transient methods in fluids to accelerate convergence for incompressible flows, chemically reacting flows, and other numerically stiff fluid calculations. [Oakley and Knight, 1995] have done the same for structures.

Pseudo time stepping procedures also apply to unsteady problems. The analogy is not quite the same in this case, but the technique remains to step forward in pseudo time until the residual of the unsteady physical problem is converged at the new physical time level.

The pseudo time stepping techique is straightforward and [Venkateswaran and Merkle, 1995] categorize the Newton iterative method as a subset of pseudo time stepping. In the Newton method, a set of equations is linearized around a state vector to form a Jacobian matrix. The set of linear equations has the form $Ax = b$. This can be rewritten as a residual equation $Ax - b = R^*(w)$ where w is a vector of the dependent variables. The system can now be stepped forward in pseudo time such that $R^* \to 0$ using the equation

$$\frac{\partial w}{\partial t^*} + R^*(w) = 0 \tag{3-13}$$

or

$$Q\frac{\partial w}{\partial t^*} + R^*(w) = 0 \tag{3-14}$$

where * denotes the fictitious time [Alonso and Jameson, 1994], and Q is some mass artificial matrix. Substituting and rearranging, this becomes

$$\frac{\partial w}{\partial t^*} + Ax = b \tag{3-15}$$

or

$$Q\frac{\partial w}{\partial t^*} + Ax = b \tag{3-16}$$

For clarity, b is actually the residual of the physical system of equations, steady or unsteady; A is the Jacobian of the system of equations; Q is a fictitious mass matrix chosen for convenience and $x = \Delta w$. Redefining $R(w) = -b$ to be the negative residual of the physical equation set and substituting yields

$$\frac{\partial w}{\partial t^*} + \frac{\partial R(w)}{\partial w}\Delta w = -R(w) \tag{3-17}$$

or

$$Q\frac{\partial w}{\partial t^*} + \frac{\partial R(w)}{\partial w}\Delta w = -R(w) \tag{3-18}$$

This shows the Newton method is actually a pseudo time stepping method with an infinite step size for the pseudo time term, t^*. The pseudo time term acts as a sink term to dampen errors in the Newton iterations [Venkateswaran and Merkle, 1995] and the mass matrix may be selected to improve convergence properties.

Some [Venkateswaran and Merkle, 1995] [Shuen, Chen and Choi, 1992] have developed the pseudo time set of equations by adding a pseudo time term to the physical set of equations prior to linearization. This makes it easier to determine the role the term can play in aiding convergence. In this approach, $Q\frac{\partial w}{\partial t^*} + R(w) = 0$ is linearized to produce

$$\frac{\partial\left(Q\frac{\partial w}{\partial t^*}\right)}{\partial w}\Delta w + \frac{\partial R(w)}{\partial w}\Delta w = -Q\frac{\partial w}{\partial t^*} - R(w) \tag{3-19}$$

or,

$$\frac{\partial R(w)}{\partial w}\Delta w = -Q\frac{\partial w}{\partial t^*} - R(w) \tag{3-20}$$

Rearranging gives

$$Q\frac{\partial w}{\partial t^*} + \frac{\partial R(w)}{\partial w}\Delta w = -R(w) \tag{3-21}$$

The result is the same as before and there are no pseudo time terms in the right hand side. Some have included the pseudo time terms on the right hand side but experiments showed this greatly slowed convergence for the Newton/pseudo time method.

The pseudo time stepping procedure is discrete in time so

$$\left[Q\frac{1}{\Delta t^*}+\frac{\partial R(w)}{\partial w}\right]\Delta w=-R(w) \tag{3-22}$$

or

$$\left[Q+\Delta t^*\frac{\partial R(w)}{\partial w}\right]\Delta w=-\Delta t^* R(w) \tag{3-23}$$

This is a new set of linear equations of the form $A^*x=b$ or $\left[\frac{1}{\Delta t^*}Q+A\right]x=b$. Any number of time stepping methods, direct inversion, or iterative matrix techniques may be used to solve the linear equation set. An iterative GMRES solver is used in the current effort.

Ultimately, the goal is to find the best linear combination of the vectors in the column space of A to represent the data vector b. The set of coefficients is given by the solution, x, of the Newton system $Ax=b$. The pseudo time approach adds to A the column space of Q scaled by the pseudo time step size Δt^*. The time step may be held constant, changed with each iteration, varied in a local time stepping fashion throughout the domain, or some combination. Local time stepping intuitively holds the most promise but some set of rules must be provided to select the best pseudo time step size based on grid size, velocity gradient, or some other measure. The literature on adaptive grids contains many measures to trigger grid refinement. These may also be useful for selecting local time step sizes. A fixed time step size is used in the current effort.

Preconditioned GMRES Linear Equation Solver (Inner Iteration)

As stated earlier, the Newton statement $f'(x)\Delta x=-f(x)$ forms a linear equation set $Ax=b$. Normally, the choice is to solve the set of equations using a direct or iterative method. In the current effort, the left hand side is treated as a matrix-vector product representing the directional derivative using

the matrix-free forward difference approximation. This eliminates the possibility of using a direct solver because there is no matrix to invert. Instead, the GMRES (Generalized Minimum RESidual) method of [Saad and Schultz, 1986] is used and is implemented as described in [Kelly, 1995].

GMRES, as the name implies, solves the linear set of equations for x by minimizing the residual, $r = b - Ax$. This is implemented as a solution to the least squares problem

$$\text{minimize}_x \|b - Ax\|_2 \tag{3-24}$$

It is actually a generalization of the conjugate gradient method extended to unsymmetric matrices.

The key element in the GMRES concept is the formation and orthogonalization of a set of basis vectors for the "Krylov" subspace. The Krylov subspace for $Ax = b$ is defined as $\{r_0, Ar_0, A^2r_0, \ldots, A^{n-1}r_0\}$ for $A_{n\times n}$. Thus, the subspace is defined by the set of vectors generated by the product of a starting vector, r_0, and the powers of the matrix, A^i for $0 \le i \le n$. The starting vector, r_0, is arbitrary and is normally taken as $r_0 = b$. Notice that the matrix-vector product of A with the previous Krylov vector in the subspace forms each subsequent vector in the Krylov subspace (e.g. $A^3r_0 = A \cdot A^2r_0$). Also, the eigenvalues of the matrix, A, essentially determine the number of Krylov vectors needed to approximate the full n-dimensional subspace. A^ir_0 tends to zero for every r_0 if (and only if) the largest eigenvalue of A is less than one [Noble, p 373]. The behavior of the powers of the matrix, A^i, is controlled by the relative sizes of the eigenvalues. These will converge at different rates according to λ_j^i for the ith power of the jth eigenvalue. If A can be factored into a diagonal matrix as $A = XDX^{-1}$, then X contains the eigenvectors of A if A is normal. [Saad and Schultz, p866] show the convergence of the GMRES method is governed by the condition of the X matrix and a minimum polynomial in the maximum eigenvalue of A [Kelly, pp 34-36].

$$\frac{\|r_k\|}{\|r_0\|} \le \kappa_2(X) \max_{z\in\rho(A)} |\bar{p}_k(z)| \tag{3-25}$$

This is included for reference only and is intended to show the importance of not only the eigenvalue distribution, but the condition of the diagonalizing transformation matrix as well. Appendix A provides background information on the sensitivity of eigenvalues and eigenvectors to perturbations.

It is also interesting to note that even in their original work, [Saad and Schultz, 1986] used Incomplete Lower-Upper (ILU) and Modified Incomplete Lower-Upper (MILU) algebraic preconditioners to improve convergence rates. These have become de facto standard preconditioners in most practical fluid flow applications. [Barrett, et al, 1993] provides a great synopsis of various preconditioning methods, costs, and tradeoffs. [Axelsson, 1994] provides detailed descriptions and theoretical basis for preconditioners and their appropriate uses based on matrix eigenvalue structure. The problem with most preconditioners, such as ILU, is the left hand side Jacobian matrix must be explicitly formed at least once to use them [Tidriri, 1995] [Pueyo and Zingg, 1997].

The basic intent of a preconditioner is to form a system of equations such that for

$$MAx = Mb \tag{3-26}$$

then

$$\left\| I - MA \right\|_2 = \rho < 1 \tag{3-27}$$

The errors, e_k, in the iteration sequence are expected to decrease with additional iterations. While the actual errors in a particular iteration are not known until the solution is known, it is useful to know how the error behaves with respect to the known residual at each step. This is shown by [Kelly, p36] to be related to the condition of the preconditioned system of equations by

$$\frac{\left\| e_k \right\|_2}{\left\| e_0 \right\|_2} \le \kappa_2(MA) \frac{\left\| r_k \right\|_2}{\left\| r_0 \right\|_2} \tag{3-28}$$

From a practical standpoint, all this means the condition of the matrix associated with the directional derivative (matrix-vector product) should be well conditioned to provide rapid convergence. This may be done by forming the vector, Ax_i, and then modifying it as MAx_i, or by changing the set of equations so that A is well conditioned from the outset. It is more than interesting to note that the left preconditioning matrix represented by M may be implemented through an addition instead of a multiplication step. For example, if

$$Q + A = MA \tag{3-29}$$

then

$$Q = (M - I)A \tag{3-30}$$

or

$$M = QA^{-1} + I \tag{3-31}$$

Alternately, a right preconditioning operator could be implemented for $Ax = b$ such that

$$AMM^{-1}x = b \tag{3-32}$$

or

$$AMs = b \tag{3-33}$$

where $s = M^{-1}x$. At convergence, $x \approx 0$ so $s \approx 0$ also satisfies convergence criteria and there is no need to ever solve for x or modify b. As above, an additive matrix will have the same action as matrix multiplication if $M = I + A^{-1}Q$. The composite equation becomes

$$A(I + A^{-1}Q)(I + A^{-1}Q)^{-1}x = b \tag{3-34}$$

or

$$A(I + A^{-1}Q)s = b \tag{3-35}$$

or

$$(A + Q)s = b \tag{3-36}$$

In the current effort, the eigenvalues of the physical problem are modified so that A is reasonably well conditioned instead of using a matrix preconditioner, M. The approach is conceptually the same as shown above where a matrix, Q, is added to the A matrix. Q is developed based on controlling the continuum eigenvalues of the system of equations represented in A. Since A represents a Jacobian for the system of equations $f(x)$, the result is an approximate Newton method,

$$[Q(x) + f'(x)]\Delta x = -f(x) \tag{3-37}$$

The matrix Q may be viewed as an additive preconditioner, a multiplicative preconditioner, a penalty or LaGrange multiplier term, or a pseudo time stepping term. In the sections to follow, Q is developed and justified as a pseudo time stepping term but the behavior of the solution iterates seem to indicate it behaves as a penalty or LaGrange multiplier term.

Technically, the GMRES algorithm is classified as a nonstationary iterative method and generates a sequence of orthogonal vectors (basis vectors for the Krylov subspace). The vectors are created as indicated in the description of the Krylov subspace and made orthogonal through the modified Gram-Schmidt procedure (described in [Golub and Van Loan, pp 218-219] and summarized in Appendix B). As the orthogonal vectors are formed, the coefficients are stored in an upper Hessenberg matrix. The overall procedure is referred to as the Arnoldi iteration [Golub and Van Loan, p 501]. The orthogonalization process only breaks down if the solution to $Ax = b$ is in the space already created. As a practical consideration, the coefficients generated by the modified Gram-Schmidt process must be scaled. For example, if the modified Gram-Schmidt process is used to turn a set of independent vectors $v_1,\ldots,v_q$ into an orthogonal set $u_1,\ldots,u_q$, the coefficients of the process are $\frac{\left(u_i, v_j^{i-1}\right)}{\left(u_i, u_i\right)}$ where $\left(u_i, u_i\right)$ refers to the inner product operation. If an element of u_i is of a significantly different order of magnitude than the other elements, it will dominate the orthogonalization process and mask the influence of lesser but equally important components. Nondimensionalizing the set of equations normally avoids this problem. In the current effort, all equations and calculations are kept in dimensional form rather than nondimensionalizing so, for example, the mass and energy equations were scaled to match the momentum equation to overcome the six order of magnitude differences in the numerical values. [Kelly, pp 37-46] describes the GMRES idea and other practical considerations for implementation. The GMRES algorithms used in this effort are modified versions of that provided by [Kelly, 1995].

A problem with the GMRES algorithm is that all iteration vectors must be saved rather than just the current iteration. If the solution does not converge quickly, the computer memory requirements may become overwhelming. For this reason (and maybe the only good reason) the restarted GMRES algorithm

is used most often. In restarted GMRES, a limit is imposed on the number of Krylov vectors stored. If the solution has not converged, the best solution is found over the limited set of Krylov vectors, the vectors are erased, and the process starts over with the "best solution" from the last set as the new starting point. This procedure definitely slows convergence and there are no general theories on convergence of the restarted GMRES method. Finding the "right" number of vectors to use for the restart point is a matter of experience (read trial and error) and luck. Keeping more vectors before restarting does not necessarily lead to faster solutions [Edwards, et al, 1994]. Also, the number of restarts allowed may be limited. Again, more restarts do not necessarily lead to faster solutions. [McHugh, Knoll, and Keyes, 1996] limited the number of Krylov vectors to 50 for a two dimensional, low speed combustion problem and did not use restarts at all. Their goal was to apply preconditioners on subdomains of the global flowfield and accelerate the GMRES convergence so an adequate solution could be found on the rank 50 Krylov subspace. The subdomain flowfields were solved using ILU and BILU preconditioners with varying (limited) degrees of success.

Although not implemented in the current effort, domain decomposition within a single set of field equations is not unlike partitioning between fluid and structure in a coupled field problem. Single field decompositions have historically been based on geometry and are sized to fit in the computer hardware of the day. The above philosophy also implies numerical domain decomposition methods based on eigenvalue distributions could be very effective. Decompositions based on eigenvalues may have great promise on parallel architectures but have not been found in the literature. In the current context, Newton-Krylov-Schwarz methods are used with domain decomposition but are usually based on computational efficiency or geometry with heavy reliance on local (subdomain) preconditioners to provide rapid convergence. [Cai, Keyes, and Venkatakrishnan, 1995] use Newton-Krylov-Schwarz decomposition to assess an ILU based approximation as a Schwarz preconditioner for elliptic and potential flow equations. [McHugh, Knoll, and Keyes, 1996] partitioned the field to make best use of ILU and BILU type preconditioners but with limited success for a stiff system of low speed combustion equations. [Gropp, Keyes, McInnes, and Tidriri, 1997] pull together a variety of algorithms and implement with a focus on architecture and "coarse grain" parallelism. They use additive Schwarz "preconditioning" to split the problem and minimize inter-processor communications while using block Jacobi and ILU preconditioners

for the inner iterations. They solve the 3-D transonic Euler equations using pseudo time stepping. The domain decomposition is based on computational efficiency rather than matrix or continuum eigenvalues and is effective if the inner iteration preconditioners perform well. [Kaushik, Keyes, and Smith, 1998] extend the method further still to a 3-D unstructured code using pseudo time stepping with application to compressible and incompressible flows. They selectively use Chorin [Chorin, 1967] artificial compressibility for low speed blocks. Most of their focus is on scalability and minimizing communications overhead between processors. They use one Newton iteration per pseudo time step, limit the Krylov subspace to 30 vectors, and allow only one restart. They are coming close to the concept of using the pseudo time step to control eigenvalues and eliminate the need for matrix preconditioning but never quite go in that direction.

Inexact Newton-GMRES with Variable Forcing Function

Direct methods for inverting matrices present all or nothing propositions; either the matrix is completely inverted or it isn't. On the other hand, iterative methods may be stopped at any point if the successive (and hopefully converging) trial solutions are considered "good enough" or meet some predetermined limits. If the set of linear equations comes from a Newton linearization, an approximate solution to the linear set does not rigorously satisfy the Newton formulation. [Kelly, p 95] refers to this as an "inexact Newton" method. This opens the question of how the approximate solution from the linear (inner) iteration affects the Newton (outer) step. The answer comes by defining acceptance criteria for inexact Newton iterations as

$$\left\| f'(x)\Delta x + f(x) \right\| \le \eta_c \left\| f(x) \right\| \tag{3-38}$$

or

$$\left\| Q(x)\Delta x + f'(x)\Delta x + f(x) \right\| \le \eta_c \left\| f(x) \right\| \tag{3-39}$$

where $0 < \eta_c < 1$ is referred to as the "forcing term." The forcing term is used to set specific criteria for acceptance of an approximate solution to the linear equation set. In effect, if the approximate solution produces a linear residual that is sufficiently small compared to the nonlinear residual, then the approximate solution is accepted. [Kelly, pp 95-100] provides the theoretical aspects of the forcing

function and its possible impact on the quadratic convergence desired in a Newton iterative method. In the Newton-GMRES method, the forcing term criteria is used to set termination criteria and GMRES continues adding vectors to the Krylov subspace until the forcing term is satisfied.

If the forcing term is small, a heavy burden is placed on the inner iteration to provide as accurate of a solution as possible and the early outer iterations may be over emphasized when the current iterate, x_n, is still far from the actual solution, x_*. This may lead to a large number of GMRES iterations and/or restarts. Conversely, if the forcing term is close to one, the burden is shifted to the nonlinear outer iteration and may require a large number of nonlinear steps to converge. The best choice for a fixed forcing term is a balancing act between the work associated with inner and outer iterations. [Pueyo and Zingg, 1997] selected $\eta_c = 0.1$ as the optimum for fluid flow. However, there is no requirement for the forcing term to be fixed so η_c may be replaced by a sequence of forcing terms, $\{\eta_n\}$. At startup, while x_n is still far from the actual solution, x_*, η_n may be near one. As the nonlinear iteration begins to converge, η_n is reduced to force better and better linear solutions from the Jacobian. Finally, η_n may be close to one again in the last iteration to meet termination criteria without over solving the linear step. If the linear solution cannot meet the forcing function goals on a given step, the "best solution" is used and the global Newton logic takes over. The forcing function algorithm must then loosen the requirements for the next step. The impact of the variable forcing function can be seen throughout the solution process, regardless of the physical equation set. The algorithm applied in the code is referred to as the "Eisenstat-Walker" method and is described in [Kelly, pp 104-106]. Again, the baseline code also comes from [Kelly, 1995].

The Eisenstat-Walker method is summarized as

$$\eta_n = \min\left(\eta_{max}, \max\left(\eta_n^c, .5\tau_t / \|f(x_n)\|\right)\right) \tag{3-40}$$

where

$$\eta_n^c = \begin{cases} \eta_{max} & n = 0 \\ \min\left(\eta_{max}, \eta_n^a\right) & n > 0, \gamma\eta_{n-1}^2 \le .1 \\ \min\left(\eta_{max}, \max\left(\eta_n^a, \gamma\eta_{n-1}^2\right)\right) & n > 0, \gamma\eta_{n-1}^2 > .1 \end{cases} \tag{3-41}$$

and

$$\eta_n^a = \gamma \frac{\left\| f(x_n) \right\|^2}{\left\| f(x_{n-1}) \right\|^2} \tag{3-42}$$

The effect and impact of the various terms in the Eisenstat-Walker forcing function are shown graphically in Figure 3-2.

In the code developed for this research effort, $\gamma = .9$, $\eta_{max} = .5$ and $\tau_t \approx 1 \times 10^{-6}$. τ_t is based on the termination criteria described below. Since these are arbitrary parameters, η_{max} was tested over the range $.01 < \eta_{max} < .9999$. γ was not changed. The best overall values are shown. The parameter $\gamma \eta_{n-1}^2 > .1$ is a safeguarding step to avoid sharp decreases in η_n generated by a "lucky guess" step. The value .1 is also arbitrary. The graph shows a historical sequence of the Eisenstat-Walker forcing function given a randomly generated but converging set of nonlinear residuals. The forcing function sequence changes to accommodate the set of nonlinear residuals.

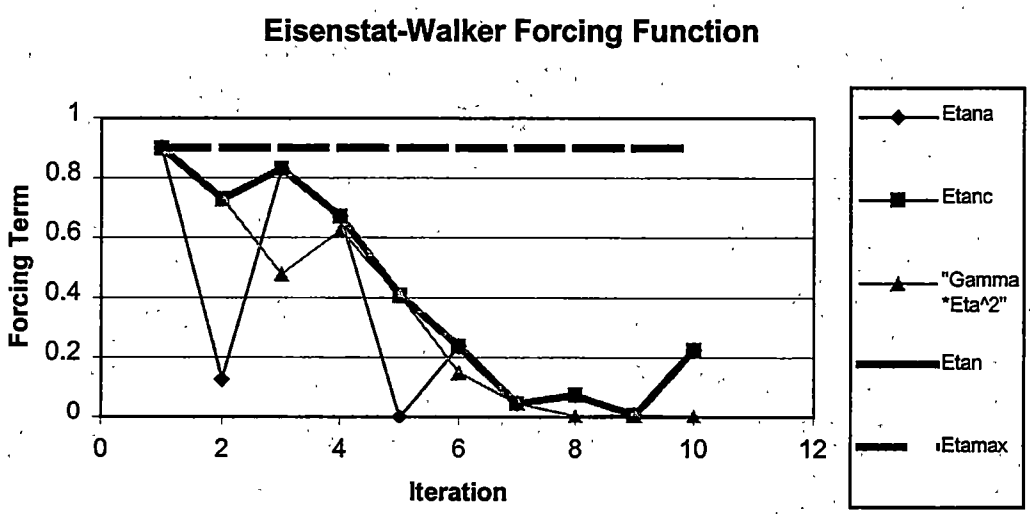


Figure 3-2: Sample Eisenstat-Walker Forcing Function

Termination Criteria

The forcing function essentially determines the nonlinear reduction goal for a given Newton step. For the $Ax = b$ system, the i iterative solution steps for Ax continue until $\left\|A_n x_n^i + b_n\right\| \le \eta_n \left\|b_n\right\|$ or some hard limit is reached. If the initial iterate $x_0 = 0$, then $r_0 = Ax_0 + b = b$. [Kelly, pp 33-35] shows $\left\|r_i\right\|_2 \le \rho^i \left\|r_0\right\|_2$ or $\left\|Ax_i + b\right\|_2 \le \rho^i \left\|b\right\|_2$ where ρ is the maximum eigenvalue of A. This is essentially the forcing function definition and shows the value of controlling the spectral radius of A throughout the nonlinear iteration process. A well-conditioned system may turn ill-conditioned as the solution evolves or vice versa. This is especially true if the grid geometry is changing. It also implies more or more exact linear solution iterates are needed as the forcing function is reduced. In practice, more exact GMRES solutions were generally needed as the solution converged and the convergence rate goals set by the forcing function accelerated. Hard limits are set for the maximum number of Krylov vectors and the maximum number of GMRES restarts.

The nonlinear iteration also has termination criteria based on hard limits and residual reduction goals. The hard limit for maximum number of nonlinear iterations is set to 20. This was only reached when doing parametric studies of other parameters. The Armijo step length reduction was set to a maximum of eight reductions for any given Newton step. This could probably have been reduced because no attempt successfully converged after needing more than two or three step reductions. Some made it to seven step length reductions on a given step only to fail at the limit on a subsequent step. The step length reductions turned out to be a good early indicator of stagnation in the nonlinear solution process. Even single step length reductions set back the convergence rate and typically reset the Eisenstat-Walker forcing function to the maximum value.

The termination criterion for nonlinear residual reduction was based on a relative and absolute measure of the residual. Convergence for the nonlinear iterations was defined as

$$\left\|f(x_n)\right\| \le \tau_a + \tau_r \left\|f(x_0)\right\| \qquad \textbf{(3-43)}$$

where τ_a is an absolute minimum threshold and τ_r is a relative reduction parameter [Kelly, pp 72-73]. In the code developed for this research effort, $\tau_a = 1 \times 10^{-6}$ and $\tau_r = 1 \times 10^{-6}$. Thus, the solution process was terminated when the nonlinear residual norm was six orders of magnitude below the starting value or below 1×10^{-6}. The absolute and relative thresholds do not have to be the same and should reflect realistic objectives for the project.

CHAPTER 4

FLUID FLOW PRECONDITIONING

Background

Solving fluid flow problems numerically often leads to problems with stiff systems of equations. The stiffness can slow or stagnate convergence and may be driven by numerical artifacts such as grid aspect ratios or physical aspects of the flow such as a wide speed range. There are numerous strategies to resolve these stiffness issues but these primarily fall into three major categories. In no particular order, the first major approach is one of algebraic preconditioning. It does not rely on any theoretical analysis of the fluid system to be solved but attacks the linear algebra problem. The approach is basically to first form a linearized system of equations for a given geometry. The structure and elements of the

$$\begin{aligned} Ax &= b \\ P^{-1}Ax &= P^{-1}b \end{aligned} \tag{4-1}$$

coefficient matrix, A, ultimately determine the numerical stiffness. A typical approach is to form the preconditioning matrix, P, based on features of the coefficient matrix. This may be the diagonal, $P = diag(A)$, for a simple point Jacobi method or $P \approx LU$ for a complete or incomplete factorization method [Barrett, et. al., 1993]. This basic approach is very common in practical use but usually requires formation of the coefficient matrix, A, at least once and storage of the preconditioning matrix P^{-1}.

The second approach is to subdivide the fluid flow region into overlapping or contiguous blocks where each block is selected to have similar properties and low stiffness. Each block is solved semi-independently of the other blocks. The solution for one block provides updated boundary conditions for adjacent blocks and the entire process is iterated to convergence. This approach covers domain decomposition or partitioned strategies as well as the multi-grid methods. The approach to the composite problem can be viewed as a multiplicative preconditioner where the blocking strategy is

$$\begin{aligned} Ax &= b \\ B^{-1}Ax &= B^{-1}b \end{aligned} \tag{4-2}$$

implemented through the operator, B. In this case, the preconditioning operator combines a sequence of operations to restrict the domain, solve on the subdomain, then extrapolate out to the full domain [Barrett, 1993]. The preconditioner may be successfully implemented without the coarse grid global solve famous in the multi-grid method but it will require more iterations to reach convergence. Computer hardware limitations and geometry have historically driven the blocking strategy but numerical stiffness could also be a driver.

The third and more elegant approach is to attack the physical problem to remove ill conditioning. This approach is prevalent in structural calculations where beam, plate, shell, and other specialized elements implement different forms, assumptions and boundary conditions for the basic structural equations. In fluid flow, different codes are often used for creeping, low speed, compressible and supersonic speeds but these are typically not used in combination. Real world situations such as transonic flow over a wing actually have viscous dominated boundary layers, low speed separation or stagnation regions, and compressible core flow with supersonic pockets. Ad hoc methods such as changing variables and under/over integration of variables sometimes act to change the basic equation but [Chorin, 1967] really opened the door with the pseudo-compressibility method. His early success using time-marching pressure methods in low speed flow has been greatly expanded and now compressible flow codes may be used for all speed regimes. [Turkel and Arnone, 1993], [Turkel, 1987], [Lee and van Leer, 1993], [Godfrey, Walters and van Leer, 1993] [Merkle, 1995] and others have developed methods to control the characteristic speeds within a fluid by using an artificial mass matrix with a time-stepping approach to steady state. [Saleem, Pulliam, and Cheer, 1993] take a different tack using eigenvalue annihilation and eigenvalue shifting to accelerate convergence.

It is worth pointing out the three methods are not exclusive and a single problem could actually use time derivative preconditioning equations with an algebraic preconditioner on each grid of a domain decomposition or multi-grid problem. This may represent the ultimate in convergence acceleration.

Newton Time Stepping with Preconditioning

The method implemented in the current effort is based on the time derivative preconditioning approach of [Merkle, 1995]. His approach has been well tested [Reed and Anderson, 1997] [Zori and Rajagopalan, 1997] with uniformly good results for a variety of methods that use time stepping to approach a converged steady state. This research effort implements a significantly different formulation of the fluid flow equations where a Newton method is used to linearize equations and the linear problem statement is solved using a Krylov based (GMRES) iterative solver. The fluid flow equations are discretized using an unstructured finite element formulation combining aspects of the Taylor Weak Statement (TWS) [Baker and Kim, 1987] and the latest Streamwise Upwind Petrov Galerkin (SUPG) method [Aliabadi and Tezduyar, 1993].

The pure Newton method (as opposed to approximate Newton methods such as ADI) represents the limit of artificial time stepping methods with an infinitely large time step. It has not been widely used for fluid flow problems due to practical problems associated with the size, formation, and inversion of the Jacobian matrix. [Venkatakrishnan, 1989] studied the convergence behavior of the Newton time stepping method for the unsteady fluid flow equations using large time steps to converge to a steady state. He noted the left hand side formed a modified Newton statement and the time term acted as a damping term. R is

$$\left(\frac{I}{\Delta t}+\frac{\partial R}{\partial w}\right)\Delta w = -R \qquad \textbf{(4-3)}$$

the residual of the steady fluid flow equation set at each of the nodes (continuity, Y and Y momentum, energy) and w is the set of unknown fluid flow variables (pressure, X velocity, Y velocity, and temperature) at those nodes [Wood, 1990]. Because he used a direct inversion technique, node numbering and matrix bandwidths were significant issues. He was typically able to achieve convergence in less than 15 Newton (outer) iterations and increased the pseudo time step size with each iteration. This reduced the influence of the time term as the solution approached convergence.

[Shuen, Chen, and Choi, 1992] expanded the "artificial time stepping to steady state" approach to a dual time stepping approach for truly unsteady problems. They note the convergence of the iterative process is determined by the eigenvalue characteristics of the pseudo time space. To control the

eigenvalues, they included a time derivative preconditioning mass matrix on the pseudo time term based on the work of [Merkle and Choi, 1988] [Merkle and Choi, 1987] [Choi and Merkle, 1993]. They used a modified strongly implicit procedure (MSIP) to split the left hand side matrix into upper and lower triangular matrices and solved the coupled system through sequential inversion of these matrices. For large aspect ratio nozzle flows with air dissociation and recombination, the method converged to near machine zero in around 1000 steps. The residual was six to ten orders of magnitude smaller than the non-preconditioned case for the same number of iterations.

[Venkateswaran and Merkle, 1995] explored the underlying mechanisms for preconditioning in dual time stepping calculations. They also noted the Newton method is a subset of the pseudo time procedure where the step size is infinite. The time term can be used to advantage, though, to control the eigenvalue structure of the iteration and improve convergence. The modified Newton or pseudo time algorithm now has the form

$$\left(\frac{\Gamma}{\Delta t}+\frac{\partial R}{\partial w}\right)\Delta w=-R \tag{4-4}$$

where Γ is a preconditioning matrix and R is the residual of the steady or unsteady fluid flow equation set. They show how the terms in the preconditioning matrix must change depending on the physical time step size and physical time scales present.

[Buelow, Venkateswaran and Merkle, 1995] address the issue of preconditioning in conjunction with local time stepping to remove or mitigate the effects of grid aspect ratio. The local time step should be selected based on

$$\Delta t = Max\left[\frac{CFL\Delta x}{u+c},\frac{CFL\Delta y}{v+c}\right] \tag{4-5}$$

where u and v are the velocity components and c is the acoustic speed. Preconditioning artificially controls the acoustic speed so it is close to the particle velocity. The $Max[\cdot]$ operator takes care of disparities in the time scales for particle speed changes, grid size and aspect ratios. This is suspiciously similar to the scaling for the τ parameter in the TWS/SUPG derivation but the connection was not pursued.

The basic goal of the preconditioning strategies above is to remove or control stiffness in the numerical system of equations. Time derivative preconditioning effectively controls the physical eigenvalues with the pseudo time stepping procedure. The time step size can be global or localized in each element. Local time stepping with the time scale switch above targets the adverse effects of grid aspect ratio. Chapter 5 describes the scaling parameters used to make the dimensional fluid flow equations have similar magnitudes so one equation or the other does not dominate the residual vectors. It now stands to reason that this could be used with a Newton-GMRES solver to speed convergence of the GMRES (inner) iterations.

The Time Derivative Preconditioning Matrix

[Merkle, 1995] follows a very logical path to develop a time derivative preconditioning matrix for steady fluid flow equations based, essentially, on the Jacobian associated with a change of primary variables. Instead of using the conservation variables,

$$Q = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho\varepsilon + \dfrac{\rho V^2}{2} \end{Bmatrix} \tag{4-6}$$

a set of primitive "viscous" variables are used

$$Q_v = \begin{Bmatrix} p \\ u \\ v \\ T \end{Bmatrix} \tag{4-7}$$

The inviscid fluid flow equation is converted to the primitive variables through the normal chain rule of

$$\frac{\partial Q}{\partial t} + \frac{\partial E(Q)}{\partial x} + \frac{\partial F(Q)}{\partial y} = 0 \tag{4-8}$$

calculus. The conservative flux vectors can still be calculated from the primitive variables so the only change is to the time derivative:

$$\frac{\partial Q}{\partial Q_v}\frac{\partial Q_v}{\partial t}+\frac{\partial E(Q_v)}{\partial x}+\frac{\partial F(Q_v)}{\partial y}=0 \tag{4-9}$$

where

$$\frac{\partial Q}{\partial Q_v}=\begin{bmatrix} \partial\rho/\partial p & \partial\rho/\partial u & \cdots \\ \partial\rho u/\partial p & & \\ \vdots & & \end{bmatrix} \tag{4-10}$$

or

$$\frac{\partial Q}{\partial Q_v}=\begin{bmatrix} \rho_p & 0 & 0 & \rho_T \\ \rho_p u & \rho & 0 & \rho_T u \\ \rho_p v & 0 & \rho & \rho_T v \\ \rho_p h^o-\left(1-\rho h_p\right) & \rho u & \rho v & \rho_T h^o+\rho h_T \end{bmatrix} \tag{4-11}$$

The subscript on the terms within the matrix denotes a partial derivative with respect to the subscripted variable. This provides a transformation of variables and the Jacobian matrix in front of the time derivative is now a "mass" matrix for the new variables. The conserved quantities in the flux vectors are now calculated from more primitive variables but the flux vectors have not changed.

$$E=\begin{Bmatrix} \rho u \\ \rho u^2+p \\ \rho uv \\ \left(\rho\varepsilon+\frac{\rho V^2}{2}+p\right)u \end{Bmatrix} \tag{4-12}$$

The subscript denotes a partial derivative of the primary variable with respect to the subscripted one and $h^o=h+\frac{V^2}{2}$ is the stagnation enthalpy.

The flux Jacobians with respect to the primitive variables will be needed for the left hand side. The first of these, $A_v=\frac{\partial E}{\partial Q_v}$, is given by

$$A_v = \begin{bmatrix} \rho_p u & \rho & 0 & \rho_T u \\ \rho_p u^2 + 1 & 2\rho u & 0 & \rho_T u^2 \\ \rho_p uv & \rho v & \rho u & \rho_T uv \\ \left[\rho_p h^o + \rho h_p\right] u & \rho\left(h^o + u^2\right) & \rho uv & u\left(\rho_T h^o + \rho h_T\right) \end{bmatrix} \tag{4-13}$$

As always, the eigenvalues of the fluid flow equation are given by

$$\lambda = \{u - c, u, u, u + c\} \tag{4-14}$$

The significant benefit of the primitive variable transformation is in the detailed relationship between the partial derivatives and the acoustic speed

$$c = \frac{\rho h_T}{\rho_T\left(1 - \rho h_p\right) + \rho \rho_p h_T} \tag{4-15}$$

This reduces to the more familiar form $c = \sqrt{\gamma RT}$ for a perfect gas.

Since the time transient is artificial, the mass matrix can be modified to accelerate convergence without changing the converged solution. Note the flux Jacobians must not be modified. Let

$$\Gamma_v = \begin{bmatrix} \rho'_p & 0 & 0 & \rho'_T \\ \rho'_p u & \rho & 0 & \rho'_T u \\ \rho'_p v & 0 & \rho & \rho'_T v \\ \rho'_p h^o - h'_p & \rho u & \rho v & \rho'_T h^o + \rho h'_T \end{bmatrix} \tag{4-16}$$

The variables marked by the apostrophe will be modified. There are now four parameters to select based on the type of fluid and velocity considerations. If preconditioning is not needed, the parameters may be simply returned to their physical values. For a perfect gas with constant specific heats, these are

$$\begin{aligned} \rho'_p &= P/RT \\ \rho'_T &= -P/RT^2 \\ h'_p &= 1 - h_p = 1 \\ h'_T &= C_p \end{aligned} \tag{4-17}$$

[Merkle, 1995] provides a concise analysis for selecting the preconditioning parameters and extends them to viscous flows. His findings for Euler flows are summarized as

$$\begin{aligned} \rho'_p &= k_p / V_{ref}^2 \\ \rho'_T &= k_T \rho_T \\ h'_p &= \text{arbitrary} \\ h'_T &= k_h h_T \end{aligned} \tag{4-18}$$

where V_{ref} is the magnitude of the local grid velocity at each node; k_p equals γ or 1; and the free parameters k_T, h'_p, and k_h are between 0 and 1. In this research effort, I used

$$\begin{aligned} \rho'_p &= 1 / V_{ref}^2 \\ \rho'_T &= 0 \\ h'_p &= 1 \\ h'_T &= C_p \end{aligned} \tag{4-19}$$

where C_p is the specific heat at constant pressure. This gives a preconditioning mass matrix of

$$\Gamma_v = \begin{bmatrix} 1/V_{ref}^2 & 0 & 0 & 0 \\ u/V_{ref}^2 & \rho & 0 & 0 \\ v/V_{ref}^2 & 0 & \rho & 0 \\ h^o/V_{ref}^2 - 1 & \rho u & \rho v & \rho C_p \end{bmatrix} \tag{4-20}$$

The preconditioned Euler equations are now given by

$$\Gamma_v \frac{\partial Q_v}{\partial t} + \frac{\partial E(Q_v)}{\partial x} + \frac{\partial F(Q_v)}{\partial y} = 0 \tag{4-21}$$

or, in linearized form,

$$\Gamma_v \frac{\partial Q_v}{\partial t} + A_v \frac{\partial Q_v}{\partial x} + B_v \frac{\partial Q_v}{\partial y} = 0 \tag{4-22}$$

Discrete Implementation

From the Taylor Weak Statement (TWS) derivation in Chapter 5,

$$\Gamma_v \frac{\partial Q_v}{\partial t} + \frac{\partial E(Q_v)}{\partial x} + \frac{\partial F(Q_v)}{\partial y} = \left[I + \tau \left(\frac{\partial}{\partial x} A_v \Gamma_v^{-1} + \frac{\partial}{\partial y} B_v \Gamma_v^{-1} \right) \right] \left(\Gamma_v \frac{Q_v^{n+1} - Q_v^n}{\Delta t} + D\left(\frac{\partial E(Q_v)}{\partial x} \right) + D\left(\frac{\partial F(Q_v)}{\partial y} \right) \right) + H.O.T.\left(\Delta t^2, \Delta x^k, \Delta y^k \right) \quad \text{(4-23)}$$

where D is a spatial discretization operator of arbitrary order. This has the basic form

$$\Gamma_v \frac{\partial Q_v}{\partial t} + \frac{\partial E(Q_v)}{\partial x} + \frac{\partial F(Q_v)}{\partial y} = [W] \left(\Gamma_v \frac{Q_v^{n+1} - Q_v^n}{\Delta t} + D\left(\frac{\partial E(Q_v)}{\partial x} \right) + D\left(\frac{\partial F(Q_v)}{\partial y} \right) \right) \quad \text{(4-24)}$$

where

$$W = I + \tau \left(\frac{\partial}{\partial x} A_v \Gamma_v^{-1} + \frac{\partial}{\partial y} B_v \Gamma_v^{-1} \right) \quad \text{(4-25)}$$

is a weight function. In the TWS derivation, the flux operators in the weight function generate numerical diffusion exactly tuned to offset negative diffusion associated with the inconsistency of the simple discretization (Chapter 5). The diffusion terms are also eigenvalue driven and contain the inverse of the preconditioning mass matrix. Fortunately, the inverse still has a relatively simple form given by

$$\Gamma_v^{-1} = \begin{bmatrix} V_{ref}^2 & 0 & 0 & 0 \\ -u/\rho & 1/\rho & 0 & 0 \\ -v/\rho & 0 & 1/\rho & 0 \\ \left(2V_{ref}^2 - h^o\right)/\rho C_p & -u/\rho C_p & -v/\rho C_p & 1/\rho C_p \end{bmatrix} \quad \text{(4-26)}$$

Thus, the weighting term is also properly preconditioned.

On a moving grid (Chapter 6), the time derivative and weight terms change to reflect the new variables and the preconditioning matrix

$$\frac{d}{dt}\iiint_{V(t)} W_{ij}\Gamma_v Q_{v_j}\,dV + \iiint_{V(t)} W_{ij}\left[\frac{\partial}{\partial x_k}Q_j(u_k - w_k) + \frac{\partial}{\partial x_k}\delta_{km}p_m\right]dV = 0 \qquad \textbf{(4-27)}$$

In the Newton time stepping method, the right hand side does not include the pseudo time term but the left hand side does. Using the GMRES method, the left hand side can use a linearized form of the fluid flow equations or, since only the matrix vector product is needed, it can also use the right hand side conservation form to create the directional derivatives. To speed up the vector iterations on the left hand side, I used an element-by-element approach to both calculate the nonlinear conservation form for the right hand side and store the element's linearized matrix for later use on the left hand side. The global matrix was never assembled and the left hand side vectors were formed element by element.

$$\frac{\sum_{i=1}^{n}\left[\left(\frac{\Gamma_v}{\Delta t}\right)_{e(i)} + \left(\frac{\partial R_{RHS}}{\partial Q_v}\right)_{e(i)}\right]\left(Q_v + \varepsilon\Delta Q_v\right)_{e(i)}}{\varepsilon} - \frac{\sum_{i=1}^{n}\left[\left(\frac{\Gamma_v}{\Delta t}\right)_{e(i)} + \left(\frac{\partial R_{RHS}}{\partial Q_v}\right)_{e(i)}\right]\left(Q_v\right)_{e(i)}}{\varepsilon} = -\sum_{i=1}^{n} R_{RHS_{e(i)}} \qquad \textbf{(4-28)}$$

In residual form, this looks like

$$\frac{\sum_{i=1}^{n} R\left(Q_v + \varepsilon\Delta Q_v\right)_{LHS_{e(i)}} - \sum_{i=1}^{n} R\left(Q_v\right)_{LHS_{e(i)}}}{\varepsilon} = -\sum_{i=1}^{n} R_{RHS_{e(i)}} \qquad \textbf{(4-29)}$$

The actual discrete form of the equations used to generate the right and left hand side residuals must accommodate purely internal flow regions, solid wall boundaries, internal/external flow boundaries, and combinations of these.

Since the right hand side does not include pseudo time, the time derivative terms and the grid velocity are dropped for steady state problems. These will be included on the left hand side. Dropping to two dimensions

$$R_{RHS} = \sum_{k=1}^{n}\iint_{V(t)_{e(k)}} W_{ij}\left[\frac{\partial}{\partial x}E + \frac{\partial}{\partial y}F\right]dV_{e(k)} \qquad \textbf{(4-30)}$$

where

$$W = I + \tau\left(\frac{\partial}{\partial x}A_v\Gamma_v^{-1} + \frac{\partial}{\partial y}B_v\Gamma_v^{-1}\right) \tag{4-31}$$

The I term is simply one in a scalar world. It is the identity matrix here but will be used as a placeholder for the discrete interpolation functions.

Separating the weight function into parts

$$\begin{aligned} R_{RHS} &= \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[I + \tau\left(\frac{\partial}{\partial x}A_v\Gamma_v^{-1} + \frac{\partial}{\partial y}B_v\Gamma_v^{-1}\right)\right]\left[\frac{\partial}{\partial x}E + \frac{\partial}{\partial y}F\right]dV_{e(k)} \\ &= \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} [I]\left[\frac{\partial}{\partial x}E + \frac{\partial}{\partial y}F\right]dV_{e(k)} \\ &\quad + \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[\tau\left(\frac{\partial}{\partial x}A_v\Gamma_v^{-1} + \frac{\partial}{\partial y}B_v\Gamma_v^{-1}\right)\right]\left[\frac{\partial}{\partial x}E + \frac{\partial}{\partial y}F\right]dV_{e(k)} \end{aligned} \tag{4-32}$$

or

$$R_{RHS} = \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} [I]\left[\nabla\cdot\begin{Bmatrix}E\\F\end{Bmatrix}\right]dV_{e(k)} + \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[\tau\nabla\cdot\begin{Bmatrix}A_v\Gamma_v^{-1}\\B_v\Gamma_v^{-1}\end{Bmatrix}\right]\left[\nabla\cdot\begin{Bmatrix}E\\F\end{Bmatrix}\right]dV_{e(k)} \tag{4-33}$$

The first term on the right hand side can be rewritten using a Green-Gauss transformation on the first term to create a combination of volume and surface integrals. Using

$$\begin{aligned} \nabla\cdot I\begin{Bmatrix}E\\F\end{Bmatrix} &= \nabla I\cdot\begin{Bmatrix}E\\F\end{Bmatrix} + I\nabla\cdot\begin{Bmatrix}E\\F\end{Bmatrix} \\ I\nabla\cdot\begin{Bmatrix}E\\F\end{Bmatrix} &= \nabla\cdot I\begin{Bmatrix}E\\F\end{Bmatrix} - \nabla I\cdot\begin{Bmatrix}E\\F\end{Bmatrix} \end{aligned} \tag{4-34}$$

and substituting,

$$\begin{aligned} R_{RHS} = &\sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \nabla\cdot I\begin{Bmatrix}E\\F\end{Bmatrix}dV_{e(k)} - \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} [\nabla I]\cdot\begin{Bmatrix}E\\F\end{Bmatrix}dV_{e(k)} \\ &+ \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[\tau\nabla\cdot\begin{Bmatrix}A_v\Gamma_v^{-1}\\B_v\Gamma_v^{-1}\end{Bmatrix}\right]\left[\nabla\cdot\begin{Bmatrix}E\\F\end{Bmatrix}\right]dV_{e(k)} \end{aligned} \tag{4-35}$$

Now, transform the first term to get

$$R_{RHS} = \sum_{k=1}^{n} \oint_{S(t)_{e(k)}} I\begin{Bmatrix} E \\ F \end{Bmatrix} \cdot \hat{n} dS_{e(k)} - \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} [\nabla I] \cdot \begin{Bmatrix} E \\ F \end{Bmatrix} dV_{e(k)}$$
$$+ \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[\tau \left(\frac{\partial}{\partial x} A_v \Gamma_v^{-1} + \frac{\partial}{\partial y} B_v \Gamma_v^{-1} \right) \right] \left[\frac{\partial}{\partial x} E + \frac{\partial}{\partial y} F \right] dV_{e(k)} \tag{4-36}$$

This form of the fluid flow equation set is very useful for all elements and includes a preconditioned numerical diffusion term.

The boundary conditions for the fluid elements are completely implemented through the surface integral. For purely internal flow regions, the surface integrals cancel upon assembly and are not needed. The pressure term in the surface integral must be included for solid wall boundaries but the flux terms are zero. The surface flux is required for inflow/outflow boundaries but the variables are determined based on the characteristics of the flow. For subsonic inflow, velocity and temperature are set as Dirichlet conditions and the pressure is calculated. For subsonic outflow, the pressure is set and the velocity and temperature are calculated. It takes some bookkeeping to keep straight which edges actually require surface integral calculations for combination elements such as a corner element with a solid wall on one edge, an inflow/outflow on an adjacent edge, and internal flow on the others.

For the actual discrete implementation, the primitive variables are stored and calculated at the nodes. In the element by element approach, the various matrices, $\left(A_v, B_v, \Gamma_v, \Gamma_v^{-1}\right)$, are also calculated at the nodes from the current primitive variables and joined with the interpolating functions. On the right hand side, the flux vectors act as the unknown nodal variables but these are easily calculated from the primitive variables.

Let

$$N = \begin{bmatrix} \{N_i\} & 0 & 0 & 0 \\ 0 & \{N_i\} & 0 & 0 \\ 0 & 0 & \{N_i\} & 0 \\ 0 & 0 & 0 & \{N_i\} \end{bmatrix} \tag{4-37}$$

represent the appropriate interpolation function hyper-matrix of a given order for an element and N_s be its companion for surface integration. The right hand side residual comes from the assembly of the element

residuals. In two dimensions, the surface integrals are simply line integrals scaled by the length of the edge and the orientation of the normal to the line. The element integrals are formed by Gauss quadrature.

$$
\begin{aligned}
R_{RHS} = & \sum_{k=1}^{n} \sum_{sides} \oint_{S(t)_{e(k)}} n_x \left[N_s N_s^T \right] dS_{e(k)} \widetilde{E} + \sum_{k=1}^{n} \sum_{sides} \oint_{S(t)_{e(k)}} n_y \left[N_s N_s^T \right] dS_{e(k)} \widetilde{F} \\
& - \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[\frac{\partial N}{\partial x} N^T \right] dV_{e(k)} \widetilde{E} \\
& + \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[\tau \left(\frac{\partial N}{\partial x} A_v \Gamma_v^{-1} + \frac{\partial N}{\partial y} B_v \Gamma_v^{-1} \right) \right] \left[\frac{\partial N^T}{\partial x} \right] dV_{e(k)} \widetilde{E} \\
& - \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[\frac{\partial N}{\partial y} N^T \right] dV_{e(k)} \widetilde{F} \\
& + \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[\tau \left(\frac{\partial N}{\partial x} A_v \Gamma_v^{-1} + \frac{\partial N}{\partial y} B_v \Gamma_v^{-1} \right) \right] \left[\frac{\partial N^T}{\partial y} \right] dV_{e(k)} \widetilde{F}
\end{aligned}
\tag{4-38}
$$

There are several ways to form the terms like $\frac{\partial N}{\partial x} A_v \Gamma_v^{-1}$. One is to interpolate the primitive variables and form the matrices at the Gauss point such that $A_v \Gamma_v^{-1} = f\left(\left\{ N^T \widetilde{p}, \quad N^T \widetilde{u}, \quad N^T \widetilde{v}, \quad N^T \widetilde{T}, \right\} \right)$ This requires the matrices to be formed from scratch for each Gauss point in each element and is time consuming.

If the matrices are formed at the nodes, they can be reused. The question now centers on the proper interpolation method. They could be linearly interpolated from the nodes as some weighted average quantity such that

$$
\overline{A}_v \overline{\Gamma}_v^{-1} = N_i \left(A_v \Gamma_v^{-1} \right)_i \tag{4-39}
$$

However, this ignores the derivation where the gradient operation is supposed to be acting on the preconditioned flux Jacobians.

The best approach is to combine the nodal matrices with the interpolation function. The basic procedure is to form the various matrices, $\left(A_v, B_v, \Gamma_v, \Gamma_v^{-1} \right)$, at the nodes. Then form the specific matrix combinations at the nodes (such as $A_v \Gamma_v^{-1}$). Next, create a directional interpolation hyper-matrix. For

example, if a_{ij} represents the elements of $A_v\Gamma_v^{-1}$ at a given node, then a_{ijm} represents the elements of $A_v\Gamma_v^{-1}$ at node m. The matrix element at each node must be interpolated separately so the combination

$$\begin{aligned}\frac{\partial N}{\partial x}A_v\Gamma_v^{-1} &= \frac{\partial N_m}{\partial x}\left(A_v\Gamma_v^{-1}\right)_m \quad \text{no sum on } m \\ &= \frac{\partial N_m a_{ijm}}{\partial x}\end{aligned} \tag{4-40}$$

In matrix form, the m and i indices are vertical and the j index is horizontal. This creates a full matrix with the first row of each of the nodal matrices multiplied by the interpolation function in the top block. To avoid confusion, let the element have three nodes. The $\frac{\partial N}{\partial x}A_v\Gamma_v^{-1}$ term will be 12×4 with the ordering given by

$$\frac{\partial N}{\partial x}A_v\Gamma_v^{-1} = \left[\begin{array}{cccc} N_1a_{111} & N_1a_{121} & N_1a_{131} & N_1a_{141} \\ N_2a_{112} & N_2a_{122} & N_2a_{132} & N_2a_{142} \\ N_3a_{113} & N_3a_{123} & N_3a_{133} & N_3a_{143} \\ \hline N_1a_{211} & \cdots & & \\ \vdots & & & \\ \hline N_1a_{311} & \cdots & & \\ \vdots & & & \\ \hline N_1a_{411} & \cdots & & \\ \vdots & & & \end{array}\right] \tag{4-41}$$

This corresponds to a nodal variable ordering

$$\tilde{Q}_v = \left\{\begin{array}{c} p_1 \\ \vdots \\ u_1 \\ \vdots \\ v_1 \\ \vdots \\ T_1 \\ \vdots \end{array}\right\} \tag{4-42}$$

The matrix ordering will obviously change if the variable ordering changes. All the matrices are formed using the same approach. The nodal matrices are re-used at each Gauss point within an element and on adjacent elements. I formed the matrices for a block of about 40 nodes at a time.

The left hand side residual equations are similar but the linearized form of the fluid flow equations are used. The left hand side also includes the pseudo time and grid velocity terms. The element integration matrices are formed at the same time as the right hand side residual and stored for left hand side use. The order of the interpolation is thus the same as for the right hand side.

Starting with the weighted integral equation for fluid flow on a moving grid, variants of the equation show how the grid motion creates a source (or sink) for the conservation variables. The linearized form is created around the primitive variables and gives an indication of the terms required in the left hand side analysis.

$$\begin{aligned}
&\frac{d}{dt}\iiint_{V(t)} W_{ij}\Gamma_{v}Q_{v_j}\,dV + \iiint_{V(t)} W_{ij}\left[\frac{\partial}{\partial x_k}Q_j(u_k - w_k) + \frac{\partial}{\partial x_k}\delta_{km}p_m\right]dV = 0 \\
&\frac{d}{dt}\iiint_{V(t)} W_{ij}\Gamma_{v}Q_{v_j}\,dV + \iiint_{V(t)} W_{ij}\left[\frac{\partial}{\partial x_k}E_{jk} - \frac{\partial}{\partial x_k}Q_j w_k\right]dV = 0 \\
&\frac{d}{dt}\iiint_{V(t)} W_{ij}\Gamma_{v}Q_{v_j}\,dV + \iiint_{V(t)} W_{ij}\left[A_{v_{jmk}}\frac{\partial}{\partial x_k}Q_{v_m} - \frac{\partial}{\partial x_k}Q_j w_k\right]dV = 0
\end{aligned} \qquad \textbf{(4-43)}$$

The left hand side Jacobian matrix-vector product is actually the difference between two residual vectors. Each of these is similar to a linearized version of the conservation form. They do not have to be equivalent to the linearized form as long as their difference approximates the Jacobian of the conservation form. For example, E is a homogeneous function of degree one in the conservation variables but not the primitive variables so $E \neq A_v Q_V$. However, its differential gives $dE = A_v dQ_V$. Using a finite difference approximation to the differential, $dE \approx \dfrac{A_v(Q_v + \varepsilon\Delta Q_v) - A_v Q_v}{\varepsilon}$ so $E = A_v Q_v$ is a valid statement in the left hand side residual expression. These residual vectors are formed using the current solution vector of primitive variables and the latest guess at the perturbed primitive variable vector. Since the element integration matrices are stored during the formation of the right hand side residual, these matrices do not change during the GMRES iteration process. The baseline (zero perturbation) left hand

side residual is formed at the start of the GMRES process and stored. Each new Krylov vector then requires only a single evaluation of the left hand side residual operator.

The conservation form is used for most of the left hand side residual formation. The terms are linearized at the last possible step to avoid confusion over how to treat the flux Jacobians on a finite element. Dropping to two dimensional form and switching to vector notation,

$$\begin{aligned} R_{LHS} = &\sum_{k=1}^{n} \frac{d}{dt} \iint_{V(t)_{e(k)}} \left[I + \tau \nabla \cdot \begin{Bmatrix} A_v \Gamma_v^{-1} \\ B_v \Gamma_v^{-1} \end{Bmatrix} \right] [\Gamma_v Q_v] dV_{e(k)} \\ &+ \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} [I] \left[\nabla \cdot \begin{Bmatrix} E \\ F \end{Bmatrix} \right] dV_{e(k)} + \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[\tau \nabla \cdot \begin{Bmatrix} A_v \Gamma_v^{-1} \\ B_v \Gamma_v^{-1} \end{Bmatrix} \right] \left[\nabla \cdot \begin{Bmatrix} E \\ F \end{Bmatrix} \right] dV_{e(k)} \\ &- \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} [I][\nabla \cdot Q\vec{w}] dV_{e(k)} - \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[\tau \nabla \cdot \begin{Bmatrix} A_v \Gamma_v^{-1} \\ B_v \Gamma_v^{-1} \end{Bmatrix} \right] [\nabla \cdot Q\vec{w}] dV_{e(k)} \end{aligned} \tag{4-44}$$

Again, the Green-Gauss theorem transforms the second and fourth terms where

$$\begin{aligned} I\nabla \cdot \begin{Bmatrix} E \\ F \end{Bmatrix} &= \nabla \cdot I \begin{Bmatrix} E \\ F \end{Bmatrix} - \nabla I \cdot \begin{Bmatrix} E \\ F \end{Bmatrix} \\ I\nabla \cdot Q\vec{w} &= \nabla \cdot IQ\vec{w} - \nabla I \cdot Q\vec{w} \end{aligned} \tag{4-45}$$

Substituting into the residual equation gives

$$\begin{aligned} R_{LHS} = &\sum_{k=1}^{n} \frac{d}{dt} \iint_{V(t)_{e(k)}} \left[I + \tau \nabla \cdot \begin{Bmatrix} A_v \Gamma_v^{-1} \\ B_v \Gamma_v^{-1} \end{Bmatrix} \right] [\Gamma_v Q_v] dV_{e(k)} \\ &+ \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \nabla \cdot I \begin{Bmatrix} E \\ F \end{Bmatrix} dV_{e(k)} - \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} [\nabla I] \cdot \begin{Bmatrix} E \\ F \end{Bmatrix} dV_{e(k)} \\ &+ \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[\tau \nabla \cdot \begin{Bmatrix} A_v \Gamma_v^{-1} \\ B_v \Gamma_v^{-1} \end{Bmatrix} \right] \left[\nabla \cdot \begin{Bmatrix} E \\ F \end{Bmatrix} \right] dV_{e(k)} \\ &- \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \nabla \cdot IQ\vec{w} dV_{e(k)} + \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} [\nabla I] \cdot Q\vec{w} dV_{e(k)} \\ &- \sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[\tau \nabla \cdot \begin{Bmatrix} A_v \Gamma_v^{-1} \\ B_v \Gamma_v^{-1} \end{Bmatrix} \right] [\nabla \cdot Q\vec{w}] dV_{e(k)} \end{aligned} \tag{4-46}$$

Now transform the volume integrals to surface integrals on the second and fifth terms to get

$$
\begin{aligned}
R_{LHS} = & \sum_{k=1}^{n} \frac{d}{dt} \iiint_{V(t)_{e(k)}} \left[I + \tau \nabla \cdot \begin{Bmatrix} A_v \Gamma_v^{-1} \\ B_v \Gamma_v^{-1} \end{Bmatrix} \right] [\Gamma_v Q_v] dV_{e(k)} \\
& + \sum_{k=1}^{n} \oint_{S(t)_{e(k)}} I \begin{Bmatrix} E \\ F \end{Bmatrix} \cdot \hat{n} dS_{e(k)} - \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} [\nabla I] \cdot \begin{Bmatrix} E \\ F \end{Bmatrix} dV_{e(k)} \\
& + \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} \left[\tau \nabla \cdot \begin{Bmatrix} A_v \Gamma_v^{-1} \\ B_v \Gamma_v^{-1} \end{Bmatrix} \right] \left[\nabla \cdot \begin{Bmatrix} E \\ F \end{Bmatrix} \right] dV_{e(k)} \\
& - \sum_{k=1}^{n} \oint_{S(t)_{e(k)}} IQ\vec{w} \cdot \hat{n} dS_{e(k)} + \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} [\nabla I] \cdot Q\vec{w} dV_{e(k)} \\
& - \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} \left[\tau \nabla \cdot \begin{Bmatrix} A_v \Gamma_v^{-1} \\ B_v \Gamma_v^{-1} \end{Bmatrix} \right] [\nabla \cdot Q\vec{w}] dV_{e(k)}
\end{aligned}
\tag{4-47}
$$

Finally, it is time to linearize the flux vector terms but the chain rule must be satisfied. If $G = \frac{\partial E}{\partial x}$, then

$$
dG = \frac{\partial G}{\partial Q_v} dQ_v = \frac{\partial}{\partial Q_v} \frac{\partial E}{\partial x} dQ_v = \frac{\partial}{\partial x} \frac{\partial E}{\partial Q_v} dQ_v = \frac{\partial}{\partial x} A_v dQ_v \tag{4-48}
$$

Substituting for the flux vectors gives

$$
\begin{aligned}
R_{LHS} = & \sum_{k=1}^{n} \frac{d}{dt} \iiint_{V(t)_{e(k)}} \left[I + \tau \nabla \cdot \begin{Bmatrix} A_v \Gamma_v^{-1} \\ B_v \Gamma_v^{-1} \end{Bmatrix} \right] [\Gamma_v Q_v] dV_{e(k)} \\
& + \sum_{k=1}^{n} \oint_{S(t)_{e(k)}} I \begin{Bmatrix} A_v Q_v \\ B_v Q_v \end{Bmatrix} \cdot \hat{n} dS_{e(k)} - \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} [\nabla I] \cdot \begin{Bmatrix} A_v Q_v \\ B_v Q_v \end{Bmatrix} dV_{e(k)} \\
& + \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} \left[\tau \nabla \cdot \begin{Bmatrix} A_v \Gamma_v^{-1} \\ B_v \Gamma_v^{-1} \end{Bmatrix} \right] \left[\nabla \cdot \begin{Bmatrix} A_v Q_v \\ B_v Q_v \end{Bmatrix} \right] dV_{e(k)} \\
& - \sum_{k=1}^{n} \oint_{S(t)_{e(k)}} IQ\vec{w} \cdot \hat{n} dS_{e(k)} + \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} [\nabla I] \cdot Q\vec{w} dV_{e(k)} \\
& - \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} \left[\tau \nabla \cdot \begin{Bmatrix} A_v \Gamma_v^{-1} \\ B_v \Gamma_v^{-1} \end{Bmatrix} \right] [\nabla \cdot Q\vec{w}] dV_{e(k)}
\end{aligned}
\tag{4-49}
$$

The left hand side gets a little more complicated in discrete form but it repeats the same basic analysis procedure as the right hand side. Again, let N and N_s be the interpolation functions. Expand all

the products and discretize each term. The interpolation terms will be coupled with the various nodal matrices as described in the right hand side derivation. The surface integrals are only needed for inflow/outflow boundaries and to correct for pressure force contributions to the momentum along solid walls. Otherwise, the surface flux terms either cancel or are zero. If pseudo time stepping is used, the Δt term is included in the preconditioning matrix. Since the integration matrices are stored and recalled from a right hand side calculation, the volume integral doesn't actually change with time so the first term in equation **(4-49)** is actually

$$\sum_{k=1}^{n} \iint_{V(t)_{e(k)}} \left[I + \tau \nabla \cdot \begin{Bmatrix} A_v \Gamma_v^{-1} \\ B_v \Gamma_v^{-1} \end{Bmatrix} \right] \left[\Gamma_v \dot{Q}_v \right] dV_{e(k)} \qquad \textbf{(4-50)}$$

The correct volume integral change-with-time would be created if the matrices were reformed with each statement of the left hand side residual. The simplification (and speedup) did not seem to have any adverse effects since this is artificial transient. Also, the simplified term is correct for a fixed grid or one with constant nodal velocities. The full discrete statement for the left hand side residual is expanded in equation **(4-51)**.

Notice there are two primary sets of unknown nodal variables: the primitive variables, Q_v, in the first six terms and the x and y direction grid velocities, $\vec{w}$, in the second six terms. The left hand side residual statement with grid motion now has more unknowns than equations per node so additional information is needed to bring closure. The grid motion equation set will provide the additional equations. The grid motion serves as the intermediary for coupling the fluid to the structure but the pseudo transient grid motion is only needed for the tightly coupled approach. In the loosely coupled approach, the fluid grid is constant and cancels in the finite difference approximation to the derivative.

In equation **(4-51)**, the conserved variables, Q, are treated as fixed quantities within the fluid domain and the source (or sink) in the conserved variables mentioned above comes from grid motion along a gradient of these variables. Again, if the grid motion is zero or constant, these terms are not needed since they will cancel. This hold in both physical and pseudo time for the loosely coupled approach.

$$
\begin{aligned}
R_{LHS} = & \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} \left[N + \tau \left(\frac{\partial N}{\partial x} A_v \Gamma_v^{-1} + \frac{\partial N}{\partial y} B_v \Gamma_v^{-1} \right) \right] \left[N^T \frac{\Gamma_v}{\Delta t} \right] dV_{e(k)} \tilde{Q}_v \\
& + \sum_{k=1}^{n} \sum_{sides} \oint_{S(t)_{e(k)}} n_x \left[N_s \right] \left[N_s^T A_v \right] dS_{e(k)} \tilde{Q}_v + \sum_{k=1}^{n} \sum_{sides} \oint_{S(t)_{e(k)}} n_y \left[N_s \right] \left[N_s^T B_v \right] dS_{e(k)} \tilde{Q}_v \\
& - \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} \left[\frac{\partial N}{\partial x} \right] \left[N^T A_v \right] dV_{e(k)} \tilde{Q}_v \\
& - \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} \left[\frac{\partial N}{\partial y} \right] \left[N^T B_v \right] dV_{e(k)} \tilde{Q}_v \\
& + \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} \left[\tau \left(\frac{\partial N}{\partial x} A_v \Gamma_v^{-1} + \frac{\partial N}{\partial y} B_v \Gamma_v^{-1} \right) \right] \left[\frac{\partial N^T}{\partial x} A_v \right] dV_{e(k)} \tilde{Q}_v \\
& + \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} \left[\tau \left(\frac{\partial N}{\partial x} A_v \Gamma_v^{-1} + \frac{\partial N}{\partial y} B_v \Gamma_v^{-1} \right) \right] \left[\frac{\partial N^T}{\partial y} B_v \right] dV_{e(k)} \tilde{Q}_v \\
& - \sum_{k=1}^{n} \sum_{sides} \oint_{S(t)_{e(k)}} n_x \left[N_s N_s^T Q \right] dS_{e(k)} \tilde{w}_x - \sum_{k=1}^{n} \sum_{sides} \oint_{S(t)_{e(k)}} n_y \left[N_s N_s^T Q \right] dS_{e(k)} \tilde{w}_y \\
& + \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} \left[\frac{\partial N}{\partial x} N^T Q \right] dV_{e(k)} \tilde{w}_x \\
& - \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} \left[\tau \left(\frac{\partial N}{\partial x} A_v \Gamma_v^{-1} + \frac{\partial N}{\partial y} B_v \Gamma_v^{-1} \right) \right] \left[\frac{\partial N^T}{\partial x} Q \right] dV_{e(k)} \tilde{w}_x \\
& + \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} \left[\frac{\partial N}{\partial y} N^T Q \right] dV_{e(k)} \tilde{w}_y \\
& - \sum_{k=1}^{n} \iiint_{V(t)_{e(k)}} \left[\tau \left(\frac{\partial N}{\partial x} A_v \Gamma_v^{-1} + \frac{\partial N}{\partial y} B_v \Gamma_v^{-1} \right) \right] \left[\frac{\partial N^T}{\partial y} Q \right] dV_{e(k)} \tilde{w}_y
\end{aligned}
\tag{4-51}
$$

The boundary conditions for the fluid element are still completely implemented through the surface integral. For purely internal flow regions, the surface integrals all cancel upon assembly and are not needed. The pressure term in the surface integral must be included for solid wall boundaries but the flux terms are zero. The wall may be moving though so the grid velocity surface integrals must also be included. The surface flux is required for inflow/outflow boundaries but the variables are determined based

on the characteristics of the flow. The inflow/outflow boundaries may be moving so the grid velocity surface integrals must be considered.

CHAPTER 5

DERIVATION OF A TAYLOR WEAK STATEMENT

One Dimensional Inviscid Fluid Flow with Conservation Variables

The conservation form of the single dimensional equation for inviscid fluid flow is given by

$$\frac{\partial q}{\partial t}+\frac{\partial f(q)}{\partial x}=0 \tag{5-1}$$

where q is a vector of conserved variables. This can be expanded as a time Taylor series to describe the evolutionary process in discrete time as the system moves from the current (initial) state and asymptotically approaches the desired steady (or pseudo steady) state.

$$q^{n+1}=q^{n}+\frac{\partial q^{n}}{\partial t}\Delta t+\frac{\partial^{2} q^{n}}{\partial t^{2}}\frac{\Delta t^{2}}{2!}+\frac{\partial^{3} q^{n}}{\partial t^{3}}\frac{\Delta t^{3}}{3!}+\vartheta\left(\Delta t^{4}\right) \tag{5-2}$$

Solving for $\frac{\partial q}{\partial t}$ gives

$$\frac{\partial q^{n}}{\partial t}=\frac{q^{n+1}-q^{n}}{\Delta t}-q_{tt}\frac{\Delta t}{2!}-q_{ttt}\frac{\Delta t^{2}}{3!}-\vartheta\left(\Delta t^{3}\right) \tag{5-3}$$

where $q_{tt}=\frac{\partial^{2} q}{\partial t^{2}}$, etc.

Using a similar approach, the space dimension may be expanded about a point to a desired order of accuracy as

$$\frac{\partial f(q)}{\partial x}=D\left(f(q)\right)\pm\vartheta\left(\Delta x^{k}\right) \tag{5-4}$$

where $D\left(f(q)\right)$ describes a generic spatial discretization of the flux vector. Substituting the Taylor series into the original fluid flow equation gives

$$\frac{\partial q}{\partial t}+\frac{\partial f(q)}{\partial x}=\frac{q^{n+1}-q^{n}}{\Delta t}+D\left(f\left(q^{n}\right)\right)-q_{tt}^{n}\frac{\Delta t}{2!}-q_{ttt}^{n}\frac{\Delta t^{2}}{3!}\pm\vartheta\left(\Delta t^{3},\Delta x^{k}\right) \tag{5-5}$$

Now, since $q_t=-f_x$ and the functions are continuous in time and space, then

$$q_{tt} = -(f_x)_t = -(f_t)_x = -(f_q q_t)_x = -\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial q}\frac{\partial q}{\partial t}\right) \tag{5-6}$$

Defining $A \equiv \frac{\partial f}{\partial q}$ yields several useful, equivalent statements for q_{tt}:

$$q_{tt} = -\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right) = -\frac{\partial}{\partial x}\left(A\left(-\frac{\partial f}{\partial x}\right)\right) = \frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right) = \frac{\partial}{\partial x}\left(AA\frac{\partial q}{\partial x}\right) \tag{5-7}$$

[Donea, 1984] used a time Taylor series expansion as outline above to initiate the Taylor-Galerkin method for convective problems using finite elements. The new scheme used only the spatial form of q_{tt} (and similar terms for q_{ttt}) to restate the convective problem as

$$\underbrace{\frac{\partial q}{\partial t} + \frac{\partial f(q)}{\partial x}}_{\textit{original PDE}} = \underbrace{\underbrace{\frac{q^{n+1}-q^n}{\Delta t} + D\left(f\left(q^n\right)\right)}_{\textit{basic discrete}} - \frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right)\frac{\Delta t}{2!} - q^n_{ttt}\frac{\Delta t^2}{3!} \pm \vartheta\left(\Delta t^3, \Delta x^k\right)}_{\textit{enhanced discrete}} \tag{5-8}$$

If the basic form of the discrete equation is solved exactly $\left(\frac{q^{n+1}-q^n}{\Delta t} + D\left(f\left(q^n\right)\right) = 0\right)$, the solution satisfies an equivalent partial differential equation

$$\frac{\partial q}{\partial t} + \frac{\partial f(q)}{\partial x} = -\frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right)\frac{\Delta t}{2!} \pm \vartheta\left(\Delta t^2, \Delta x^k\right) \tag{5-9}$$

This new differential equation contains a negative viscosity term that makes the equation unstable. In order to satisfy the original partial differential equation (PDE), the discrete equation must include the derivative of the flux gradient term. This also makes the finite element solution process stable.

$$\frac{q^{n+1}-q^n}{\Delta t} + D\left(f\left(q^n\right)\right) = \frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right)\frac{\Delta t}{2!} \tag{5-10}$$

[Donea, 1984] also includes the q_{ttt} term and reformulates is as a mixed space/time derivative

$$q_{ttt} = \left[\frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right)\right]_t = \frac{\partial}{\partial x}\left[A\frac{\partial}{\partial x}\frac{\partial f}{\partial t}\right] = \frac{\partial}{\partial x}\left[A\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial q}\frac{\partial q}{\partial t}\right)\right] = \frac{\partial}{\partial x}\left[A\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right)\right] \tag{5-11}$$

He includes the mixed derivative as a correction to the mass matrix for the time term. This leads to

$$\left(I-\frac{\Delta t^2}{3!}\frac{\partial}{\partial x}A\frac{\partial}{\partial x}A\right)\frac{q^{n+1}-q^n}{\Delta t}+D\left(f\left(q^n\right)\right)=\frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right)\frac{\Delta t}{2!} \tag{5-12}$$

or

$$\left(I-\frac{\Delta t^2}{3!}\frac{\partial}{\partial x}A\frac{\partial}{\partial x}A\right)\frac{q^{n+1}-q^n}{\Delta t}+\left(I-\frac{\Delta t}{2!}\frac{\partial}{\partial x}A\right)D\left(f\left(q^n\right)\right)=0 \tag{5-13}$$

The additional term in the spatial discretization adds artificial viscosity while the extra term in the mass matrix (since it could be written in space as $\frac{\Delta t^2}{3!}\frac{\partial}{\partial x}\left\{A\frac{\partial}{\partial x}\left[A\left(-\frac{\partial q}{\partial x}\right)\right]\right\}$) influences the dispersive component of the truncation error.

The equation reveals the diffusion and dispersion roles of the various terms but the only real measure of control is in the form of the time discretization (Euler, backward Euler, leapfrog, Crank-Nicholson, etc) and the order of the truncation. A more flexible approach comes from [Baker & Kim, 1987] where Donea's method is generalized to the Taylor weak statement (TWS). The basic approach is the same but the higher order time derivatives are formed from a combination of the possible time and space forms derived.

$$q_{tt}=-\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right)=\frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right) \tag{5-14}$$

q_{ttt} follows a similar development.

[Baker & Kim, 1987] form a "convex" sum of the forms above to get

$$q_{tt}=\alpha\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right)+\beta\frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right) \tag{5-15}$$

where $-\alpha+\beta=1$. (Note: $\alpha+\beta=1$ gives the same answer.) This can be generalized for a system of equations where α and β are diagonal matrices and $-\alpha+\beta=I$. The diagonal elements do not have to be equal in the given α or β matrices but must satisfy the convex relationship.

Substituting for q_{tt} in the original equation gives

$$\frac{\partial q}{\partial t}+\frac{\partial f(q)}{\partial x}=\frac{q^{n+1}-q^{n}}{\Delta t}+D\left(f\left(q^{n}\right)\right)$$
$$-\left[\alpha\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right)+\beta\frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right)\right]\frac{\Delta t}{2!} \qquad \textbf{(5-16)}$$
$$-q_{ttt}^{n}\frac{\Delta t^{2}}{3!}\pm\vartheta\left(\Delta t^{3},\Delta x^{k}\right)$$

This gives a great deal of control within the constraints of the "convex" summation. [Baker & Kim, 1987] go on to show how specific choices for α and β (also γ and μ for the q_{ttt} term) are actually equivalent to numerous popular algorithms of the time. Their landmark completion of the TWS provides a framework for analysis of various finite element algorithms for fluid flow.

Setting $\beta=1+\alpha$

$$\frac{\partial q}{\partial t}+\frac{\partial f(q)}{\partial x}=\frac{q^{n+1}-q^{n}}{\Delta t}+D\left(f\left(q^{n}\right)\right)$$
$$-\left[\alpha\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right)+(1+\alpha)\frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right)\right]\frac{\Delta t}{2!} \qquad \textbf{(5-17)}$$
$$-q_{ttt}^{n}\frac{\Delta t^{2}}{3!}\pm\vartheta\left(\Delta t^{3},\Delta x^{k}\right)$$

and

$$\underbrace{\frac{\partial q}{\partial t}+\frac{\partial f(q)}{\partial x}}_{\textit{original PDE}}=\underbrace{\underbrace{\frac{q^{n+1}-q^{n}}{\Delta t}+D\left(f\left(q^{n}\right)\right)}_{\textit{basic discrete}}-\alpha\frac{\Delta t}{2!}\left[\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right)+\frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right)\right]}_{\textit{enhanced discrete}} \qquad \textbf{(5-18)}$$
$$-\frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right)\frac{\Delta t}{2!}-q_{ttt}^{n}\frac{\Delta t^{2}}{3!}\pm\vartheta\left(\Delta t^{3},\Delta x^{k}\right)$$

If the enhanced discrete algorithm is solved exactly, the above form implies a negative diffusion term is still present as generated by the second order time manipulation. It may be eliminated by interactions with the third order time term but it may also be avoided.

If the convex constraint is relaxed so that $-\alpha+\beta=0$ or $\alpha=\beta$, the extra diffusion term is not produced. For $\alpha=\beta\neq 0$, this is the equivalent of stating $q_{tt}=0$. From above,

$$q_{tt} = -\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right) = \frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right) \tag{5-19}$$

For a general convex (weighted) summation where $a+b=c$, q_{tt} may be written

$$cq_{tt} = -a\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right) + b\frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right) \tag{5-20}$$

or

$$q_{tt} = -\frac{a}{c}\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right) + \frac{b}{c}\frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right) \tag{5-21}$$

In this case $a=-\alpha, b=\beta, c=0$ so

$$q_{tt} = \frac{\alpha}{0}\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right) + \frac{\beta}{0}\frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right) \tag{5-22}$$

Either $q_{tt} = -\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right) = \frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right) \equiv 0$ or $q_{tt} \to \infty$ for $\alpha = \beta \neq 0$. The problem (maybe) comes in justifying the conditions in which $q_{tt} = 0$. If it is only used as a pseudo-transient to relax to a steady state, it may not matter. It is possible to enforce $q_{tt} = 0$ in the solution process but this implies q_t is constant. An alternative may be to manipulate the time Taylor series by adding a forward and backward time step to form a leapfrog time scheme that eliminates q_{tt} altogether. Baker and Kim made the same observations, noting that assuming $q_{tt} = 0$ voids the exact Taylor series. They show that the choice $\alpha = \beta \neq 0$ with third order time terms set to zero corresponds to the [Raymond and Garder, 1976] formulation.

Setting $\alpha = \beta \neq 0$ leads to the form

$$\begin{aligned}\frac{\partial q}{\partial t} + \frac{\partial f(q)}{\partial x} = \frac{q^{n+1}-q^n}{\Delta t} + D\left(f\left(q^n\right)\right) \\ -\alpha\left[\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right) + \frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right)\right]\frac{\Delta t}{2!} \\ -q_{ttt}^n\frac{\Delta t^2}{3!} \pm \vartheta\left(\Delta t^3, \Delta x^k\right)\end{aligned} \tag{5-23}$$

If the convex terms are changed so that $\alpha+\beta=0$ (or rearranging to $-\alpha=\beta\neq 0$), the equation changes slightly to

$$\frac{\partial q}{\partial t}+\frac{\partial f(q)}{\partial x}=\frac{q^{n+1}-q^{n}}{\Delta t}+D\left(f\left(q^{n}\right)\right)+\alpha\left[\frac{\partial}{\partial x}\left(A\frac{\partial q}{\partial t}\right)+\frac{\partial}{\partial x}\left(A\frac{\partial f}{\partial x}\right)\right]\frac{\Delta t}{2!}-q^{n}_{ttt}\frac{\Delta t^{2}}{3!}\pm\vartheta\left(\Delta t^{3},\Delta x^{k}\right) \tag{5-24}$$

Notice the discrete portion of the equation as it may also be written as

$$\frac{\partial q}{\partial t}+\frac{\partial f(q)}{\partial x}=\left(I+\alpha\frac{\Delta t}{2!}\frac{\partial}{\partial x}A\right)\left[\frac{q^{n+1}-q^{n}}{\Delta t}+D\left(f\left(q^{n}\right)\right)\right]-q^{n}_{ttt}\frac{\Delta t^{2}}{3!}\pm\vartheta\left(\Delta t^{3},\Delta x^{k}\right) \tag{5-25}$$

where $\frac{\partial}{\partial x}A$ is an operator.

This form does not produce negative viscosity and essentially reconstructs the Streamline-Upwind Petrov- Galerkin (SUPG) method of [Brooks and Hughes, 1982]. It is also interesting to note the leading error term above is a dispersion term. This is consistent with the documented need [Hughes II, 1986] for a discontinuity capturing term to control oscillations in flow regions with sharp internal gradients such as shocks or boundary layers.

The evolving SUPG method has been thoroughly studied and extensively documented in technical literature. [Hughes et al. Parts I - IV, 1986; Tezduyar and Park, 1986; Tezduyar et al., 1987; Tezduyar, 1992; Johnson, 1992; Hughes et al. Part VI, 1987; Hughes, 1987; Hughes, 1992; Tezduyar, 1992; Aliabadi and Tezduyar, 1993; LeBeau et al., 1993; Idelsohn et al., 1996] The most extensive treatment is given to selecting the form of α as a scalar term or a matrix of intrinsic time scales and the development of a discontinuity capturing term. [Hughes, 1987, p. 278] noted the need for a deeper understanding of the process for selecting the intrinsic time scales.

One Dimensional Inviscid Fluid Flow with Primitive Variables

If the original fluid flow equation is modified by a change to primitive variables, the equation may be written

$$Q_v \frac{\partial w}{\partial t} + \frac{\partial f_v(w)}{\partial x} = 0 \tag{5-26}$$

or

$$Q_v \frac{\partial w}{\partial t} = -\frac{\partial f_v(w)}{\partial x} \tag{5-27}$$

or

$$\frac{\partial w}{\partial t} + Q_v^{-1} \frac{\partial f_v(w)}{\partial x} = 0 \tag{5-28}$$

or

$$\frac{\partial w}{\partial t} = -Q_v^{-1} \frac{\partial f_v(w)}{\partial x} \tag{5-29}$$

where Q_v is the jacobian matrix for the transformed variables and $f_v(w)$ is the flux function based on the new variables. The development of an appropriate discrete formulation follows the same approach as before but with some alternative versions.

$$w^{n+1} = w^n + w_t^n \Delta t + w_{tt}^n \frac{\Delta t^2}{2!} + \vartheta(\Delta t^3) \tag{5-30}$$

and

$$Q_v w^{n+1} = Q_v w^n + Q_v w_t^n \Delta t + Q_v w_{tt}^n \frac{\Delta t^2}{2!} + \vartheta(\Delta t^3) \tag{5-31}$$

Therefore,

$$\frac{\partial w}{\partial t} = \frac{w^{n+1} - w^n}{\Delta t} - \frac{\partial^2 w^n}{\partial t^2} \frac{\Delta t}{2!} - \vartheta(\Delta t^2) \tag{5-32}$$

or

$$Q_v \frac{\partial w}{\partial t} = Q_v \frac{w^{n+1} - w^n}{\Delta t} - Q_v \frac{\partial^2 w^n}{\partial t^2} \frac{\Delta t}{2!} - \vartheta(\Delta t^2) \tag{5-33}$$

Now, $w_{tt} = -\left(Q_v^{-1}\frac{\partial f_v}{\partial x}\right)_t$ and $Q_v w_{tt} = -\left(\frac{\partial f_v}{\partial x}\right)_t$ so $w_{tt} = -Q_v^{-1}\left(\frac{\partial}{\partial x}\frac{\partial f_v}{\partial t}\right)$ and

$Q_v w_{tt} = -\left(\frac{\partial}{\partial x}\frac{\partial f_v}{\partial t}\right)$. A choice arises where $\frac{\partial f_v}{\partial t}$ can be transformed in either the primitive or

conservation variables. Primitive variables lead to

$$\begin{aligned} w_{tt} &= -Q_v^{-1}\frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\frac{\partial w}{\partial t}\right) \\ &= -Q_v^{-1}\frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\left(-Q_v^{-1}\right)\frac{\partial f_v}{\partial x}\right) \\ &= Q_v^{-1}\frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\left(Q_v^{-1}\right)\frac{\partial f_v}{\partial x}\right) \end{aligned} \tag{5-34}$$

and

$$\begin{aligned} Q_v w_{tt} &= -\frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\frac{\partial w}{\partial t}\right) \\ &= -\frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\left(-Q_v^{-1}\right)\frac{\partial f_v}{\partial x}\right) \\ &= \frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\left(Q_v^{-1}\right)\frac{\partial f_v}{\partial x}\right) \end{aligned} \tag{5-35}$$

while conservation variables may be used to create the forms

$$\begin{aligned} w_{tt} &= -Q_v^{-1}\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial q}\frac{\partial q}{\partial t}\right) \\ &= -Q_v^{-1}\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial q}\left(-\frac{\partial f}{\partial x}\right)\right) \\ &= Q_v^{-1}\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial q}\left(\frac{\partial f}{\partial x}\right)\right) \end{aligned} \tag{5-36}$$

and

$$
\begin{aligned}
Q_v w_{tt} &= -\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial q}\frac{\partial q}{\partial t}\right) \\
&= -\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial q}\left(-\frac{\partial f}{\partial x}\right)\right) \\
&= \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial q}\left(\frac{\partial f}{\partial x}\right)\right)
\end{aligned}
\tag{5-37}
$$

The four versions are equivalent statements of the time derivative of the original fluid flow equation if Q_v is the physical transformation jacobian $\frac{\partial q}{\partial w}$.

$$
\frac{\partial q}{\partial t} = -\frac{\partial f}{\partial x} = -A\frac{\partial q}{\partial x} = -A\frac{\partial q}{\partial w}\frac{\partial w}{\partial x} = -AQ_v\frac{\partial w}{\partial x} \tag{5-38}
$$

$$
\frac{\partial q}{\partial t} = Q_v\frac{\partial w}{\partial t} = -\frac{\partial f_v}{\partial x} = -A_v\frac{\partial w}{\partial x} \tag{5-39}
$$

so $AQ_v = A_v$ and $A = A_v Q_v^{-1}$. However, if $Q_v \to \Gamma_v$ is viewed as a preconditioning mass matrix chosen for convenience on the pseudo-transient, then $A \neq A_v\Gamma_v^{-1}$. (note: A_v is always based on the physical variable transformation.)

Returning to the second time derivatives, $w_{tt} = Q_v^{-1}\frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\left(Q_v^{-1}\right)\frac{\partial f_v}{\partial x}\right)$ and

$Q_v w_{tt} = \frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\left(Q_v^{-1}\right)\frac{\partial f_v}{\partial x}\right)$, form the convex summation as before where $\alpha + \beta = 0 \to -\alpha = \beta$:

$$
\begin{aligned}
w_{tt} &= -\alpha Q_v^{-1}\frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\frac{\partial w}{\partial t}\right) + \beta Q_v^{-1}\frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\left(Q_v^{-1}\right)\frac{\partial f_v}{\partial x}\right) \\
&= -\alpha Q_v^{-1}\frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\left(\frac{\partial w}{\partial t} + Q_v^{-1}\frac{\partial f_v}{\partial x}\right)\right) \\
&= -\alpha Q_v^{-1}\frac{\partial}{\partial x}\left(A_v\left(\frac{\partial w}{\partial t} + Q_v^{-1}\frac{\partial f_v}{\partial x}\right)\right)
\end{aligned}
\tag{5-40}
$$

or

$$
\begin{aligned}
Q_v w_{tt} &= -\alpha \frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\frac{\partial w}{\partial t}\right) + \beta \frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\left(Q_v^{-1}\right)\frac{\partial f_v}{\partial x}\right) \\
&= -\alpha \frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\left(\frac{\partial w}{\partial t} + Q_v^{-1}\frac{\partial f_v}{\partial x}\right)\right) \\
&= -\alpha \frac{\partial}{\partial x}\left(A_v\left(\frac{\partial w}{\partial t} + Q_v^{-1}\frac{\partial f_v}{\partial x}\right)\right)
\end{aligned}
\tag{5-41}
$$

For a given matrix β or α, in general, $Q_v^{-1}\beta \neq \beta Q_v^{-1}$, even if β is diagonal. However, if $\beta = bI$, then $Q_v^{-1}bI = bIQ_v^{-1}$. The implication is that α and β should be scalar in order to make the two forms above equivalent.

Moving to the space dimension, the discrete form may be written as

$$
\frac{\partial f_v(w)}{\partial x} = D\left(f_v(w)\right) \pm \vartheta\left(\Delta x^k\right) \tag{5-42}
$$

where $D\left(f_v(w)\right)$ describes a generic spatial discretization of the flux vector. Substituting the two time Taylor series and the space discretization into the two versions of the original fluid flow equation gives three alternative versions of the equivalent partial differential equation. Using the w_{tt} form gives

$$
\begin{aligned}
\frac{\partial w}{\partial t} + Q_v^{-1}\frac{\partial f_v(w)}{\partial x} = \frac{w^{n+1} - w^n}{\Delta t} + Q_v^{-1} D\left(f_v\left(w^n\right)\right) + \frac{\Delta t}{2!}\alpha Q_v^{-1}\frac{\partial}{\partial x}\left(A_v\left(\frac{\partial w}{\partial t} + Q_v^{-1}\frac{\partial f_v}{\partial x}\right)\right) \\
- w_{ttt}^n \frac{\Delta t^2}{3!} \pm \vartheta\left(\Delta t^3, \Delta x^k\right)
\end{aligned}
\tag{5-43}
$$

Multiply equation by Q_v to get

$$
\begin{aligned}
Q_v\frac{\partial w}{\partial t} + \frac{\partial f_v(w)}{\partial x} = Q_v\frac{w^{n+1} - w^n}{\Delta t} + D\left(f_v\left(w^n\right)\right) + \frac{\Delta t}{2!}Q_v\alpha Q_v^{-1}\frac{\partial}{\partial x}\left(A_v\left(\frac{\partial w}{\partial t} + Q_v^{-1}\frac{\partial f_v}{\partial x}\right)\right) \\
- Q_v w_{ttt}^n \frac{\Delta t^2}{3!} \pm \vartheta\left(\Delta t^3, \Delta x^k\right)
\end{aligned}
\tag{5-44}
$$

Using the $Q_v w_{tt}$ form leads to

$$Q_v\frac{\partial w}{\partial t}+\frac{\partial f_v(w)}{\partial x}=Q_v\frac{w^{n+1}-w^n}{\Delta t}+D\left(f_v\left(w^n\right)\right)+\frac{\Delta t}{2!}\alpha\frac{\partial}{\partial x}\left(A_v\left(\frac{\partial w}{\partial t}+Q_v^{-1}\frac{\partial f_v}{\partial x}\right)\right)$$
$$-Q_v w_{ttt}^n\frac{\Delta t^2}{3!}\pm\vartheta\left(\Delta t^3,\Delta x^k\right) \tag{5-45}$$

Again, the two forms are the same if α is a scalar.

Equation **(5-43)** can be factored as

$$\frac{\partial w}{\partial t}+Q_v^{-1}\frac{\partial f_v(w)}{\partial x}=\left(I+\frac{\Delta t}{2!}\alpha Q_v^{-1}\frac{\partial}{\partial x}A_v\right)\left(\frac{w^{n+1}-w^n}{\Delta t}+Q_v^{-1}D\left(f_v\left(w^n\right)\right)\right)$$
$$-w_{ttt}^n\frac{\Delta t^2}{3!}\pm\vartheta\left(\Delta t^3,\Delta x^k\right) \tag{5-46}$$

Equations **(5-44)** and **(5-45)** can also be factored as

$$Q_v\frac{\partial w}{\partial t}+\frac{\partial f_v(w)}{\partial x}=\left(I+\frac{\Delta t}{2!}\overbrace{\underbrace{Q_v\alpha Q_v^{-1}}_{C}\frac{\partial}{\partial x}A_vQ_v^{-1}}^{E}\right)\left(Q_v\frac{w^{n+1}-w^n}{\Delta t}+D\left(f_v\left(w^n\right)\right)\right)$$
$$-Q_v w_{ttt}^n\frac{\Delta t^2}{3!}\pm\vartheta\left(\Delta t^3,\Delta x^k\right) \tag{5-47}$$

and

$$Q_v\frac{\partial w}{\partial t}+\frac{\partial f_v(w)}{\partial x}=\left(I+\frac{\Delta t}{2!}\alpha\frac{\partial}{\partial x}A_vQ_v^{-1}\right)\left(Q_v\frac{w^{n+1}-w^n}{\Delta t}+D\left(f_v\left(w^n\right)\right)\right)$$
$$-Q_v w_{ttt}^n\frac{\Delta t^2}{3!}\pm\vartheta\left(\Delta t^3,\Delta x^k\right) \tag{5-48}$$

In general, for a full matrix, Q_v, and diagonal α, $C=Q_v\alpha Q_v^{-1}$ is not diagonal. However, $C=Q_v\alpha Q_v^{-1}$ is similar to α. Similar matrices have the same characteristic polynomial and the same eigenvalues so even if α is a diagonal matrix of intrinsic time scales rather than a scalar, the form $C=Q_v\alpha Q_v^{-1}$ should not produce a different result than using α alone. Equations **(5-47)** and **(5-48)** are equivalent if α is a scalar.

A second similarity comes from the term $E = Q_v \alpha Q_v^{-1} \frac{\partial}{\partial x} A_v Q_v^{-1} = Q_v N Q_v^{-1}$ where

$N = \alpha Q_v^{-1} \frac{\partial}{\partial x} A_v$. Again, the similar matrix forms have the same eigenvalues even though the eigenvectors will be different. The term $N = \alpha Q_v^{-1} \frac{\partial}{\partial x} A_v$ is the same as the term in equation **(5-46)**. Equation **(5-48)** has the same terms but the order is different: $\alpha \frac{\partial}{\partial x} A_v Q_v^{-1}$. For scalar α, the difference is pre vs. post multiplying by Q_v^{-1}. At this point there is strong enough evidence to assume α is a scalar.

Equation **(5-48)** gives the correct form for an "upwinded" weighting term. This represents a modified version of the original SUPG form that accommodates a time term with a general mass matrix. It shows how the basic conservation variable equations may be transformed to any variables without having to rely on an initial linearized form of the equation, conversion to entropy variables, or on symmetry arguments for Q_v and A_v. It does not require an assumption of divergence free flow, $\nabla \cdot [A_v, B_v] = 0$. At worst it shows α to be a diagonal matrix related to $C = Q_v \alpha Q_v^{-1}$ and, at best, α may be safely treated as a scalar. The flux jacobian matrix product is still to the right of the gradient operator so it captures the effect desired by [Aliabadi and Tezduyar, 1993] because the SUPG term becomes stronger or weaker based on the flux jacobian gradient. Thus the flux gradient and physical dimensions are connected without trying to develop a different α for each physical dimension or artificially introducing a gradient based weighting.

For pseudo transient solutions, the mass matrix is often modified to provide better convergence properties. The form of the weighting term does not change but the weight must reflect the modified version of the mass matrix. Replacing Q_v with Γ_v,

$$\Gamma_v \frac{\partial w}{\partial t} + \frac{\partial f_v(w)}{\partial x} = \left(I + \frac{\Delta t}{2!} \alpha \frac{\partial}{\partial x} A_v \Gamma_v^{-1} \right) \left(\Gamma_v \frac{w^{n+1} - w^n}{\Delta t} + D\left(f_v\left(w^n \right) \right) \right)$$
$$- \Gamma_v w_{ttt}^n \frac{\Delta t^2}{3!} \pm \vartheta\left(\Delta t^3, \Delta x^k \right)$$
$$= \Gamma_v \frac{w^{n+1} - w^n}{\Delta t} + D\left(f_v\left(w^n \right) \right) \tag{5-49}$$
$$+ \frac{\Delta t}{2!} \alpha \frac{\partial}{\partial x} \left(A_v \Gamma_v^{-1} \left(\Gamma_v \frac{w^{n+1} - w^n}{\Delta t} + D\left(f_v\left(w^n \right) \right) \right) \right)$$
$$- \Gamma_v w_{ttt}^n \frac{\Delta t^2}{3!} \pm \vartheta\left(\Delta t^3, \Delta x^k \right)$$

This is the one-dimensional version of the partial differential equation to use in the Taylor weak statement Notice the modified mass matrix does not change the form of A_v. Also, the ever-present dispersion term reflects the change in the mass matrix so any discontinuity capturing term would be modified accordingly.

Two Dimensional Inviscid Fluid Flow with Primitive Variables

Extension to two or more dimensions is straightforward using the steps outlined above. The two dimensional PDE for inviscid fluid flow may be written

$$Q_v \frac{\partial w}{\partial t} + \frac{\partial f_v(w)}{\partial x} + \frac{\partial g_v(w)}{\partial y} = 0 \tag{5-50}$$

for general variables. Rearranging,

$$Q_v \frac{\partial w}{\partial t} = - \frac{\partial f_v(w)}{\partial x} - \frac{\partial g_v(w)}{\partial y}$$
$$\frac{\partial w}{\partial t} = - Q_v^{-1} \frac{\partial f_v(w)}{\partial x} - Q_v^{-1} \frac{\partial g_v(w)}{\partial y} \tag{5-51}$$

The time Taylor series expansion gives

$$Q_v w^{n+1} = Q_v w^n + Q_v \frac{\partial w^n}{\partial t} \Delta t + Q_v \frac{\partial^2 w^n}{\partial t^2} \frac{\Delta t^2}{2!} + Q_v \frac{\partial^3 w^n}{\partial t^3} \frac{\Delta t^3}{3!} + \vartheta\left(\Delta t^4 \right) \tag{5-52}$$

or

$$Q_v \frac{\partial w}{\partial t} = Q_v \frac{w^{n+1} - w^n}{\Delta t} - Q_v \frac{\partial^2 w^n}{\partial t^2} \frac{\Delta t}{2!} - Q_v \frac{\partial^3 w^n}{\partial t^3} \frac{\Delta t^3}{3!} - \vartheta(\Delta t^3) \tag{5-53}$$

Taking the time derivative of the original PDE in equation **(5-50)** gives

$$Q_v w_{tt} = -\frac{\partial}{\partial t}\left(\frac{\partial f_v}{\partial x} + \frac{\partial g_v}{\partial y}\right) \tag{5-54}$$

so

$$Q_v w_{tt} = -\left(\frac{\partial}{\partial x}\frac{\partial f_v}{\partial t} + \frac{\partial}{\partial y}\frac{\partial g_v}{\partial t}\right) \tag{5-55}$$

and

$$\begin{aligned} Q_v w_{tt} &= -\frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\frac{\partial w}{\partial t}\right) - \frac{\partial}{\partial y}\left(\frac{\partial g_v}{\partial w}\frac{\partial w}{\partial t}\right) \\ &= -\frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\left(-Q_v^{-1}\right)\left(\frac{\partial f_v}{\partial x} + \frac{\partial g_v}{\partial y}\right)\right) - \frac{\partial}{\partial y}\left(\frac{\partial g_v}{\partial w}\left(-Q_v^{-1}\right)\left(\frac{\partial f_v}{\partial x} + \frac{\partial g_v}{\partial y}\right)\right) \\ &= \frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\left(Q_v^{-1}\right)\left(\frac{\partial f_v}{\partial x} + \frac{\partial g_v}{\partial y}\right)\right) + \frac{\partial}{\partial y}\left(\frac{\partial g_v}{\partial w}\left(Q_v^{-1}\right)\left(\frac{\partial f_v}{\partial x} + \frac{\partial g_v}{\partial y}\right)\right) \end{aligned} \tag{5-56}$$

As before, form the convex summation of the second time derivative terms where

$\alpha + \beta = 0 \rightarrow -\alpha = \beta$ to get

$$\begin{aligned} Q_v w_{tt} &= -\alpha\left[\frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\frac{\partial w}{\partial t}\right) + \frac{\partial}{\partial y}\left(\frac{\partial g_v}{\partial w}\frac{\partial w}{\partial t}\right)\right] \\ &\quad + \beta\left[\frac{\partial}{\partial x}\left(\frac{\partial f_v}{\partial w}\left(Q_v^{-1}\right)\left(\frac{\partial f_v}{\partial x} + \frac{\partial g_v}{\partial y}\right)\right) + \frac{\partial}{\partial y}\left(\frac{\partial g_v}{\partial w}\left(Q_v^{-1}\right)\left(\frac{\partial f_v}{\partial x} + \frac{\partial g_v}{\partial y}\right)\right)\right] \\ &= -\alpha\frac{\partial}{\partial x}\left[\frac{\partial f_v}{\partial w}\left(\frac{\partial w}{\partial t} + Q_v^{-1}\frac{\partial f_v}{\partial x} + Q_v^{-1}\frac{\partial g_v}{\partial y}\right)\right] \\ &\quad -\alpha\frac{\partial}{\partial y}\left[\frac{\partial g_v}{\partial w}\left(\frac{\partial w}{\partial t} + Q_v^{-1}\frac{\partial f_v}{\partial x} + Q_v^{-1}\frac{\partial g_v}{\partial y}\right)\right] \\ &= -\alpha\frac{\partial}{\partial x}\left[A_v\left(\frac{\partial w}{\partial t} + Q_v^{-1}\frac{\partial f_v}{\partial x} + Q_v^{-1}\frac{\partial g_v}{\partial y}\right)\right] - \alpha\frac{\partial}{\partial y}\left[B_v\left(\frac{\partial w}{\partial t} + Q_v^{-1}\frac{\partial f_v}{\partial x} + Q_v^{-1}\frac{\partial g_v}{\partial y}\right)\right] \end{aligned} \tag{5-57}$$

Using similar approximations, the space dimensions may be written as

$$\frac{\partial f_v(w)}{\partial x} = D\left(f_v(w)\right) \pm \vartheta\left(\Delta x^k\right) \tag{5-58}$$

$$\frac{\partial g_v(w)}{\partial y} = D\left(g_v(w)\right) \pm \vartheta\left(\Delta y^k\right) \tag{5-59}$$

where $D(\cdot)$ describes a generic spatial discretization of the flux vectors.

Combining **(5-57)** with **(5-53)** and substituting that result, along with **(5-58)** and **(5-59)**, into the original flow equation **(5-50)** gives

$$\begin{aligned} Q_v \frac{\partial w}{\partial t} + \frac{\partial f_v(w)}{\partial x} + \frac{\partial g_v(w)}{\partial y} = Q_v \frac{w^{n+1} - w^n}{\Delta t} + D\left(f_v(w)\right) + D\left(g_v(w)\right) \\ + \alpha \frac{\Delta t}{2!} \frac{\partial}{\partial x}\left[A_v \left(\frac{\partial w}{\partial t} + Q_v^{-1} \frac{\partial f_v}{\partial x} + Q_v^{-1} \frac{\partial g_v}{\partial y} \right) \right] \\ + \alpha \frac{\Delta t}{2!} \frac{\partial}{\partial y}\left[B_v \left(\frac{\partial w}{\partial t} + Q_v^{-1} \frac{\partial f_v}{\partial x} + Q_v^{-1} \frac{\partial g_v}{\partial y} \right) \right] \\ - Q_v \frac{\partial^3 w^n}{\partial t^3} \frac{\Delta t^2}{3!} + \vartheta\left(\Delta t^3, \Delta x^k, \Delta y^k\right) \end{aligned} \tag{5-60}$$

or, moving the dispersion term into the roundoff error,

$$\begin{aligned} Q_v \frac{\partial w}{\partial t} + \frac{\partial f_v(w)}{\partial x} + \frac{\partial g_v(w)}{\partial y} = Q_v \frac{w^{n+1} - w^n}{\Delta t} + D\left(f_v(w)\right) + D\left(g_v(w)\right) \\ + \alpha \frac{\Delta t}{2!} \frac{\partial}{\partial x}\left[A_v Q_v^{-1} \left(Q_v \frac{\partial w}{\partial t} + \frac{\partial f_v}{\partial x} + \frac{\partial g_v}{\partial y} \right) \right] \\ + \alpha \frac{\Delta t}{2!} \frac{\partial}{\partial y}\left[B_v Q_v^{-1} \left(Q_v \frac{\partial w}{\partial t} + \frac{\partial f_v}{\partial x} + \frac{\partial g_v}{\partial y} \right) \right] \\ + \vartheta\left(\Delta t^2, \Delta x^k, \Delta y^k\right) \end{aligned} \tag{5-61}$$

The factored form is

$$Q_v \frac{\partial w}{\partial t} + \frac{\partial f_v(w)}{\partial x} + \frac{\partial g_v(w)}{\partial y}$$
$$= \left[I + \alpha \frac{\Delta t}{2!} \left(\frac{\partial}{\partial x} A_v Q_v^{-1} + \frac{\partial}{\partial y} B_v Q_v^{-1} \right) \right] \left(Q_v \frac{w^{n+1} - w^n}{\Delta t} + D(f_v) + D(g_v) \right) \quad \textbf{(5-62)}$$
$$+ \vartheta\left(\Delta t^2, \Delta x^k, \Delta y^k\right)$$

where $\frac{\partial}{\partial x} A_v Q_v^{-1}$ and $\frac{\partial}{\partial y} B_v Q_v^{-1}$ are operators. Notice $A_v Q_v^{-1}$ and $B_v Q_v^{-1}$ are used and not $\left[A_v Q_v^{-1}\right]^T$ and $\left[B_v Q_v^{-1}\right]^T$. This is somewhat confusing in the literature but the transposed matrices do not provide the proper dimensional units unless they are symmetric and then the transpose is irrelevant.

Wave Analysis of Continuum Equation for Inviscid Fluid Flow in Primitive Variables

After the change of variables described above, the partial differential equation for inviscid fluid flow may be written

$$Q_v \frac{\partial w}{\partial t} + \frac{\partial f_v(w)}{\partial x} + \frac{\partial g_v(w)}{\partial y} = 0 \quad \textbf{(5-63)}$$

Let $w = \hat{w} e^{i(k_x x + k_y y - \omega t)}$ where k_x and k_y are the components of the wave number vector in the x and y directions and $\omega = \omega_r + \omega_c$ is the (possibly complex) angular velocity. Linearizing the PDE and substituting for w gives

$$-i\omega Q_v \hat{w} e^{i(k_x x + k_y y - \omega t)} + i k_x A_v \hat{w} e^{i(k_x x + k_y y - \omega t)} + i k_y B_v \hat{w} e^{i(k_x x + k_y y - \omega t)} = 0 \quad \textbf{(5-64)}$$

or

$$\omega = k_x Q_v^{-1} A_v + k_y Q_v^{-1} B_v \quad \textbf{(5-65)}$$

so

$$\frac{\omega}{k} = \frac{k_x}{k} Q_v^{-1} A_v + \frac{k_y}{k} Q_v^{-1} B_v \text{ where } k = \left(k_x^2 + k_y^2\right)^{1/2} \quad \textbf{(5-66)}$$

The wave number vector, $\vec{k} = \{k_x, k_y\}^T$, gives the direction of propagation of waves in the domain and the ratio $\frac{\omega}{k}$ describes the phase velocity [Hirsch I, p149]. The phase velocity is real because time and all space derivatives are the same order.

Wave Analysis of Discrete Equation for Inviscid Fluid Flow in Primitive Variables

The discretized partial differential equation yields to a similar wave analysis. The specific approach depends on the objective. In this case, the objective is to determine the form of the free parameter α introduced in the derivation of the modified form of the fluid flow equations. To simplify matters, let

$$R_h^n = Q_v \frac{w^{n+1} - w^n}{\Delta t} + D\left(f_v(w)\right) + D\left(g_v(w)\right) \tag{5-67}$$

be the residual of the basic discrete form of the partial differential equation. The full discrete form is given by

$$\begin{aligned} & Q_v \frac{w^{n+1} - w^n}{\Delta t} + D\left(f_v(w)\right) + D\left(g_v(w)\right) \\ & + \alpha \frac{\Delta t}{2!} \frac{\partial}{\partial x} \left[A_v Q_v^{-1} \left(Q_v \frac{w^{n+1} - w^n}{\Delta t} + D\left(f_v(w)\right) + D\left(g_v(w)\right) \right) \right] \\ & + \alpha \frac{\Delta t}{2!} \frac{\partial}{\partial y} \left[B_v Q_v^{-1} \left(Q_v \frac{w^{n+1} - w^n}{\Delta t} + D\left(f_v(w)\right) + D\left(g_v(w)\right) \right) \right] = 0 \end{aligned} \tag{5-68}$$

Substituting for R_h^n leads to

$$R_h^n + \alpha \frac{\Delta t}{2!} \frac{\partial}{\partial x} \left[A_v Q_v^{-1} R_h^n \right] + \alpha \frac{\Delta t}{2!} \frac{\partial}{\partial y} \left[B_v Q_v^{-1} R_h^n \right] = 0 \tag{5-69}$$

This is a partial differential equation in space with the residual of the basic discrete form as the independent variable vector. Using a semi-discrete version of the wave analysis technique let $R_h^n = \hat{R} e^{i(k_x x + k_y y - \omega t)}$. Substituting and simplifying leaves

$$\hat{R}e^{i(k_x x+k_y y-\omega t)}+\alpha\frac{\Delta t}{2!}ik_x\left[A_v Q_v^{-1}\hat{R}\right]e^{i(k_x x+k_y y-\omega t)}+\alpha\frac{\Delta t}{2!}ik_y\left[B_v Q_v^{-1}\hat{R}\right]e^{i(k_x x+k_y y-\omega t)}=0 \quad \textbf{(5-70)}$$

Removing the common terms simplifies equation **(5-70)** to

$$I+\alpha\frac{\Delta t}{2!}ik_x A_v Q_v^{-1}+\alpha\frac{\Delta t}{2!}ik_y B_v Q_v^{-1}=0 \quad \textbf{(5-71)}$$

Derivation of the SUPG τ Parameter

Solving for the free parameter, α, yields

$$\begin{aligned}\alpha&=\frac{2i}{\Delta t}\left[k_x A_v Q_v^{-1}+k_y B_v Q_v^{-1}\right]^{-1}I\\&=\frac{2i}{\Delta t k}\left[n_x A_v Q_v^{-1}+n_y B_v Q_v^{-1}\right]^{-1}I\end{aligned} \quad \textbf{(5-72)}$$

where $n_x=\frac{k_x}{k}$ and $n_y=\frac{k_y}{k}$ (remember $\frac{1}{i}=-i$). The eigenvalues of the matrix $\left[n_x A_v Q_v^{-1}+n_y B_v Q_v^{-1}\right]$, associated with the arbitrary direction of propagation of a unit wave number vector $\hat{k}=\left\{n_x,n_y\right\}$, largely determine the behavior of the solutions of the Euler equations [Hirsch II, p150]. The eigenvalues are given by the set $\left\{\vec{u}\cdot\vec{k},\vec{u}\cdot\vec{k}+ck,\vec{u}\cdot\vec{k}-ck\right\}$ [Hirsch II, p177] for multi-dimensional flow where the full wave number vector, $\vec{k}$, represents the number of waves over a distance 2π in the direction of the vector and has units of 1/length. Thus, if Q_v is the Jacobian for the variable transformation, then

$$\alpha=\frac{2i}{\Delta t\left(\vec{u}\cdot\vec{k}+ck\right)} \quad \textbf{(5-73)}$$

where $\vec{u}$ is the particle velocity vector and c is the acoustic speed. Since all the terms involving α have the form $\alpha\frac{\Delta t}{2!}$, let

$$\tau = \alpha \frac{\Delta t}{2!} = \frac{2i}{\Delta t\left(\vec{u}\cdot\vec{k} + ck\right)} \frac{\Delta t}{2!}$$

$$= \frac{1}{\left(\vec{u}\cdot\vec{k} + ck\right)} i \qquad \text{(5-74)}$$

$$= \frac{1}{k\left(\vec{u}\cdot\hat{k} + c\right)} i$$

The τ parameter has units of time and represents an intrinsic time scale for the gradient of the residual terms. Again, a great deal of effort in the SUPG literature has gone into developing a proper understanding of the term and its affect on the accuracy and convergence rates of fluid flow problems.

This general form of the τ parameter illustrates the role the term plays in the overall equation and can take on different specific forms depending on the particular choice of $\vec{k}$. As an example, it is very similar to the space-time/SUPG parameter derived by [Aliabadi & Tezduyar, 1993] and reinforced by [Ray, et al., 1997].

[Aliabadi & Tezduyar, 1993] use the normalized gradient vector of the 2-norm of the conservation variables as the direction for the wave number vector and use the infinity-norm of the product of the element dimension and the wave vector component as the measure of length. A parameter a is used to adjust for the time integration algorithm.

$$\tau = \frac{\max\left(ah_i\beta_i\right)}{\left(\left|\vec{u}\cdot\hat{\beta}\right| + c\right)} i \text{ (no sum on repeated index) where } \hat{\beta} = \frac{\nabla\|q\|_2^2}{\left\|\nabla\|q\|_2^2\right\|_2} \qquad \text{(5-75)}$$

The equivalent form using the wave number vector is $\tau = \dfrac{1}{k\left(\vec{u}\cdot\hat{k} + c\right)}$ where k has magnitude given by

$$k = \frac{1}{\max\left(ah_i\beta_i\right)} = \frac{1}{\left\|ah_i\beta_i\right\|_\infty} \text{ and direction } \hat{k} = \frac{\nabla\|q\|_2^2}{\left\|\nabla\|q\|_2^2\right\|_2}. \qquad \text{(5-76)}$$

The derivation using the TWS formulations indicates the τ parameter should be independent of the time discretization method so the parameter a is theoretically unnecessary. Practical experience may dictate otherwise. The gradient of the norm of the conservation variables seems to provide a legitimate

direction for defining the wave number vector direction. The wave number vector magnitude (reference length: **(5-76)**) selects the reference length as the direction with the largest gradient changes over the given element dimensions. This leads to the largest intrinsic time scale along the Cartesian directions **(5-75)**.

[Aliabadi & Tezduyar, 1993]'s method was tested in this research effort for reference and provides convergence rates similar to the best method below. It also suffers from the same problems to be described in the analysis of the method. The method is described here for reference only to show the utility of the general form derived above for obtaining an understanding of choices for the τ parameter.

Several formulations for τ were developed and tested in the present effort. Most were developed using ad hoc eigenvalue or CFL justifications before applying the wave number vector derivation above. The new theoretical basis immediately led to an improved selection for the wave number vector, $\vec{k}$, and improved computational results.

The various forms derived and tested for the τ parameter fall into four basic categories with some variations within the categories. The categories were not known or defined at the time and the development was ad hoc. The categories come from after-the-fact theoretical analyses and are distinguished by the fundamental directions for the wave number vector, $\vec{k}$.

In the first category, the direction of the wave vector was selected parallel to the velocity vector, $\vec{u}$, but various magnitudes were tested. These schemes were fairly successful and were used for most of the effort prior to the development of the theoretical basis.

Schemes **(5-77)** through **(5-81)** are variations of the same idea where Δx and Δy represent directional length dimensions for each individual element in the grid. Scheme **(5-81)** provides an additional variation on the theme where the speed of a fluid signal in the element is based on an artificial acoustic velocity modified to match the particle velocity. Equations (5-82) and (5-83) are similar in concept but the dimensions L_x and L_y represent global dimensions rather than element dimensions.

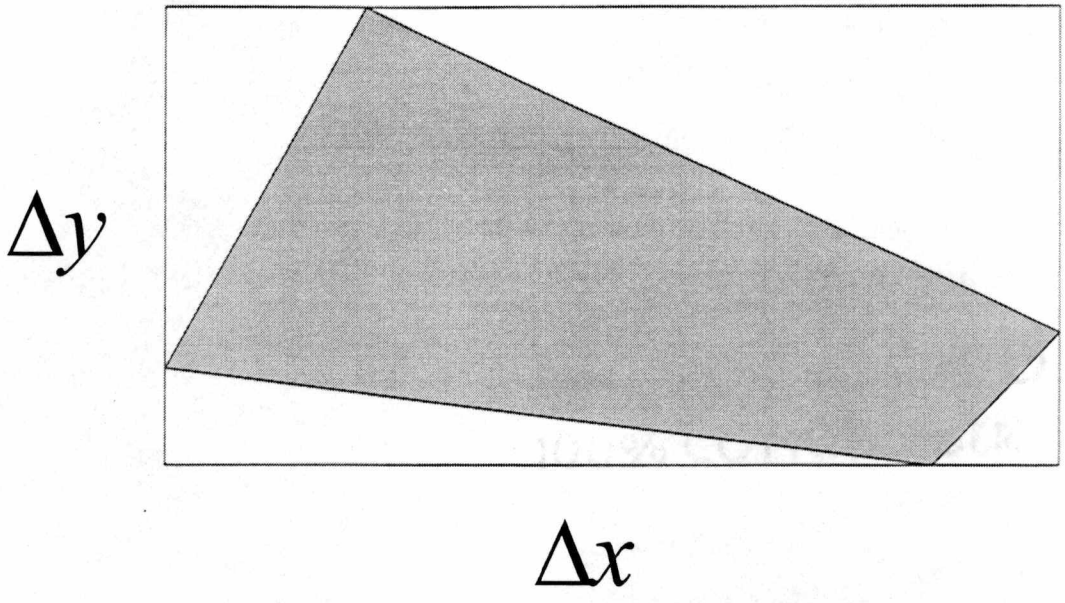


Figure 5-1: Minimally Bounding Box

In schemes **(5-77)** through **(5-81)**, Δx is calculated as $\max\left(\left|XN_{\substack{element\\nodes}}\right|\right) - \min\left(\left|XN_{\substack{element\\nodes}}\right|\right)$ where XN is a vector of the x location of the element nodes. Δy is calculated the same way. This essentially describes a Cartesian box completely but minimally bounding the entire element (Figure 5-1.)

$$\tau = \frac{2\left(\Delta x^2 + \Delta y^2\right)^{1/2}}{\left(\|\vec{u}\| + c\right)} \qquad \hat{k} = \frac{\vec{u}}{\|\vec{u}\|} \qquad \|\vec{k}\| = \frac{1}{2\left(\Delta x^2 + \Delta y^2\right)^{1/2}} \tag{5-77}$$

$$\tau = \frac{\left(\Delta x^2 + \Delta y^2\right)^{1/2}}{\left(\|\vec{u}\| + c\right)} \qquad \hat{k} = \frac{\vec{u}}{\|\vec{u}\|} \qquad \|\vec{k}\| = \frac{1}{\left(\Delta x^2 + \Delta y^2\right)^{1/2}} \tag{5-78}$$

$$\tau = \frac{\left(\Delta x^2 + \Delta y^2\right)^{1/2}}{2\left(\|\vec{u}\| + c\right)} \qquad \hat{k} = \frac{\vec{u}}{\|\vec{u}\|} \qquad \|\vec{k}\| = \frac{2}{\left(\Delta x^2 + \Delta y^2\right)^{1/2}} \tag{5-79}$$

$$\tau = \frac{\left(\Delta x^2 + \Delta y^2\right)^{1/2}}{\pi\left(\|\vec{u}\| + c\right)} \qquad \hat{k} = \frac{\vec{u}}{\|\vec{u}\|} \qquad \|\vec{k}\| = \frac{\pi}{\left(\Delta x^2 + \Delta y^2\right)^{1/2}} \tag{5-80}$$

$$\tau = \frac{\left(\Delta x^2 + \Delta y^2\right)^{1/2}}{2\left(\|\vec{u}\| + \|\vec{u}\|\right)} \qquad \hat{k} = \frac{\vec{u}}{\|\vec{u}\|} \qquad \|\vec{k}\| = \frac{2}{\left(\Delta x^2 + \Delta y^2\right)^{1/2}} \tag{5-81}$$

L_x is calculated as $\max\left(\left|X_{\substack{global\\nodes}}\right|\right) - \min\left(\left|X_{\substack{global\\nodes}}\right|\right)$ where X is a vector of the x location of all the element nodes in the fluid domain. L_y is calculated the same way. This essentially describes a Cartesian box completely but minimally bounding the entire fluid grid.

$$\tau = \frac{\left(L_x^2 + L_y^2\right)^{1/2}}{2\pi\left(\|\vec{u}\| + c\right)} \qquad \hat{k} = \frac{\vec{u}}{\|\vec{u}\|} \qquad \|\vec{k}\| = \frac{2\pi}{\left(L_x^2 + L_y^2\right)^{1/2}} \tag{5-82}$$

$$\tau = \frac{\left(L_x^2 + L_y^2\right)^{1/2}}{\pi\left(\|\vec{u}\| + c\right)} \qquad \hat{k} = \frac{\vec{u}}{\|\vec{u}\|} \qquad \|\vec{k}\| = \frac{\pi}{\left(L_x^2 + L_y^2\right)^{1/2}} \tag{5-83}$$

The second category has the wave vector perpendicular to the velocity vector $\left(\hat{k} \cdot \vec{u} = 0\right)$. These were generally unsuccessful. Δx, Δy, L_x, and L_y are the same as above.

$$\tau = \frac{\left(\Delta x^2 + \Delta y^2\right)^{1/2}}{c} \qquad \hat{k} \text{ perpendicular to } \frac{\vec{u}}{\|\vec{u}\|} \qquad \|\vec{k}\| = \frac{1}{\left(\Delta x^2 + \Delta y^2\right)^{1/2}} \tag{5-84}$$

$$\tau = \frac{\left(L_x^2 + L_y^2\right)^{1/2}}{\pi c} \qquad \hat{k} \text{ perpendicular to } \frac{\vec{u}}{\|\vec{u}\|} \qquad \|\vec{k}\| = \frac{\pi}{\left(L_x^2 + L_y^2\right)^{1/2}} \tag{5-85}$$

$$\tau = \frac{\left(L_x^2 + L_y^2\right)^{1/2}}{\pi^2 c} \qquad \hat{k} \text{ perpendicular to } \frac{\vec{u}}{\|\vec{u}\|} \qquad \|\vec{k}\| = \frac{\pi^2}{\left(L_x^2 + L_y^2\right)^{1/2}} \tag{5-86}$$

The third category reflects the wave number vectors developed after the theoretical analysis. These target the well-known $2\Delta x$ waves and the wave number vector directions reflect the shape of the individual grid elements.

The schemes in equations **(5-87)** and **(5-88)** use the full wave number vector with one wave over a length 2Δ in each of the Cartesian directions. Scheme **(5-87)** uses the physical acoustic velocity to set the time scale for signals in the element while scheme **(5-88)** uses a modified acoustic velocity set equal to the magnitude of the velocity vector. Schemes **(5-89)** and **(5-92)** produce minimum time scales based on the direction with the shortest time of flight for an acoustic disturbance. Scheme **(5-94)** produces a maximum time of flight for an acoustic disturbance moving downstream.

Schemes **(5-87)** and **(5-88)** essentially reflect (Figure 5-2) twice the time of flight for an acoustic signal to transit the diagonal of a Cartesian box completely but minimally bounding the entire element. As a minimum, three grid nodes are required to support a full wave. This is equivalent to half a wave between two grid nodes or twice the time of flight required on a single element to support a full wave.

$$\tau = \frac{1}{\left|\frac{u}{2\Delta x}\right| + \left|\frac{v}{2\Delta y}\right| + c\frac{\left(\Delta x^2+\Delta y^2\right)^{1/2}}{2\Delta x\Delta y}} \qquad \vec{k} = \begin{Bmatrix} \frac{1}{2\Delta x} \\ \frac{1}{2\Delta y} \end{Bmatrix} \qquad \left\|\vec{k}\right\| = \frac{\left(\Delta x^2+\Delta y^2\right)^{1/2}}{2\Delta x\Delta y} \tag{5-87}$$

$$\tau = \frac{1}{\left|\frac{u}{2\Delta x}\right| + \left|\frac{v}{2\Delta y}\right| + \left\|\vec{u}\right\|\frac{\left(\Delta x^2+\Delta y^2\right)^{1/2}}{2\Delta x\Delta y}} \qquad \vec{k} = \begin{Bmatrix} \frac{1}{2\Delta x} \\ \frac{1}{2\Delta y} \end{Bmatrix} \qquad \left\|\vec{k}\right\| = \frac{\left(\Delta x^2+\Delta y^2\right)^{1/2}}{2\Delta x\Delta y} \tag{5-88}$$

Notice (Figure 5-3) that this is different than traveling a distance equal to twice the length of the diagonal but in the direction of the velocity vector as shown in category one. In the second case (Figure 5-3) the full velocity magnitude is used. The first case (Figure 5-2) only uses the component of the velocity in the direction of the diagonal.

$$\tau = \min\left(\frac{2\Delta x}{|u|+c}, \frac{2\Delta y}{|v|+c}\right) \qquad \vec{k} = \begin{Bmatrix} \frac{1}{2\Delta x} \\ 0 \end{Bmatrix} \text{ or } \vec{k} = \begin{Bmatrix} 0 \\ \frac{1}{2\Delta y} \end{Bmatrix} \tag{5-89}$$

$$\tau = \min\left(\frac{\Delta x}{|u|+c}, \frac{\Delta y}{|v|+c}\right) \qquad \vec{k} = \begin{Bmatrix} \frac{1}{2\Delta x} \\ 0 \end{Bmatrix} \text{ or } \vec{k} = \begin{Bmatrix} 0 \\ \frac{1}{2\Delta y} \end{Bmatrix} \tag{5-90}$$

$$\tau = \max\left(\frac{2\Delta x}{|u|+c}, \frac{2\Delta y}{|v|+c}\right) \qquad \vec{k} = \begin{Bmatrix} \frac{1}{2\Delta x} \\ 0 \end{Bmatrix} \text{ or } \vec{k} = \begin{Bmatrix} 0 \\ \frac{1}{2\Delta y} \end{Bmatrix} \tag{5-91}$$

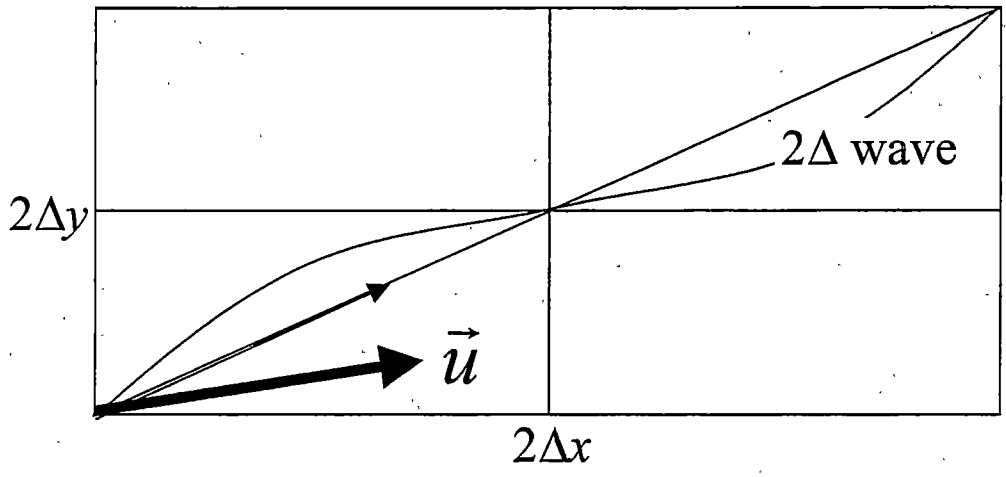


Figure 5-2: Wave Number Vector in Direction of Diagonal

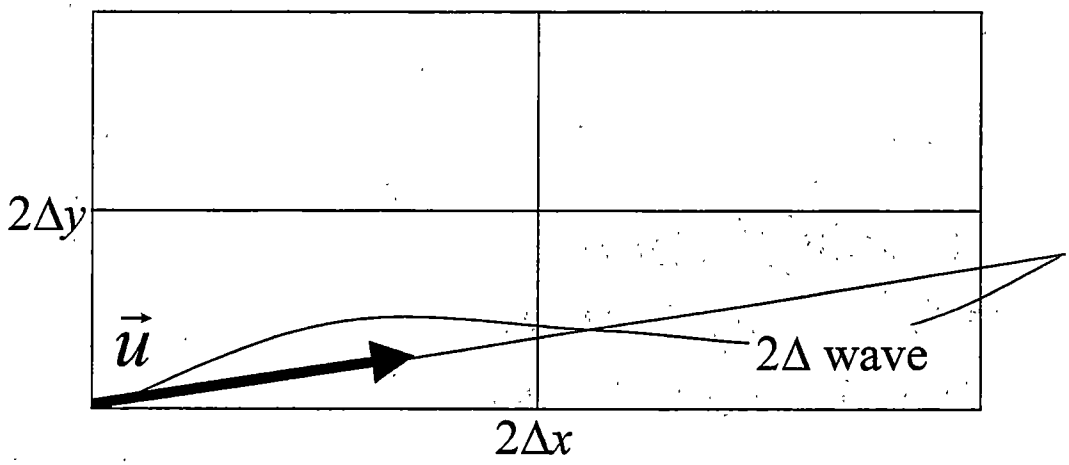


Figure 5-3: Wave Number Vector in Direction of Velocity Vector

$$\tau = \min\left(\frac{2\Delta x}{|u|+c}, \frac{2\Delta y}{|v|+c}\right) \qquad \vec{k} = \left\{\begin{matrix}\frac{1}{2\Delta x}\\ 0\end{matrix}\right\} \text{ or } \vec{k} = \left\{\begin{matrix}0\\ \frac{1}{2\Delta y}\end{matrix}\right\} \tag{5-92}$$

$$\tau = \min\left(\frac{\Delta x}{|u|+c}, \frac{\Delta y}{|v|+c}\right) \qquad \vec{k} = \left\{\begin{matrix}\frac{1}{2\Delta x}\\ 0\end{matrix}\right\} \text{ or } \vec{k} = \left\{\begin{matrix}0\\ \frac{1}{2\Delta y}\end{matrix}\right\} \tag{5-93}$$

$$\tau = \max\left(\frac{2\Delta x}{|u|+c}, \frac{2\Delta y}{|v|+c}\right) \qquad \vec{k} = \left\{\begin{matrix}\frac{1}{2\Delta x}\\ 0\end{matrix}\right\} \text{ or } \vec{k} = \left\{\begin{matrix}0\\ \frac{1}{2\Delta y}\end{matrix}\right\} \tag{5-94}$$

The fourth category essentially covers all other cases. The primary method tested did not relate to a wave number vector at all. It was a polynomial in $\|\vec{u}\|$ and c developed to algebraically minimize the ratio of different eigenvalues in a limit. The free parameter p provided the best results for this category at $p = 5$ and the τ has the form

$$\tau = \frac{\left(L_x^2 + L_y^2\right)^{1/2}\left(\|\vec{u}\|^2 + \frac{c^2}{p}\right)}{\pi p\left(c\|\vec{u}\|^2 + \|\vec{u}\|c^2\right)} \tag{5-95}$$

Summary

Chapter 4 provided the background and detailed approach for using time derivative preconditioning to control the eigenvalues, and thus any ill conditioning, of the inviscid fluid flow equations in the continuum. The original fluid flow equation set was linearized using a classic Newton method to create a new equation set with the nonlinear fluid equations on the right hand side and a linearized, incremental form of the equations on the left hand side. The time derivative preconditioning term was added in pseudo time to the left hand side and shown to be a generalization of the Newton method. The modified Newton statement was then discretized using a Petrov-Galerkin finite element approach with the weighting function derived in Chapter 5.

In Chapter 5, I used the Taylor Weak Statement (TWS) approach to derive appropriate diffusion terms to create a stable and consistent finite element formulation for a generic discretization with a generic time preconditioning term. The Galerkin approach of TWS was shown to be equivalent to upwind methods

typically referred to as Streamwise Upwind Petrov Galerkin (SUPG). I derived the appropriate form for the SUPG weight function using primitive variables and time preconditioning. Also, a theoretical basis was developed for the intrinsic time scale parameter common to all SUPG weight functions.

CHAPTER 6

FLUID FLOW ON A MOVING OR DEFORMING GRID

Grid Motion Strategies

Many problems in fluid mechanics and fluid-structure interaction dictate the use of computational domains that move relative to some inertial system and/or deform from their original shapes. There are three primary alternatives for addressing the changing grid locations and shapes. The Chimera method [Benek, et al., 1986] uses multiple overset grids to describe complex shapes. The individual grids are typically fixed in shape but may be put in motion where one grid moves through another according to some prescribed or calculated path [Alonso, Martinelli, and Jameson, 1995]. Practical implementations set boundary conditions in one grid by interpolating from nodes in another grid or by locally adaptive grid nodes that "snap-to" a background grid or some other grid node [Janus & Whitfield, 1990]. Grids may be butted against each other without overlap for constrained motion problems such as propellers [Sheng, et al., 1997] and rotor flowfields in turbomachinery. The second primary strategy focuses on stretching initial grids [Batina, 1989] [Tezduyar, Behr & Liou, 1992] to a limit and "regridding" [Lohner, 1992]. The third strategy centers on continuously adaptive gridding where grid nodes and cells are added or removed as local gradients or some forcing function dictate [Soni & Yang, 1992] [Thompson, 1985].

All three strategies have been used with great success and cover a wide range of applications. However, in spite of their fundamental differences in approach, they all require some technique for satisfying fluid flow conservation laws on a non-inertial grid, some technique for setting local flow conditions from information at surrounding locations, and some technique for establishing the new locations of individual grid nodes. In the current effort, an Arbitrary Lagrange Euler method is used to satisfy the fluid conservation laws. Only one fluid grid is used. The fluid grid deforms as if it is a linear elastic solid continuum based on displacement boundary conditions rather than as a discrete set of points or a network of springs. No points are added or removed and the grid does not adapt to flowfield conditions or algebraic weight functions. Thus, no specific interpolation method is needed between "before and after" grid sets.

Derivation of a Weighted Arbitrary Lagrange Euler Statement

The "Arbitrary Lagrange Euler" (ALE) method began in the mid-1970s [Hirt, et al., 1974] [Pracht, 1975] and has become the standard approach for fluid flow problems on a moving or deforming grid. The method was soon extended to fluid-structure interaction problems [Donea, et al., 1982] [Donea, 1983]. In some cases, it has also been used to describe the structural response of solids as in [Bendiksen, 1991] and the Lawrence Livermore National Laboratories ALE3D [no reference available] code. It has been derived for particles using kinematics and for arbitrary control volumes using a combination of Reynold's transport theorem [Aris, pp. 84-85] and Leibniz's rule [Taylor & Mann, pp. 552.] In short, the method is well established and very successful.

The standard approach for a finite volume computational method is to use the control volume form of the ALE equations. For finite element methods, published derivations start with the differential form of the ALE equations and move into a weighted residual integration. The current effort starts with the time derivative of a weighted integral of the fluid flow variables where the weight function is not necessarily constant in time or space. The end result is the same but the current approach places some restrictions on the weight functions as described below.

In Leibniz theorem, $I(t) = \int_{V(t)} f(\vec{x},t)dV$ where $f(\vec{x},t)$ is any vector, scalar, or tensor function. It can be shown

$$\frac{d}{dt}I(t) = \frac{d}{dt}\iiint_{V(t)} f(\vec{x},t)dV = \iiint_{V(t)} \frac{\partial}{\partial t}[f(\vec{x},t)]dV + \iint_{\partial V(t)} f(\vec{x},t)\vec{w}\cdot\hat{n}dS \qquad \textbf{(6-1)}$$

where $\hat{n}$ is the outward normal to the surface dS and $\vec{w}$ is the absolute velocity vector of dS. The integral is taken over any arbitrary closed region and does not have to be an infinitesimal element.

Let the function $f(\vec{x},t)$ be the vector of conservation variables, $Q = \{\rho, \rho\vec{u}, \rho E\}^T$. If the integral is taken over a closed material region moving exactly with the fluid so the volume always contains the same material, then $\vec{w} = \vec{u}$ where $\vec{u}$ is the velocity of the fluid and the above becomes

$$\frac{d}{dt}I(t) = \frac{d}{dt}\iiint_{V(t)} QdV = \iiint_{V(t)} \frac{\partial}{\partial t}[Q]dV + \iint_{\partial V(t)} Q\vec{u}\cdot\hat{n}dS \qquad \textbf{(6-2)}$$

Using the Green-Gauss theorem on the surface integral and combining with the volume integral,

$$\frac{d}{dt}I(t)=\frac{d}{dt}\iiint_{V(t)}QdV=\iiint_{V(t)}\left[\frac{\partial}{\partial t}Q+\nabla\cdot Q\vec{u}\right]dV \tag{6-3}$$

The term in brackets in the last integral reproduces the Eulerian form of the fluid flow equations without the force terms (pressure, viscosity, heat flux, etc.). If $\vec{w}\neq\vec{u}$,

$$\frac{d}{dt}I(t)=\frac{d}{dt}\iiint_{V(t)}QdV=\iiint_{V(t)}\left[\frac{\partial}{\partial t}Q+\nabla\cdot Q\vec{w}\right]dV \tag{6-4}$$

where $\vec{w}$ is the arbitrary absolute velocity vector of the surface of the volume. Substituting the Euler equations

$$\frac{\partial}{\partial t}Q+\nabla\cdot Q\vec{u}+\nabla p=0 \tag{6-5}$$

for the first term in the brackets leads to

$$\frac{d}{dt}I(t)=\frac{d}{dt}\iiint_{V(t)}QdV=\iiint_{V(t)}\left[-\left(\nabla\cdot Q\vec{u}+\nabla p\right)+\nabla\cdot Q\vec{w}\right]dV \tag{6-6}$$

Rearranging the terms,

$$\frac{d}{dt}\iiint_{V(t)}QdV+\iiint_{V(t)}\left[\nabla\cdot Q(\vec{u}-\vec{w})+\nabla p\right]dV=0 \tag{6-7}$$

or

$$\frac{d}{dt}\iiint_{V(t)}QdV+\iint_{\partial V(t)}\left[Q(\vec{u}-\vec{w})+\nabla p\right]\cdot\hat{n}dS=0 \tag{6-8}$$

This is an integral statement of the Arbitrary Lagrange Eulerian set of equations for fluid flow.

Now consider a weighted integration over an arbitrarily moving region. The change to indicial notation (hopefully) removes some of the ambiguity from the operators.

$$\frac{d}{dt}\iiint_{V(t)}W_iQ_jdV=\iiint_{V(t)}\left[\left(\frac{\partial}{\partial t}W_{ij}\right)Q_j+W_{ij}\left(\frac{\partial}{\partial t}Q_j\right)\right]dV+\iint_{\partial V(t)}W_{ij}Q_jw_kn_kdS \tag{6-9}$$

The last integral is transformed using the Green-Gauss theorem to give

$$\frac{d}{dt}\iiint_{V(t)} W_{ij}Q_j dV = \iiint_{V(t)}\left[\left(\frac{\partial}{\partial t}W_{ij}\right)Q_j + W_{ij}\left(\frac{\partial}{\partial t}Q_j\right) + \frac{\partial}{\partial x_k}\left(W_{ij}Q_j w_k\right)\right]dV$$
$$= \iiint_{V(t)}\left[\left(\frac{\partial}{\partial t}W_{ij}\right)Q_j + W_{ij}\left(\frac{\partial}{\partial t}Q_j\right) + \left(\frac{\partial}{\partial x_k}W_{ij}\right)\left(Q_j w_k\right) + W_{ij}\left(\frac{\partial}{\partial x_k}Q_j w_k\right)\right]dV \quad \textbf{(6-10)}$$

Rearranging and grouping terms,

$$\frac{d}{dt}\iiint_{V(t)} W_{ij}Q_j dV = \iiint_{V(t)}\left[\left(\frac{\partial}{\partial t}W_{ij}\right) + w_k\left(\frac{\partial}{\partial x_k}W_{ij}\right)\right]Q_j dV$$
$$+\iiint_{V(t)} W_{ij}\left[\left(\frac{\partial}{\partial t}Q_j\right) + \left(\frac{\partial}{\partial x_k}Q_j w_k\right)\right]dV \quad \textbf{(6-11)}$$

As before, substitute the Euler equations for the time derivative in the second RHS integral to get

$$\frac{d}{dt}\iiint_{V(t)} W_{ij}Q_j dV = \iiint_{V(t)}\left[\left(\frac{\partial}{\partial t}W_{ij}\right) + w_k\left(\frac{\partial}{\partial x_k}W_{ij}\right)\right]Q_j dV$$
$$+\iiint_{V(t)} W_{ij}\left[-\left(\frac{\partial}{\partial x_k}Q_j u_k + \frac{\partial}{\partial x_k}\delta_{km}p_m\right) + \left(\frac{\partial}{\partial x_k}Q_j w_k\right)\right]dV \quad \textbf{(6-12)}$$

Rearranging gives

$$\frac{d}{dt}\iiint_{V(t)} W_{ij}Q_j dV + \iiint_{V(t)} W_{ij}\left[\frac{\partial}{\partial x_k}Q_j\left(u_k - w_k\right) + \frac{\partial}{\partial x_k}\delta_{km}p_m\right]dV$$
$$= \iiint_{V(t)}\left[\left(\frac{\partial}{\partial t}W_{ij}\right) + w_k\left(\frac{\partial}{\partial x_k}W_{ij}\right)\right]Q_j dV \quad \textbf{(6-13)}$$

The terms on the left hand side represent the weighted version of the integral ALE equation set derived above. However, the right hand side terms show the weight function will act as a source for Q_j if the term in brackets is not set equal to zero. The term in brackets is simply the passive convective transport equation for the weight function and indicates that a non-constant weight function must change as the location of the nodes in the finite element change. Thus, the weight function must satisfy

$$\frac{\partial}{\partial t}W_{ij} + \vec{w}\cdot\nabla W_{ij} = 0 \quad \textbf{(6-14)}$$

or the source term in Q_j must be included.

Geometric Conservation Laws

[Thomas and Lombard, 1978] coined the phrase "geometric conservation laws" in describing the Leibniz integral operation

$$\frac{d}{dt}\iiint_{V(t)} dV = \iint_{\partial V} \vec{w}\cdot\hat{n}dS \tag{6-15}$$

They essentially pointed out the need to treat the moving and distorting volumes associated with grid motion in a conservative manner. The process is straightforward in finite volume and finite element methods but may be elusive with finite difference methods. In fact, even geometric variable transformations to body fitting, cylindrical, or spherical coordinates may violate geometric conservation if the metrics and Jacobians are not treated properly. [Tamura and Fujii, 1993] quantify the possible errors. The geometric conservation laws are satisfied identically in the integral form of the ALE equations and the space-time methods of [Tezduyar, Behr and Liou., 1992].

CHAPTER 7

STRUCTURAL EQUATIONS

There are many techniques and levels of approximation employed for modeling structures and structural dynamics. All start with a set of differential or integral equations and are either solved in closed form based on a set of limiting assumptions and boundary conditions or discretized in some fashion and solved numerically. The fundamental level is a set of continuum differential equations for solid mechanics based on linear or non-linear elasticity known as the Navier equations. The giants of applied calculus such as Bernoulli and Euler used limiting assumptions to create special forms of the equations applicable to simple shapes. Bernoulli-Euler beams provide great insight analytically and may also be used as a starting point for discrete models. Many other analytic models have been developed for special classes of structural elements such as thick and thin plates, shells, rods, etc. These take advantage of the unique natures of simple structural elements and their tendency to retain those natures under normal loading conditions. Combinations of these specialized models are probably the most widely used basis for discrete models in modern structural analysis [MacMurdy, Guruswamy and Kapania, 1994]. The behavior of each specific shape could also be solved using the Navier equations but the specialized models take advantage of additional knowledge and typically provide tremendous computational advantages.

Discrete models also come in many shapes, sizes and functionality depending on the need (Figure 7-1). They may be simple approximations [Yang and Lohner, 1997] with variable cross section and lumped mass or fully three-dimensional models [Ray, Wren and Tezduyar, 1997] with continuously variable material properties and distributed mass. Combination models are common. For low frequency structural dynamics, complicated models are often used to solve for the vibratory mode shapes and frequencies. The lowest (say 10) mode shapes are then combined in a linear summation to provide a simplified model of the structural response [Guruswamy and Tu, 1994] [Yang and Lohner, 1997]. This technique is especially common in fluid-structure interaction problems where typically only the first one or

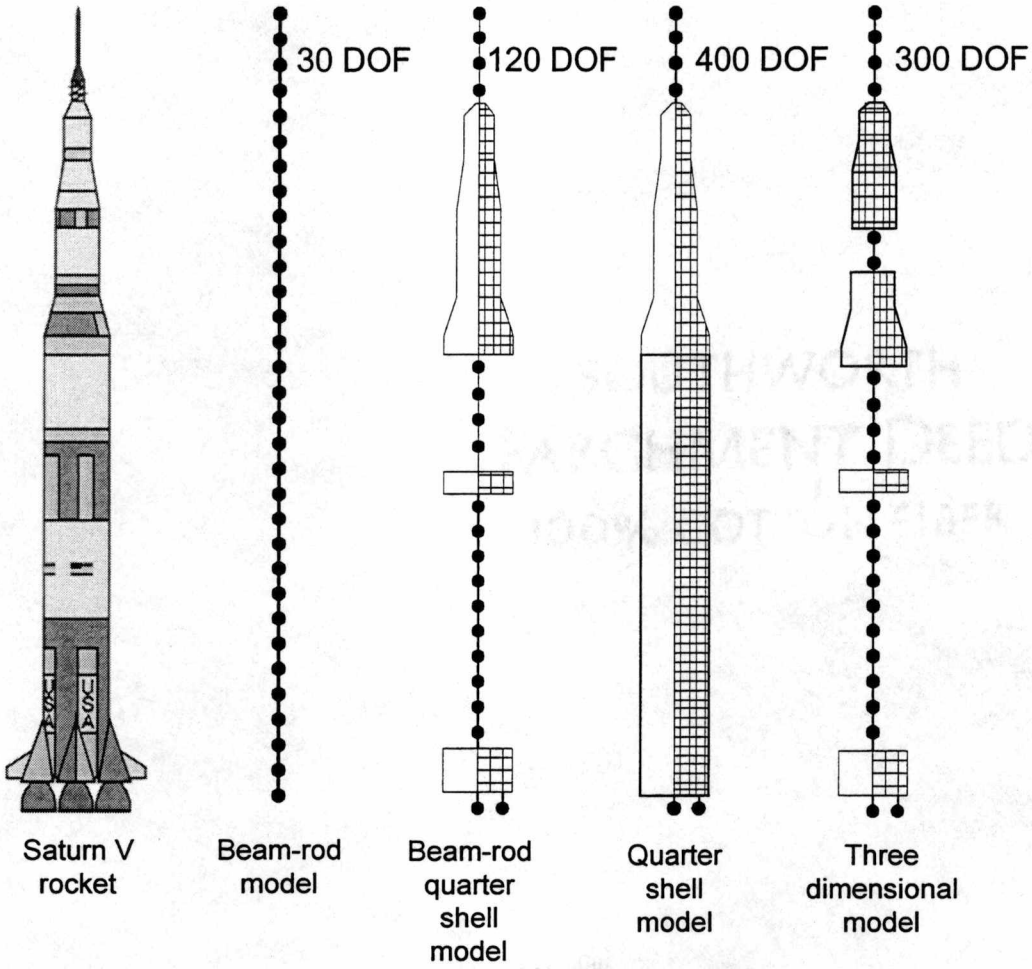


Figure 7-1: Structural Models [Craig, 1981]

two structural modes are important. These models are limited to linear perturbations around the baseline loading conditions.

While many techniques are available, finite element discretization of the Navier equations for solid mechanics based on displacement variables was used in the current effort.

Solid Equations

Linear elastic solids deform under load and return to their original configuration when the load is removed. The linear restriction comes from linear stress-strain relationships in material properties and assumptions associated with small deformations. Most metallic structures operate in the linear range of

material properties but substances such as rubber and composite laminates do not. Structures with nonlinear material properties may still exhibit elastic deformation under load and return to their original shape when the load is removed. The continuum equations are still valid for plastic deformation but the material properties are forever changed and the structure will not return to its original configuration when the load is removed. Nonlinear material properties are important in many fluid-structure interaction problems such as composite wing aircraft; antenna and radar enclosures; and concrete/steel bridges in cross flow.

Nonlinearity in structural models may also come from large deformation under relatively small applied loads. This does not imply nonlinear material properties or plastic strains. Very flexible structures such as ropes, watch springs and fly fishing rods may exhibit very large geometry changes with very little strain. Large geometry changes can be critical behavior for fluid-structure interactions such as slender missiles (javelins), high aspect ratio wings, and towed cables.

Without derivation, the Navier equations of motion for continuum solids with linear isotropic material properties are written as [Malvern, 1969]

$$\rho\frac{\partial^2\vec{u}}{\partial t^2}=\rho\vec{b}+\mu\nabla^2\vec{u}+(\lambda+\mu)\nabla\left(\nabla\cdot\vec{u}\right) \tag{7-1}$$

or

$$\rho\frac{\partial^2 u_i}{\partial t^2}=\rho b_i+\mu\frac{\partial^2 u_i}{\partial X_k\partial X_k}+(\lambda+\mu)\frac{\partial^2 u_k}{\partial X_i\partial X_k} \tag{7-2}$$

where λ and μ are the Lame' coefficients ($\lambda=\dfrac{\upsilon E}{(1+\upsilon)(1-2\upsilon)}$ and $\mu=\dfrac{E}{2(1+\upsilon)}$), $\vec{u}$ is a vector of the displacements (u and v) in the x and y directions, and $\vec{b}$ is a body force. The material properties determine the Young's modulus E [1], the Poisson ratio υ [2], and the material density ρ. The Navier

[1] Young's modulus represents the ratio of the longitudinal stress to the corresponding longitudinal strain
[2] Poisson ratio is the numerical value of the ratio of the contraction in a transverse direction to the corresponding extension in the longitudinal direction.

equations describe the relationship between inertial, internal, and body forces acting on a structure due to displacements. They must be extended to cover generalized stresses due to nonlinear or non-isotropic material properties, large deformations, and non-conservative frictional force.

There are many forms for the structural equations using combinations of stress, strain and various deformation descriptions as the dependent variables. Each has advantages in certain applications but the irreducible form using the deformation variables is the most useful for fluid-structure applications because the structural displacement provides the interactive coupling. The stress and strains within the structure are obviously important to its behavior and health and the deformation field may be post-processed to provide the information.

Keeping deformation as the independent variable, adding damping terms and dropping body forces, the structural dynamics equation set is generally written

$$M\frac{\partial^2\vec{u}}{\partial t^2}+C\frac{\partial\vec{u}}{\partial t}+K(\vec{u})=\vec{f}_{external}$$
$$\vec{f}_{inertia}+\vec{f}_{friction}+\vec{f}_{internal}=\vec{f}_{external} \qquad (7\text{-}3)$$

The inertia term is not a true force but may be treated as a pseudo force in the momentum equation. The density is now included in the mass matrix, the damping matrix, C, may be a linear function of the mass if it is limited to material damping of a homogeneous material or some other predefined relationship, and the internal force vector may be a nonlinear function of the displacements. The fluid pressure supplies the external force. The real key in structural calculations is the form of the stiffness matrix for the internal force vector. This is where the battle is won or lost and where nearly all the simplifying assumptions or nonlinear complications are implemented. The stiffness matrix is the same whether the problem is static or dynamic.

In the classic sense, the internal force vector for linear elasticity comes from the divergence of the Cauchy stress tensor, $\vec{f}_{internal}=\operatorname{div}T$. For linear elastic materials, the stress elements are related to the strains by a simple relationship $\sigma=D(\varepsilon-\varepsilon_{initial})+\sigma_{initial}$. The subscripted terms represent initial strains and residual stresses. The matrix D is an elasticity matrix containing the material properties. For

plane stress problems, $D = \frac{E}{1-\upsilon^2}\begin{bmatrix} 1 & \upsilon & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & \frac{1-\upsilon}{2} \end{bmatrix}$ and for general two dimensional elasticity, or plane strain, the matrix is $D = \frac{E(1-\upsilon)}{(1+\upsilon)(1-2\upsilon)}\begin{bmatrix} 1 & \upsilon/(1-\upsilon) & 0 \\ \upsilon/(1-\upsilon) & 1 & 0 \\ 0 & 0 & (1-2\upsilon)/2(1-\upsilon) \end{bmatrix}$

[Zienkiewicz and Taylor, Vol 1]. This can be easily expanded to three dimensions and for anisotropic materials.

For nonlinear problems, the divergence operator is still relevant in the current coordinates. For small displacements, these can be considered the same as the initial coordinates but large displacements invalidate this assumption and require more complex treatment. Nonlinear material properties are implemented in the relationship between stress and strain. Using the notation described in [Zienkiewicz and Taylor, Vol 2, p 213 and 285], the internal force vector may be written as $\vec{f}_{internal} = \int_V \overline{B}^T \sigma dV$ where $\overline{B}^T$ is now a linear or nonlinear differential operator in space relative to a specified coordinate system. The stress term is still given by $\sigma = D(\varepsilon - \varepsilon_{initial}) + \sigma_{initial}$ for most cases. This may be expanded to show the relationship with deformation through the strain by noting that $d\sigma = \frac{\partial \sigma}{\partial \varepsilon}\frac{\partial \varepsilon}{\partial u} du$. The nonlinear material properties are contained in the tangential elasticity matrix $D_T = \frac{\partial \sigma}{\partial \varepsilon}$ and the differential strain operator is given by $\overline{B} = \frac{\partial \varepsilon}{\partial u}$. The matrix $\overline{B}$ can now be defined by its linear and nonlinear components as $\overline{B} = B_0 + B_L(u)$ respectively. The B_0 term contains the linear infinitesimal strain relations and $B_L(u)$ is the nonlinear component dependent on the deformation [Zienkiewicz and Taylor, Vol 2, 1991].

It is important to make the distinction about the reference state for the analysis. In the Lagrangian (rather than Eulerian) frame of reference all the deformations are referred to the original configuration. For general continuum solids, the Green/St. Venant descriptions of Lagrangian strain components for finite deformations are used [Malvern, 1969]. The second Piola-Kirchoff stresses are the symmetric stresses conjugate to the Green/St. Venant strains in the reference state [Malvern, 1969]. Development of a complete tangent stiffness matrix for this formulation led to a great deal of confusion and some disagreement in the literature between 1965 and 1980 [Hibbitt, Marcal and Rice, 1970] [Stricklin, et al, 1972] [Yamada, 1972] [Belytschko and Hsieh, 1973] [McNamara and Marcal, 1973] [Bathe, Ramm and Wilson, 1975] [Cescotto, Frey and Fonder, 1979]. Large deflection, large rotation problems were typically dealt with according one of three classes [Knight, Macy and McCleary, 1989]:

1. the *total Lagrangian approach* where the deformed configuration is referenced to the initial undeformed configuration
2. the *updated Lagrangian approach* where the deformation increment due to the load increment is reference back to an updated local coordinate system with residual stresses accounting for current deformation
3. the *corotational method* where each element has a local coordinate system that rotates but does not deform with the element

Though each approach has some appealing properties, [Cescotto, Frey and Fonder, 1979] declared the total Lagrangian approach to be the most general and the one that allows for history dependent constitutive laws. [Zienkiewicz and Taylor, Vol 2, 1991] base all nonlinear analysis on the total Lagrangian approach and that is the method implemented in the current effort. The appropriate stiffness matrices generated by the approach will be detailed below closely following the technique and notation described in [Zienkiewicz and Taylor, Vol 1 and Vol 2].

The Green and St. Venant finite strain elements are given by

$$\varepsilon_x = \frac{\partial u}{\partial x} + \frac{1}{2}\left[\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2\right]$$
$$\varepsilon_y = \frac{\partial v}{\partial y} + \frac{1}{2}\left[\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2\right] \qquad \textbf{(7-4)}$$
$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} + \left[\frac{\partial u}{\partial x}\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\frac{\partial v}{\partial y}\right]$$

For small deformations, the higher order terms are ignored and ε_x and ε_y are interpreted as extensional strains and γ_{xy} is a shear strain. (Here, γ_{xy} is actually twice the shear strain given by Green and St. Venant; written this way for convenience in forming ε_L.) The terms inside the brackets reflect the nonlinear contributions and are used for small strain, large deformation problems as well as large strain. Note these represent the complete finite-strain for large and small deformations and not just a second order approximation to it. The derivatives are taken relative to the initial frame of reference.

From above, the differential strain operator is $\overline{B} = B_0 + B_L(u)$. The small strain operator, B_0, is given by

$$B_0 = \begin{bmatrix} \dfrac{\partial}{\partial x} & 0 \\ 0 & \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix} \qquad \textbf{(7-5)}$$

and for small strain, $\varepsilon_0 = B_0 \begin{Bmatrix} u \\ v \end{Bmatrix}$ where ε_0 is the linear strain component. The full strain comes from $\varepsilon = \varepsilon_0 + \varepsilon_L$ where ε_L is a nonlinear strain. The total strain may be small but still requires the nonlinear component if the deformation is large. The nonlinear differential strain component is a bit more complicated and

$$B_L = \begin{bmatrix} \theta_x^T & 0 \\ 0 & \theta_y^T \\ \theta_y^T & \theta_x^T \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} & 0 \\ 0 & \frac{\partial}{\partial y} \end{bmatrix} \text{ where } \theta_x^T = \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \end{bmatrix} \text{ and } \theta_y^T = \begin{bmatrix} \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix} \tag{7-6}$$

The nonlinear strain is $\varepsilon_L = \frac{1}{2} B_L \begin{Bmatrix} u \\ v \end{Bmatrix}$.

As stated earlier, the stiffness matrix is the same for a static or dynamic problem. Thus a static analysis is used to develop the stiffness matrix based on the principle of virtual work. The process is to apply a virtual displacement to each node and equate the work done by the external forces with the internal work. [Malvern, pp 235-248] provides a detailed derivation based on the Cauchy stress in local coordinates and based on finite strains and the second Piola-Kirchoff stress in the initial (Lagrangian) coordinates. The final form shows the virtual work terms due to internal stresses as the integral of a scalar product of the second Piola-Kirchoff stress tensor and the variation of the Green/St. Venant strain tensor integrated over the initial volume of the element.

$$\delta W = \int_{V_0} \tilde{T} \cdot\cdot \delta E dV_0 \tag{7-7}$$

From above, the strain tensor variation is $\overline{B}$ and the symmetric stress tensor may be written as a stress vector σ. If δu is the virtual deformation, this may be rewritten as

$$\delta W = \delta u \int_{V_0} \overline{B}^T \sigma dV_0 \tag{7-8}$$

and the virtual work balance equation becomes

$$\delta u \int_{V_0} \overline{B}^T \sigma dV_0 = \delta u \cdot \bar{f}_{external} \tag{7-9}$$

The external forces may be point or distributed loads on the surface or body forces over the entire volume of the element. The virtual work due to internal forces is distributed over the region. Since the equality must hold for any deformation, the deformation term may be eliminated to leave

$$\int_{V_0} \overline{B}^T \sigma dV_0 = \vec{f}_{external} \tag{7-10}$$

or

$$\int_{V_0} \overline{B}^T \sigma dV_0 - \vec{f}_{external} = 0 \tag{7-11}$$

This is the conservative form of the summation of the internal and external forces used on the right-hand side of a Newton iterative process. Substituting the stress relationship $\sigma = D(\varepsilon - \varepsilon_{initial}) + \sigma_{initial}$ and reiterating $\varepsilon = \varepsilon_0 + \varepsilon_L = \left(B_0 + \frac{1}{2}B_L\right)\vec{u}$, the internal force term becomes

$$\int_{V_0} \overline{B}^T D\varepsilon \, dV_0 = \int_{V_0} \overline{B}^T D\left(B_0 + \frac{1}{2}B_L\right)\vec{u} \, dV_0 \tag{7-12}$$

For the iterative process, a linearized version is needed for the left hand side. The *tangent stiffness matrix,* $d\int_{V_0} \overline{B}^T \sigma dV_0$ provides the needed iteration matrix. In the standard Newton method,

$$\left(d\int_{V_0} \overline{B}^T \sigma dV_0\right)\Delta u = -\left(\int_{V_0} \overline{B}^T \sigma dV_0 - \vec{f}_{external}\right) \tag{7-13}$$

To evaluate the left hand side, the definition of the strain is needed where $d\varepsilon = \overline{B}du$. Now,

$$d\left(\int_{V_0} \overline{B}^T \sigma dV_0\right) = \int_{V_0} \frac{\partial \overline{B}^T}{\partial u} \sigma dV_0 + \int_{V_0} \overline{B}^T \frac{\partial \sigma}{\partial u} dV_0 \tag{7-14}$$

Since, $\overline{B} = B_0 + B_L(u)$ then

$$\frac{\partial \overline{B}}{\partial u} = \frac{\partial B_L}{\partial u} \tag{7-15}$$

and

$$\frac{\partial \sigma}{\partial u} = D_T \frac{\partial \varepsilon}{\partial u} \tag{7-16}$$

but $\frac{\partial \varepsilon}{\partial u} = \overline{B}$ so

$$\frac{\partial \sigma}{\partial u} = D_T \overline{B} \tag{7-17}$$

Combining all this produces

$$d\left(\int_{V_0} \overline{B}^T \sigma dV_0\right) = \int_{V_0} \frac{\partial B_L^T}{\partial u} \sigma dV_0 + \int_{V_0} \overline{B}^T D \overline{B} dV_0 \tag{7-18}$$

The first term on the right-hand side is commonly referred to as the *initial stress matrix* or *geometric matrix,* K_σ, and is dependent on the stress level.

$$\begin{aligned} K_\sigma &= \int_{V_0} \frac{\partial B_L^T}{\partial u} \sigma dV_0 \\ &= \int_{V_0} G^T M G dV_0 \end{aligned} \tag{7-19}$$

where

$$M = \begin{bmatrix} \sigma_x & 0 & \tau_{xy} & 0 \\ 0 & \sigma_x & 0 & \tau_{xy} \\ \tau_{xy} & 0 & \sigma_y & 0 \\ 0 & \tau_{xy} & 0 & \sigma_y \end{bmatrix} \text{ and } G = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} & 0 \\ 0 & \frac{\partial}{\partial y} \end{bmatrix} \tag{7-20}$$

The second term may be written as $\overline{K} = \int_{V_0} \overline{B}^T D \overline{B} dV_0 = K_0 + K_L$ where

$$K_0 = \int_{V_0} B_0^T D B_0 dV_0 \tag{7-21}$$

is the normal small deformation matrix. The matrix K_L is due to the nonlinear strain and is expanded as

$$K_L = \int_{V_0} \left(B_0^T D B_L + B_L^T D B_0 + B_L^T D B_L\right) dV_0 \tag{7-22}$$

K_L is commonly referred to as *the initial displacement matrix* or the *large displacement matrix*. The total stiffness matrix or tangential stiffness matrix is thus the sum of the three matrices above [Zienkiewicz and Taylor, Vol 2, pp 285-287].

$$K_T = K_0 + K_\sigma + K_L \tag{7-23}$$

The tangential stiffness matrix for nonlinear deformation is now seen to be the small displacement stiffness matrix corrected by an initial stress matrix and an initial displacement matrix.

The tangential stiffness matrix, K_T, may or may not be required for matrix free implementations of the Newton method. If it is approximated by a directional derivative, then

$$\frac{\partial R(u)}{\partial u}\Delta u \approx \frac{R(u+h\Delta u) - R(u)}{h} \tag{7-24}$$

The various stiffness matrices (K_0, K_σ, K_L) are not needed if the residual $R(u)$ is formed from the nonlinear equations on each iteration since they are artifacts of linearization. The forward differencing in the directional derivative takes care of the effects. If, however, a linearized form of the residual $R(u)$ is used for speed or convenience, all the relevant stiffness matrices should be included. A linearized form is used in this research effort and the three stiffness matrices are included as needed by defining a priori whether the solution process is to be linear or nonlinear.

Discrete Implementation

Discrete implementation is straight forward based on the matrices and integrals described above. An element by element approach is used to form the integration matrices and the residual vectors are assembled from these. An inexact Newton method is used to linearize the equation set and is solved iteratively using the GMRES algorithm. This is written as

$$\frac{R_LHS(u+h\Delta u) - R_LHS(u)}{h} = -R_RHS \tag{7-25}$$

The left and right-hand residual vectors could use the same algorithm. However, the left hand side residual module was implemented separately for speed and convenience in implementing additional terms or adjustments to the left hand side calculation.

For nonlinear calculations not dependent on time (steady), the right-hand side residual comes from

$$\begin{aligned} R_RHS &= \int_{V_0} \overline{B}^T D\varepsilon \, dV_0 - \int_A \vec{f}_{external} dA \\ &= \int_{V_0} \overline{B}^T D\left(B_0 + \frac{1}{2}B_L\right) dV_0 \begin{Bmatrix} u \\ \vdots \\ v \\ \vdots \end{Bmatrix} - \int_A \vec{f}_{external} dA \\ &= K_{\substack{element \\ nonlinear}} \begin{Bmatrix} u \\ \vdots \\ v \\ \vdots \end{Bmatrix} - \int_A \vec{f}_{external} dA \end{aligned} \tag{7-26}$$

D is the generalized elasticity matrix and for plane stress is written as

$$D = \frac{E}{1-\upsilon^2} \begin{bmatrix} 1 & \upsilon & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & \frac{1-\upsilon}{2} \end{bmatrix} \tag{7-27}$$

B_0 is the infinitesimal strain/displacement matrix from equation **(7-5)** and is written as

$$B_0 = \begin{bmatrix} \frac{\partial N^T}{\partial x} & 0 \\ 0 & \frac{\partial N^T}{\partial y} \\ \frac{\partial N^T}{\partial y} & \frac{\partial N^T}{\partial x} \end{bmatrix} \tag{7-28}$$

where N is the finite element shape function vector. The term 0 is actually a zero vector with the same number of components as the shape function. Linear triangle, bilinear quadrilateral, and 8-node biquadratic quadrilateral elements were implemented.

B_L implements the nonlinear terms in the strain/displacement relationship and is given by

$$B_L = \begin{bmatrix} \theta_x^T & 0 \\ 0 & \theta_y^T \\ \theta_y^T & \theta_x^T \end{bmatrix} \begin{bmatrix} \frac{\partial N^T}{\partial x} & 0 \\ 0 & \frac{\partial N^T}{\partial x} \\ \frac{\partial N^T}{\partial y} & 0 \\ 0 & \frac{\partial N^T}{\partial y} \end{bmatrix} \text{ where } \theta_x = \begin{bmatrix} \frac{\partial N^T}{\partial x} & 0 \\ 0 & \frac{\partial N^T}{\partial x} \end{bmatrix} \begin{Bmatrix} u \\ \vdots \\ v \\ \vdots \end{Bmatrix}, \; \theta_y = \begin{bmatrix} \frac{\partial N^T}{\partial y} & 0 \\ 0 & \frac{\partial N^T}{\partial y} \end{bmatrix} \begin{Bmatrix} u \\ \vdots \\ v \\ \vdots \end{Bmatrix} \quad \textbf{(7-29)}$$

In the right-hand side residual equation, the external force $\int_A \vec{f}_{external} dA$ is integrated over the **deformed** area of the element. This provides the same nodal forces as if the external loads were transformed to the initial coordinate system and taken over the undeformed element shape. The nodal forces are integrated in a separate module using line integration.

The calculation procedures are somewhat simplified for steady, linear (small deformation) problems. [Zienkiewicz and Taylor, Vol 1, pp 468-474] describes a process for forming the residual with about a 25% reduction in the operation count compared to forming $B_0^T D B_0$ explicitly. In this case,

$$\begin{aligned} R_RHS &= \int_{V_0} B_0^T D \varepsilon \, dV_0 - \int_A \vec{f}_{external} dA \\ &= \int_{V_0} B_0^T D B_0 \, dV_0 \begin{Bmatrix} u \\ \vdots \\ v \\ \vdots \end{Bmatrix} - \int_A \vec{f}_{external} dA \qquad \textbf{(7-30)} \\ &= K_{\substack{element \\ linear}} \begin{Bmatrix} u \\ \vdots \\ v \\ \vdots \end{Bmatrix} - \int_A \vec{f}_{external} dA \end{aligned}$$

where

$$K_{\substack{element \\ linear}} = \lambda W_{ac} + \mu\left(W_{ca} + block_trace\left(W_{ac}\right)\right) \qquad \textbf{(7-31)}$$

λ and μ are the Lame constants, and the W matrices are defined by

$$W_{ac} = \int_{V_0} \begin{bmatrix} \frac{\partial N}{\partial x}\frac{\partial N^T}{\partial x} & \frac{\partial N}{\partial x}\frac{\partial N^T}{\partial y} \\ \frac{\partial N}{\partial y}\frac{\partial N^T}{\partial x} & \frac{\partial N}{\partial y}\frac{\partial N^T}{\partial y} \end{bmatrix} dV_0 \tag{7-32}$$

$$W_{ca} = \int_{V_0} \begin{bmatrix} \frac{\partial N}{\partial x}\frac{\partial N^T}{\partial x} & \frac{\partial N}{\partial y}\frac{\partial N^T}{\partial x} \\ \frac{\partial N}{\partial x}\frac{\partial N^T}{\partial y} & \frac{\partial N}{\partial y}\frac{\partial N^T}{\partial y} \end{bmatrix} dV_0 \tag{7-33}$$

$$block_trace(W_{ac}) = \int_{V_0} \begin{bmatrix} \frac{\partial N}{\partial x}\frac{\partial N^T}{\partial x} + \frac{\partial N}{\partial y}\frac{\partial N^T}{\partial y} & 0 \\ 0 & \frac{\partial N}{\partial x}\frac{\partial N^T}{\partial x} + \frac{\partial N}{\partial y}\frac{\partial N^T}{\partial y} \end{bmatrix} dV_0 \tag{7-34}$$

In the code, W_{ac} is formed by element integration and the other matrices are simple rearrangements of it.

For unsteady problems, the second order system

$$M\frac{\partial^2 \vec{u}}{\partial t^2} + C\frac{\partial \vec{u}}{\partial t} + K(\vec{u}) = \vec{f}_{external} \tag{7-35}$$

may be rewritten as a first order system. To do so, let $\vec{w} = \frac{\partial \vec{u}}{\partial t}$ be the velocity. Taking the time derivative, $\frac{\partial^2 \vec{u}}{\partial t^2} = \frac{\partial \vec{w}}{\partial t}$. Substituting into the original second order equation and writing the two equations as a set,

$$\begin{aligned} M\frac{\partial \vec{w}}{\partial t} + C\frac{\partial \vec{u}}{\partial t} + K(\vec{u}) &= \vec{f}_{external} \\ \frac{\partial \vec{u}}{\partial t} &= \vec{w} \end{aligned} \tag{7-36}$$

Rearranging gives

$$C\frac{\partial \vec{u}}{\partial t} + M\frac{\partial \vec{w}}{\partial t} + K(\vec{u}) - \vec{f}_{external} = 0$$
$$\frac{\partial \vec{u}}{\partial t} - \vec{w} = 0 \tag{7-37}$$

If the stiffness term $K(\vec{u})$ is linearized and assembled element by element, the first order in time linearized equation set becomes

$$R_RHS = \begin{bmatrix} C & M \\ I & 0 \end{bmatrix} \frac{\partial \begin{Bmatrix} \vec{u} \\ \vec{w} \end{Bmatrix}}{\partial t} + \begin{bmatrix} K_{element} & 0 \\ 0 & -I \end{bmatrix} \begin{Bmatrix} \vec{u} \\ \vec{w} \end{Bmatrix} - \begin{Bmatrix} \vec{f}_{external} \\ 0 \end{Bmatrix} \tag{7-38}$$

This provides direct access to the nodal velocities, $\vec{w}$, and the discrete time step may be implemented in a simple forward or backward Euler step. The damping matrix C may be ignored, written as a scalar multiple of the mass $C = cM$, or some other more intricate relation. In this research effort, it is written as a scalar multiple of the mass matrix due to simplicity and the artificial nature of the pseudo time transients.

Pseudo time stepping terms are treated as a first order system in time as described above. However, the pseudo time terms are not used on the right-hand side of the equation. In order to keep the residual vector the same length on the left and right-hand sides, the right-hand side for a steady problem with pseudo time stepping is formed as

$$R_RHS = \begin{bmatrix} K_{element} & 0 \\ 0 & 0 \end{bmatrix} \begin{Bmatrix} \vec{u} \\ \vec{w} \end{Bmatrix} - \begin{Bmatrix} \vec{f}_{external} \\ 0 \end{Bmatrix} \tag{7-39}$$

The second row of the matrix is not actually formed but illustrates an important point. The nodal velocity, $\vec{w}$, has no role in the right-hand side of a steady equation. If the identity matrix is included in the stiffness hypermatrix, $\vec{w}$ makes a contribution to the right-hand side residual. This acts as a penalty term on the velocity and it will eventually damp out, but adds many more unnecessary steps to the process. The right-hand side of an unsteady problem with pseudo time stepping remains exactly the same as given above and, again, there are no pseudo time terms used in the right-hand side.

In the current effort, the left hand side of the Newton equation is formed using linearized approximations to the right-hand side residual. This may not be the best approach for fast, parallel machines but it is the fastest for a single processor with sufficient memory. Although it is not required by the matrix free technique, the linearized element integration matrices are created and stored during the formation of the right-hand side residual and recalled for rapid calculation of the left hand side residuals. Thus, each left hand side residual vector is the assembly of the matrix vector products of the linearized element integration matrices and the current estimate, $\vec{u}^p$, for the deformation vector.

$$R_LHS_e = \left[Q_{element} + K_{element}\right]\left\{\vec{u} + h\Delta\vec{u}^p\right\}_{element} - \vec{f}\left(\vec{u} + h\Delta\vec{u}^p\right)_{external} \tag{7-40}$$

The matrix Q generically represents the physical and pseudo time contributions to the left hand side residual and K generically represents the tangent stiffness. The external force is included as it may change with the deformation increment for follower loads and as part of a tightly coupled fluid-structure calculation.

The complete left hand side residual is actually the difference between two vectors. The first is stored from the first iteration and used repeatedly as the zero increment step. The second is formed with each new estimate for the deformation increment.

$$\frac{R_LHS\left(\vec{u} + h\Delta\vec{u}^p\right) - R_LHS\left(\vec{u}\right)}{h} \tag{7-41}$$

The left hand side residuals follow the exact same methodology described above and the total pseudo time, physical time, nonlinear equation comes from.

$$R_LHS = \left(\begin{bmatrix} \frac{cM_{pseudo}}{\Delta\tau} & \frac{M_{pseudo}}{\Delta\tau} \\ \frac{I}{\Delta\tau} & 0 \end{bmatrix} + \begin{bmatrix} \frac{C_{physical}}{\Delta t} & \frac{M_{physical}}{\Delta t} \\ \frac{I}{\Delta t} & 0 \end{bmatrix} + \begin{bmatrix} (K_0 + K_\sigma + K_L) & 0 \\ 0 & -I \end{bmatrix} \right) \cdot \left(\begin{Bmatrix} \vec{u} \\ \vec{w} \end{Bmatrix} + h \begin{Bmatrix} \Delta\vec{u} \\ \Delta\vec{w} \end{Bmatrix}^p \right) - \begin{Bmatrix} \vec{f}\left(\vec{u} + h\Delta\vec{u}^p\right)_{external} \\ 0 \end{Bmatrix} \quad (7\text{-}42)$$

The terms not needed are simply not included. For steady problems without pseudo time stepping, this simply reduces to

$$R_LHS = \left[K_0 + K_\sigma + K_L\right] \cdot \left(\{\vec{u}\} + h\{\Delta\vec{u}^p\} \right) - \vec{f}\left(\vec{u} + h\Delta\vec{u}^p\right)_{external} \quad (7\text{-}43)$$

for nonlinear deformations or

$$R_LHS = \left[K_0\right] \cdot \left(\{\vec{u}\} + h\{\Delta\vec{u}^p\} \right) - \vec{f}\left(\vec{u} + h\Delta\vec{u}^p\right)_{external} \quad (7\text{-}44)$$

for linear deformations.

Locking of Thin Structures

The equations and deformation variables outlined above are general in nature and have application to a wide range of structural configurations. However, they often lead to convergence stagnation or "locking" when discretely applied to thin structures. There are a variety of explanations for the phenomena but they all essentially boil down to the interaction between shear and bending modes within the structure. For thin structures, bending modes should dominate but strain energy may get trapped in shear modes that appear during the numerical solution process. This destroys convergence and gives the structure the appearance of being stiffer than it actually is. Thus, even converged solutions may be far from the actual solution [Donea and Lamain, 1987].

A first explanation of the "locking" comes from basic wave analysis. For an isotropic solid, bending (longitudinal) waves travel at $\xi_{bending} = \sqrt{\frac{\lambda + 2\mu}{\rho}}$ where ρ is the density, λ and μ are the Lame coefficients describe earlier. The shear (transverse) waves travel at $\xi_{shear} = \sqrt{\frac{\mu}{\rho}}$. The ratio of the wave speeds depends only on the Poisson ratio of the material and the ratio is not large (ill conditioned) unless the material is nearly incompressible (as $\upsilon \to .5$ the bending (longitudinal) wave speed approaches infinity.)

$$\begin{aligned} \frac{\xi_{bending}}{\xi_{shear}} &= \sqrt{\frac{\lambda + 2\mu}{\mu}} = \sqrt{2 + \frac{\lambda}{\mu}} \\ &= \sqrt{2 + \frac{\frac{\upsilon E}{(1+\upsilon)(1-2\upsilon)}}{\frac{E}{2(1+\upsilon)}}} \\ &= \sqrt{2\frac{(1-\upsilon)}{(1-2\upsilon)}} \end{aligned} \tag{7-45}$$

However, time scales depend on the aspect ratio of the structure as well as the material properties. If the time for a wave to transit a dimension of the structure is given by $time = \frac{dimension}{wave\ speed}$, taking a ratio of time scales (for example a bending wave transiting the large dimension and a shear wave transiting the small dimension) gives

$$\begin{aligned} \frac{time_{long}}{time_{short}} &= \frac{\frac{L}{\xi_{bending}}}{\frac{t}{\xi_{shear}}} = \frac{L}{t}\frac{\xi_{shear}}{\xi_{bending}} \\ &= \frac{L}{t}\sqrt{\frac{(1-2\upsilon)}{2(1-\upsilon)}} \end{aligned} \tag{7-46}$$

The ratio of time scales is proportional to the aspect ratio of the structure and aspect ratios of 100,000 or more are not unusual. Thus, shear (and bending) modes in the short dimension will transit the thickness and be established in much less time than modes in the long dimension. This traps or "locks" strain energy early in the solution process, which must eventually migrate to the bending mode in the long dimension. For the example given, it is tempting to try to manipulate the material properties with an artificial Poisson ratio to offset the effects of the aspect ratio. This essentially creates an artificial longitudinal (bending) wave speed fast enough to transit the long dimension in the time a shear wave would transit the short dimension. A Poisson ratio very near 0.5 (specifically $\frac{4999999999}{9999999999}$) would offset the aspect ratio but the elasticity matrix, D, would be incorrect with the material appearing too stiff.

Although it was not vigorously pursued in this effort, logic would suggest a strong analogy exists with artificial wave speeds created and controlled by time derivative preconditioning techniques used in fluid flow analysis. A future effort should investigate this possibility beyond the current dynamic relaxation techniques used in structures. [Stolarski and Belytschko, Part I and II, 1986] and others have already developed the mode decomposition topic but not in the form of time derivative preconditioning.

Structural calculation procedures have an advantage over those for fluid analyses in that a beam, thin plate, or shell typically remains a beam, thin plate or shell throughout the solution process. Thus, special forms of the continuum equations may be devised to better describe the behavior of the element. This is a hallmark of finite element analysis and beam, plate and shell elements abound.

In their introduction to simple beam theory, [Timoshenko and Gere, 1972] show the transverse displacement as a fourth order relation to the load

$$\frac{d^4 \Delta y}{dx^4} = \frac{q}{EI} \tag{7-47}$$

where q is the load and I is the moment of inertia for the cross section. The fundamental assumption for small deflections of thin beams (Bernoulli-Euler) is that cross section cuts perpendicular to the neutral axis remain perpendicular even as the neutral axis deforms. A second fundamental assumption is that direct

stresses in the transverse direction are small [Zienkiewicz and Taylor, Vol 2, p 3]. A direct finite element implementation of the fourth order deformation equation above would require C^1 continuity (7-continuous slope) on any element. In practice, displacement and slope of the neutral axis are used as the dependent variables and require only C^0 continuity. This essentially decouples the rotation of the neutral axis from its displacement and provides a surrogate variable such that C^1 continuity is no longer required. This approach is actually a more correct physical implementation because the fourth order equation is a result of the small deflection assumption. In the full derivation, the moment causing curvature in the beam is proportional to the change in slope along the beam

$$\frac{d\theta}{ds} = \frac{d\left[\arctan\left(\frac{dw}{dx}\right)\right]}{ds} \tag{7-48}$$

rather than the small angle approximation along the original orientation given by

$$\theta \approx \tan\theta = \frac{dw}{dx}, \quad ds \approx dx \rightarrow \frac{d\theta}{ds} \approx \frac{d^2w}{dx^2} \tag{7-49}$$

Thus the change in slope along the beam is actually related to the deflection of the beam by

$$\frac{d\theta}{ds} = \frac{\frac{d^2w}{dx^2}}{\left[1+\left(\frac{dw}{dx}\right)^2\right]^{\frac{3}{2}}} \tag{7-50}$$

and the approximation $\frac{d\theta}{ds} = \frac{d^2w}{dx^2}$ is a statement that $\frac{dw}{dx} << 1$ [Timoshenko and Gere, p 168-170].

[Zienkiewicz and Taylor, Vol 2] develop the approach using a general theory for thick plates with thin plates as a limiting case. They assume plane sections normal to the neutral axis remain plane (but not necessarily normal) and direct strains in the transverse direction are negligible. The equations developed are based on moment, normal, and shear stress resultants from cross sectional cuts of the structure. The

dependent variables are the neutral axis deflection and slope and changes are taken relative to the original orientation. The stress resultant equations are given by

$$\begin{aligned} M &= -\frac{Et^3}{12}\frac{\partial \theta}{\partial x} \\ P &= Et\frac{\partial u_0}{\partial x} \\ S &= \beta Gt\left(-\theta + \frac{\partial w_0}{\partial x}\right) \end{aligned} \tag{7-51}$$

M, P, and S are the moment, normal, and shear stress resultants; u_0 and w_0 are the normal and transverse displacements; t is the thickness, and G is the shear modulus (purposely stated independently of E). β is a shape factor particular to the cross section. The equilibrium equations are

$$\begin{aligned} \frac{\partial M}{\partial x} + S &= 0 \\ \frac{\partial S}{\partial x} + f_{external} &= 0 \\ \frac{\partial P}{\partial x} &= 0 \end{aligned} \tag{7-52}$$

In the thin limit, the shear modulus $G \to \infty$ and $S = -\theta + \frac{\partial w_0}{\partial x} = 0$. This is essentially an implementation of the time scale observation made above based on wave speeds (i.e. shear waves in the thin direction effectively move with infinite speed relative to the rest of the structure). It also re-implements the Bernoulli-Euler assumption that sections normal to the neutral axis remain normal (developed independently through small angle assumptions in equation **(7-49)**.)

It is important to note that the specialized beam/plate formulation still does not avoid locking behavior. In practice, using a lower order interpolation function (under-integrating) for the shear deformation equation avoids locking. The under-integration acts as an artificial diffusive operator and filters or damps out high frequency modes [Stolarski and Belytschko, 1983]. This is again reminiscent of techniques developed to apply compressible fluid codes to incompressible flows. However, with under-integrated elements, special care must be taken (e.g. "patch tests" [Zienkiewicz and Taylor, Vol 1]) to

ensure spurious grid modes such as "hour-glassing" are not generated by rank deficiencies in the tangent stiffness matrix [Belytschko and Bachrach, 1986] [Wriggers and Reese, 1996].

CHAPTER 8

DYNAMIC RELAXATION IN STRUCTURES

Background

Dynamic relaxation techniques for structural equations first began to appear in the literature around 1965. They were first used in explicit time-stepping algorithms and followed artificial transients to reach a converged steady state. They are now widely used in fluid flow calculations but do not seem to have a strong following in the structural world. Nevertheless, the basic technique is sound for structural equations and seems to get dusted off about every 15 years for new applications or implementation concepts.

[Underwood, 1983] presented a thorough review of the explicit dynamic relaxation technique and literature on the subject, assorted methods for selecting equation parameters, and demonstrated application of the method for highly nonlinear problems such as snap-through calculations of an arch (the oil can problem). The explicit dynamic relaxation method requires a large number of iterations to converge in artificial time for each load increment but ultimately converges to the nonlinear solution without overshoot.

[Oakley and Knight, 1995] essentially pick up dynamic relaxation where [Underwood, 1983] left off. They focused on the implementation issues of the adaptive dynamic relaxation (ADR) scheme on vector and parallel computer platforms for an explicit pseudo-transient to steady state. They applied the method to a highly nonlinear analysis of automotive tires. It is important to note that [Oakley and Knight, 1995] were interested in physical transient solutions as well as static solutions. However, they never applied the ADR method as a pseudo-transient in a dual time stepping mode as has been done in fluid flow calculations [Venkateswaran and Merkle, 1995] [Shuen, Chen and Choi, 1992]. They also purposely dismiss implicit time stepping methods because of the problems associated with ill-conditioning of the Jacobian matrix in the Newton step.

In the present effort, an adaptive dynamic relaxation scheme (ADR) is developed as part of an implicit pseudo transient algorithm for a static structure. Artificial mass and damping terms are added to remove numerical stiffness due to ill-conditioning of the Jacobian matrix in the Newton step, and thus ease the burden of solving the matrix with a GMRES algorithm.

Implicit Adaptive Dynamic Relaxation

The equations for structural dynamics are most commonly written as

$$M\frac{\partial^2\vec{u}}{\partial t^2}+C\frac{\partial\vec{u}}{\partial t}+K(\vec{u})=\vec{f}_{external}$$
$$\vec{f}_{inertia}+\vec{f}_{friction}+\vec{f}_{internal}=\vec{f}_{external} \tag{8-1}$$

The equation set is obviously second order in time with a first order (in time) damping term. This same basic structure is used for both pseudo transient and physical transient problems. If a pseudo transient is to be used as part of a dual time stepping mode during a physical transient, additional inertia and friction terms are added for the pseudo transient.

$$M_\tau\frac{\partial^2\vec{u}}{\partial\tau^2}+C_\tau\frac{\partial\vec{u}}{\partial\tau}+M\frac{\partial^2\vec{u}}{\partial t^2}+C\frac{\partial\vec{u}}{\partial t}+K(\vec{u})=\vec{f}_{external} \tag{8-2}$$

The dual time stepping is not pursued in the current effort.

The second order system may be rewritten as a first order system using the trivial relation

$$\vec{v}=\frac{\partial\vec{u}}{\partial t} \tag{8-3}$$

or a slightly more complex relation

$$\vec{v}=A\frac{\partial\vec{u}}{\partial t}+B\vec{u} \tag{8-4}$$

[Park and Felippa, 1983] elaborate on the more complex version in their analysis of coupled systems. The trivial case is a specific implementation where $A=I$ and $B=0$. The trivial case was used in the current effort but other choices for A and B may bear fruit. If the damping matrix, C, is taken as a scalar multiple of the mass matrix, $C=cM$, the first order system is given by

$$\begin{bmatrix} cM & M \\ I & 0 \end{bmatrix}\frac{\partial\begin{Bmatrix}\vec{u}\\ \vec{v}\end{Bmatrix}}{\partial t}+\begin{bmatrix} K & 0 \\ 0 & -I \end{bmatrix}\begin{Bmatrix}\vec{u}\\ \vec{v}\end{Bmatrix}=\begin{Bmatrix}\vec{f}_{external}\\ 0\end{Bmatrix} \tag{8-5}$$

Some license has been taken writing the (possibly nonlinear) stiffness term as a matrix, K. The method is not limited to linear stiffness problems and the product $K\vec{u}$ should be read as the appropriate internal force vector $K(\vec{u})$.

For an explicit pseudo transient to steady state, this would be the starting point for developing the iteration scheme. [Underwood, 1983] and [Oakley and Knight, 1995] start here with the second order set of equations. However, for the implicit approach used in the current effort, an inexact Newton method is used to form an incremental left hand side operator that implements the pseudo transient.

The left hand side operator is formed from a finite difference approximation to a Jacobian for the right hand side. This is written as

$$\frac{R_LHS(\vec{u}+h\Delta\vec{u})-R_LHS(\vec{u})}{h}=-R_RHS(\vec{u}) \tag{8-6}$$

where $R_RHS(\vec{u})\rightarrow 0$ as the solution converges. For now, let

$$R_RHS(\begin{Bmatrix}\vec{u}\\ \vec{v}\end{Bmatrix})=\begin{bmatrix}cM & M\\ I & 0\end{bmatrix}\frac{\partial\begin{Bmatrix}\vec{u}\\ \vec{v}\end{Bmatrix}}{\partial t}+\begin{bmatrix}K & 0\\ 0 & -I\end{bmatrix}\begin{Bmatrix}\vec{u}\\ \vec{v}\end{Bmatrix}-\begin{Bmatrix}\vec{f}_{external}\\ 0\end{Bmatrix} \tag{8-7}$$

As the solution converges to a steady state, the time derivative term drops out. The ultimate time step is to push to convergence in a single step. Thus, the artificial time derivative term is now dropped from the right hand side so as not to create right hand side residual errors that must be removed through additional outer

$$R_RHS(\begin{Bmatrix}\vec{u}\\ \vec{v}\end{Bmatrix})=\begin{bmatrix}K & 0\\ 0 & 0\end{bmatrix}\begin{Bmatrix}\vec{u}\\ \vec{v}\end{Bmatrix}-\begin{Bmatrix}\vec{f}_{external}\\ 0\end{Bmatrix} \tag{8-8}$$

iterations. The $-I$ term is also dropped from the stiffness matrix to avoid a residual if the velocity does not go to zero in a single step. This residual would drive a non-zero velocity in the left hand side increment. The goal is not to force $\frac{\partial\vec{u}}{\partial t}=\vec{v}=0$ but $K\vec{u}-\vec{f}_{external}=0$.

The left hand side uses the same equation but keeps the time derivative terms. Since the left hand side calculation is actually a difference, the external force vector could be dropped if the forces are fixed

$$R_LHS(\begin{Bmatrix}\vec{u}\\ \vec{v}\end{Bmatrix})=\begin{bmatrix}cM & M\\ I & 0\end{bmatrix}\frac{\partial\begin{Bmatrix}\vec{u}\\ \vec{v}\end{Bmatrix}}{\partial t}+\begin{bmatrix}K_T & 0\\ 0 & -I\end{bmatrix}\begin{Bmatrix}\vec{u}\\ \vec{v}\end{Bmatrix}-\begin{Bmatrix}\vec{f}_{external}\\ 0\end{Bmatrix} \tag{8-9}$$

for the iteration. The external force vector may be included for a tightly coupled solution procedure or deformation dependent load. Also note the physical stiffness matrix, K, may be formed as the original,

nonlinear version $K(\vec{u})$ or, since the operator will simulate a Jacobian, it may be written as a tangent stiffness matrix, $K_T = \dfrac{\partial K(\vec{u})}{\partial \vec{u}}$. The result is the same since

$$\frac{K(\vec{u}+h\Delta\vec{u})-K(\vec{u})}{h} \approx K_T\Delta\vec{u} = \frac{hK_T\Delta\vec{u}+K_T\vec{u}-K_T\vec{u}}{h} = \frac{K_T(\vec{u}+h\Delta\vec{u})-K_T(\vec{u})}{h} \tag{8-10}$$

In the current effort, the element tangent stiffness matrices were formed during the right hand side evaluation and stored for rapid generation of the left hand side vectors. It should be noted, this was not required to form the left hand side vectors but presented the best combination of speed and memory on the computer hardware used.

In a discrete form, the left hand side may now be written in matrix form as

$$\begin{aligned}\frac{R_LHS(\vec{u}+h\Delta\vec{u})-R_LHS(\vec{u})}{h} &\approx \left[\frac{1}{\Delta t}\begin{bmatrix} cM & M \\ I & 0 \end{bmatrix} + \begin{bmatrix} K_T & 0 \\ 0 & -I \end{bmatrix}\right]\begin{Bmatrix}\Delta\vec{u}\\ \Delta\vec{v}\end{Bmatrix} \\ &\approx \begin{bmatrix} \dfrac{cM}{\Delta t}+K_T & \dfrac{M}{\Delta t} \\ \dfrac{I}{\Delta t} & -I \end{bmatrix}\begin{Bmatrix}\Delta\vec{u}\\ \Delta\vec{v}\end{Bmatrix}\end{aligned} \tag{8-11}$$

If an infinitely large time step is used, $\Delta t \to \infty$ and the matrix reduces to

$$\begin{bmatrix} K_T & 0 \\ 0 & -I \end{bmatrix}\begin{Bmatrix}\Delta\vec{u}\\ \Delta\vec{v}\end{Bmatrix} \tag{8-12}$$

The equations are uncoupled and, since the right hand side velocity residual is forced to zero as described above, the active terms simplify to

$$K_T\Delta\vec{u} = -\left(K(\vec{u}) - \vec{f}_{external}\right) \tag{8-13}$$

If $\Delta\vec{q} \equiv \begin{Bmatrix}\Delta\vec{u}\\ \Delta\vec{v}\end{Bmatrix}$ is taken as $\Delta\vec{q} = \vec{q}^{n+1} - \vec{q}^{n}$, an error analysis leads to an appropriate selection of parameters for the pseudo time stepping terms. Let the error in a given solution be given by

$$e^n = q^n - q^* \tag{8-14}$$

where q^* is the actual discrete solution. Substituting $q^n = e^n + q^*$ gives

$$\begin{bmatrix} \frac{cM}{\Delta t}+K_T & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & -I \end{bmatrix} e^{n+1} + \begin{bmatrix} \frac{cM}{\Delta t}+K_T & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & -I \end{bmatrix} q^* - \begin{bmatrix} \frac{cM}{\Delta t}+K_T & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & -I \end{bmatrix} e^n - \begin{bmatrix} \frac{cM}{\Delta t}+K_T & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & -I \end{bmatrix} q^* = -R_RHS\left(e^n + q^*\right) \tag{8-15}$$

or

$$\begin{bmatrix} \frac{cM}{\Delta t}+K_T & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & -I \end{bmatrix} e^{n+1} - \begin{bmatrix} \frac{cM}{\Delta t}+K_T & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & -I \end{bmatrix} e^n = -R_RHS\left(e^n + q^*\right) \tag{8-16}$$

Assume the right hand side stiffness matrix is linear and the external load vector does not change with the deformation over the current time step. Now, $K_T\vec{u} = K(\vec{u})$. Expanding the right hand side gives

$$\begin{bmatrix} \frac{cM}{\Delta t}+K_T & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & -I \end{bmatrix} e^{n+1} - \begin{bmatrix} \frac{cM}{\Delta t}+K_T & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & -I \end{bmatrix} e^n = -\begin{bmatrix} K & 0 \\ 0 & -I \end{bmatrix} e^n \underbrace{- \begin{bmatrix} K & 0 \\ 0 & 0 \end{bmatrix} q^* + \begin{Bmatrix} \vec{f}_{external} \\ 0 \end{Bmatrix}}_{=0} \tag{8-17}$$

Grouping like terms and simplifying,

$$\begin{bmatrix} \frac{cM}{\Delta t}+K_T & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & -I \end{bmatrix} e^{n+1} = \begin{bmatrix} \frac{cM}{\Delta t} & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & 0 \end{bmatrix} e^n \tag{8-18}$$

or

$$e^{n+1} = \begin{bmatrix} \frac{cM}{\Delta t}+K_T & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & -I \end{bmatrix}^{-1} \begin{bmatrix} \frac{cM}{\Delta t} & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & 0 \end{bmatrix} e^n \tag{8-19}$$

The matrix product on the right hand side represents a gain matrix for the implicit equation set. It is well known that for convergence, the spectral radius of the gain matrix must be less than unity. Likewise, the spectral radius of the inverse of the gain matrix must be greater than unity. In this case, the inverse of the gain matrix is easier to deal with so it will be used to select the dynamic relaxation parameters.

The inverse gain matrix is

$$GAIN^{-1} = \begin{bmatrix} \frac{cM}{\Delta t} & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & 0 \end{bmatrix}^{-1} \begin{bmatrix} \frac{cM}{\Delta t} + K & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & -I \end{bmatrix} = \begin{bmatrix} I & -\Delta t I \\ \Delta t M^{-1} K & (1 + c\Delta t) I \end{bmatrix} \tag{8-20}$$

Get the eigenvalues of $GAIN^{-1}$ by forming the characteristic polynomial using

$$\begin{bmatrix} I - DD & -\Delta t I \\ \Delta t M^{-1} K & (1 + c\Delta t) I - DD \end{bmatrix} \tag{8-21}$$

where $DD = dd \cdot I$. Thus,

$$DD^2 - (c\Delta t + 2) DD + (c\Delta t + 1) I + \Delta t^2 M^{-1} K = 0 \tag{8-22}$$

This can be restated in terms of the eigenvalues of $M^{-1}K$ by replacing $M^{-1}K$ with $-\lambda I$ where λ is an eigenvalue of $M^{-1}K$ and satisfies the standard relation $(M^{-1}K - \lambda I)\vec{u}_{\substack{eigen- \\ vector}} = 0$. Making the substitutions, the problem becomes

$$dd^2 I - (c\Delta t + 2) ddI + (c\Delta t + 1 + \Delta t^2 \lambda) I = 0 \tag{8-23}$$

Solving for dd, the eigenvalues of $GAIN^{-1}$ are by

$$dd = \frac{(c\Delta t + 2) \pm \sqrt{(c\Delta t)^2 - 4\Delta t^2 \lambda}}{2} \tag{8-24}$$

where λ is an eigenvalue of $M^{-1}K$.

There are now three possibilities for dd depending whether the argument of the square root operator is zero, positive, or negative. If the argument is zero,

$$(c\Delta t)^2 - 4\Delta t^2 \lambda = 0 \tag{8-25}$$

and

$$dd = \frac{c\Delta t + 2}{2} \tag{8-26}$$

Since dd is an eigenvalue of the inverse gain matrix, $|dd|>1$ is required for convergence. Substituting, and solving for the damping coefficient

$$\begin{aligned} 1 &< \frac{c\Delta t+2}{2} \\ 2 &< c\Delta t+2 \\ c\Delta t &> 0 \\ c &> 0 \end{aligned} \tag{8-27}$$

since Δt is always positive. This is not much of a restriction and merely states the damping should resist motion. Returning to the argument of the radical

$$\begin{aligned} (c\Delta t)^2 - 4\Delta t^2\lambda &= 0 \\ c^2 &= 4\lambda \\ c &= \pm 2\sqrt{\lambda} \end{aligned} \tag{8-28}$$

but $c>0$ from above so

$$c = 2\sqrt{\lambda} \tag{8-29}$$

For the case where the term under the radical is positive, again $|dd|>1$ is required for convergence so

$$\begin{aligned} dd &= \frac{(c\Delta t+2)\pm\sqrt{(c\Delta t)^2-4\Delta t^2\lambda}}{2} \\ 1 &< \frac{(c\Delta t+2)\pm\sqrt{(c\Delta t)^2-4\Delta t^2\lambda}}{2} \\ 2 &< (c\Delta t+2)\pm\sqrt{(c\Delta t)^2-4\Delta t^2\lambda} \\ c\Delta t &> \mp\sqrt{(c\Delta t)^2-4\Delta t^2\lambda} \\ (c\Delta t)^2 &> (c\Delta t)^2-4\Delta t^2\lambda \\ 4\Delta t^2\lambda &> 0 \end{aligned} \tag{8-30}$$

Convergence is satisfied for all positive eigenvalues of $M^{-1}K$ regardless of c or Δt but only for positive eigenvalues, λ.

The last case is for the argument of the radical less than zero. This unfolds as

$$
\begin{aligned}
dd &= \frac{(c\Delta t+2) \pm \sqrt{(c\Delta t)^2 - 4\Delta t^2 \lambda}}{2} \\
1 &< \frac{(c\Delta t+2) \pm i\sqrt{4\Delta t^2 \lambda - (c\Delta t)^2}}{2} \\
2 &< (c\Delta t+2) \pm i\sqrt{4\Delta t^2 \lambda - (c\Delta t)^2} \\
0 &< c\Delta t \pm i\sqrt{4\Delta t^2 \lambda - (c\Delta t)^2}
\end{aligned}
\tag{8-31}
$$

using the relation $|a \pm bi| = (a^2 + b^2)^{1/2}$ for the magnitude of complex numbers, the last term further reduces to give

$$
\begin{aligned}
0 &< \left((c\Delta t)^2 + 4\Delta t^2 \lambda - (c\Delta t)^2\right)^{1/2} \\
0 &< \left(4\Delta t^2 \lambda\right)^{1/2} \\
4\Delta t^2 \lambda &> 0
\end{aligned}
\tag{8-32}
$$

This is the same restriction as for the case of the positive radical and is satisfied for all positive eigenvalues of $M^{-1}K$.

Since dd represents the eigenvalues of the inverse gain matrix, $GAIN^{-1}$, larger values for dd should lead to faster convergence. In equation **(8-24)**, either one or two eigenvalues are possible depending on the terms under the radical. [Oakley and Knight, 1995] put forth the argument that one eigenvalue will always lead to faster convergence than the other if two are allowed. To avoid this situation, they set the terms for the explicit gain matrix based on real repeated or imaginary eigenvalues. For the implicit case, this essentially means $(c\Delta t)^2 - 4\Delta t^2 \lambda \le 0$. Upon simplification, this limits the damping parameter so that $0 < c \le 2\sqrt{\lambda_0}$ where λ_0 is the smallest (positive) eigenvalue of $M^{-1}K$. The damping is now tuned to the lowest eigenvalue. If the system is not positive definite, the damping is set to zero. Notice the argument only affects the damping coefficient and not the time step size. The only restriction on the time step size so far is $\Delta t > 0$, which is always the case.

As a reminder, the eigenvalues of the inverse of the gain matrix should be greater than unity for rapid convergence. Looking back to the original statement

$$dd = \frac{(c\Delta t + 2) \pm \sqrt{(c\Delta t)^2 - 4\Delta t^2 \lambda}}{2}$$
$$= 1 + \Delta t \left[\frac{c \pm \sqrt{c^2 - 4\lambda}}{2} \right] \tag{8-33}$$

it is obvious larger (finite) values of Δt could lead to faster convergence. However, as $\Delta t \to \infty$, $dd \to \infty$ or $dd^{-1} \to 0$. As the step size becomes infinite, the system of equations should return to the original (non-ADR) rate of convergence rate. The gain for the entire eigenvalue spectrum should be less than unity but it is not zero as the above would imply. Therefore, additional analysis is needed based on the ratio of the maximum to minimum eigenvalues.

System convergence is controlled by the ratio of fastest to slowest eigenvalues. If

$$gg_{\max} \equiv (dd_{\min})^{-1} = \left(\frac{c\Delta t + 2 \pm i\sqrt{4\Delta t^2 \lambda - (c\Delta t)^2}}{2} \right)^{-1}$$
$$= (a \pm bi)^{-1} \tag{8-34}$$

then

$$|dd_{\min}|^{-1} = (a^2 + b^2)^{-1/2}$$
$$= \left[\frac{(c\Delta t)^2 + 4c\Delta t + 4 + 4\Delta t^2 \lambda - (c\Delta t)^2}{4} \right]^{-1/2} \tag{8-35}$$
$$= \left[\Delta t^2 \lambda_0 + c\Delta t + 1 \right]^{-1/2}$$

and

$$gg_{\max} = \left[\frac{1}{\Delta t^2 \lambda_0 + c\Delta t + 1} \right]^{1/2} \tag{8-36}$$

The eigenvalue ratio for the gain matrix is the same as the eigenvalue ratio for the inverse gain matrix. The two are linked as

$$\frac{gg_{\max}}{gg_{\min}} = \frac{\dfrac{1}{dd_{\min}}}{\dfrac{1}{dd_{\max}}} = \frac{dd_{\max}}{dd_{\min}} \tag{8-37}$$

Substituting for dd, this becomes

$$\left(\frac{|dd_{\max}|}{|dd_{\min}|}\right) = \left[\frac{\Delta t^2 \lambda_m + c\Delta t + 1}{\Delta t^2 \lambda_0 + c\Delta t + 1}\right]^{1/2} \tag{8-38}$$

Now the restriction $0 < c \le 2\sqrt{\lambda_0}$ may be rewritten $c = \sqrt{\alpha\lambda_0}$ where $0 < \alpha \le 4$. Substituting for c,

$$\left(\frac{|dd_{\max}|}{|dd_{\min}|}\right) = \left[\frac{\Delta t^2 \lambda_m + \Delta t\sqrt{\alpha\lambda_0} + 1}{\Delta t^2 \lambda_0 + \Delta t\sqrt{\alpha\lambda_0} + 1}\right]^{1/2} \tag{8-39}$$

Now, it is easy to see

$$\lim_{\Delta t \to \infty}\left[\frac{\Delta t^2 \lambda_m + \Delta t\sqrt{\alpha\lambda_0} + 1}{\Delta t^2 \lambda_0 + \Delta t\sqrt{\alpha\lambda_0} + 1}\right]^{1/2} \to \sqrt{\frac{\lambda_m}{\lambda_0}} \tag{8-40}$$

so the system convergence rate returns to the non-ADR rate when the step size becomes infinite.

In order for the system to converge evenly, the ratio of eigenvalues should be as close to unity as possible. This ensures all the system modes move at about the same rate or according to the same time scale. If R represents this eigenvalue ratio, then

$$R^2 = \frac{\Delta t^2 \lambda_m + \Delta t\sqrt{\alpha\lambda_0} + 1}{\Delta t^2 \lambda_0 + \Delta t\sqrt{\alpha\lambda_0} + 1} \tag{8-41}$$

Solving for the step size gives

$$\Delta t = \frac{\sqrt{\alpha\lambda_0}\left(R^2 - 1\right) \pm \left(R^2 - 1\right)\sqrt{\alpha\lambda_0\left(R^2 - 1\right) + 4\left(\lambda_m - \lambda_0 R^2\right)}}{2\left(\lambda_m - \lambda_0 R^2\right)} \tag{8-42}$$

Assume $\lambda_0 << \lambda_m$. This is generally true for structural mechanics problems [Oakley and Knight, 1995] (and is the basic reason preconditioning is needed for conjugate gradient methods). Also assume R^2 is of

order one, $R^2 = \vartheta(1)$. Now $R^2\lambda_0 << \lambda_m$ and $\alpha\lambda_0 << \lambda_m$. Substituting and rearranging the time step relationship above leads to the simplified form

$$\begin{aligned}
\Delta t &\approx \frac{\sqrt{\alpha\lambda_0}\left(R^2-1\right) \pm \left(R^2-1\right)\sqrt{4\left(\lambda_m\right)}}{2\left(\lambda_m\right)} \\
&\approx \frac{\left(R^2-1\right)\left[\sqrt{\alpha\lambda_0} \pm \sqrt{4\lambda_m}\right]}{2\lambda_m} \\
&\approx \frac{\left(R^2-1\right)\left[\sqrt{4\lambda_m}\right]}{2\lambda_m} \\
\Delta t &\approx \frac{R^2-1}{\sqrt{\lambda_m}}
\end{aligned} \tag{8-43}$$

This is the same form achieved by [Oakley and Knight, 1995] with $R^2 - 1 = 2$ or $R = \frac{|gg_{\max}|}{|gg_{\min}|} = \sqrt{3}$.

An alternate analysis is based on the condition of the system. Let $\kappa = \frac{\lambda_m}{\lambda_0}$ and rearrange the equation for the time step to form the condition term:

$$\begin{aligned}
\Delta t &= \frac{\sqrt{\alpha\lambda_0}\left(R^2-1\right) \pm \left(R^2-1\right)\sqrt{\alpha\lambda_0\left(R^2-1\right)+4\left(\lambda_m-\lambda_0R^2\right)}}{2\left(\lambda_m-\lambda_0R^2\right)} \cdot \frac{\sqrt{\lambda_0}}{\sqrt{\lambda_0}} \\
&= \frac{\sqrt{\alpha}\left(R^2-1\right) \pm \left(R^2-1\right)\sqrt{\alpha\left(R^2-1\right)+4\left(\frac{\lambda_m}{\lambda_0}-R^2\right)}}{2\left(\frac{\lambda_m}{\lambda_0}-R^2\right)\sqrt{\lambda_0}} \\
&= \frac{\sqrt{\alpha}\left(R^2-1\right) \pm \left(R^2-1\right)\sqrt{\alpha\left(R^2-1\right)+4\left(\kappa-R^2\right)}}{2\left(\kappa-R^2\right)\sqrt{\lambda_0}} \\
\Delta t &= \left(R^2-1\right)\frac{\sqrt{\alpha} \pm \sqrt{4\kappa-(4-\alpha)R^2-\alpha}}{2\left(\kappa-R^2\right)\sqrt{\lambda_0}}
\end{aligned} \tag{8-44}$$

By definition, $1 < R^2 < \frac{\lambda_m}{\lambda_0}$, $0 < \alpha \le 4$ and $\Delta t > 0$. Only the positive root satisfies all these constraints. In the limit of zero damping, $\alpha = 0$ leads to

$$\Delta t = \left(R^2 - 1\right)\frac{\sqrt{4\kappa - 4R^2}}{2\left(\kappa - R^2\right)\sqrt{\lambda_0}}$$

$$= \left(R^2 - 1\right)\frac{\sqrt{\kappa - R^2}}{\left(\kappa - R^2\right)\sqrt{\lambda_0}} \tag{8-45}$$

$$= \frac{\left(R^2 - 1\right)}{\sqrt{\kappa - R^2}\sqrt{\lambda_0}}$$

If $\kappa >> R^2$, this again reduces to

$$\Delta t = \frac{R^2 - 1}{\sqrt{\lambda_m}} \tag{8-46}$$

If not,

$$\Delta t = \frac{R^2 - 1}{\sqrt{\lambda_m - \lambda_0 R^2}} \tag{8-47}$$

If R is selected to be small but within it's stated constraints such as in the case above from [Oakley and Knight, 1995] where $R = \sqrt{3}$, then

$$\Delta t = \frac{2}{\sqrt{\lambda_m - 3\lambda_0}} \tag{8-48}$$

In the limit of full damping where $\alpha = 4$,

$$\Delta t = \left(R^2 - 1\right)\frac{\sqrt{4} + \sqrt{4\kappa - 4}}{2\left(\kappa - R^2\right)\sqrt{\lambda_0}}$$

$$= \left(R^2 - 1\right)\frac{1 + \sqrt{\kappa - 1}}{\left(\kappa - R^2\right)\sqrt{\lambda_0}} \tag{8-49}$$

Using the same scenarios as the zero damping case, $\kappa >> R^2$ (and $\kappa >> 1$ by default since $R^2 > 1$) again leads to

$$\Delta t = \frac{R^2 - 1}{\sqrt{\lambda_m}} \tag{8-50}$$

If not,

$$\Delta t = \left(R^2 - 1\right)\left[\frac{\sqrt{\lambda_0} + \sqrt{\lambda_m - \lambda_0}}{\left(\lambda_m - \lambda_0 R^2\right)}\right] \tag{8-51}$$

Substituting a desirable value such as $R = \sqrt{3}$ leads to

$$\Delta t = 2\left[\frac{\sqrt{\lambda_0} + \sqrt{\lambda_m - \lambda_0}}{\left(\lambda_m - 3\lambda_0\right)}\right] \tag{8-52}$$

The end result is that regardless of the damping parameter selected, if the system is ill conditioned and good convergence is desired, the condition $\kappa >> R^2$ holds true and, somewhat arbitrarily using $R = \sqrt{3}$, the time step size is given by

$$\Delta t = \frac{2}{\sqrt{\lambda_m}}$$

where λ_m is the largest eigenvalue of $M^{-1}K$.

Although it has not been stated previously, M and Δt are linked and only one may be selected arbitrarily. I selected M. The particular value selected will be adjusted by the damping coefficient, c, and the time step, Δt. In summary,

$$0 < c \le 2\sqrt{\lambda_0} \tag{8-53}$$

and

$$\Delta t = \frac{R^2 - 1}{\sqrt{\lambda_m}} \tag{8-54}$$

where λ_0 and λ_m are the minimum and maximum eigenvalues respectively of $M^{-1}K$. Also, if $\frac{\lambda_m}{\lambda_0}$ describes the condition number of $M^{-1}K$ then R^2 can be viewed as the desired condition number of the ADR equation set. I selected $R^2 = 3$.

Eigenvalue Estimates

Since λ_0 and λ_m are generally not known ahead of time and may change as a nonlinear solution progresses, they should be estimated or bounded using established techniques such as Gerschgorin circle theorem, Rayleigh quotients, Lanczos iterations, power iterations, etc.

Gerschgorin's circle theorem places an upper bound on the maximum eigenvalue where

$$\lambda_m \le \max_i \sum_{j=1}^{n} \frac{|K_{ij}|}{M_{ii}}$$

A disadvantage of the method is the K matrix must be formed in order for the terms to be assembled for the summation. This only provides an estimate of the upper bound and was not implemented.

The Rayleigh quotient or Rayleigh iteration provides an estimate of the eigenvalue associated with the eigenvector closest to a starting vector. Most of the references to the Rayleigh quotient in the structural literature assume it will estimate the eigenvalue for the fundamental mode. This is only true if the deformation vector is closer to the fundamental mode vector than any other eigenvector. However, the starting vector is typically very simple and associates with the fundamental mode and the minimum eigenvalue. In this case,

$$\lambda_0 = \omega_0^2 = \frac{u^T K u}{u^T M u}$$

[Golub and Van Loan, 1989] describe the cubic convergence of the Rayleigh iteration method where u is a starting vector. [Oakley and Knight, 1995] use the nodal velocity for the starting vector and do not iterate. Other common choices for the starting vector are the nodal deformation and the residual vector. Again, the Rayleigh method, whether iterated or single pass, provides an estimate of the eigenvalue associated with the closest eigenvector to the trial vector, u. This method was tested and used to verify the relative accuracy of the minimum eigenvalue estimate from the Arnoldi method but was not implemented.

In the current effort, Arnoldi iterations were used to simultaneously estimate the largest and smallest eigenvalues. It was arbitrarily stopped after twenty iterations but this proved to be sufficient for

the simple geometries tested. The damping coefficient is selected as $c = 2\sqrt{\lambda_0}$ and $\Delta t = \frac{2}{\sqrt{\lambda_m}}$. The pseudo mass is taken as a multiple of the physical mass.

Arnoldi Process for Extreme Eigenvalue Estimates

The Lanczos iteration method provides estimates for both the maximum and minimum eigenvalues. [Golub and Van Loan, 1989] devote an entire chapter to describing the method and the theoretical mechanisms that make it work. The extreme eigenvalues converge fairly quickly [Strang, 1986, pp 416-417] [Golub and Van Loan, 1989, p 475] and do not require explicit formation of the matrix since the only requirement is the matrix-vector product. Lanczos only applies to symmetric matrices and forms a tridiagonal matrix, $T = Q^T\left(M^{-1}K\right)Q$, that is similar to $M^{-1}K$. Similar matrices have the same eigenvalues. In the Lanczos iterative process, the extreme eigenvalues of $T_{1:n,1:n}$ are estimates of the eigenvalues of $M^{-1}K$ and become successively better with each iteration, n. However, $M^{-1}K$ is not necessarily symmetric even if M is diagonal and K is symmetric. In structural calculations, K is symmetric for linear applications and unsymmetric for nonlinear ones.

For unsymmetric K, the techniques is basically the same using Arnoldi iterations. [Saleem, Pulliam, and Cheer, 1993] used the Arnoldi process extensively while mapping the eigenvalue footprint of the ARC2D fluid flow code under various conditions. They showed the largest eigenvalue of the ARC2D gain matrix operator obtained from Arnoldi iterations matched the convergence rate of the code to within 10^{-3} in most cases. [Golub and Van Loan, 1989] describe the Arnoldi process to form an upper Hessenberg matrix, H, where $H = Q^T\left(M^{-1}K\right)Q$ at each iteration rather than a tridiagonal matrix, T, as was the case in the (symmetric) Lanczos process. The basic Arnoldi process is to form a sequence of vectors, q_i, where $Q = \left[q_1, \ldots q_n\right]$ satisfies $M^{-1}Kq_j = \sum_{i=1}^{j+1} h_{ij}q_i$. The q_i are formed as the vectors spanning a Krylov subspace, $span\left\{q_1, M^{-1}Kq_1, \ldots, \left(M^{-1}K\right)^{j-1}q_1\right\}$, where each new vector is made

orthogonal to the others through a modified Gram-Schmidt process. The components of the upper Hessenberg matrix, H, are simply the scaling factors generated by the Gram-Schmidt orthogonalization. The eigenvalues of the upper left $m \times m$ block of H are estimates of the eigenvalues of $M^{-1}K$. Larger blocks provide estimates for more eigenvalues but the extreme eigenvalues of these blocks of H tend to converge to the extreme eigenvalues of $M^{-1}K$ long before the $m \times m$ block reaches the full $n \times n$ size of H.

The number of Lanczos or Arnoldi iterations to perform is not definite. More iterations could provide better estimates but are more costly. It is possible to track convergence of the maximum and minimum eigenvalues of the T or H coefficient matrices generated by the iterations and stop when the values start to converge. However, this requires finding eigenvalues after each iteration. The work required to do so grows approximately as n^3 where n is the iteration number. Another possibility is to check for loss of orthogonality of the q_i vectors generated as this indicates convergence to an eigenvector. It is not a very predictable estimate though.

The Kaniel-Paige convergence theory [Golub and Van Loan, 1989] shows the estimated eigenvalues, θ_m and θ_0, will approach the actual eigenvalues from the middle of the spectrum as $\lambda_m \geq \theta_m^n$ and $\theta_0^n \geq \lambda_0$. Thus, the estimated eigenvalues are not conservative estimates as $0 < c \leq 2\sqrt{\lambda_0} \leq 2\sqrt{\theta_0^n}$ and $0 < \Delta t \leq \frac{2}{\sqrt{\lambda_m}} \leq \frac{2}{\sqrt{\theta_m^n}}$. The estimates using a fixed number (20) of iterations proved to be either fully converged or "close enough" in practice and did not incur the additional work of finding eigenvalues for multiple matrices. The eigenvalues of the $20x20$ matrix were found using the built-in eigenvalue function in MATLAB.

CHAPTER 9

FLUID GRID MOTION

Background

The hallmark of fluid-structure interaction problems is the coupling between the changing shape or orientation of the structure and the changing load distribution driving or being created by the changing shape. In short, either the fluid or the structure may be the actual driver but the result is a changing shape or orientation for the fluid domain. Some practical examples include the gross domain shape changes associated with a folded airbag inflation [Mestreau and Lohner, 1996]; a blast wave smashing a vehicle or building [Baum, et.al., 1996]; large but constrained motion of an automotive piston with detailed combustion flame front and heat transfer; and the flutter response of a transonic wing with control inputs [Guillot and Friedmann, 1994]. Since most practical problems do not have analytic solutions, the analysis is typically performed on a discrete grid that must somehow change with the shape and still accommodate the necessary details of the fluid flow and meet "grid quality" criteria.

There are numerous techniques to update the fluid grid. Some bring new nodes into the calculation to capture flowfield details, some redistribute the existing nodes, and some [Alonso and Jameson, 1994] [Alonso, Martinelli and Jameson, 1995] need only update the orientation of the nodes with the grid treated as a rigid body motion. All the methods reviewed position the wetted structure boundary nodes based on the motion of the structure. Whether or not these boundary nodes can move or slide along the solid boundary depends on the need. Normally, the boundary nodes are "glued" to the wetted surface and only move as the surface moves. Inflow and outflow boundary nodes may stay fixed in an inertial frame, move as if attached to a point on the structure by a rigid body, or deform depending on the need. Some use a combination approach by fixing the boundary nodes and stretching the adjoining cells to some limit before "snapping" the boundary nodes to a new attachment point on a smooth and inertial background grid [Sheng, et.al., 1997]. The inflow and outflow boundary conditions must be updated accordingly and a given node or cell edge may switch from inflow to outflow or vice versa.

Most nodes are internal to the flow domain and their position must be determined by some means. Experience and logic suggest the battle may be won or lost by how the nodes near (but not on) the

boundary are treated. Cell size, aspect ratio, distortion, orthogonality to the surface, etc, all affect the quality of the flow calculation and convergence properties.

Most fluid flow "solution adaptive" grid methods are also relevant to fluid-structure interaction problems. One technique for fluid-structure interactions is to move the boundaries and allow the interior grid to adapt to the new fluid flow requirements. [Thompson, 1985] provides a comprehensive overview of solution adaptive grid methods and describes the strengths and weaknesses of each approach. Innovative techniques and data structures have continued to evolve with extensions to three dimensions and unstructured grids. Each technique requires a strategy or criteria for determining regions for refinement and adaption as well as a means to alter the mesh [Biswas and Strawn, 1993]. Algebraic methods represent one class of solution adaptive grid technique for altering the mesh. [Kania, 1996] provides a typical example and describes an algebraic algorithm using three specifications to control near-wall clustering of grid nodes, smoothing, and adaption strength. The strategy for adaption does not have to be based on flow conditions and may be global or localized. For example, [Yang and Lohner, 1997] only allow mesh movement in the region near the moving structure. Algebraic methods may also be used to generate very quickly a new grid with each deformation increment. They must be used cautiously, however, to avoid violating the "geometric conservation law" [Thomas and Lombard, 1978] [Lesoinne and Farhat, 1996]. Other solution adaptive techniques include minimizing distributed weight functions and solving partial differential equations. [Soni and Yang, 1992] describe a general-purpose technique for developing solution adaptive grids against a fixed background mesh.

A second strategy for mesh motion is based on stretching an initial grid. [Batina, 1989] outlines a strategy using a network of nodes connected by linear and torsional springs. In essence, the edges of the elements are treated as linear springs with torsional springs at the nodes to control skewness. The springs may have a uniform stiffness or it may vary. This way, stiff torsional springs at boundaries maintain orthogonality or some original angle and other, less critical, regions may bear most of the burden. A closely related method is a continuum approach to mesh motion. In this case, the mesh is viewed as the discretization of a continuum "ether" that deforms as a solid with fixed or variable stiffness properties. In essence, the deformations in the physical structure are treated as Dirichet boundary conditions for the imaginary structure. The equations to be solved are the linear or non-linear Navier equations for solid

isotropic media. The location and velocity of the fluid grid nodes are updated as if they were another structural calculation. This continuum approach is the method implemented in the current effort and will be described in more detail below.

A third strategy is based on using fast grid generators to create a new grid with each step. The data from the old grid is interpolated to the new grid. This is a critical step as the interpolation may be diffusive or non-conservative.

A fourth possible strategy not found in the literature is based on domain decomposition. In this approach, a series of overlapping fixed grids would be attached to a flexible body. As the body deforms, the amount of grid overlap would change in a "Chinese fan" fashion. The actual grid motion would be a rigid body motion. An advantage here would be the potential for coarse grain parallel computations. The doubly defined regions on the surface would require careful consideration before transferring pressure and viscous loads to the surface. These issues have all been addressed individually [Benek, et.al., 1986] but, again, have not been found in the literature for fluid-structure interactions.

The final approach is again a combination of the techniques above. [Lohner, 1992] describes the basic technique for large-scale grid motion as stretching the initial grid to some limit and then regridding. Any valid approach may be used for stretching the grid but some strategy is needed for defining the limit criteria to trigger regridding. The regridding may be global or localized and the values for the new grid are interpolated from the old grid. Unstructured grids are more amenable to this approach and clever data structures can greatly improve the speed [Biswas and Strawn, 1993].

Discrete Implementation

In explicit calculations of node locations using structural codes, a numerical disturbance may arise in the shape of a sawtooth pattern of node displacements along a grid line. This is referred to as hourglassing (Figure 9-1) when it occures in two adjacent grid lines. It is basically a $2\Delta x$ type wave and comes from a lack of coupling between adjacent vertices [Pracht, 1975]. Giving the nodes some mass so they require an acceleration to move effectively controls the mode. This couples each node to its nearest

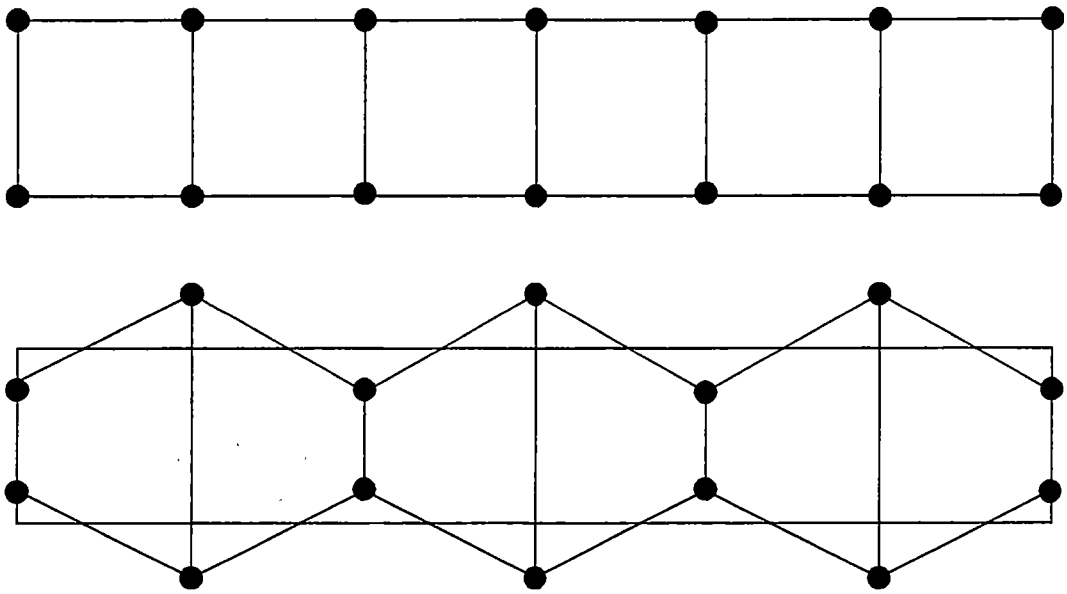


Figure 9-1: Zero Energy Mode (Hourglassing)

neighbors and reduces the relative velocities between them. This issue is only included for reference and is not evident in implicit solvers.

The continuum equations for grid motion are the same as those for a structure:

$$M\frac{\partial^2\vec{u}}{\partial t^2}+C\frac{\partial\vec{u}}{\partial t}+K\vec{u}=f_{ext} \tag{9-1}$$

M, C and K are mass, damping and stiffness matrices with artificial material properties for the imaginary structure. $\vec{u}=\begin{Bmatrix}u\\w\end{Bmatrix}$ is a deformation vector measured relative to an initial state. Since the imaginary structure is entirely artificial, the initial state can be the original undeformed grid or simply the most recent grid. The original grid is preferred for oscillatory motion so the fluid grid nodes return to their original positions each time the structure returns to the neutral position. Updating the grid may create path dependence so that the original grid is never recreated. Comparison of different cycles then requires interpolation between grids. "Stress relieving" or updating the grid with each step may provide some grid quality benefits for large scale, one-way motion but the baseline grid was used in the current effort.

The second order system of equations was reduced to a first order system for consistency with the structural calculation and to ease the coupled implementation. The ALE fluid flow equations incorporate a grid velocity term as a modification to the flux vector. In the first order system, the grid nodal velocity, $\vec{v} = \begin{Bmatrix} v_x \\ v_y \end{Bmatrix}$, is defined as a dependent variable and the grid pseudo-structure system becomes

$$\begin{bmatrix} c \cdot M & M \\ I & 0 \end{bmatrix} \frac{\partial}{\partial t} \begin{Bmatrix} \vec{u} \\ \vec{v} \end{Bmatrix} + \begin{bmatrix} K & 0 \\ 0 & -I \end{bmatrix} \begin{Bmatrix} \vec{u} \\ \vec{v} \end{Bmatrix} = \begin{Bmatrix} \vec{f}_{ext} \\ \vec{0} \end{Bmatrix} \tag{9-2}$$

where the frictional damping matrix is assumed proportional to the mass. This form provides direct access to the position and velocity variables needed for the fluid integration regions and flux adjustments.

The external force vector, f_{ext}, is set to zero and the deformation from the physical structure is implemented as Dirichlet boundary conditions for the position and velocity for boundary nodes on the wetted surface. For the test problems selected, the inflow and outflow boundary nodes are also set by Dirichlet conditions on the position and velocity.

The discrete system of equations **(9-3)** is derived directly from a Galerkin weak statement using homogeneous material properties. The value for the scalar damping term, c, was determined by eigenvalue analysis when testing dynamic relaxation techniques (described in Chapter 8) for the grid motion. However, dynamic relaxation was not used for grid motion other than during experimentation. The scalar damping term, c, was normally set to zero as the position and velocity of all boundary nodes were determined by Dirichlet boundary conditions. Variable material properties distributed across the grid could also have been implemented but were unnecessary. This could have been used to make the pseudo-structure very rigid near the walls to maintain grid orthogonality and flexible in less critical regions.

Only the linear elastic stiffness matrix for plane stress was implemented in this study. The linear elastic stiffness matrix determined the distribution of internal nodes only as all boundary nodes were set using Dirichlet conditions.. The individual element contributions to nodal forces were calculated element-by-element and the global residual force vector determined by assembly of the individual elements. Through this procedure, global matrices were never created or needed. The weak statement form of equation **(9-2)** with the assumptions above is given as

$$WS = S_e \left\{ \begin{array}{l} \rho \cdot \iint_{\Omega_e(t)} \begin{bmatrix} \{N_1\}\{N_1\}^T & 0 \\ 0 & \{N_1\}\{N_1\}^T \end{bmatrix} d\Omega \dfrac{\partial}{\partial t} \begin{Bmatrix} v_x \\ v_y \end{Bmatrix}_e \\ \iint_{\Omega_e(t)} \begin{bmatrix} \{N_1\}\{N_1\}^T & 0 \\ 0 & \{N_1\}\{N_1\}^T \end{bmatrix} d\Omega \dfrac{\partial}{\partial t} \begin{Bmatrix} u \\ w \end{Bmatrix}_e \end{array} \right\} + \left\{ \begin{array}{l} \iint_{\Omega_e(t)} \left[B^T D B \right] d\Omega \begin{Bmatrix} u \\ w \end{Bmatrix}_e \\ -\iint_{\Omega_e(t)} \begin{bmatrix} \{N_1\}\{N_1\}^T & 0 \\ 0 & \{N_1\}\{N_1\}^T \end{bmatrix} d\Omega \begin{Bmatrix} v_x \\ v_y \end{Bmatrix}_e \end{array} \right\} \tag{9-3}$$

where

$$B = \begin{bmatrix} \dfrac{\partial \{N_1\}^T}{\partial x} & 0 \\ 0 & \dfrac{\partial \{N_1\}^T}{\partial y} \\ \dfrac{\partial \{N_1\}^T}{\partial y} & \dfrac{\partial \{N_1\}^T}{\partial x} \end{bmatrix} \tag{9-4}$$

and

$$D = \begin{bmatrix} D11 & D12 & 0 \\ D12 & D11 & 0 \\ 0 & 0 & D33 \end{bmatrix} \tag{9-5}$$

$$D11 = \frac{E}{1-\upsilon^2},\ D12 = \frac{\upsilon E}{1-\upsilon^2} \text{ and } D33 = \frac{(1-\upsilon)E}{2(1-\upsilon^2)}$$

D represents the plane stress form of the linear elasticity matrix, E is the Young's modulus, υ the Poisson ratio, ρ the density, and S_e the finite element assembly operator . The material properties are completely artificial and may be set to any convenient and beneficial value. For simplicity, they were set to the same values as the actual structure.

A shortcut version of the stiffness matrix is actually implemented in the code where $K = \iint \left[B^T D B \right] d\Omega$ is replaced by (see equations 29-33 in Chapter 7):

$$K_{ac} = D12 \cdot W_{ac} + D33 \cdot \left(W_{ca} + \delta_{ac} W_{bb} \right) \tag{9-6}$$

$$W = \begin{bmatrix} \frac{\partial \{N_1\}}{\partial x} \cdot \frac{\partial \{N_1\}^T}{\partial x} & \frac{\partial \{N_1\}}{\partial x} \cdot \frac{\partial \{N_1\}^T}{\partial y} \\ \frac{\partial \{N_1\}}{\partial y} \cdot \frac{\partial \{N_1\}^T}{\partial x} & \frac{\partial \{N_1\}}{\partial y} \cdot \frac{\partial \{N_1\}^T}{\partial y} \end{bmatrix} \tag{9-7}$$

The tensor notation is included to highlight the block transpose and the block trace operator. This does not follow the exact definition as given in [Zienkiewicz and Taylor, Vol 1, pp 468-474] for the first Lame' coefficient but it is equivalent to the $B^T DB$ system and produces the correct results against analytic solutions with a reduced operation count.

The boundary conditions are all controlled by Dirichlet conditions so specific values are set for $\vec{u}$ and $\vec{v}$. This decreases the number of degrees of freedom in the system. For a residual based iterative solver, the residual corresponding to each boundary node value must also be set to zero. This creates a two step process for implementing boundary conditions. In step one, the predetermined boundary values for $\vec{u}$ and $\vec{v}$ are overlaid to the state variable input vector. The global residual vector is formed as usual from assembly of the individual elements using the state variable input vector. In step two, the elements of the residual vector associated with the set boundary conditions are set to zero.

A final step in the solution process is included to accommodate the modified Gram Schmidt orthogonalization in the GMRES solver. The first two equations in the system have large values (typical $E = 30 \times 10^6 \, {}^{lb_f}/_{in^2}$) associated with the stiffness matrix, K. The second two equations do not have any such amplifiers so the residual values will be on the order of the input variables. This leads to a stiffness in the system of equations due the disparity in residual values and an under-emphasis on the second set of equations during the solution process. Fortunately a simple and effective solution is to simply scale the residuals to a common order of magnitude. Since the code implemented contains dimensional units, the scaling must be appropriate to the units. The scaling values for a typical Young's modulus $E = 4.32 \times 10^9 \, {}^{lb_f}/_{ft^2}$ (notice square feet instead of square inches) are 1×10^{-5} for the first two equations and 1×10^3 for the second two.

Solution Process

The deformed physical structure provides boundary conditions for the pseudo structure that describes the fluid grid. The deformed fluid grid is found using a Newton-GMRES iterative process modified for pseudo time stepping. There are two artificial processes to consider. The first implements artificial material properties to treat the fluid grid region as if it were an actual solid structure. The material properties (mass, damping, stiffness) should be chosen to create a smooth distribution of grid nodes. A low mass/high stiffness combination would most closely approximate a rigid body motion while high mass/low stiffness would have a more fluid-like behavior. The time terms would be treated as actual physical transients for unsteady fluid structure interactions. The second artificial process implements pseudo time stepping in the solution process to reduce numerical stiffness by adding more artificial mass and damping time terms to the equation, but now in pseudo time. These may be selected to be the same as the artificial properties just described but not necessarily. The pseudo time material properties (mass, damping, stiffness) should be chosen to speed convergence of the inner iteration process and would most likely come from some analysis such as the ADR process described in Chapter 8. The full set of unsteady equations with pseudo time would be

$$\begin{bmatrix} c^* \cdot M^* & M^* \\ I & 0 \end{bmatrix} \frac{\partial}{\partial t^*} \begin{Bmatrix} \vec{u} \\ \vec{v} \end{Bmatrix} + \begin{bmatrix} c \cdot M & M \\ I & 0 \end{bmatrix} \frac{\partial}{\partial t} \begin{Bmatrix} \vec{u} \\ \vec{v} \end{Bmatrix} + \begin{bmatrix} K & 0 \\ 0 & -I \end{bmatrix} \begin{Bmatrix} \vec{u} \\ \vec{v} \end{Bmatrix} = R\left(\begin{Bmatrix} \vec{u} \\ \vec{v} \end{Bmatrix} \right) \tag{9-8}$$

where M does not necessarily equal M^* and so on. If a steady solution is needed, the pseudo time terms are kept and the other time terms are dropped.

For a steady problem in pseudo time, the first order system of equations

$$\begin{bmatrix} c \cdot M & M \\ I & 0 \end{bmatrix} \frac{\partial}{\partial t^*} \begin{Bmatrix} \vec{u} \\ \vec{v} \end{Bmatrix} + \begin{bmatrix} K & 0 \\ 0 & -I \end{bmatrix} \begin{Bmatrix} \vec{u} \\ \vec{v} \end{Bmatrix} = \begin{Bmatrix} \vec{f}_{ext} \\ \vec{0} \end{Bmatrix} \tag{9-9}$$

may be written in the standard pseudo time form

$$Q \frac{\partial z}{\partial t^*} + K^+ z = R(z) \tag{9-10}$$

All the pseudo time artificial terms are contained in the matrix Q, z is a vector of the dependent variables, $R(z)$ is the residual and $K^+ = \begin{bmatrix} K & 0 \\ 0 & -I \end{bmatrix}$.

In the coupled solution process, a matrix-free restarted GMRES solver is used to create the search directions for the solution vector. The matrix (though never formed) is held fixed throughout the linear (inner) solve. The system has the form

$$\left[\frac{1}{\Delta t^*}\begin{bmatrix} c^* \cdot M^* & M^* \\ I & 0 \end{bmatrix} + \begin{bmatrix} K & 0 \\ 0 & -I \end{bmatrix}\right] \cdot \begin{Bmatrix} \Delta\vec{u} \\ \Delta\vec{v} \end{Bmatrix} = -\begin{bmatrix} K & 0 \\ 0 & 0 \end{bmatrix}\begin{Bmatrix} \vec{u} \\ \vec{v} \end{Bmatrix} \quad \textbf{(9-11)}$$

The velocity is obviously zero for a steady problem but the terms are kept on the left hand side to feed to the ALE fluid equations during tightly coupled (simultaneous fluid-structure-grid) solution transients. They are not passed during a loosely coupled (cycling through fluid, structure, grid) solution process but are still necessary when using pseudo time for the first-order-in-time system of equations for the physical structure. The identity term in the stiffness matrix must be dropped on the right-hand side since it was an artifact of the unsteady (rather than pseudo time) derivation. The term would be retained along with the other unsteady terms on both the left and right hand sides for truly time-varying problems.

During the linear solution of a tightly coupled approach to the fluid-structure system, the displacement of the physical structure updates the Dirichlet boundary conditions for the fluid grid's pseudo structure with each new GMRES search vector. For the loosely coupled approach, these boundary conditions are held fixed during the inner iterations of the linear solver and updated with each outer iteration. Each approach should ultimately converge to the same solution. The intent of the current effort is to see if the tightly coupled solution process can reach convergence faster than the loosely coupled process can.

CHAPTER 10

FLUID-STRUCTURE COUPLING

In 1955, [Bisplinghoff, Ashley & Halfman, 1955] and [Fung, 1955] redefined the term *aeroelasticity* to include interactions between aerodynamic, elastic, and inertial forces. They subdivided the field into dynamic and static aeroelastic phenomena depending on the strength and relevance of the inertial forces. This was not a departure from earlier thinking so much as the addition of the dynamic component. The concepts and terminology are directly applicable to all fluid-structure interactions, regardless of whether the fluid is air or any other fluid. In 1962, [Bisplinghoff, and Ashley, 1962] extended the analysis further and created the landmark work in aeroelasticity theory. The basic theory has not changed since but a great deal of effort has been expended on developing the rules, tools, and techniques for specific aeroelasticity applications. Examples include fixed and rotary wing aircraft; turbomachinery; cables, bridges, buildings, stacks and chimneys, and other civil engineering applications; flexible aerospace applications; and now optimal design [Hajela, 1986].

Continuum Interactions

[Bisplinghoff, Ashley & Halfman, 1955] used a diagram (Figure 10-1) referred to as the "aeroelastic triangle of forces" to indicate the varying degrees of influence of the aerodynamic, elastic, and inertial forces. The forces change over time and with actual fluid loading conditions. In a typical aircraft design, the structural aspects such as wing stiffness and inertial properties are mostly fixed but must operate over a wide range of fluid loading conditions. On the other hand, structural stiffness and inertial properties may also change as fuel is removed from wing bladders or temperatures change. Wing surface temperatures on the runway may exceed 100 degrees but then drop below zero at altitude; or they may exceed 500 degrees due to aerodynamic friction and compressibility at very high speeds. Thus, an aeroelastic assessment is a point assessment at actual physical conditions and does not lend itself to fully non-dimensional treatment.

The "close-up" details of fluid-structure interaction are intuitive and well known but system behavior is elusive. At the conceptual level, the structure influences the fluid through the physical shape of

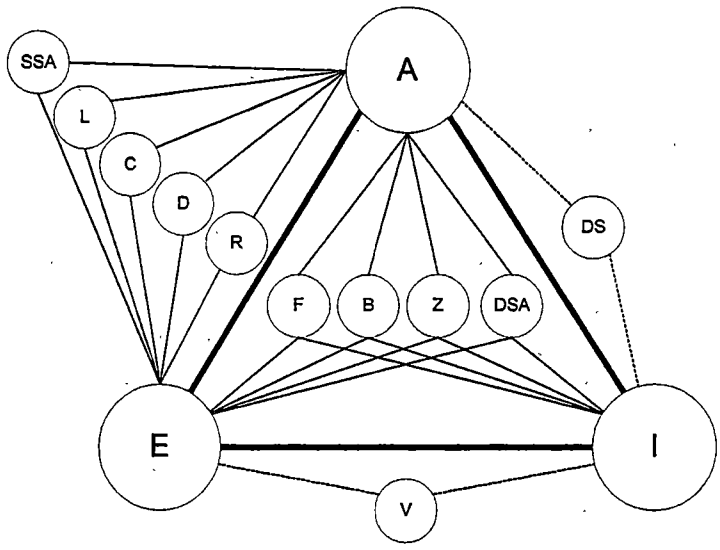


Figure 10-1: Aeroelastic Triangle of Forces [Bisplinghoff, Ashley & Halfman, 1955]

	Primary forces		Aeroelastic phenomena
A:	Aerodynamic force	SSA:	Aeroelastic effects on static stability
E:	Elastic force	L:	Load distribution
I:	Inertial force	C:	Control effectiveness
		D:	Divergence
		R:	Control system reversal
	Related Fields	F:	Flutter
V:	Mechanical vibrations	B:	Buffeting
DS:	Dynamic stability (rigid body)	Z:	Dynamic response
		DSA:	Aeroelastic effects on dynamic stability

Aeroelastic effects on stability (SA) - influence of elastic deformations of the structure on dynamic and static airplane stability

Load distribution (L) - influence of elastic deformations of the structure on the distribution of aerodynamic pressures over the structure

Control effectiveness (C) - influence of elastic deformations of the structure on the controllability of an airplane

Divergence (D) - a static instability of a lifting surface of an aircraft in flight, at a speed called the divergence speed, where the elasticity of the lifting surface plays an essential role in the instability

Control system reversal (R) - a condition occurring in flight, at a speed called the control reversal speed, at which the intended effects of displacing a given component of the control system are completely nullified by elastic deformations of the structure

Flutter (F) - a dynamic instability occurring in an aircraft in flight, at a speed called the flutter speed, where the elasticity of the structure plays an essential part in the instability

Buffeting (B) - transient vibrations of aircraft structural components due to aerodynamic impulses produced by the wake behind wings, nacelles, fuselage pods, or other components of the airplane

Dynamic response (Z) - transient response of aircraft structural components produced by rapidly applied loads due to gusts, landing, gun reactions, abrupt control motions, moving shock waves, or other dynamic loads

the structure and the velocity of moving surfaces. Likewise, the fluid influences the structure through the load distribution where the fluid load vector may be broken into the pressure load normal to the surface and the frictional load tangent to the surface. The magnitude of the fluid friction force per unit area is typically insignificant relative to the pressure force per unit area and is often ignored. However, the aggregate (rather than per-unit-area) friction force over the surface (e.g. the wing of a Boeing 747) can be significant.. Also, the nonlinear effects of viscosity on the fluid pressure distribution through boundary layer separation, shock location, wake behavior, etc can drastically alter the fluid behavior. In other words, many fluid-structure interaction problems may be adequately solved with inviscid fluid assumptions but fluid viscosity should not be ignored without some preliminary analysis. The test cases used in this research effort were steady state with subsonic internal flow and a favorable pressure gradient so fluid viscosity was excluded.

In an elastic continuum, disturbances propagate as waves until they are diffused or they encounter some kind of boundary. Both the fluid and the structure are elastic media and the fluid-structure interface represents a physical boundary. Elastic waves in the fluid and structural continuum respond to the boundary (Figure 10-2) according to the physical laws as reflected or refracted waves [Brillouin, 1953] [Brillouin, 1960] [Redwood, 1960] [Breskhovskikh and Goncharov, 1994]. Just as bending light waves make objects appear to bend through a glass of water, elastic waves bend (refract) if the media on the sides of the boundary have different properties. Since any pressure field may be described by its wave components through Fourier analysis, the application extends beyond acoustic problems.

Snell's law of refraction applies to elastic waves as well as electromagnetic waves and describes the amount of wave bending relative to the surface normal with the relation

$$V_2 \sin\theta_1 = V_1 \sin\theta_2 \tag{10-1}$$

or

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{V_1}{V_2} \tag{10-2}$$

The change in phase velocity of an elastic disturbance as it strikes the boundary determines the change in direction of the disturbance propagation. If the change in velocity is large, there is a critical angle for the incident wave beyond which the wave will be totally reflected. During this total reflection, the field amplitude in the substance with the faster phase velocity decays exponentially away from the surface. The

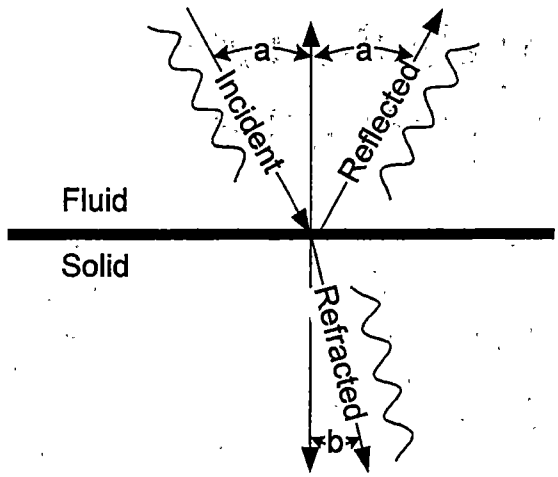


Figure 10-2: Fluid-Solid Interface

constant phase surfaces are perpendicular to the boundary and travel with the trace velocity parallel to the boundary [Ingard, 1988]. These are essentially Rayleigh or surface waves.

As an example, suppose the actual materials at the fluid-structure interface are air and steel. If the air has an acoustic (phase) velocity of $V_{air} = 330\frac{m}{s}$ and the steel has $V_{steel} = 5100\frac{m}{s}$, then the critical angle is given by

$$\sin\theta_{\substack{air \\ critical}} = \frac{V_{air}}{V_{steel}} = \frac{330}{5100} \longrightarrow \theta_{\substack{air \\ critical}} = 3.7 \text{ degrees} \qquad \textbf{(10-3)}$$

If the disturbance strikes the surface at an angle greater than 3.7 degrees, the disturbance will be reflected and may have a trace component parallel to the surface. In the air-steel example, if the fluid disturbance strikes the surface 1° off the normal, the disturbance will penetrate the solid at 15.6^o.

An elastic wave in air is a pressure disturbance acting normal to the surface. However, this is only a component of the total force vector and the tangential friction force must be considered (Figure 10-3). The total force vector will interact with the surface according to the size of the components. The angle of the total force vector relative to the normal is given by

$$\tan\theta_{air} = \frac{f_{\tau_{wall}}}{f_{pressure}} = \frac{\frac{1}{2}\rho u^2 C_f A}{PA} = \frac{\gamma M^2 C_f}{2} \qquad \textbf{(10-4)}$$

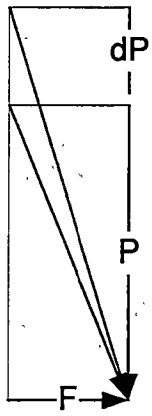


Figure 10-3: Fluid Force Vector

Using the notional critical angle above and a skin friction coefficient $C_f = .005$,

$$\tan 3.7 = \frac{\gamma M^2 (.005)}{2} \longrightarrow M \leq 4.3$$

the skin friction does not cause the force vector to exceed the critical angle below $Mach \leq 4.3$. The same type of analysis can be used to show a refraction angle of 10° (arbitrary) corresponds to an incident angle of $.64^o$. Mach numbers below $Mach \leq 1.8$ are within this incident angle for the example parameters.

Viscous forces may be important in determining the aeroelastic effects on *static* stability. However, viscous affects on *dynamic* stability will most likely be felt through the nonlinear relationships between viscosity and pressure in the fluid equations. This will likely manifest itself through boundary layer transitions, boundary layer separation and unsteady wakes creating a time dependent pressure distribution.

It is also interesting to note the influence of a fluid-structure boundary on the time it takes for a fluid "signal" to reach a new location. The solid does not change the time of flight in the fluid but provides an alternative path for the signal to follow (Figure 10-4). Using the material properties in the example above let the disturbance and receiver be $10cm$ off the surface and $1m$ apart. The direct signal path through the fluid only will take $tof_{fluid} = \frac{1}{330} = 3 \times 10^{-3} s$. However, the signal can take the composite

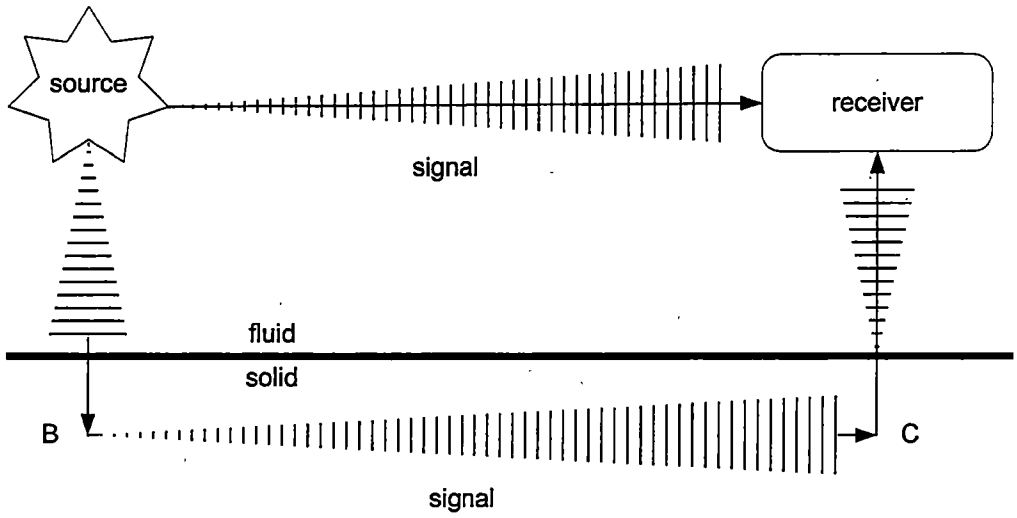


Figure 10-4: Signal Paths

path through the structure in $tof_{composite} = \frac{.1}{330} + \frac{1}{5100} + \frac{.1}{330} = 8 \times 10^{-4} s$. The signal following the composite path arrives almost 75% faster than the signal following the direct path. Thus, it is possible for the effects of a disturbance to be felt at other locations before the fluid signal has arrived. This can provide the equivalent of a feed-forward or anticipatory system response. It may be most evident in structural response to large disturbances such as blast waves but could be important in triggering boundary layer separation for even small disturbances.

The structure may actually be modeled using a variety of techniques depending on the application and the degree of detail needed. The simplest models are rigid body models allowing motion or rotation along one or two axes. The motion is resisted by elastic springs with frictional type dampers and the mass distribution is contained in the moments of inertia [Cebral and Lohner, 1997] [Alonso and Jameson, 1994] [Alonso, Martenilli, and Jameson, 1995]. An immediate extension of this is a flexible body model connected by springs and dampers. This method is not often used however, as transonic flow solutions are highly sensitive to the flexibility of the aerodynamic surfaces so the time accurate deformed shape must be determined more carefully [MacMurdy, Guruswamy, Kapania, 1994].

More sophisticated models use special elements for different regions of the structure such as beams, plates, shells, etc. The special elements are typically derived from the Navier equations for solid

mechanics with simplifying assumptions or specific boundary conditions. These models are the most common for modeling structures and may even incorporate specialize elements for particular applications such as wing-box structures [MacMurdy, Guruswamy, Kapania, 1994]. This is probably the best approach for complicated structures and the impact of individual spars or ribs in a wing-box may be determined. It also allows for non-linear and anisotropic structures such as composite laminates.

A common technique used for dynamic structural equations is to build the complete static model of the structure using plate and beam type elements and determine the eigenvalues and eigenvectors of the system. Since most fluid-structure interactions are at relatively low frequencies, only the first few structural system modes are needed. A typical modal equation model may use the first five eigenvectors of the static system to capture the relevant harmonics and provide an accurate representation of the lower frequency modes [Guruswamy, 1989] [Yang and Lohner, 1997]. However, the technique must be limited to linear deformations for the eigenvector modes to be valid [MacMurdy, Guruswamy, Kapania, 1994].

Finally, full finite element models based on the Navier equations for solid mechanics are also used [Byun and Guruswamy, 1994] [Bendiksen, 1991] [Ray, Wren, and Tezduyar, 1997]. This is the approach taken in the current effort.

Equivalence of Two and Three Field Methods

A coupled fluid-structure system physically has only two sets of field equations (the governing fluid equations and the governing structural equations). However, it will have three sets of field equations (fluid, structure, fluid grid motion) in a discrete implementation.

An uncoupled fluid-structure system may be written

$$\begin{bmatrix} fluid & 0 \\ 0 & structure \end{bmatrix} \tag{10-5}$$

This has a characteristic equation

$$\lambda^2 - (fluid + structure)\lambda + fluid \cdot structure = 0 \tag{10-6}$$

and

$$\lambda = \lambda(fluid), \lambda(structure) \tag{10-7}$$

This is an unorthodox statement of the lemma from [Golub & Van Loan, p. 333] where

$$T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix}$$
$$\lambda(T) = \lambda(T_{11}) \bigcup \lambda(T_{22}) \tag{10-8}$$

The lemma shows the eigenvalue spectrum of T to be the union of the spectrums of the diagonal submatrices. This also immediately leads to some notion of the possible ill-conditioning of a combined fluid-structure problem as the condition may be stated as a ratio of the largest and smallest eigenvalues. The condition number of the fluid-structure problem may be estimated as

$$\kappa = \frac{\max\left(\lambda(structure)\right)}{\min\left(\left|\lambda(fluid)\right|\right)} \tag{10-9}$$

assuming the eigenvalues of the structure are much larger (i.e. faster) than those of the fluid. This notion is shown graphically on the complex plane as a parabola of the (2nd order) characteristic equation (Figure 10-5). All the eigenvalues may lie on the real axis and the spectrum of the fluid and structure may overlap. The condition of the system may be improved by changing the shape of the parabola or translating it up and to the left so the eigenvalue spectrums are closer together.

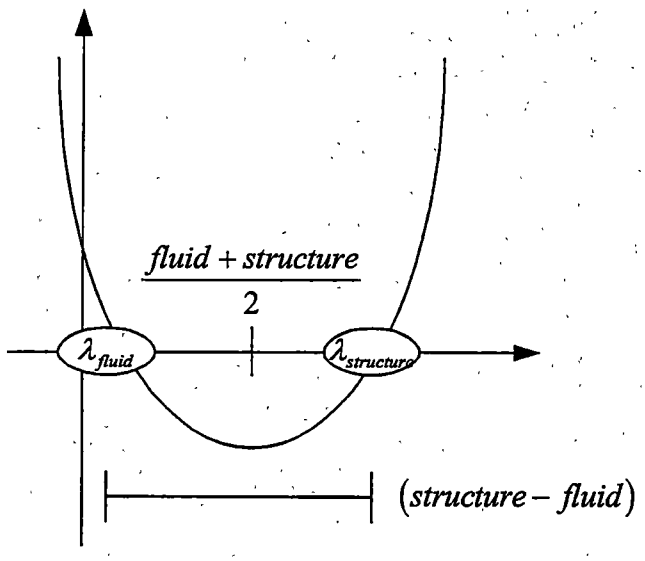


Figure 10-5: Fluid-Structure Eigenvalue Clusters

If the fluid and structure system are not coupled, one set of field equations may be scaled without affecting the other field and the overall system condition reduced dramatically. For example, let $s1$ and $s2$ be scalars or diagonal matrices used to numerically scale the fluid and structure field equations as

$$\begin{bmatrix} s1 & 0 \\ 0 & s2 \end{bmatrix} \cdot \begin{bmatrix} fluid & 0 \\ 0 & structure \end{bmatrix} = \begin{bmatrix} s1 \cdot fluid & 0 \\ 0 & s2 \cdot structure \end{bmatrix} \tag{10-10}$$

and let

$$\begin{aligned} s1 &= 1 \\ s2 &= 10^{-8} \end{aligned} \tag{10-11}$$

Now

$$\begin{bmatrix} fluid & 0 \\ 0 & structure \times 10^{-8} \end{bmatrix} \tag{10-12}$$

leads to the new system condition number

$$\kappa = \frac{\max\left(\lambda(structure)\right) \times 10^{-8}}{\min\left(\left|\lambda(fluid)\right|\right)} \tag{10-13}$$

or

$$\kappa = \frac{\max\left(\lambda(structure)\right)}{\min\left(\left|\lambda(fluid)\right|\right) \times 10^{8}} \tag{10-14}$$

Again, this is shown graphically as a parabola in the characteristic equation. Here, the parabola has been modified by the scaling parameters to bring the decoupled eigenvalue spectrums closer together (Figure 10-6).

An alternative for the uncoupled case is to take advantage of an eigenvalue shift technique given by

$$\begin{bmatrix} Fluid & 0 \\ 0 & 1 - s2 \cdot Structure \end{bmatrix} \tag{10-15}$$

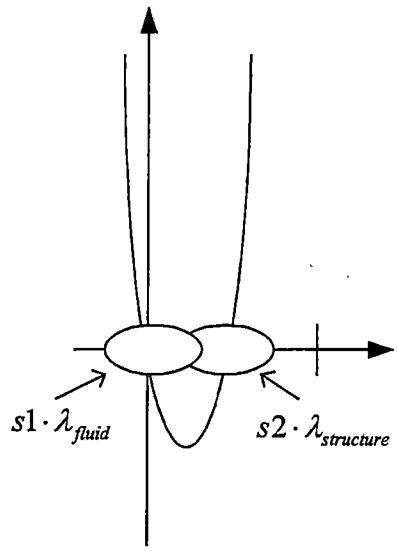


Figure 10-6: Scaled Fluid-Structure Eigenvalue Clusters

In this case, the new eigenvalue spectrum of the structure, μ_i, is related to the original structure spectrum by

$$\lambda_i = \frac{(1-\mu_i)}{s2} \tag{10-16}$$

If a typical eigenvalue in the original spectrum is $\lambda = 15{,}000$ and $s2 = 10^{-8}$, the new shifted and scaled eigenvalue is $\mu = .99985$. Scaling alone gives $\mu = 1.5 \times 10^{-4}$.

The above is an extreme case where two fields are solved simultaneously for convenience but there is no relation or interaction between the two. Many fluid-structure problems are loosely coupled or place greater emphasis on either the fluid flow or the structural motions. The normal case of a fluid-only problem may be viewed as a fluid-structure problem where the structure is assumed rigid with infinitely fast eigenvalues.

$$\begin{bmatrix} fluid & shape \\ 0 & structure \end{bmatrix}$$
$$\lambda = \lambda(fluid) \cup \lambda(structure) \tag{10-17}$$

The second set of equations is not considered in the calculation, which is the equivalent of scaling through multiplying by zero. The condition number of the system is just the condition number of the fluid equation

set. Likewise, a two step calculation to determine the pressure distribution on a structure may start with the above and add the pressure coupling for a fixed fluid field to determine a structural response.

$$\begin{bmatrix} fluid & shape \\ 0 & structure \end{bmatrix} \longrightarrow \begin{bmatrix} fluid & 0 \\ pressure & structure \end{bmatrix} \qquad \lambda = \lambda(fluid) \cup \lambda(structure) \tag{10-18}$$

If the fluid and structure are fully coupled, the shape of the structure and the pressure distribution (inviscid assumption) of the fluid are included and the system of equations becomes

$$\begin{bmatrix} fluid & shape \\ pressure & structure \end{bmatrix} \tag{10-19}$$

The fully coupled system is normally the case of interest but the system is usually decomposed to ease numerical calculations or as a natural part of a design process. If the problem is decomposed for numerical reasons, the off-diagonal elements, *pressure* and *shape*, become boundary conditions for the single field equation sets (*structure* or *fluid*, respectively). In the design process, the fluid domain shape may be a design variable (such as a wing shape) without any knowledge of its structure. The fluid flow is calculated over the given shape to determine a pressure distribution. The pressure distribution is then used as an input to a structural calculation to determine the new shape. This process is repeated to convergence. From the system of equations perspective represented by equation **(10-19)**, this is a domain or matrix decomposition.

If the system is not decomposed, this fully coupled system now has a new characteristic equation, a new eigenvalue spectrum, and a new condition number.

$$\lambda^2 - (fluid + structure)\lambda + fluid \cdot structure - pressure \cdot shape = 0 \tag{10-20}$$

and

$$\lambda \longrightarrow \frac{fluid + structure \pm \sqrt{(fluid - structure)^2 + 4 \cdot pressure \cdot shape}}{2} \tag{10-21}$$

If the *pressure* term or *shape* term is eliminated, this returns the spectrum to the union of the fluid spectrum and the structure spectrum given by equations **(10-6)** and **(10-7)**.

The characteristic equation gives a hint of the impact of the coupling on the eigenvalue spectrum. Returning to the parabola, the shape of the parabola is determined by the coefficients of λ. The offset is determined by the additional term $pressure \cdot shape$. This indicates the coupling terms cause the parabola to be translated downward by an amount $pressure \cdot shape$ but does not change the location of the vertex along the real axis. The effect is to widen the distance between the real axis crossings indicating a broader eigenvalue spectrum. Also note it is possible for the low end of the spectrum to become more negative. Again, this is shown graphically in the parabola in Figure 10-7.

Simple scaling of the equations may bring the clusters of eigenvalues together again even with the $pressure \cdot shape$ terms. The system of equations becomes

$$\begin{bmatrix} s1 \cdot fluid & s1 \cdot shape \\ s2 \cdot pressure & s2 \cdot structure \end{bmatrix} \tag{10-22}$$

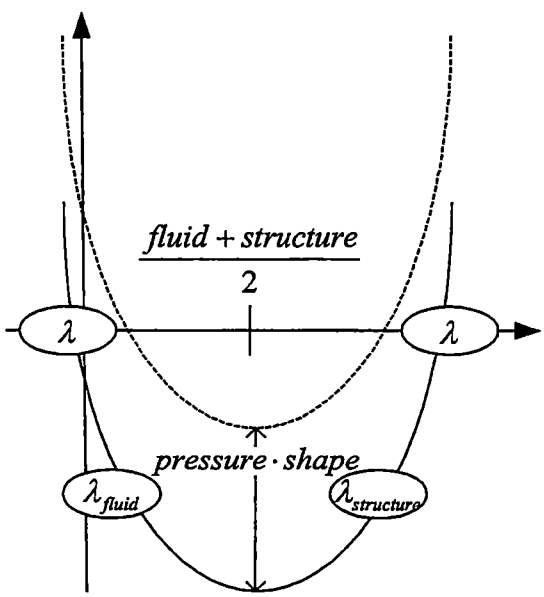


Figure 10-7: Fluid-Structure Eigenvalue Cluster Components

In the uncoupled cases, the eigenvalue spectrum is simply the union of the individual spectrums for the fluid and structure. This does not say either of the two spectrums is well conditioned; only that the coupled system condition will likely be much worse than either of the individual spectrums. Thus, the key to good conditioning for coupled fluid-structure interaction is to start with a well conditioned fluid spectrum and a well conditioned structure spectrum and scale the systems so the eigenvalues are of the same order of magnitude.

Finally, the fluid-structure interaction problem may be expressed as a two or a three field problem. In the continuum sense, only the fluid and structure actually exist. However, in discrete problems, the grid may also be considered as a third set of field equations [Farhat, et.al., 1995]. This is consistent with the continuum approach if the continuum boundary interactions are considered.

The three field discrete set of equations may be written

$$\begin{bmatrix} fluid & 0 & velocity_{grid} \\ pressure & structure & 0 \\ 0 & boundary_{\substack{fluid- \\ structure}} & grid \end{bmatrix} \begin{Bmatrix} x_{fluid} \\ x_{struct} \\ x_{grid} \end{Bmatrix} = \begin{Bmatrix} b_{fluid} \\ b_{struct} \\ b_{grid} \end{Bmatrix} \tag{10-25}$$

The three field equation set is equivalent to the two field equation set but expands on the actual form and content of the *shape* submatrix in

$$\begin{bmatrix} fluid & shape \\ pressure & structure \end{bmatrix} \tag{10-26}$$

The equivalence can be shown by the Schur complement [Golub and Van Loan, p 103] of the three field matrix. This process is known as static condensation in the finite element literature [Baker, 1983].

$$\left[\begin{array}{cc:c} fluid & 0 & velocity \\ pressure & structure & 0 \\ \hdashline 0 & boundary & grid \end{array}\right] \left\{\begin{array}{c} x_{fluid} \\ x_{struct} \\ \hdashline x_{grid} \end{array}\right\} = \left\{\begin{array}{c} b_{fluid} \\ b_{struct} \\ \hdashline b_{grid} \end{array}\right\} \tag{10-27}$$

Let the subdivided matrix be written as

$$\left[\begin{array}{c:c} FS & FG \\ \hdashline SG & GG \end{array}\right] \left\{\begin{array}{c} x_{FS} \\ \hdashline x_{GG} \end{array}\right\} = \left\{\begin{array}{c} b_{FS} \\ \hdashline b_{GG} \end{array}\right\} \tag{10-28}$$

where

$$FS = \begin{bmatrix} fluid & 0 \\ pressure & structure \end{bmatrix} \tag{10-29}$$

Solving the second set of block equations in **(10-28)**,

$$x_{GG} = GG^{-1}\left(b_{GG} - SGx_{FS}\right) \tag{10-30}$$

Substitute into the first set of block equations in **(10-28)**

$$\begin{aligned} FSx_{FS} + FGx_{GG} &= b_{FS} \\ FSx_{FS} + FG \cdot GG^{-1}\left(b_{GG} - SGx_{FS}\right) &= b_{FS} \\ \left[FS - FG \cdot GG^{-1} \cdot SG\right]x_{FS} &= b_{FS} - FG \cdot GG^{-1}b_{GG} \end{aligned} \tag{10-31}$$

Returning to the original submatrices

$$\begin{aligned} &\left[\begin{bmatrix} fluid & 0 \\ pressure & structure \end{bmatrix} - \begin{bmatrix} 0 & velocity \cdot grid^{-1} \cdot boundary \\ 0 & 0 \end{bmatrix}\right]\begin{Bmatrix} x_{fluid} \\ x_{struct} \end{Bmatrix} \\ &= \begin{Bmatrix} b_{fluid} \\ b_{struct} \end{Bmatrix} - \begin{Bmatrix} velocity \cdot grid^{-1}b_{grid} \\ 0 \end{Bmatrix} \end{aligned} \tag{10-32}$$

or

$$\begin{bmatrix} fluid & velocity \cdot grid^{-1} \cdot boundary \\ pressure & structure \end{bmatrix}\begin{Bmatrix} x_{fluid} \\ x_{struct} \end{Bmatrix} = \begin{Bmatrix} b_{fluid} - velocity \cdot grid^{-1}b_{grid} \\ b_{struct} \end{Bmatrix} \tag{10-33}$$

Equation **(10-33)** shows the two and three field problem statements are the same with all the structural feedback, static or dynamic, residing in the $shape$ term and implemented through grid motion as $velocity \cdot grid^{-1} \cdot boundary$.

[Lesoinne and Farhat, 1997] extend the three field problem even further by forming a linearized matrix N about the solution such that $\dot{q} = Nq$ where q is a vector of the all the fluid, structure and grid variables. From this, they use the inverse power series iteration to find the lowest eigenvalues of the coupled system and predict the flutter boundary of a flat panel in a supersonic airstream.

Discrete Interface and Interactions

In a discrete fluid-structure interaction problem, there is no requirement for the fluid and structure to have the same grid nodes at the interface. Some practitioners do match the nodes [Ray, Wren, and Tezduyar, 1997] [Farhat and Lin, 1990] but the fluid's grid is much finer than the structure's grid in most applications (Figure 10-9). This leads to losses of detail and load conservation when projecting results from a fine to a coarse grid. Likewise, the fine fluid grid must somehow be given the correct shape or deformation from a much coarser structural grid. The implications and techniques for dealing with the mismatch is a subject of current debate. Research to date has shown the problem is not trivial for flexible bodies and improper treatment can foul the coupled solution [MacMurdy, Guruswamy, and Kapania, 1994]. [Lesoinne and Farhat, 1996] also show the spurious modes that may be generated by not obeying geometric conservation laws for the displacement and velocity of the fluid grid at the interface. These inaccuracies in the energy exchange between the fluid and structure are due to discontinuities in the velocity of the fluid and structure at the interface.

[Smith, et.al., 1996] explore the many possibilities for interpolation and extrapolation between grids of differing densities. These are used for load projection from the fluid to the solid as well as shape changes from the solid back to the fluid. They show the strengths and weaknesses of various spline and mapping techniques and identify some definite winners and losers along with an excellent parametric study of the fluid and structure results using the different methods. The commonly used infinite-plate spline is not a strong contender but the thin-plate spline, multiquadrics, and non-uniform B-spline (NUBS) all work well.

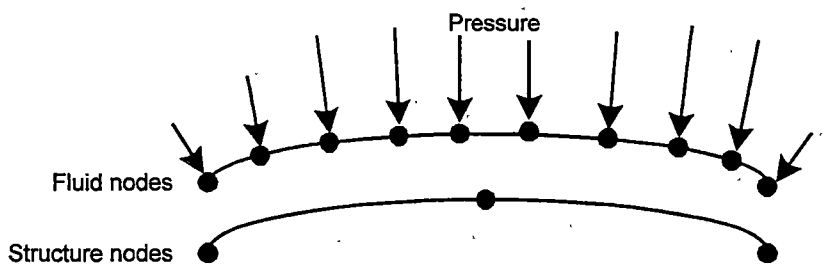


Figure 10-9: Mismatched Node Density

[MacMurdy, Guruswamy, and Kapania, 1994] use three techniques and introduce the concept of local conservation in the process. They first use a simple nearest-node interpolation to transfer forces but acknowledge that it looses the "moment arm" for pressure forces away from the structural node. They next use a local conservation scheme for the bending and twist moments in addition to the normal force. This way, the resultant forces generated by the pressure distribution are treated as a normal force plus a bending and a twist moment at each structural node. In the global assembly, the set of force conservative equations is ill-conditioned. They finally develop a twist compensation for the simple scheme and ignore the bending errors to avoid the ill-conditioning of the two moment approach.

[Cebral and Lohner, 1997] summarize several approaches and identify the need for a conservative projection of the total load. They define a scheme as conservative if the total load calculated on the fluid and solid mesh are exactly the same. This is not generally true for interpolation schemes. They point out the conservation properties are true for the virtual grid method developed by [Guruswamy and Byun, 1993] but claim the additional work for the virtual grid is unnecessary. They ultimately use a weak statement formulation on the definition of the load conservation property to enforce the requirement. This is

$$\int_{\Gamma} W^i F_{fluid} d\Gamma = \int_{\Gamma} V^j F_{solid} d\Gamma \qquad \textbf{(10-34)}$$

where W and V are weight functions and both integrals are evaluated over the same region. Deformation transfer is also addressed. A "distance vector" is used to locate the analytic fluid surface relative to a plate structural model. [Cebral and Lohner, 1997] show the advantages of using interpolated node normals rather than face normals to maintain smooth aerodynamic surfaces even if the fluid and solid have very different mesh densities.

[Farhat, et.al., 1995] also use a weighted integration over the structural elements but with a slightly different implementation. They develop a series of Gauss points on the surface of the solid element and interpolate fluid pressure to the Gauss points. If a high pressure gradient is present, more Gauss points are added. Deformation is interpolated across structural elements back to the fluid grid nodes.

In the current effort, the fluid and solid meshes are the same density and the nodes are glued together at the surface (Figure 10-10). Load projection and deformation tracking is straightforward.

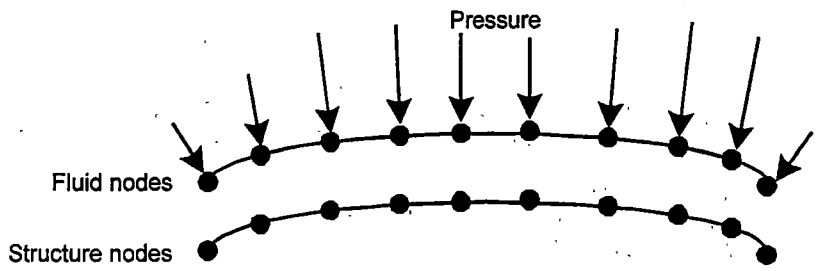


Figure 10-10: Matched Nodes

Discrete Time Synchronization

A final topic is the treatment of timing in the fluid and structure updates. If the two are solved simultaneously [Bendiksen, 1991], there is no difference in the timing. This may lead to over-solving of one or the other field as the time scales are often different. For example, many Navier-Stokes codes have a tendency for the inviscid pressure field to set up long before the full boundary layer is developed. In this case, the structure could be idle while the fluid is updated many times. Also, even for inviscid problems, fluid phenomena such as shock motion may be very fast when compared to the structural response.

[Cebral and Lohner, 1997] use different time steps for the fluid and structure and present a technique to interpolate boundary conditions between time levels. They show the problems associated with extrapolating time and present a third order Hermitian polynomial for interpolating.

[Farhat, et.al., 1995] use an explicit scheme for the fluid and an implicit for the structure. They develop a series of subcycling strategies and address stability, speed, and accuracy. The logic is that many aeroelastic problems seek the perturbation response around a steady fluid-structure system. Stable responses are typically desired but diverging responses are extremely important for determining stability bounds of aeroelastic systems. Their general logic is to initially form a steady fluid-structure solution assuming a perfectly rigid or heavily damped structure. Then, the structure is perturbed by allowing it to respond to the fluid load or giving it some initial displacement. The fluid grid is updated to conform to the new shape of the flow domain and the fluid flow is calculated on the new domain. Next, the new fluid load distribution is applied to the structure and the process is repeated. Figure 10-11 shows a time varying computation process where the fluid is updated several time steps for every structure

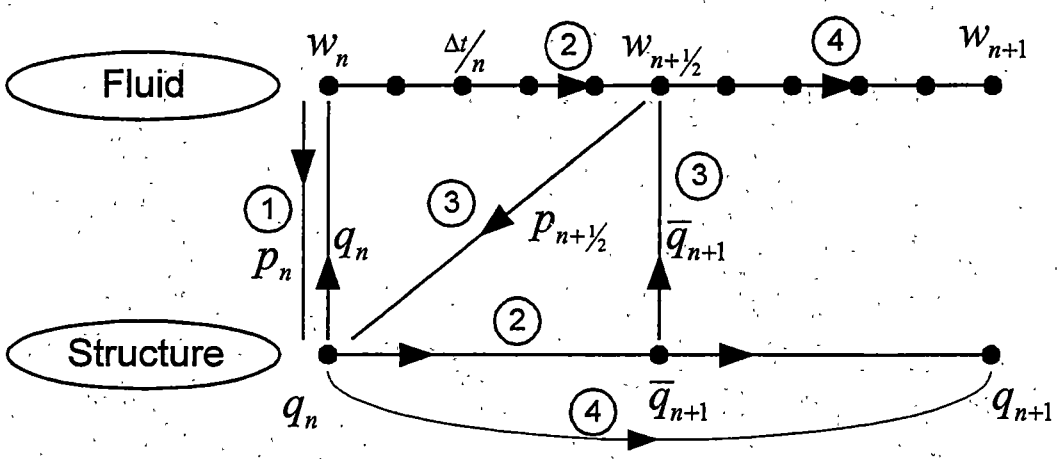


Figure 10-11: Fluid-Structure Update Strategy

update. In this particular strategy, $\overline{q}_{n+1}$ is the average structure displacement vector and w_n is the fluid state vector. Information is passed (1) between the fluid and structure using current information, (2) the fluid is stepped forward in time to $n+\frac{1}{2}$ (explicit solvers may require multiple time steps) and the structure is taken in a single predictor step to $n+1$, (3) the average structure displacement is passed to the fluid and the latest fluid pressure is passed to the structure at time n, (4) the fluid is stepped forward in time to $n+1$ and the structure is recomputed in a corrector step from n to $n+1$.

In this research project, only pseudo time was used to compute a coupled system to a steady state. In the tightly coupled solution process, the fluid, structure, and fluid grid were all calculated simultaneously so each field has the latest information from each of the other fields. In the loosely coupled solution process:

1. fluid flow was calculated on the fixed fluid grid
2. the structural response to the updated fluid pressure distribution was calculated
3. the fluid grid was updated to conform to the latest structural deformation
4. returned to (1) and the process was repeated to convergence

Different pseudo time step sizes were used in the fluid and the solid. This is roughly the equivalent of local time stepping in fluid-only calculations.

CHAPTER 11

ANALYSIS OF NUMERICAL SOLVER RESULTS

Directional Derivative Step Size Changes

Experiences with the forward difference version of the directional derivative in both fluid and structure solution procedures showed that the default h, fixed at 10^{-7}, was a good overall choice. The solution procedure was relatively insensitive to step size changes up to several orders of magnitude. In the *structure only* calculations, the residuals associated with the equations for the x and y directions were of the same order of magnitude and not scaled. For *fluid only* calculations, the residuals for the continuity equation were scaled by 1×10^{3} and the energy equation by 2×10^{-4}. The step size did not become an issue for the *fluid only* calculations even though the residuals initially spanned six orders of magnitude. The combined fluid/structure problems required residual scaling on the order of nine orders of magnitude with the already scaled fluid equations. Multiplying the fluid residual by 10^{9} moved the truncation point for the finite precision floating point number by nearly 30 bits ($2^{30}\approx1.07\times10^{9}$) or equivalently changed the least significant bit to a value of 2.22×10^{-7} and $h_{ideal}=4.7\times10^{-4}$.

Numerical experiments were conducted using the fluid flow nozzle case (Chapter 12) with the baseline scaling and the entire fluid residual multiplied (scaled) by 2×10^{8}. The baseline and the scaled residual were both tested with step sizes at $h=1\times10^{-7}$ and $h=2\times10^{-4}$. Since h_{ideal} (see Chapter 3) for the baseline is close to $h=1\times10^{-7}$, setting $h=2\times10^{-4}$ increases the step size three orders of magnitude above the ideal. For the scaled residual, the h_{ideal} is closer to $h=2\times10^{-4}$. Thus, setting $h=1\times10^{-7}$ reduces the step size to three orders of magnitude below the ideal. Table 11-1 summarizes the relevant cases tested for *fluid only* calculations with the relative and absolute stopping criteria described in Chapter 3.

All four cases converged to the exact same solution. Case 1 and case 2 followed the exact same convergence paths and the nonlinear residuals matched to at least 14 significant digits. Case 3 and case 4

Table 11-1: Effect of Residual Scaling on Directional Derivative Step Size

	Step size	Scaling	Iterations	Relative	Absolute
Case 1	1x10^-7	1	9	1x10^-6	1x10^-6
Case 2	2x10^-4	1	9	1x10^-6	1x10^-6
Case 3	2x10^-7	2x10^8	10	1x10^-6	1x10^-6
Case 4	2x10^-4	2x10^8	10	1x10^-6	1x10^-6

also matched each other to the same degree. The residual convergence paths for 1 and 2 are shown in Figure 11-1 with symbols and the paths for 3 and 4 are shown without symbols.

Cases 3and 4 had a very similar convergence path to cases 1and 2 with only a slight variation starting at Newton step five (Figure 11-2). The converged nonlinear residuals were rescaled back to the baseline and matched the baseline to 6 significant digits. The biggest difference between the case 1/2 and case 3/4 solutions had to do with termination criteria. In each case, the termination criteria were set to 1×10^{-6} for both the relative residual reduction and the absolute residual thresholds. All cases terminated against the relative threshold. The scaled cases needed one additional nonlinear iteration due to the slightly lower convergence in Newton steps five through nine. Figure 11-2 shows how the rescaled residuals for cases 3 and 4 closely follow cases 1 and 2.

The conclusion is the solution process is relatively insensitive to the actual step size used in the forward difference approximation to the directional derivative. The scaling process changed the residual by eight orders of magnitude and did create some round off errors. However, these were relatively insignificant in the overall solution process. The biggest impact due to the scaling process came from the termination criteria driving one additional nonlinear iteration for the scaled cases.

Number of Krylov Vectors vs. Restarted GMRES Tests

Restarted GMRES is practical but **unpredictable** in performance. There is no solid answer on how many Krylov vectors to carry before restarting or how many restarts to allow. Keeping more Krylov vectors in a linear solve before restarting usually gives better results but, since all the vectors must be stored, it may quickly burden available memory. Also, more keeping more vectors adds to the size (n^2) of the orthogonalization matrix and may overwhelm the practical limits of QR factorization algorithm used in

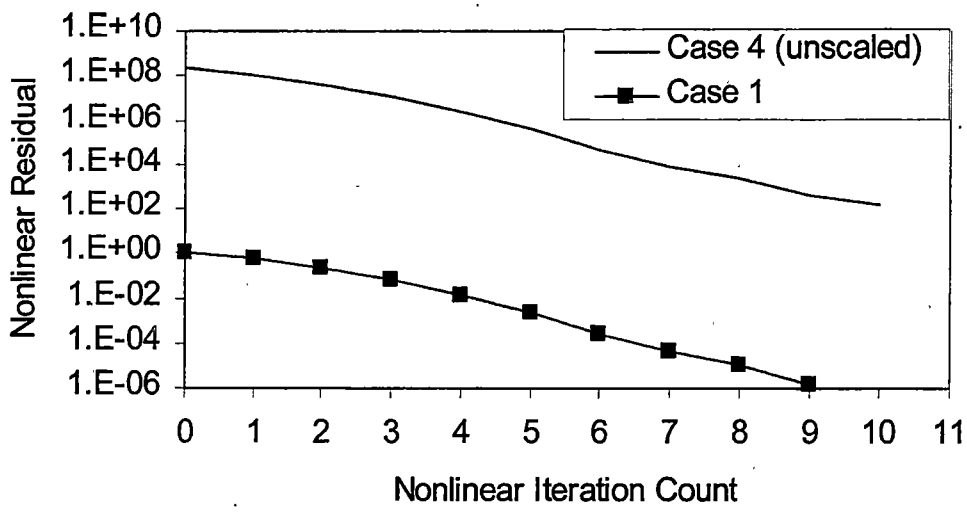


Figure 11-1: RHS Residual Convergence Paths with Scaling

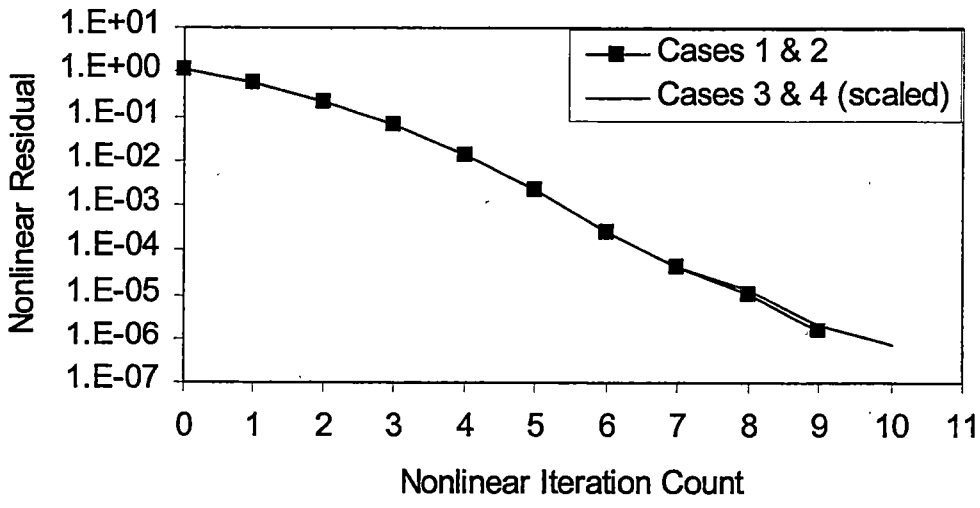


Figure 11-2: Rescaled Residual Convergence Paths

GMRES. For a fixed number of Krylov vectors, additional restarts do not necessarily lead to better convergence rates and some cases will stagnate if restarted.

Tables 11-2 and 11-3 outline a series of numerical experiments associated with GMRES restarts using the fluid nozzle test case. The only change between the cases in Table 11-2 was the number of restarts allowed. Each case had a first chance to satisfy linear convergence criteria using 40 Krylov vectors (arbitrary) before restarting. Thus, the case with zero restarts (case 1) may be considered as standard GMRES (not restarted) limited to 40 Krylov vectors. Case 1 did not converge. The time ratio describes the wall clock time of the cases relative to the baseline (case 3). The FLOP ratio is a measure of the number of floating point operations relative to the baseline. It is well correlated with the wall clock time but indicates the pure math operations without the memory or storage device read/writes. The time ratio and FLOP ratios listed in Table 11-2 are shown graphically in Figure 11-3.

The minimum number of nonlinear iterations also correlates with the lowest time and FLOP ratio. Cases 2 through 7 shown in Figure 11-4 show a minimum of eight nonlinear Newton iterations are needed for convergence. Increasing the number of restarts puts more effort into the linear (GMRES) solver to try and satisfy the increasing demands of the Eisenstat-Walker forcing function (see Chapter 3). As it turns out, cases 4 through 7 are identical in convergence until the fifth Newton step. Then, case 5 happens to stop after 400 Krylov vectors with a solution that is nearly ideal, leading to even greater convergence in step six. This can be seen in the residual histories shown in Figure 11-5.

Table 11-3 shows the actual convergence times (in seconds) for a wide variety of test conditions, all converging to the same solution (if they converged at all). Most of these will be discussed in another context in the fluid flow analysis of Chapters 12 and 13. The additional parameters will be defined there as well. Table 11-3 is included here to illustrate some important points. First, limiting the number of GMRES restarts performed better (faster convergence with equal accuracy) than allowing 18 restarts in every case, sometimes by as much as 40%! The baseline value of 18 restarts comes from the strategy of allowing GMRES the opportunity to hit every possible Krylov vector associated with the (virtual) left hand side matrix. The test case had 192 nodes with four degrees of freedom per node, or 768 total. Subtracting degrees of freedom lost for Dirichlet boundary conditions and dropping one degree of freedom for each

Table 11-2: Impact of GMRES Restarts

	Krylov Vectors	Restarts	Nonlinear Iterations	Time Ratio	FLOP Ratio
Case 1	40	0			
Case 2	40	3	18	1.482	1.495
Case 3	40	5	9	1.000	1.000
Case 4	40	7	8	0.998	1.024
Case 5	40	9	7	0.921	0.901
Case 6	40	11	8	1.193	1.243
Case 7	40	18	8	1.272	1.316

Table 11-3: GMRES Restarts with Krylov Vector Changes

	Tau=B				Tau=N	
	Wall amplification		No wall amplification			
	Pseudo time					
	NO	YES				
	Krylov vectors					
Restarts	40	40	40	88	40	56
0	-	-	-	-	-	***
2	-	-	-	-	***	-
3	-	-	3293	-	-	-
5	4535	2671	2388	1806	4552	-
7	-	-	2217	-	-	-
9	-	-	2047	-	-	-
11	-	-	2650	-	-	-
18	6642	4661	2826	-	6497	-

*** did not converge

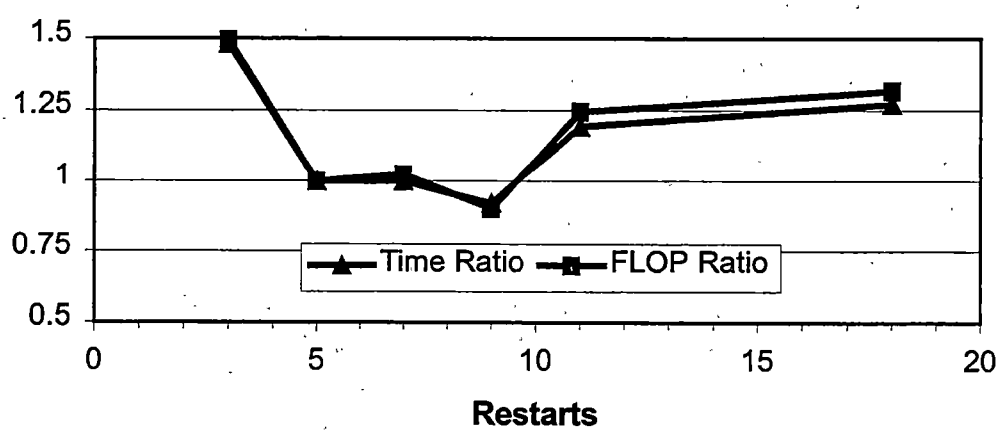


Figure 11-3: Change in Convergence Time with GMRES Restarts

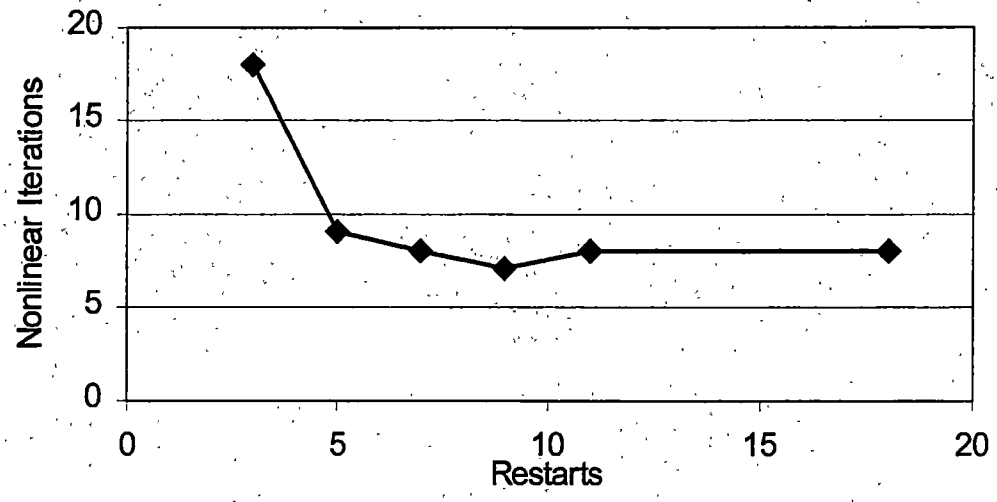


Figure 11-4: Nonlinear Iteration Count with GMRES Restarts

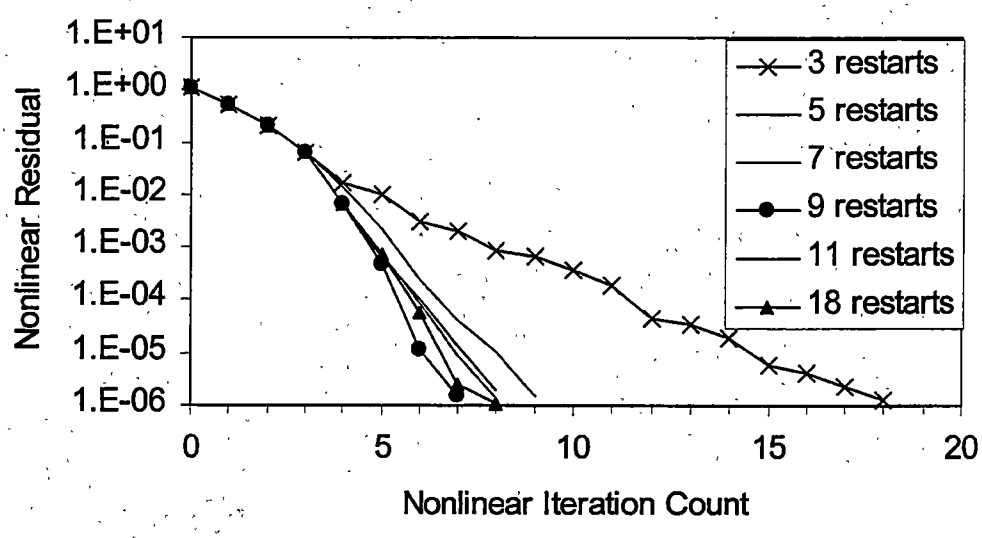


Figure 11-5: Nonlinear Convergence Histories with GMRES Restarts

wall node leaves 692 degrees of freedom. Thus, the left hand side matrix to solve is 692×692. Theory proves GMRES (without restarts) will provide the exact solution with 692 Krylov vectors or less. With 40 Krylov vectors per restart, 18 GMRES solves checks up to 720 total directions. The theory doesn't apply to restarted GMRES but it was a logical starting point for the investigation.

Table 11-3 also shows that including more Krylov vectors before restarting also improves convergence times. This proved to be true in general, especially for the structural cases. The test case with 88 Krylov vectors comes from the number of vectors required to solve the structural domain used in the coupled fluid-structure system. The need for 88 vectors was identified in *structure only* calculations which would not converge if restarted with less than 88 vectors due to ill-conditioning of the system. It is worth pointing out cases found in the literature commonly restart after forming only 20 Krylov vectors. This turned out to be unacceptably slow. The trick, of course, is to provide a problem statement (i.e. left hand side matrix) that is so well conditioned that it will converge within the alloted number of search iterations (Krylov vectors). It is also important to reiterate the cost of carrying too many Krylov vectors. First, each vector must be stored so the storage requirement goes up linearly. Second, as outlined in Chapter 3 and [Kelly, 1985], each new vector must be made orthogonal to all the previous ones. One hundred vectors turns out to be a practical limit for maintaining orthogonality. Finally, each of the vectors is used in the least squares solution process GMRES uses to find the "best" solution in the space spanned by the Krylov vectors. GMRES uses a QR algorithm for this and QR is also practically limited in accuracy to around a rank 100 matrix.

Eisenstat-Walker Forcing Functions

The Eisenstat-Walker algorithm generates a sequence of forcing functions rather than using a fixed value. The forcing function sets the nonlinear residual reduction goal for each linear solve. If the value is fixed at, say 0.5, each linear solution has a goal of producing a solution that will reduce the nonlinear residual by a factor of 2. For a direct matrix solver, this is not relevant since the solution is all-or-nothing. For an iterative solver, it sets criteria on "how good" the solution must be. [Pueyo and Zingg, 1997] selected a fixed forcing function of 0.1 as the optimum for fluid flow. With the Eisenstat-Walker

algorithm, the forcing function is variable, based on the minimum acceptance criteria specified and success at meeting goals in previous steps.

Table 11-4 shows the actual wall clock time for the fluid flow nozzle test case selecting various "minimum" reduction criteria. The minimum is a default value used as a starting point. For example, 0.9 means on the first nonlinear iteration or an iteration after a global step length reduction, the residual is to be reduced by 10%. This may be reasonable for highly nonlinear problems far from the solution as there is no sense working too hard to generate a highly accurate linear solution when all that is needed is a general direction showing improvement. A starting point of 0.1 requires a 90% reduction in the residual on the first step. This will drive additional work for the linear solver.

The Eisenstat-Walker algorithm mostly took care of itself, generating appropriate forcing functions with each step. A starting point of 0.9 was a bit too forgiving in that it did not demand enough of the linear part of the solution process. Conversely, a starting point of 0.1 placed too much emphasis on the early linear solutions while the nonlinear equation set was still far from convergence. Selecting 0.5 turned out to be a reasonable compromise with good results. This could have been set smaller for "nearly linear" problems and updating a transient solution from one time step to the next may be just that sort of case. However, fluid flow solvers typically have to deal with harsh, highly nonlinear starts from uniform initial flow conditions that instantaneously discover a solid object.

Table 11-4: Eisenstat-Walker Forcing Functions

	Restarts allowed			
EW	18	5	2	0
0.1	6773	-	-	-
0.5	6497	4552	-	@***
0.9	-	-	***	-

@ 56 Krylov vectors allowed, 40 in all others

*** did not converge

GMRES Starting Vectors

GMRES forms a Krylov subspace about an initial linear residual vector as $\{r_0, Ar_0, A^2r_0, \ldots, A^{n-1}r_0\}$, where $Ax - b = r_0$. For a Newton solver, b is the residual vector from the right hand side equations so a starting vector, x_0, must be selected. If the starting vector is the solution vector, GMRES is already done and there is no need for further iterations. However, the solution is not known so the the algorithm needs a place to start. Intuitively, all starting vectors (guesses) are not created equal and a good starting vector could lead to rapid convergence. In practice and in the literature, one guess is as good as another so the "keep it simple" strategy applies.

Table 11-5 shows the results of using three different starting vectors (all terms set to zero, all terms set to one, and the previous solution increment) for some simple fluid flow coupon test cases (described in Chapter 12). SRAND is a difficult test with flow moving through a grid slanted to the Cartesian coordinates and using a random number generator to set initial conditions (see Figure 12-3). SLANTP is the same geometry but with uniform initial conditions other than a pressure disturbance at a single node. Setting the starting vector to $x_0 = 0$ is the widely used default and causes the linear residual to match the negative residual of the right hand side, $A\{0\} - R_{rhs} = r_0$. As Table 11-5 shows, this worked as well as the other starting vectors tested and was used throughout the effort.

Table 11-5: GMRES Starting Vectors

	Cases	
X0	SRAND	SLANTP
0	17	6
1	16	6
x-xold	-	6

nonlinear iteration counts

CHAPTER 12

ANALYSIS OF FLUID FLOW TWS/SUPG RESULTS

The fluid flow solver implemented in the code uses linear triangle and bilinear quadrilateral elements. Eight-node biquadratic quadrilateral elements were included in the development but never tested. The basic fluid flow equation set is based on the two-dimensional Euler equations with the primitive variable set $\{p,u,v,T\}$ as the primary unknowns. Both steady state and time dependent equations were implemented but only the steady state equations were tested. The equations were kept in dimensional form throughout (i.e. not non-dimensionalized). The Euler equation set was discretized according to the Taylor Weak Statement derivation and tested for various choices of the scaling parameter, τ. Grid motion terms were also included.

Numerical Solver

The discrete fluid flow equation equations were numerically solved using the matrix-free Newton/GMRES methodology. The right hand side equations were kept in conservation form to create a residual vector. The left hand side solver used the linearized fluid flow equations to create multiple residual vectors. These were used in a finite difference to create an approximate Newton statement for the left hand side. The GMRES solver controlled the number of left hand side solutions attempted, restart logic, and residual reduction goals.

Total energy and entropy tracking showed the discrete implementation to be isentropic except where grids were too coarse to support the flow gradients with bilinear elements. There was never any evidence of $2\Delta x$ waves but overall convergence rates depended on the GMRES parameters and the diffusion parameter, τ.

Residual Scaling

Before getting into the test cases, it is important to show the residual scaling logic. Since the equations were left in dimensional form, the numerical values in the continuity, momentum and energy

equations were based on their consistent fundamental units. Continuity was stated in mass flux terms as $\frac{slug}{ft^2 \cdot s}$ and, for sea level conditions at 500 ft/s, has numerical order $\mathcal{O}(1)$. Momentum is stated in $\frac{lb_f}{ft^2}$ and has numerical order $\mathcal{O}(10^3)$ for the conditions stated. Energy is stated in $\frac{lb_f}{ft \cdot s}$ and has numerical order $\mathcal{O}(10^6)$ for the conditions stated. Since the numerical values in the equations differ by six orders of magnitude, a fixed percent variation in the equations would lead to a six order of magnitude difference in the residuals associated with the equations. This causes a problem for the GMRES solver when it starts the Gram-Schmidt process to make the various Krylov vectors orthogonal. When the Gram-Schmidt process makes a set of vectors $\{v_1, \ldots, v_n\}$ orthogonal by transforming to the orthogonal set $\{u_1, \ldots, u_n\}$, it calculates a coefficient $\frac{u_i^T v_j}{u_i^T u_i}$. If these vectors are actually the residual vectors, the energy term will dominate the squared sum of the terms. This same problem would arise by actually forming the left hand side Jacobian matrix. It would contain some terms that would dominate and would drive large pivots in a direct solver. If left alone, the numerical implementation would appear as a stiff system of equations. Fortunately, the situation is easily resolved by simply scaling the equations to be of the same order of magnitude. Selecting the momentum equations as the middle ground to use for the baseline, simply multiply all the continuity equations by 10^3 and all the energy equations by 10^{-3}. Scaling the residuals of the equations after formation does this most easily. The basic equations are not changed and the scaling is all in one place in the code as a post process after formation of the individual right and left hand side residuals. The scaling may change with location to emphasize a particular region or aspect of the flow field (such as boundaries) but should not be convergence adaptive as the residual norm is the measure used to determine convergence.

Coupon Test Cases

The test cases for the fluid flow equation set were simple "coupons" over a range of speeds both with and without walls and at various alignments with the flow. A simple convergent nozzle was also tested. In the most basic coupon test, the code held free stream conditions exactly and did not attempt to change or improve upon the exact solution initial condition. In other tests, the exact solution was perturbed at a single node for one or several variables.

The residual scaling effect is easily shown on a simple 3×3 element coupon (Figure 12-1) aligned with the flow. All nodes except the one marked are set at free stream conditions. The marked node is perturbed by a unit change in pressure, velocity, or temperature both with and without residual scaling. Two free stream conditions are tested: a median speed of $565\ ^{ft}/_{s}$ and a slow speed of $5\ ^{ft}/_{s}$. The twelve cases are shown in Table 12-1 and the residual reduction is shown in Figure 12-2. Since portions of the residual were scaled over six orders of magnitude, the residual norms for each iteration of each case were normalized for comparison. The dashed lines show the cases without residual scaling and the solid lines show the scaled cases. The residual reduction goal was nine orders of magnitude relative to the starting (iteration zero) residual or less than 10^{-9} absolute. All the scaled cases converged but the high and low speed pressure perturbation conditions for the unscaled cases did not (top two dashed lines). Figure 12-2 also shows the scatter in the convergence results for the unscaled cases but the fairly uniform convergence of the scaled cases. In each case, the last iteration may decrease the convergence slope because of logic in the Newton-GMRES solver that prevents "over-solving" beyond the stated goals. All the lines do not end below 10^{-9} because of the normalization process.

As one would expect, the slowest convergence for the cases that did converge was for the velocity perturbation in the low speed. The fastest convergence was for the high and low speed temperature perturbation in the unscaled cases. This is because the temperature is primarily associated with the energy and the energy residual is three to six orders of magnitude greater in numerical value than the momentum and continuity equations. The implication is that residual scaling may also be used to advantage to emphasize accuracy on a particular variable or location in the flow domain. This notion was tested but not vigorously pursued in this effort.

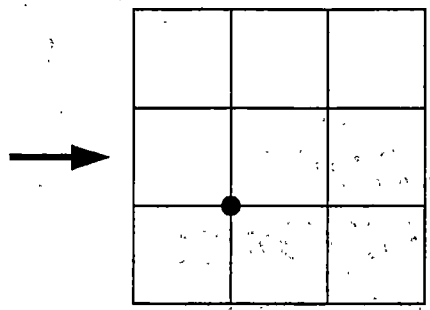


Figure 12-1: Simple Coupon

Table 12-1: Residual Scaling Test Cases

w/o scaling	P	U	V	T	Iterations
basicp	720+1	565	0	530	***
basicu	720	565+1	0	530	7
basict	720	565	0	530+1	5
slowp	720+1	5	0	530	***
slowu	720	5+1	0	530	6
slowt	720	5	0	530+1	5

*** did not converge

w/ scaling	P	U	V	T	Iterations
basicp	720+1	565	0	530	5
basicu	720	565+1	0	530	5
basict	720	565	0	530+1	5
slowp	720+1	5	0	530	5
slowu	720	5+1	0	530	5
slowt	720	5	0	530+1	5

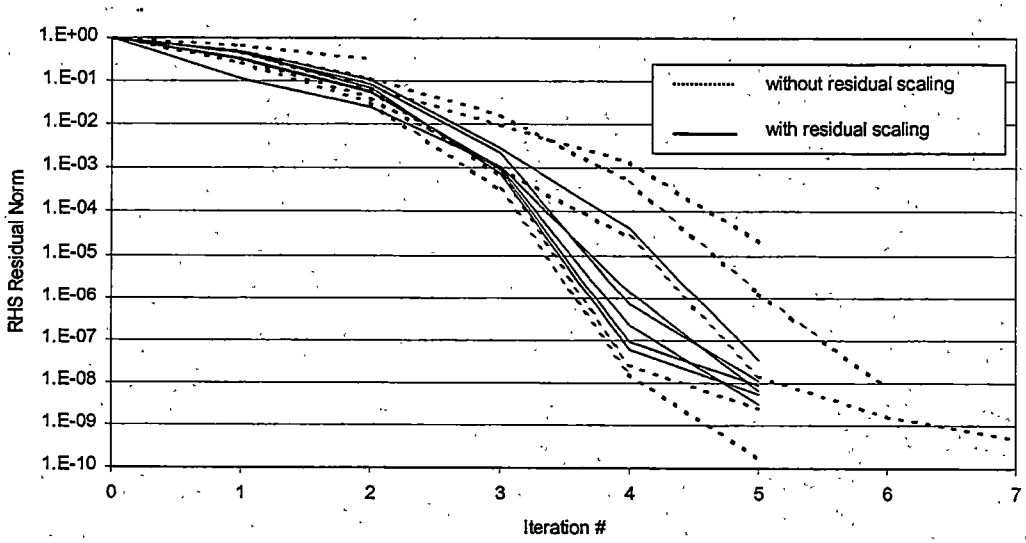


Figure 12-2: Residual Scaling Results

Several other coupon configurations were tested with residual scaling to verify the code and to test the stability of the TWS/SUPG method. The harshest was probably the slanted case with walls (Figure 12-3) where the initial conditions for every variable at every node were set using a random number generator between zero and the nominal value. The global Newton-Armillo step length reduction logic came into play and the effects of the Eisenstat-Walker forcing function can be seen as the rate of convergence changes. Figure 12-4 shows the relative influence of including time derivative preconditioning terms. The "no pseudo time" case is the baseline with residual scaling but no time derivative preconditioning. The "no local time stepping case" has time derivative preconditioning with a fixed time step of one. The "CFL" cases change the local preconditioning time step using the logic of [Buelow, Venkateswaran & Merkle, 1995]. These will be discussed more with the nozzle test cases.

The coupon test cases validated the coding, basic stability of the algorithms, speed and convergence rates, and boundary condition treatment. They all converged to the exact solution to the accuracy required to satisfy the termination criterial. Not shown are coupons with random variations to the internal node locations to vary grid size, aspect ratio and skewness or tests of mixed triangular and quadralateral elements. These were all tested and all converged at rates comparable to those already shown.

Straight Channel Test Cases

The coupon tests were fast checks of coding changes and quickly identified problem areas. A simple but much more difficult test case was a long straight channel with an aspect ratio of **18.7:1** as shown in Figure 12-5. In the test, the initial conditions on the 24×4 grid were set for uniform flow through the channel at a Mach number of .22 and the exit pressure (Dirichlet boundary condition) was changed (up or down) at startup by about $100 psf$ (16%). This was significant enough to cause a strong (nonlinear) wave that had to propagate upstream and temporarily accelerated or decelerated the flow until the wave settled. In the solution transient, the pressure wave remained very flat throughout (orthogonal to the walls) but the velocity and temperature created "signature" profiles along the walls near the exit. The

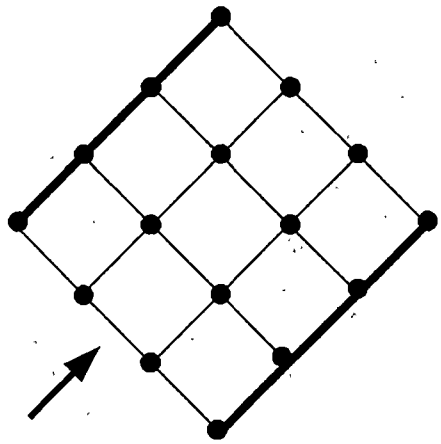


Figure 12-3: Slanted Coupon with Walls

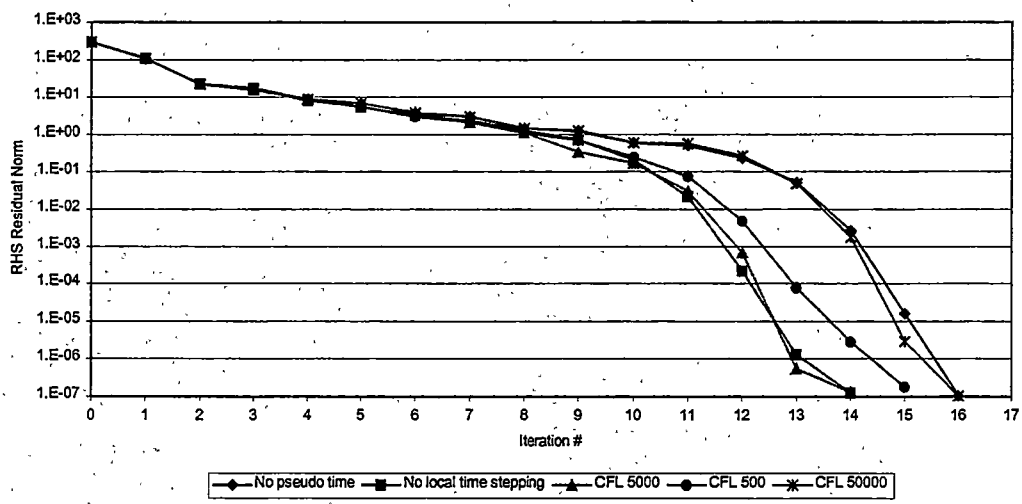


Figure 12-4: Slanted Random Variable Coupon Results

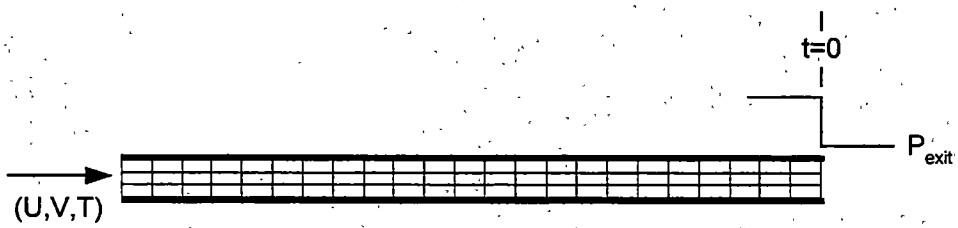


Figure 12-5: Straight Channel Test Case

profiles were very small (less than .005% max variation for the cases shown) and could be made arbitrarily small by forcing convergence of the residual to near machine zero. However, the profiles were indicative of problems to come in the nozzle test cases.

The entropy and total enthalpy profiles (aspect ratio not correct in figures) showed in Figure 12-6 and Figure 12-7 should be perfectly flat for the ideal solution at convergence. The non-dimensional entropy plot in Figure 12-6 shows a very small $\left(10^{-6}\right)$ negative entropy along the walls at the exit with entropy production along the internal nodes at the exit. There are no flow gradients, shocks, expansions or discontinuities so the profile seems to indicate a non-physical tendency either at the wall boundary or the outflow boundary. A non-linear gradient operator (shock capturing term) might correct the problem but the gradients are very small and not a part of the physical flow field.

The total enthalpy profile in Figure 12-7 shows the same pattern. The actual enthalpy is non-dimensionalized by dividing by $Cp*530$. Dividing by just Cp shows a temperature variation of .001 degrees Rankine. Since entropy and (static) enthalpy are thermodynamic state variables, the pressure and temperature primitive variables are most suspect. Obviously all the variables are linked but the temperature is the most likely suspect since it is the most strongly associated with the energy equation. The problem is not obvious, though, as the mass, momentum, and energy are all conserved and the primitive variables are correct to a high degree of accuracy even after a strong disturbance.

Coding errors were ruled out as the solution would not converge accurately here or in the coupon tests otherwise. The wall boundary and outflow boundary treatments are also correct or, again, the solution would not converge to the correct, conservative results. This was verified through numerous coding variations. Grid resolution, skewness, or alignment does not limit the solution. Tests with a uniform flow field without the initial pressure wave satisfy the right hand side residual equation with a normalized residual norm $\frac{\|RHSR\|_2}{\sqrt{NDOF}} < 5\times10^{-14}$ and no discernable pattern to the residual. Residual surfaces for the test with the pressure wave show minimal to no pattern in continuity or y-momentum but patterns similar to those above exist in the x-momemtum and energy residuals. There is no sign of any $2\Delta x$ instability but tests with a smaller value for the intrinsic time scale (viscosity parameter), τ, showed the $2\Delta x$ waves will

ENTROPY

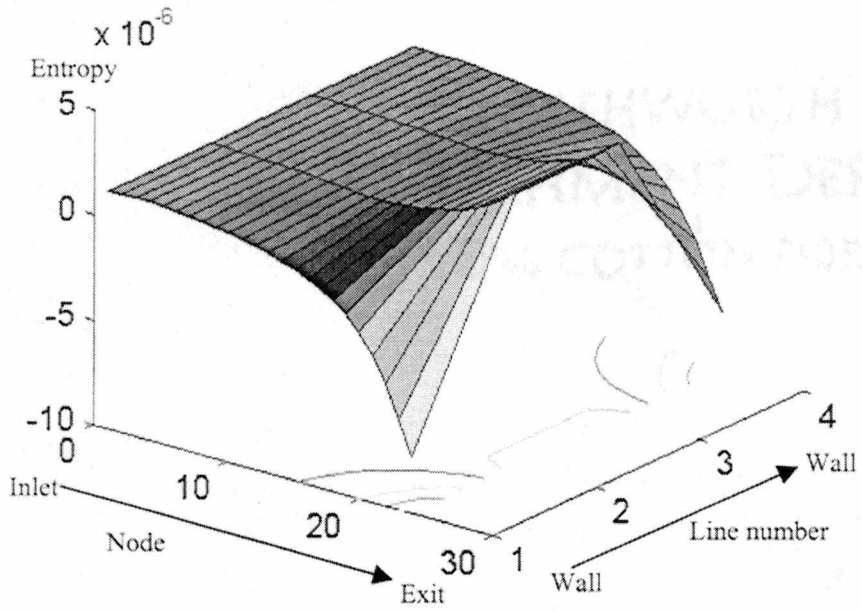


Figure 12-6: Straight Channel Entropy Profile

ENTHALPY

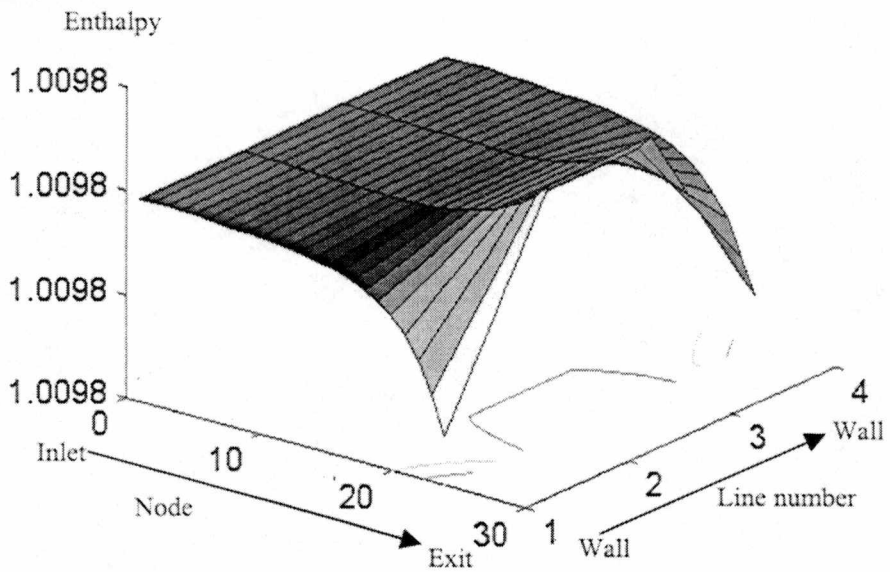


Figure 12-7: Straight Channel Total Enthalpy Profile

generate if τ is much smaller. The profile is obviously not related to $2\Delta x$ instabilities in the streamwise direction but the grid is too coarse to determine problems normal to the main flow. Tests with larger intrinsic time scales had little affect on the boundary profiles and scaling τ based on waves in the gradient direction showed no improvement over scaling to target 2Δ waves. The boundary profiles paced the rate of convergence for the global Newton statement. The solution is perfectly adequate for all practical purposes but the boundary profile is a bad sign.

In a final look, a case was run using the inverse of the physical conserved-to-primitive variable Jacobian in the SUPG-like term as opposed to the time preconditioning matrix. The preconditioning matrix was still used in the pseudo time term on the left hand side but there were no non-physical terms on the right hand side. In this case, the solution stagnated far from convergence but showed an interesting pattern in the entropy and enthalpy plots. Only the enthalpy is shown but the entropy (and temperature) were similar in nature.

Figures 12-8 and 12-9 show the curious waviness of the pattern. These are clearly not 2Δ but seem to show three complete waves along the length of the 24 nodes of the channel. The pattern is symmetric and the wall enthalpy appears to be almost 180 degrees out of phase with the interior. A quick check back to Figure 12-7 with the non-physical (preconditioning) Jacobian shows the wall and interior are also out of phase but only a half wave (fundamental mode) is present.

[LeBeau, et al, 1993] developed a Dirichlet treatment of wall boundaries by rotating the momentum interpolation functions to be normal and tangent to the wall. They were motivated by the need to get away from explicit treatments such as rotating velocity vectors at the wall. In this research project, an implicit wall treatment is used based on the transposed equations and the divergence theorem (see Chapter 4. The diffusion was left in its standard form as the weight functions would not support a second order derivative. Line integrals along the boundaries were tested both with and without diffusion terms). This did not cause the phase shift, though, as the shift was still present when the original (before transposing) equation set was used. The apparent phase shift may actually be part of a 'beat frequency' generated by the difference between two higher frequency waves. This is purely speculation but the

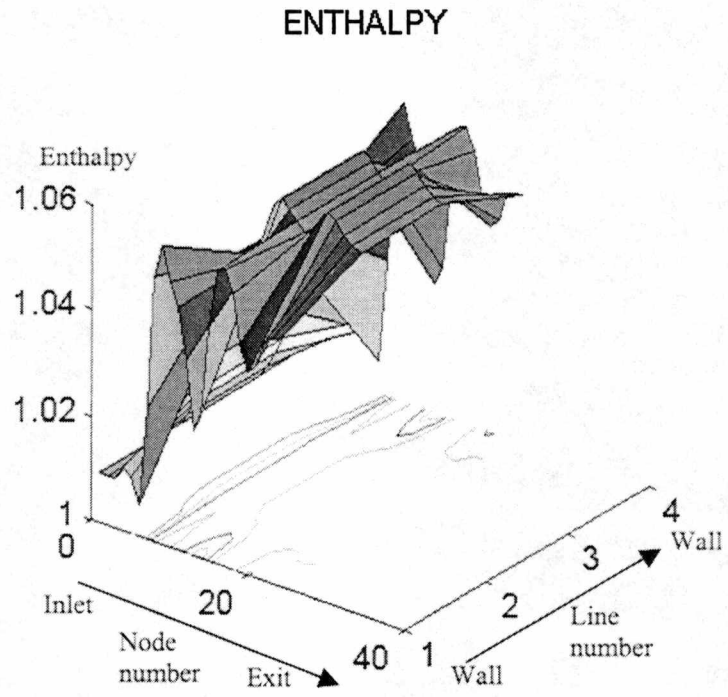


Figure 12-8: Enthalpy Surface with Physical Jacobian at Convergence Stagnation

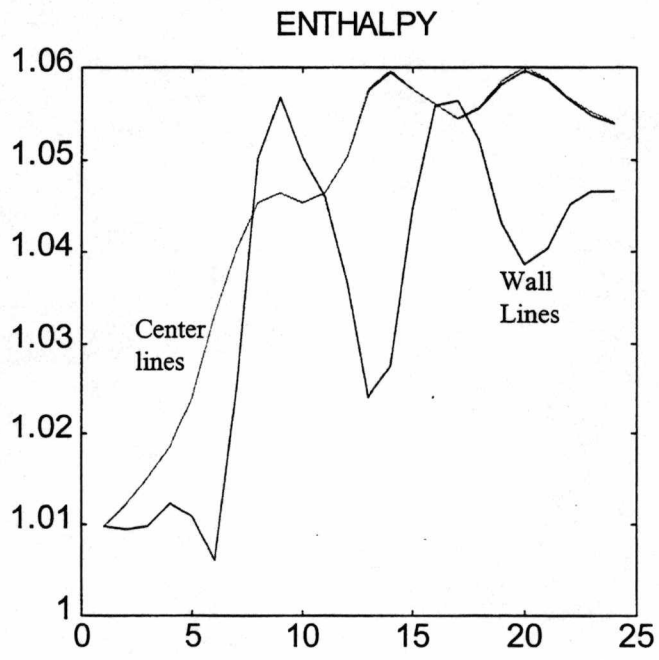


Figure 12-9: Enthalpy Plot with Physical Jacobian at Convergence Stagnation

boundaries are somehow connected to the odd profiles. Again, at convergence the profiles have damped out or reduced to miniscule levels but the transient snapshots shown in Figures 12-8 and 12-9 indicate corruption associated with the physical mass matrix that the preconditioned mass matrix somehow minimizes or avoids altogether.

Nozzle Test Cases

The coupons and straight channel test cases only had flow gradients transient to the steady solution. To provide a flow with a steady gradient, a subsonic nozzle works well. The nozzle test case is a 2-D subsonic convergent nozzle with a 2:1 area ratio and a symmetry along the centerline. Treating the centerline as a tangent surface with slip walls enforced symmetry. (Comparisons with a full nozzle calculation showed this treatment correct.) The nozzle provides a flow gradient, some aspect ratio and grid size changes, flow at various alignments to the grid, and tests solution accuracy. A coarse grid was used to test various combinations of code parameters such as the number of GMRES restarts, the number of Krylov vectors allowed per GMRES solution, options for the τ parameter, and the time derivative preconditioning logic. The coarse grid is 24×8 with a solid side wall and symmetry along the centerline. Uniform Dirichlet inflow boundary conditions $\{u, v, T\}$ are enforced at the inflow edge shown in Figure 12-10 and a uniform Dirichlet outflow boundary condition $\{p\}$ is enforced at the outflow. There was no need to overtly enforce the solid wall (tangent flow) boundary based on the weak form of the equations used. In reality, this reduces the degrees of freedom at the wall nodes by one. Several alternative approaches were tested for rigorously enforcing tangency such as penalty terms, confining GMRES search directions to wall tangent directions and zeroing out the residual associated with the lesser component of the velocity vector at the wall. There was no obvious benefit to any of the alternatives. The lost degree of freedom did not cause a problem with a reduced rank in the (virtual) left hand side matrix because GMRES was restarted (typically at 40 vectors) long before the full rank (690) was needed.

The nozzle grid used for most of the effort is shown in Figure 12-10 with no points removed or hidden. Uniform Dirichlet boundary conditions are set at the inflow and outflow nodes shown. These uniform flow conditions are too close to the nozzle for reality and would normally be set one or two pipe

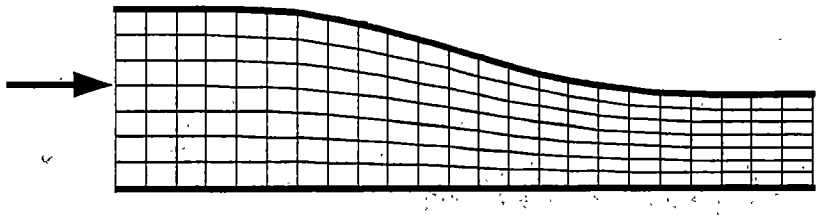


Figure 12-10: Nozzle Grid

diameters upstream and downstream on straight sections. The grid is also too coarse for best accuracy with the linear interpolating functions used but it is adequate for the purposes of this research project.

There are no shocks or discontinuous flow regions in the nozzle but the coarse grid still allowed for relatively large changes in the flow variables over a single element. The theoretical role of the discontinuity (shock) capturing term became evident in the steeper gradient regions (3/4 of the way down the nozzle) by its absence. These regions generated dispersion errors that nonlinear shock capturing terms are designed to eliminate. The proper form of the term was hinted at but not developed in the theoretical analysis. It should be completed and included in a production code even if shocks are not expected because it should speed convergence in steep gradient regions. The coarse grid of a multi-grid cycle would likely experience the same issues of large variable changes over a single element. The gradient regions did not impact the stability but seemed to pace the rate of convergence for all code configurations.

As for the accuracy of the results, they are sufficient for the bulk flow changes created by the nozzle between the inflow and outflow boundaries but slightly lacking in the detailed internal flow. As in the straight channel, enthalpy and entropy plots are the best indicators of any abnormalities. Enthalpy and entropy should be constant for a subsonic, inviscid flow but the code does not explicitly enforce these conditions. That is why they are such good measures of the quality of the calculated flow field. Figures 12-11 through 12-14 show the enthalpy and entropy surfaces along with the details from the centerline, mid flowfield (at the arrow in Figure 12-10), and wall grid lines. The profile (line charts) also highlight the theoretical results validated with a quasi 1-D analytic solution.

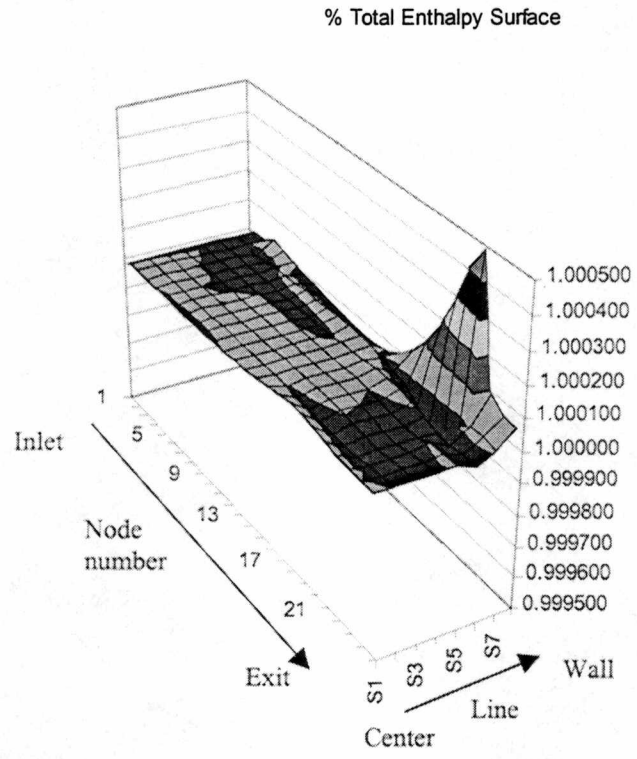


Figure 12-11: % Total Enthalpy Surface for Nozzle Flow Test Case

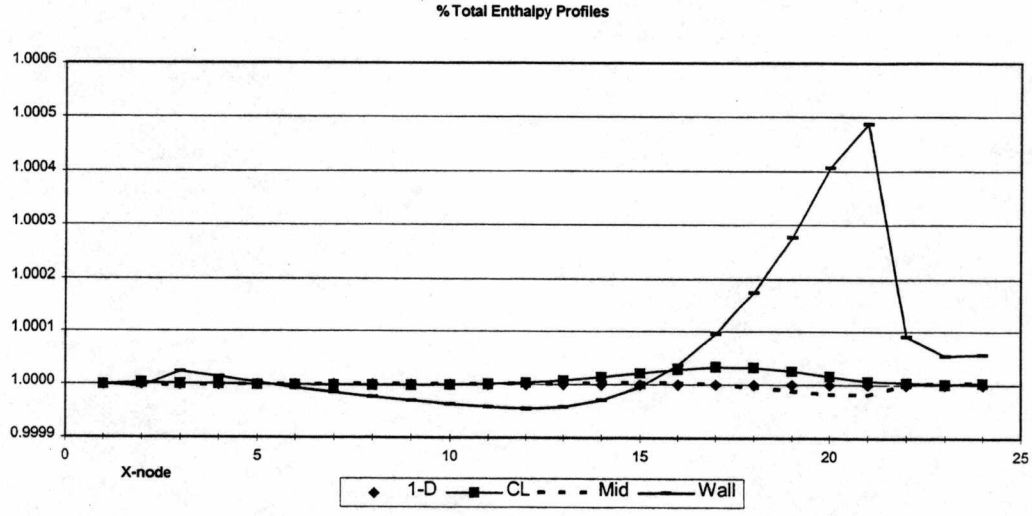


Figure 12-12: % Total Enthalpy Profile for Nozzle Flow Test Case

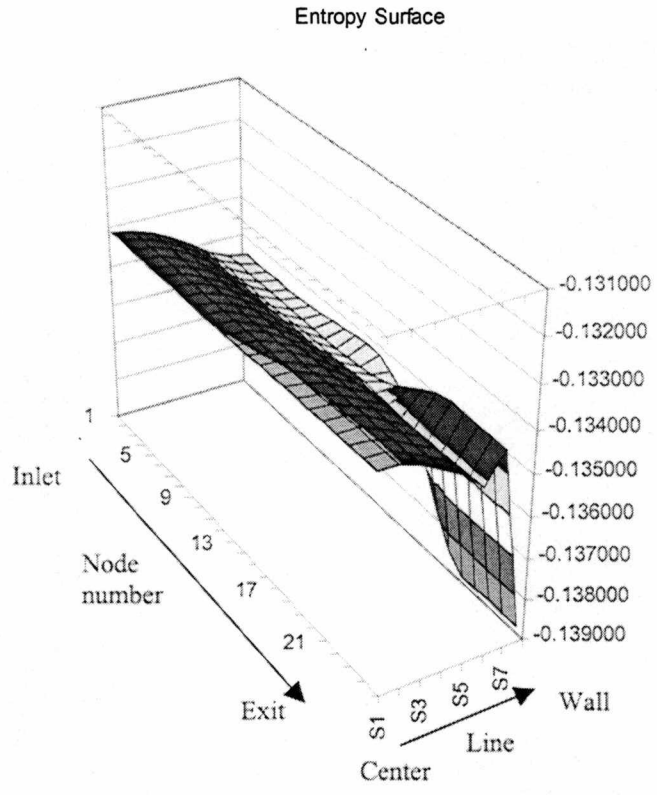


Figure 12-13: Nondimensional Entropy Surface for Nozzle Flow Test Case

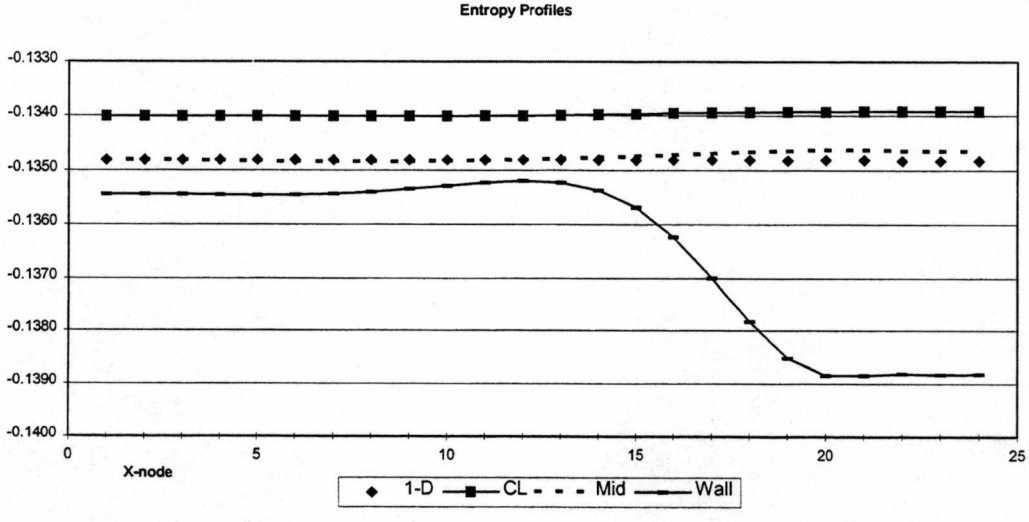


Figure 12-14: Nondimensional Entropy Profile for Nozzle Flow Test Case

Several important features are evident in Figures 12-11 through 12-14. First, the profiles are not flat so enthalpy and entropy show non-physical traits locally over the flow field near the wall. The most corrupt region is localized in the maximum curvature region of the nozzle where the flow turns to realign with the nozzle axis. The enthalpy disturbance peaks at node 21 (see Figure 12-12) along the wall at about 0.05%. This equates to about 0.25 degrees Rankine or roughly $2\,{}^{ft}/_{s}$. However, the temperature and velocity may offset each other to mask other problems. The enthalpy tries to make a quick recovery by node 22. Total enthalpy should be constant even across a shock (compression or expansion) so the energy equation behaves as if work or heat is being added and then quickly removed along the wall.

The entropy provides a different look at the problem. In Figure 12-13, the entropy should be constant but instead decreases slightly from the centerline to the wall. The real problem is the "negative entropy" generated along the wall. Entropy production is indicated by the slope of the profiles in Figure 12-14 and appears to peak at about node 17. Negative entropy production begins as the flow begins to turn back axial at the wall inflection point about half way through the nozzle (node 13). The sudden change in enthalpy between nodes 21 and 22 does not have a corresponding entropy production (negative or positive) so the mechanism is reversible. The entropy production is localized in the wall region and production peaks in the steepest flow gradient regions. The alignment with the enthalpy profile shows the negative entropy production corresponds to the increase in enthalpy. Decreases in enthalpy do not correlate well with the entropy changes.

[Baker, 1983, p178] points out a coarse grid discretization may cause negative entropy production and can be eliminated with proper refinement. [Hirsch, Vol II, pp. 519-525] expands on the need to satisfy the entropy condition for Euler flows and points out proper diffusion terms (artificial or physical) will ensure non-physical discontinuities will never appear. Both avenues were tested and will be summarized after a review of the impact the non-physical flow region has on the primitive variables: p, u, v, and T.

Figure 12-15 shows the pressure profiles for the centerline, mid flow line, and wall from the calculation and compares it to the profile from the quasi 1-D analytic solution. Surprisingly, the mid flow line matched the analytic solution better than the centerline. The wall pressure shows the impact of

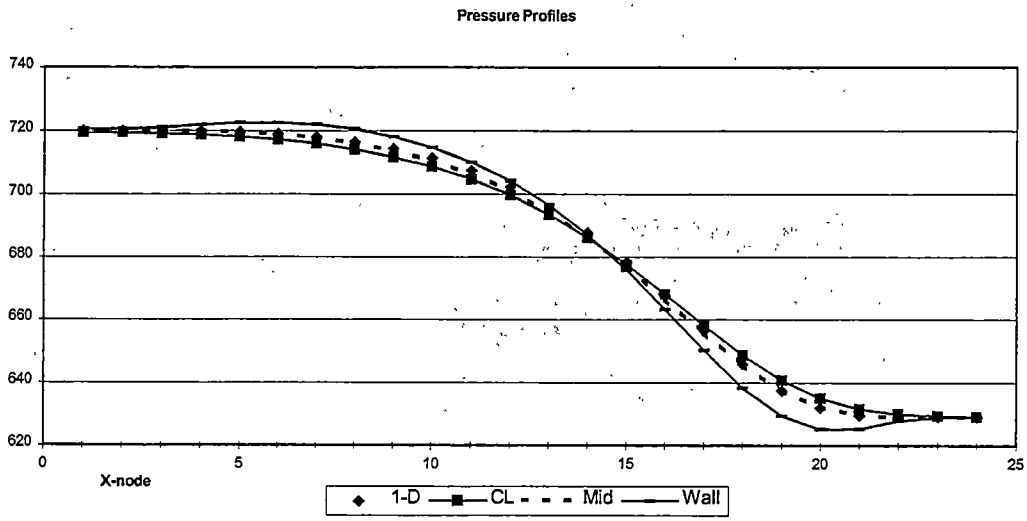


Figure 12-15: Pressure Profile for Nozzle Flow Test Case

the problem indicated by the entropy and enthalpy profiles. First, the pressure increases as the flow slows down (!) slightly before accelerating. Second, the flow over-accelerates while turning back axial causing the pressure to undershoot the exit pressure by about 0.6% (~0.03 psi). This is not enough to be a major problem but it should be better.

The same trends are apparent in the temperature profiles in Figure 12-16 as in the pressure. The deceleration before the nozzle converging appears to be physically acceptable since there is no change in entropy along the wall (nodes 2 through 8) in this region. The same is true in the deceleration region recovering from the overshoot. The pressure and temperature 'bottom out' at wall node 20 and climb as the flow slows to the exit. The wall entropy is essentially constant in this region also.

The velocity profiles in Figures 12-17 through 12-19 show the same trends already discussed in the pressure and temperature plots. The vector plot in Figure 12-20 combines the velocity components and shows the velocity vectors follow the wall contours very well and the profile is flat at the nozzle exit.

Overall, the accuracy of the nozzle test case shown in Figure 12-10 is adequate for the current research project since all calculated variables are within 1% of theoretical. Several variations were tested

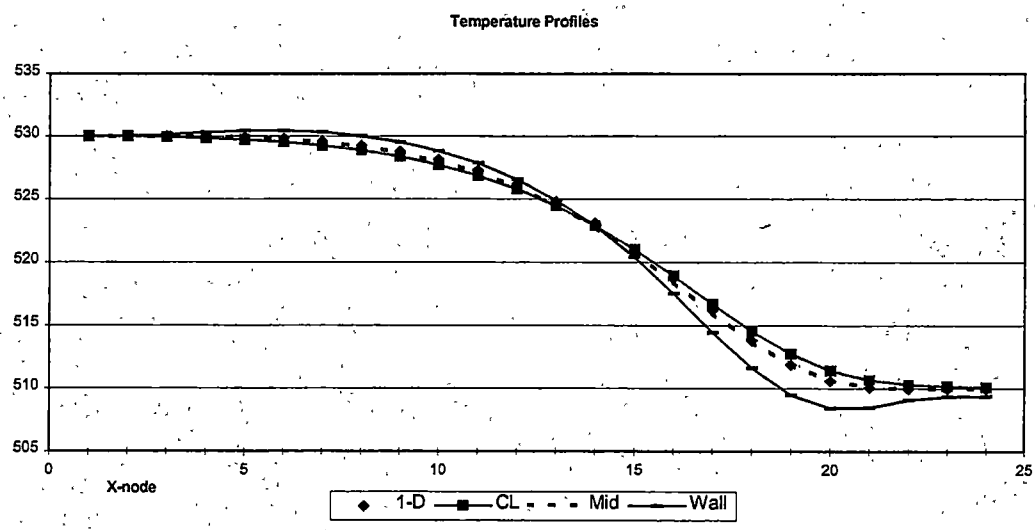


Figure 12-16: Temperature Profile for Nozzle Flow Test Case

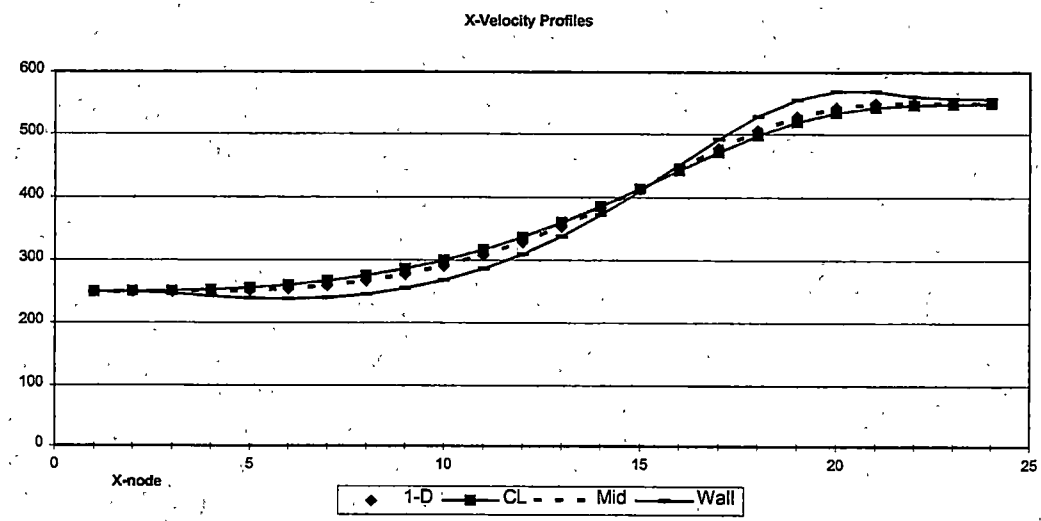


Figure 12-17: X-Velocity Profile for Nozzle Flow Test Case

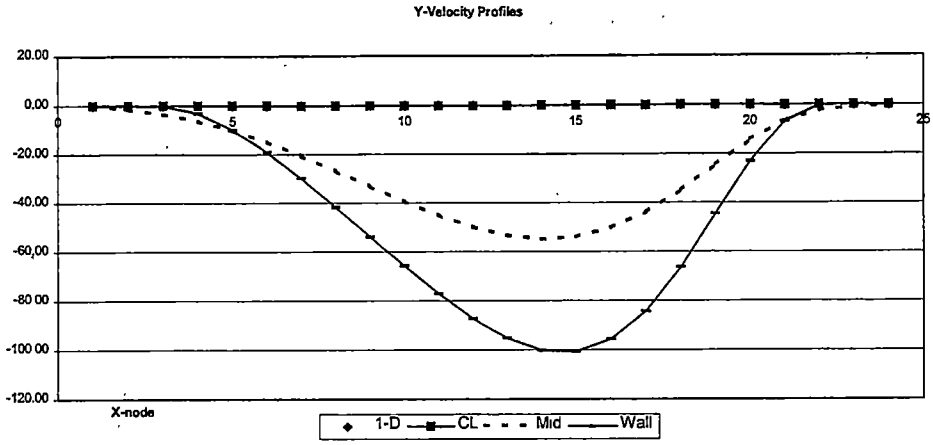


Figure 12-18: Y-Velocity Profile for Nozzle Flow Test Case

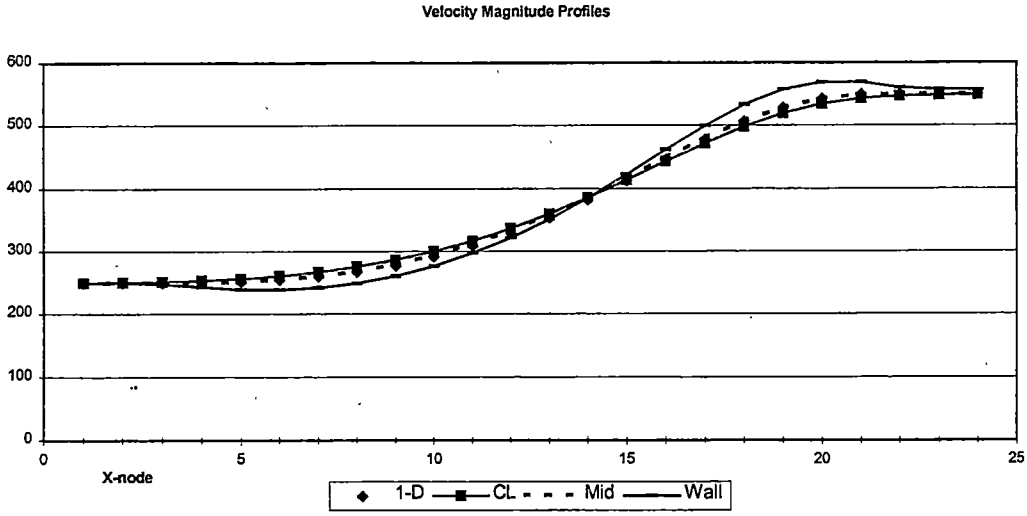


Figure 12-19: Velocity Magnitude Profile for Nozzle Flow Test Case

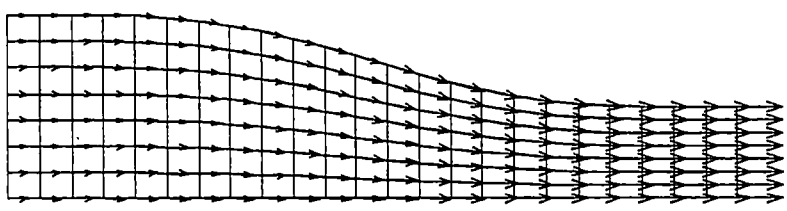


Figure 12-20: Velocity Vectors

to remove some of the known problems stated in the introduction to the nozzle section. Figure 12-21 shows an elongated entry and exit region to move the uniform boundary conditions away from the gradient regions. The boundary conditions were more uniform at the grid inlet and exit as one would expect. However, profiles at slices corresponding to the original inlet and exit locations were essentially the same as in the original configuration.

Figure 12-22 shows a refined grid along and out from the wall. It also highlights the ability of the method to accept localized grid refinement and a mixture of quadrilateral and triangular grid elements. The solution on the refined grid was better than the baseline in that the entropy and enthalpy variations along the wall were reduced (Figures 12-23 and 12-24). However, the impact on the primitive variables was small with the minimum temperature increasing about 0.2 degrees (better) but the minimum pressure dropping by 0.1 psf (worse.)

It is reasonable to assume that continued refinement would continue to reduce the errors until roundoff errors dominate. However, this is computationally expensive so a better approach would be to improve the algorithm. In addition, the fully refined grid case seems to indicate non-physical conditions are still present.

Figure 12-25 simply shows the basic nozzle without a symmetry assumption. The 24×8 grid is even coarser than the original relative to the flow gradients but a comparison of the solutions at common node locations shows no real difference. This case was tested to validate the treatment of the line of symmetry as an inviscid wall.

Finally, Figure 12-26 shows the combination of all refinements in a single case. It contains a refined grid in both the axial and laterel directions as well as extended inflow and outflow regions. The grid is 110×16 with four degrees of freedom per node (7040 total), less boundary conditions. The original grid had 768 degrees of freedom (less boundary conditions). The degrees of freedom indicate the upper bound on the computational work required to invert the (virtual) left hand side matrix by any given method. As the degrees of freedom go up, the work to solve goes up exponentially and the storage requirements go up linearly (exponentially if a full left hand side matrix is formed and

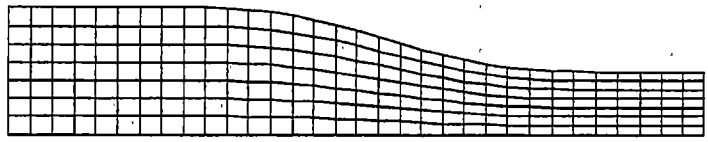


Figure 12-21: Extended In/Outflow Boundaries

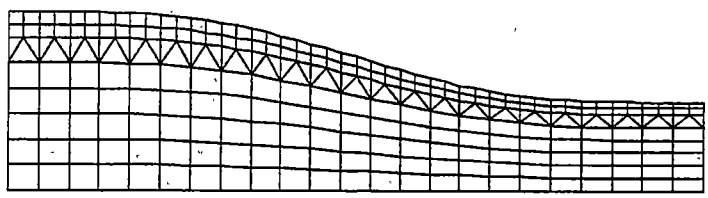


Figure 12-22: Wall Grid Refinement

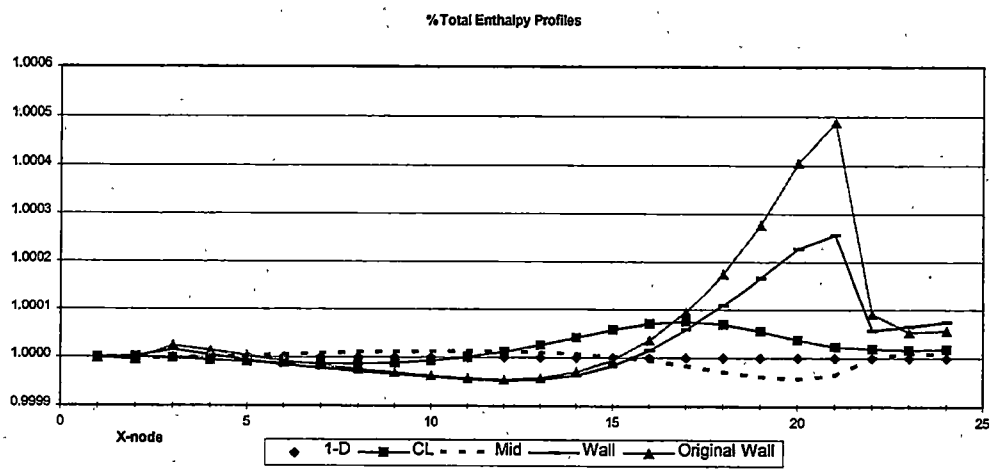


Figure 12-23: % Enthalpy with Wall Refinement

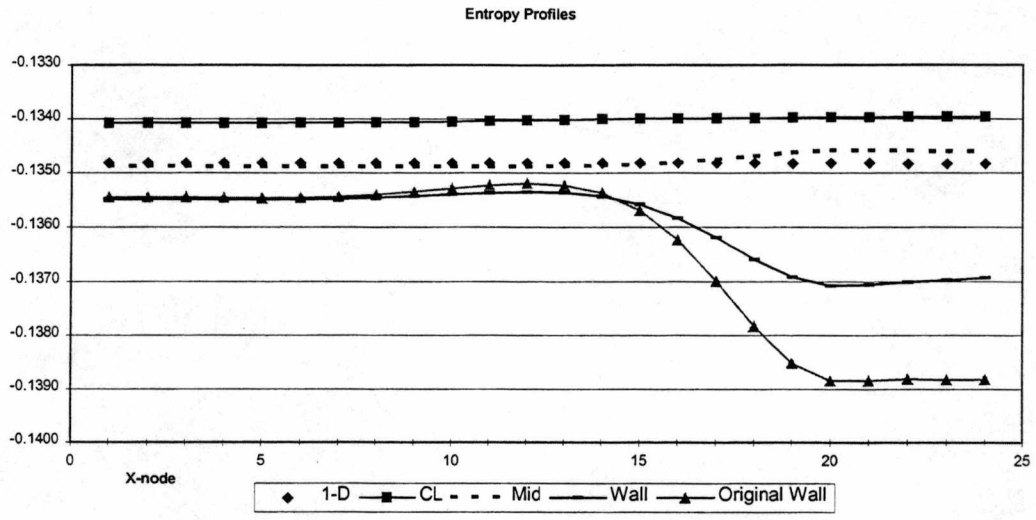


Figure 12-24: Nondimensional Entropy with Wall Refinement

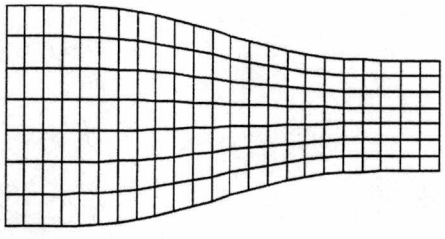


Figure 12-25: Full Nozzle Test Case

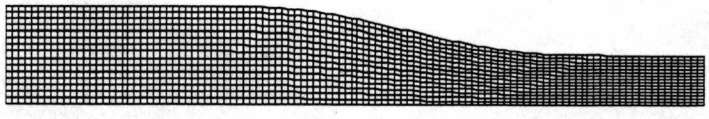


Figure 12-26: Fully Refined Grid Nozzle

stored). The basic nozzle (Figure 12-10) test case took between 30 minutes and 5 hours to run on a 200Mhz PC with all diagnostics running, depending on the method and parameters selected. (This same operations count on a single processor of a modern parallel machine would take less than 7 seconds!) The refined grid case (Figure 12-26) was estimated to need a few days to settle out all residuals but was manually stopped after about 9 hours with a normalized residual around 10^{-2}. The residual surfaces show some interesting patterns at this pre-convergence condition.

Although not converged, the refined grid solution still shows the same basic pattern as the baseline nozzle. Figures 12-27 through 12-31 seem to indicate that grid refinement alone will not resolve the issues already identified. The pressure contours are orthogonal to the centerline about 0.5 diameters upstream and downstream of the convergent region so distance from the boundaries is not an issue.

The pressure profiles in Figure 12-27 show the same bubble as the flow turns back axial after the convergent region. It also shows the pressure increases immediately afterward and then steadily decreases to the exit. The inflow region is similar where the pressure steadily increases to a peak as the nozzle wall starts to turn inward. This pressure ramp did not exist in the straight channel test case so it must be an upstream effect generated by the convergence region. The good news is the minimum pressure point is much closer to the exit pressure. Now the repressurization as the flow enters the straight exit pipe is more curious. Some of this might have damped if the solution had been allowed to continue.

The temperature profiles in Figure 12-28 show a depressed temperature at the wall and centerline and a fair amount of scatter still exists in the exit profile. The relevant point here is the wall temperature overshot the target of ~510R while the core flow is lagging. Experience with the basic nozzle shows this will eventually settle out and the wall temperatures and the mean flow will come much closer to theoretical conditions.

The entropy profiles in Figures 12-29 and 12-30 show a significant difference between the wall and centerline entropy and that of the mean flow. This is based on a slightly different version of the preconditioning matrix but indicates the wall behaves differently than the mean flow even for relatively fine grid spacing. The depressed entropy is on the wall and the centerline in this solution transient and correlates closely with the enthalpy in Figure 12-31. The enthalpy surface is not shown as it follows the entropy surface very closely. The enthalpy profiles are shown for reference.

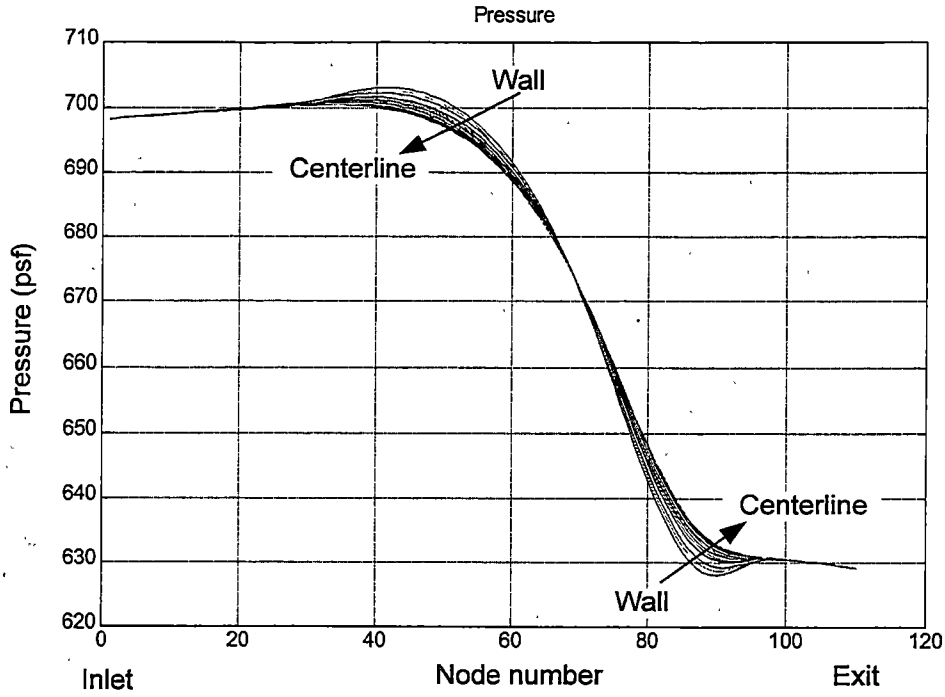


Figure 12-27: Preconvergence Refined Grid Pressure Profile

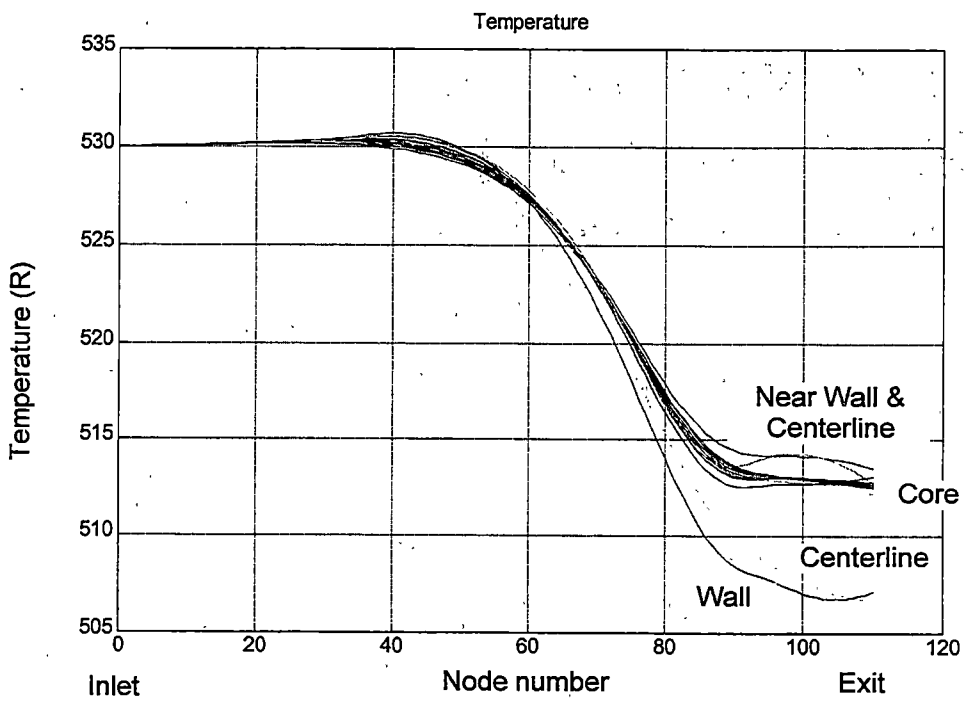


Figure 12-28: Preconvergence Refined Grid Temperature Profile

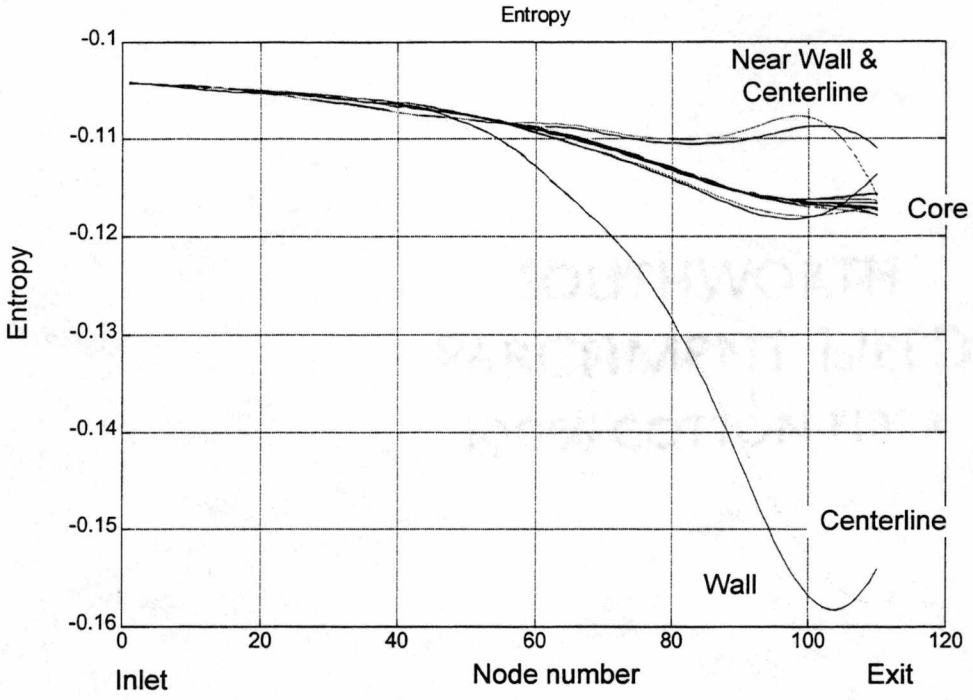


Figure 12-29: Preconvergence Refined Grid Entropy Profile

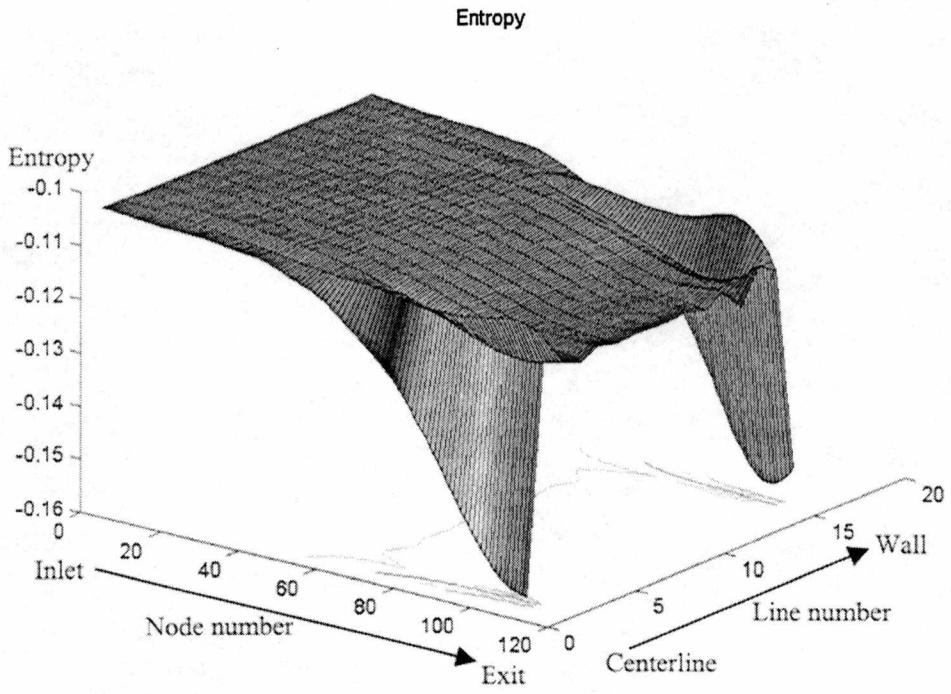


Figure 12-30: Preconvergence Refined Grid Entropy Surface

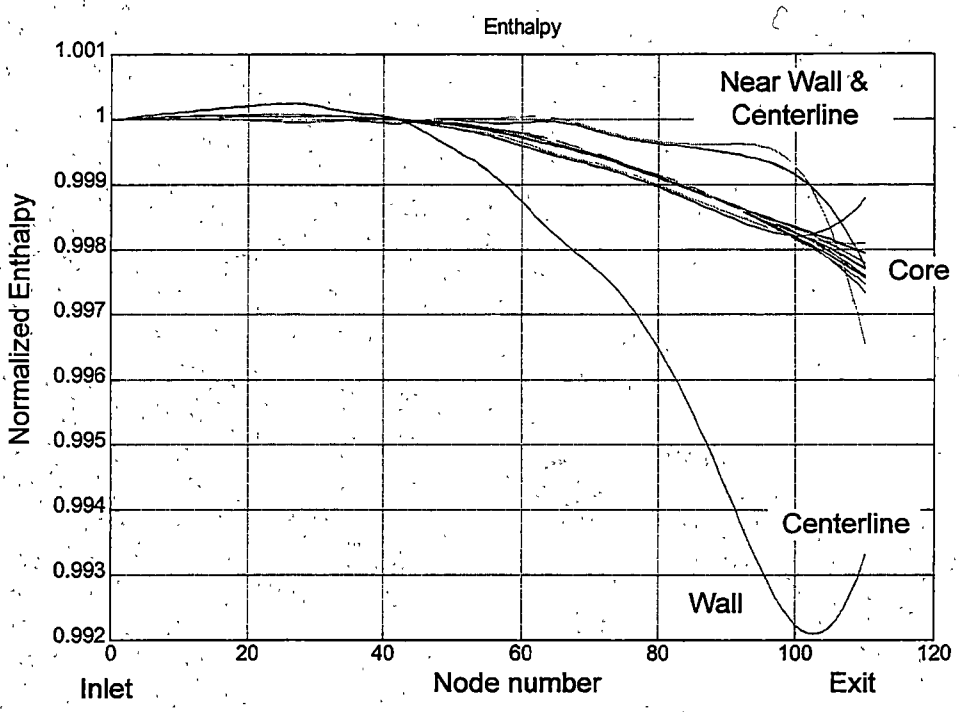


Figure 12-31: Preconvergence Refined Grid Enthalpy Profile

The group of figures (12-32 through 12-38) shows the residual profiles GMRES must eliminate. The TWS/SUPG diffusion is said to be residual based because it essentially works on the residual of a discretized set of equations $\left[N+\tau\left(\frac{\partial N}{\partial x}A_v\Gamma_v^{-1}+\frac{\partial N}{\partial y}B_v\Gamma_v^{-1}\right)\right]\{R\}$. GMRES then works with the residual of this equation set as the data vector b in the linear system $Ax=b$. The Krylov vectors in the left hand side are also based on this right hand side residual by $\{R_{rhs}, AR_{rhs}, A^2R_{rhs},\ldots\}$. In the matrix-free process, a linearized form of the right hand side is used to operate on the Krylov search directions relative to the starting vector. These linearized functions generate similar residuals since they are just linearized versions of the right hand side. The difference in the current search direction residual and the starting direction residual is then used to determine if the search direction provides adequate residual reduction. The process is not unlike going to the optometrist and flipping through various lens shapes and then combining the results to provide 'the best' overall combination. In the residual plots, greater magnitude means the equation is further from convergence, regardless of the sign.

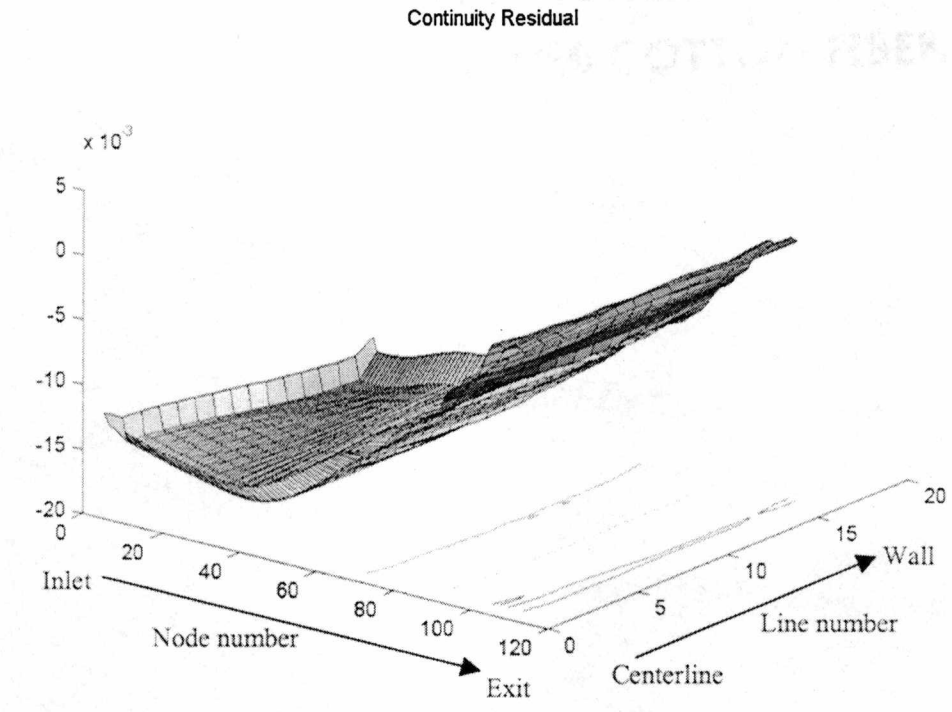


Figure 12-32: Preconvergence Continuity Residual Surface

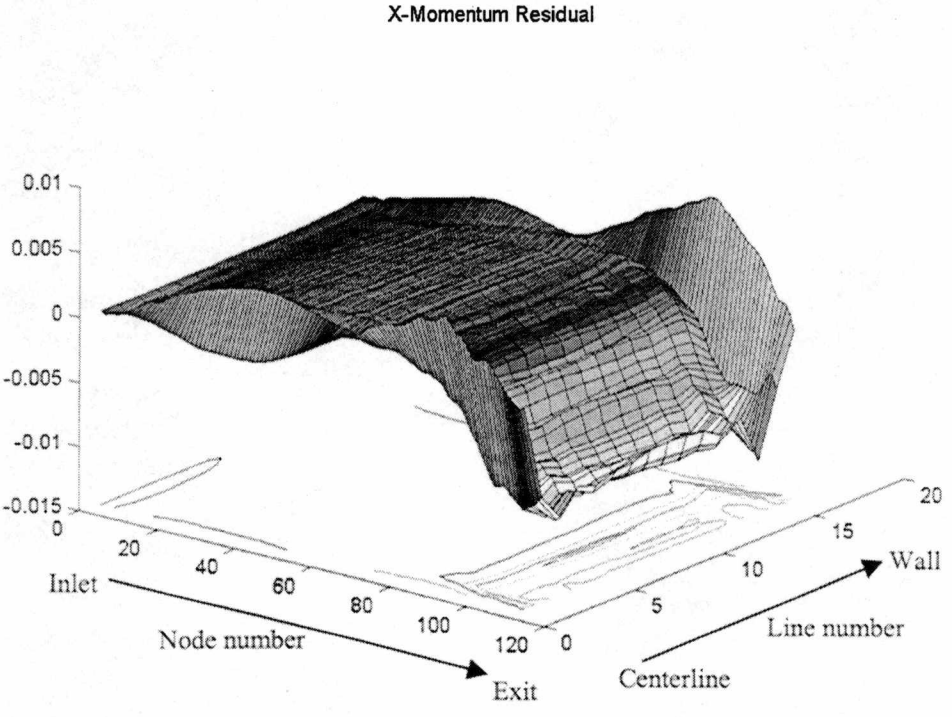


Figure 12-33: Preconvergence X-Momentum Residual Surface

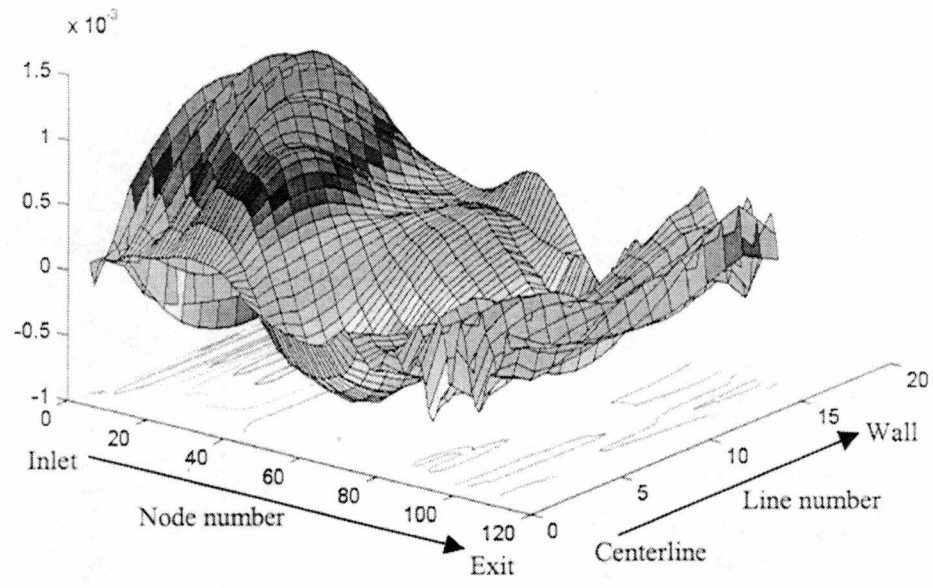


Figure 12-34: Preconvergence Y-Momentum Residual Surface

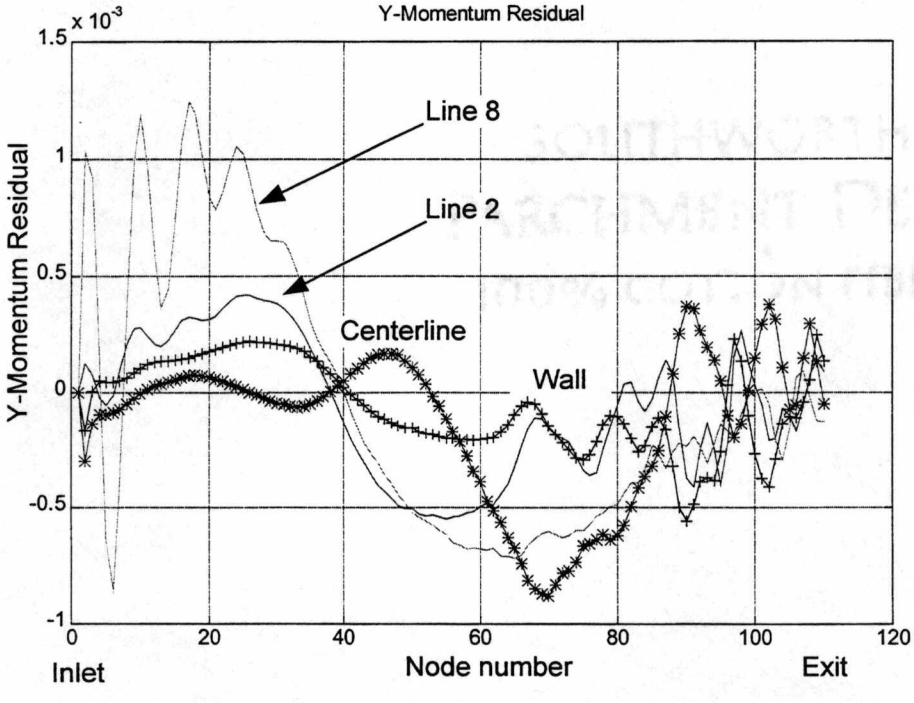


Figure 12-35: Preconvergence Y-Momentum Residual Profile

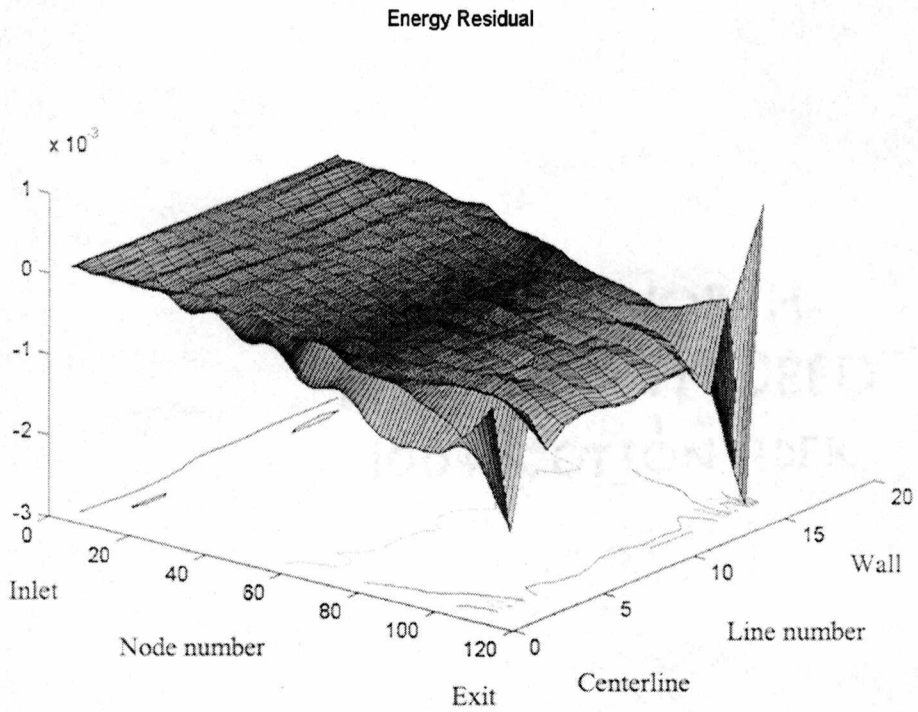


Figure 12-36: Preconvergence Energy Residual Surface

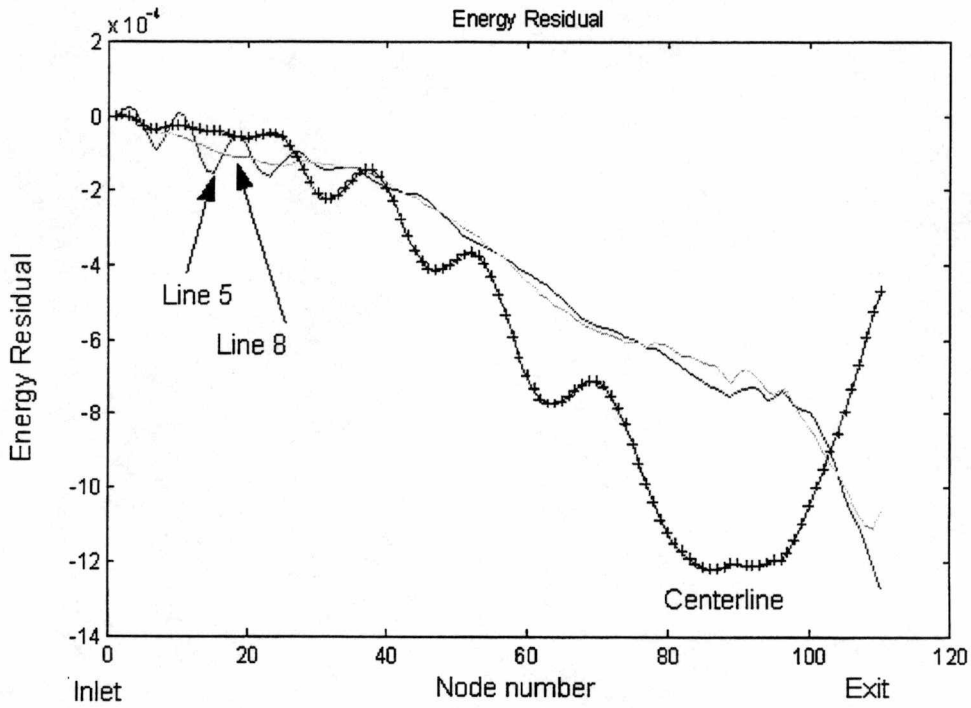


Figure 12-37: Preconvergence Energy Residual Profile

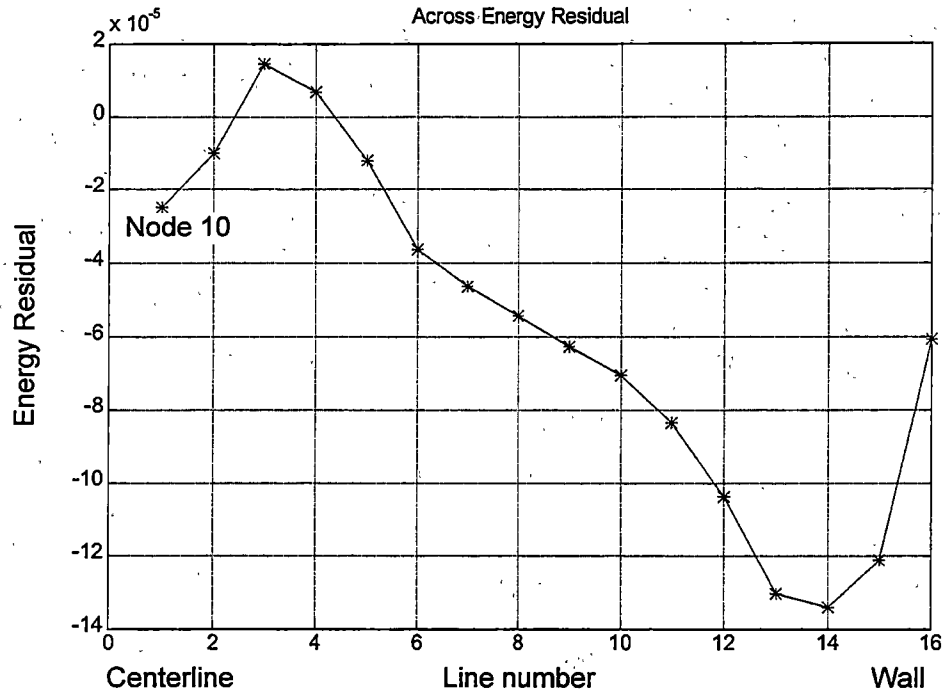


Figure 12-38: Preconvergence Energy Residual Crossflow Profile

The continuity residual surface in Figure 12-32 is fairly smooth and the down stream region seems to be converging before the upstream. The boundaries in the upstream region are also markedly different than the core flow region but in the 'good' direction. A smooth surface is intuitively less complicated and likely easier to solve. GMRES sees the surface as a vector and is looking to find a better answer by searching in a direction A-conjugate (conceptually orthogonal after mapping onto the columns of A) to the residual.

The momentum residuals are a very different story. Figure 12-33 shows the problem the X-momentum equation is having with the wall boundaries. It is disconnected from the main flow and is starting to show a long wave pattern along the wall and some 2Δ waviness normal to the wall because of the steep gradient. The search directions will be selected based on minimizing the norm of the residual so a minor improvement in the majority of points outweighs a major improvement in a small number of points. This suggested some strategies for improvement based on:

1. giving a higher weight to the wall residuals
2. minimizing the wall residuals in a search separate from the rest

3. using a linear combination of norms of the different regions to find the best solution (e.g. instead of $\|R\|$, use $\alpha_1\|R_1\| + \alpha_2\|R_2\| + \ldots + \alpha_i\|R_i\|$ where $\sum_i \alpha_i = 1$)

4. or giving a non-linear weight to outlier points so they will be emphasized in the solution

Only the method giving higher a priori weights to wall boundaries was actually tested. The wall boundary residuals were simply multiplied by a scalar. Without tracking the distributed residual reduction, it would be difficult to prove the benefit. However, this non-linear weighting based on a gradient is the essence behind the shock capturing diffusion operators. The difference is using the residual rather than a flow variable.

Artificial viscosity terms act as a filter and diffuse the energy from waves of various frequencies. The TWS/SUPG term is a particular type of directional filter acting in the flow direction. The τ term was selected and tuned to damp high frequency 2Δ waves in this direction. The residual surfaces show it is effective as there are no 2Δ waves in the flow direction. There are longer waves present in the flow direction and localized 2Δ waves in directions normal to the flow.

The X-momentum residual surface (Figure 12-33) shows a single wave along the wall with wavelength equal to the length of the flow domain. Normal to the flow, a 2Δ pattern is apparent at the wall. The Y-momentum residual surface (Figure 12-34) looks like a (small) mess but various longitudinal cross sections in Figure 12-35 show the frequency content. The highest wave taken along the mid flow region shows three complete waves over 21 nodes or 7 nodes per wave. The waves are the worst in the straight pipe sections where the Y-momentum is small and normal to the main flow and thus the damping direction.

The energy residual surface in Figure 12-36 is the most illustrative with a reasonably flat basic shape but very clear waves in different regions of the flow. The 2Δ waves normal to the flow at the exit are obvious but the inflow region shows a beautiful wave pattern of about 10 nodes per wave normal to the flow (Figure 12-37) and single waves of 16 nodes per wave normal to the flow (Figure 12-38). Meanwhile, the wall seems to have composite wave with a short wave length of 20 nodes per wave decaying into a much longer wave.

The waves are not evident in the primitive physical variables so they would be invisible to a damping scheme based on physical variable gradients. This also clearly highlights the role of the third order (dispersion) term truncated in the Taylor weak statement derivation. The derivation of the Taylor weak statement in Chapter 5 identified the role of the residual and should be explored further in light of the current findings.

The final set of results comes from "doing less" damping rather than more. At the beginning of the discussion about the nozzle test cases, the topic of the magnitude of the diffusion time scale was deferred. The short summary is to say increasing the scalar weight (simply multiply the calculated weight by 2 or 10 or something larger) showed no improvement. Likewise, decreasing the weight sometimes allowed 2Δ waves to form and slowed convergence. The most interesting test started with a converged solution and removed the damping term completely (set $\tau = 0$). After the damping was removed, the solution took four non-linear iterations to re-converge.

There was a visible effect on the profiles for the primitive variables, the entropy and the enthalpy. First, Figure 12-39 shows the entropy profile gets just a little bit worse but not by much. It also picks a little bit of 2Δ waviness but it doesn't diverge. The surprise is in the % enthalpy profile (Figure 12-40). The wall anomaly goes away! The enthalpy shows some waviness in a very close up look but the enthalpy profile is flat when compared to the converged solution for the baseline case.

Figures 12-41 through 12-44 show the changes in the primitive variables (original - current) along the wall when the damping is removed. All show the characteristic 2Δ expected for a Galerkin approach. The flow would have diverged quickly if started from the initial conditions but starting from the converged solution kept out most of the problems. All the variables changed some but the x-velocity (Figure 12-43) was the real culprit driving the enthalpy profile. The x-velocity was over $2\,fps$ higher with damping than without damping.

The pressure and temperature changes (Figures 12-41 and 12-42) were not in the right direction to eliminate the entropy problem as there was little change when the damping was removed. Unlike the enthalpy, the damping term is not causing the problem but it is not eliminating it either.

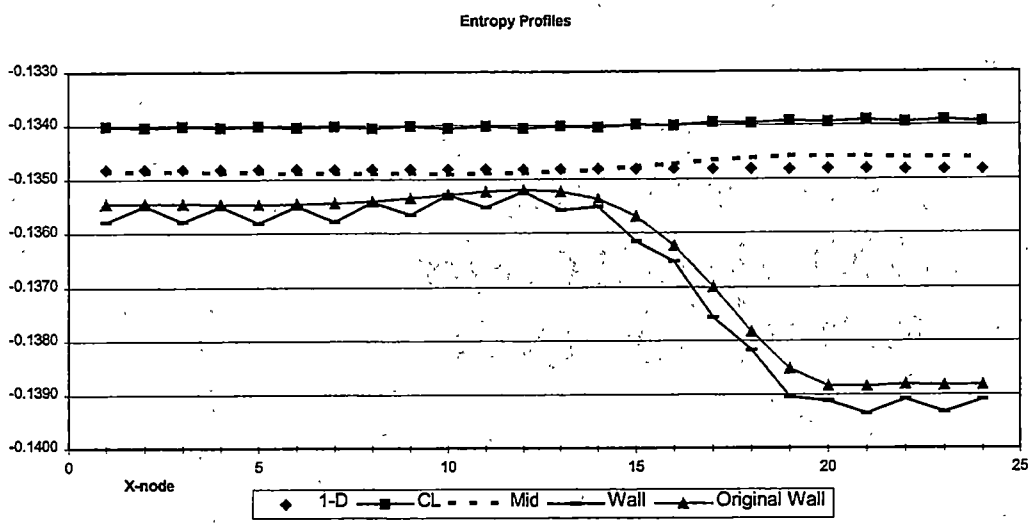


Figure 12-39: Entropy w/o Damping

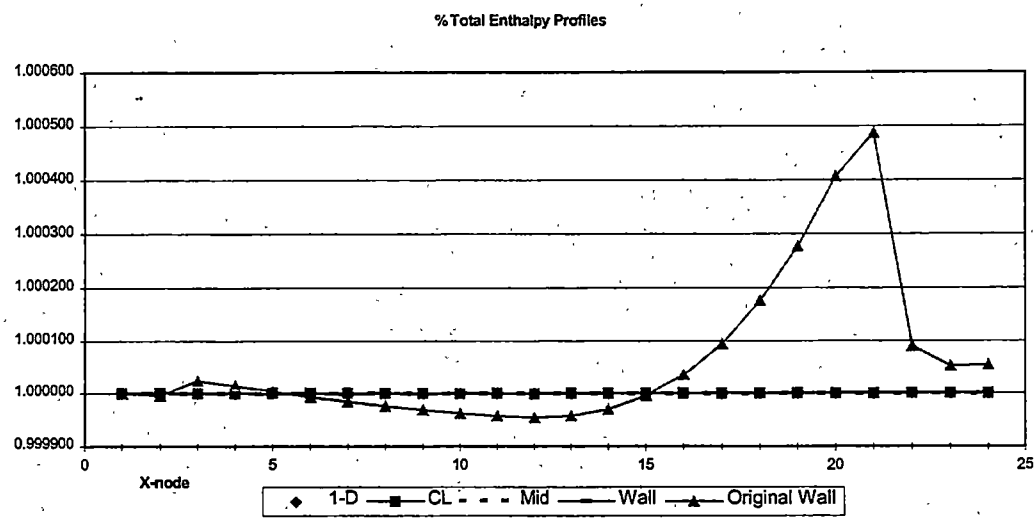


Figure 12-40: % Enthalpy w/o Damping

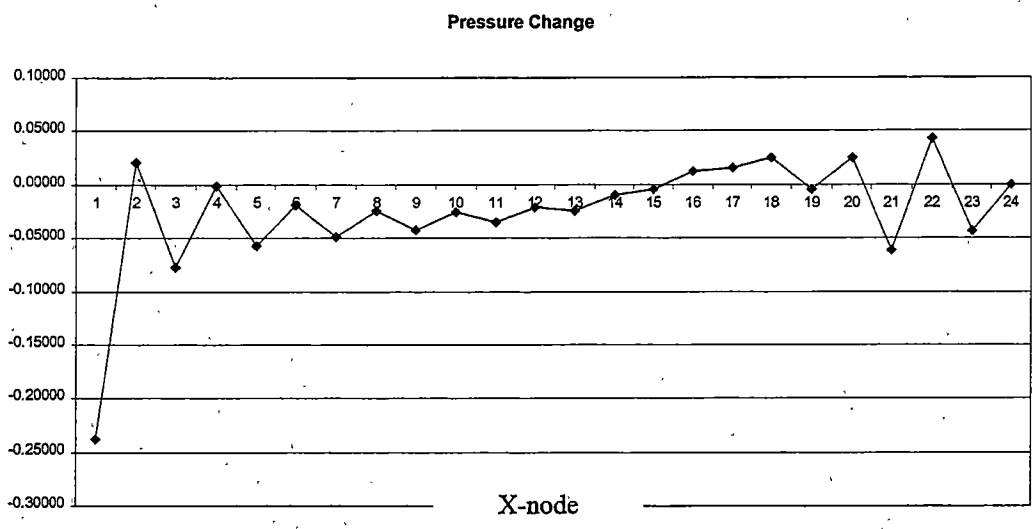


Figure 12-41: Wall Pressure Change w/o Damping

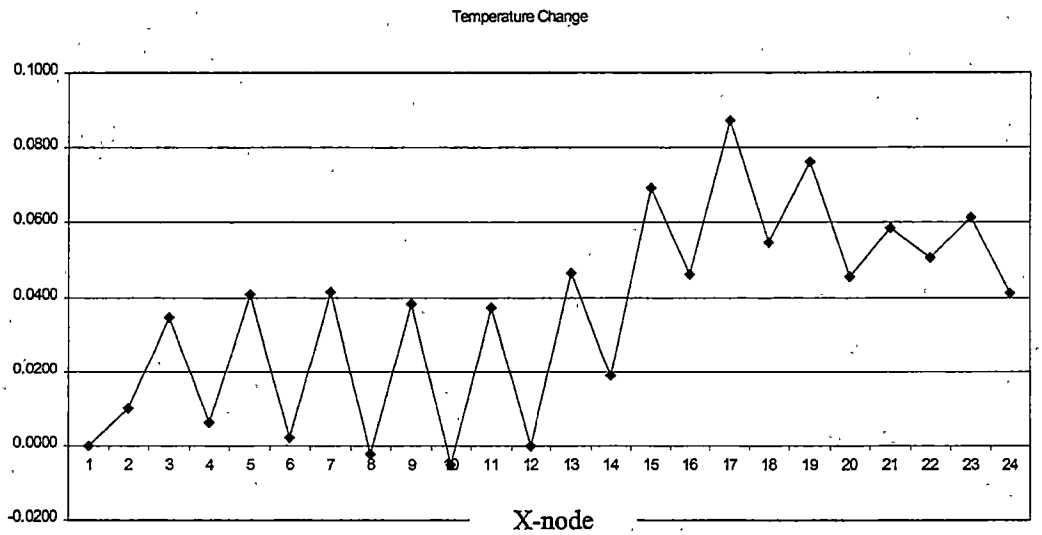


Figure 12-42: Wall Temperature Change w/o Damping

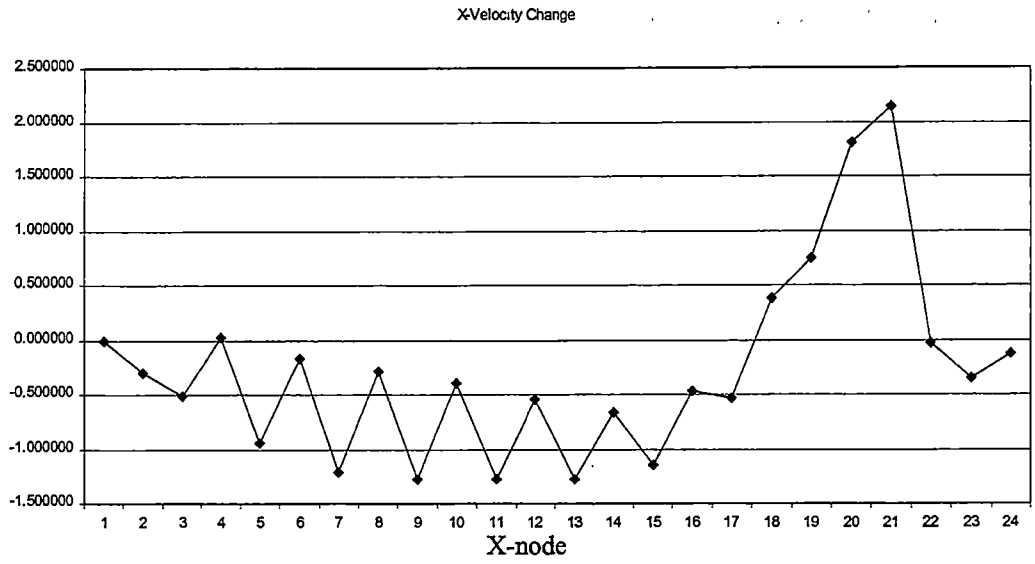


Figure 12-43: Wall X-Velocity Change w/o Damping

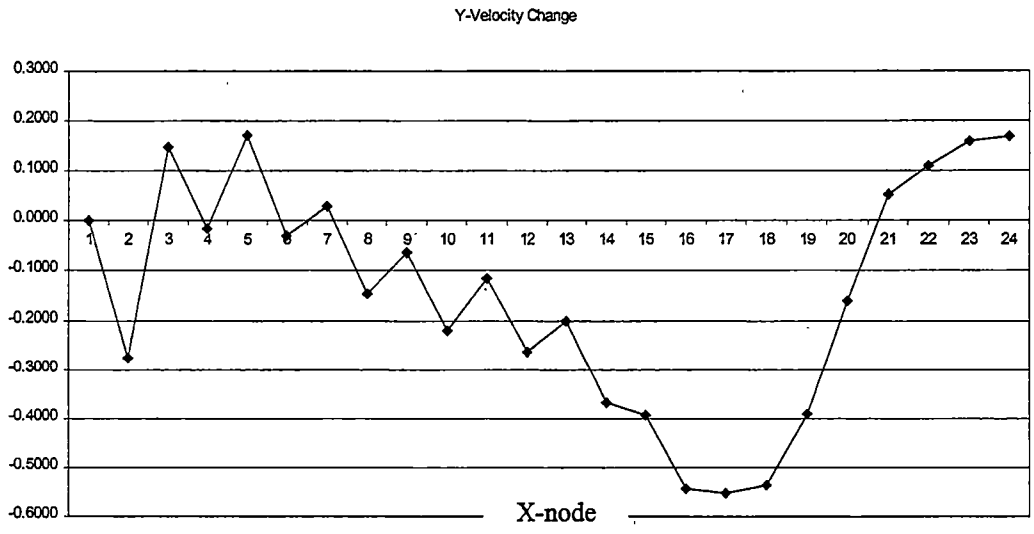


Figure 12-44: Wall Y-Velocity Change w/o Damping

The profiles at the walls for the nozzle and straight channel test cases are not unprecedented as [Hughes, et.al. II, 1986) shows advection test cases with similar features at a boundary. In that case, the profiles were in the primary variable and the boundaries were inflow/outflow boundaries. He added different versions of a shock capturing term and eliminated the boundary problem, even though there were no shocks. [Tezduyar and Park, 1986] repeated the test cases with their own version of the SUPG term and had similar results. They also controlled the problem with a shock capturing term. [LeBeau, et.al., 1993] also noted the limitations on accuracy of the SUPG method in the conservation variables without shock capture. They eliminated the accuracy problem for the conservation variables with the addition of an appropriate term.

The primitive variable set $\{p,u,v,T\}$ may be too insensitive to be the object of a non-linear gradient operator. The entropy, enthalpy, or residual vectors may be more appropriate. It is also important to reiterate the need for controlled diffusion in directions other than the streamline. These have been developed for entropy and conservation variables but not primitive variables.

Summary

The preconditioned Taylor weak statement derivation shows a strong connection with the Streamwise Upwind Petrov Galerkin method. The detailed analysis available with the Taylor weak statement provided a theoretical basis for the TWS/SUPG damping term. This led to the development of families of damping terms for the compressible fluid flow equation set using primitive variables and time derivative preconditioning. It also provided a framework for evaluating damping terms developed by ad hoc means and led to the creation of a damping term superior to any found in the literature. The TWS/SUPG equation set still has some anomalies at solid wall boundaries associated with the damping term that limit the rate of convergence and the detailed accuracy of the converged solution. However, the errors are typically less than 1% and the accuracy is sufficient for testing various approaches to coupled fluid-structure interactions.

Residual scaling greatly improved the rate of convergence of the fluid flow equation set and should be considered essential. A nondimensionalized equation set would be less sensitive to scaling issues

but not immune. Residual scaling is similar in concept to row or column pivoting in used in direct methods and is very easy to implement. It may be possible to improve the quality of solutions by using residual scaling to emphasis particular regions or variables within the flow.

The TWS/SUPG finite element equations set is applicable to any set of polynomial basis functions and was successfully tested with both triangular and quadrilateral elements.

CHAPTER 13

ANALYSIS OF FLUID FLOW PRECONDITIONING RESULTS

The fluid flow preconditioning approach was successful and deserves further investigation. Several essential elements must be correct to enjoy the full benefit but they are somewhat independent of each other and the parametric studies to follow will show the improvements are always in the direction of the physics based preconditioning approach. As a reminder, the approach is to add an artificial mass matrix to the Jacobian in the Newton iteration and treat the linear GMRES iterations as pseudo time steps. This looks like $Q\frac{\partial w}{\partial t}+\frac{\partial R(w_c)}{\partial w_c}\Delta w=-R(w_c)$ or, in discrete form, $\left[\frac{1}{\Delta t'}Q+J\right]\Delta w=-R$ where $Q\frac{\partial w}{\partial t}$ is selected to improve the condition number of the left hand side matrix.

In the cases tested, including the mass matrix worked better than leaving it out but there were nuances to the form and treatment of the matrix. Also, a Taylor Weak Statement/Streamwise Upwind Petrov Galerkin (TWS/SUPG) term was developed around the primitive flow variables $\{p,u,v,T\}$ to stabilize the fluid flow equations. In the development, the inverse of the mass matrix, Q^{-1}, shows up in the stabilization terms. In a sense, this is also an artificial term added to enhance the computation so it too can be viewed as a possible source of additive preconditioning. Finally, an effective form of the elusive time scale parameter, τ, associated with the TWS/SUPG term was finally uncovered and implemented to great effect. It would be difficult to rank order the importance of one over the other or the importance of the fluid equation set relative to the number of Krylov vectors and restarts allowed.

TWS/SUPG Terms

The mass matrix implemented is based on the time derivative preconditioning approach of [Merkle, 1995]. The basic process is to control the eigenvalues in the continuum, then don't destroy the good conditioning with the discretization. If the various time scales are controlled relative to the discretization, the condition number of the left hand side matrix should be controlled. This is not the only consideration for preconditioning a GMRES solver, but it's a reasonable start. In reality, the GMRES method is controlled

by the condition number of the eigenvector matrix of the left hand side, X, where $X\Lambda X^{-1} = \left[\frac{1}{\Delta t'}Q + J\right]$ rather than eigenvalues of the left hand side matrix. This may help explain why lumping the mass matrix (adding diagonal dominance) improved the rate of convergence.

As for the TWS/SUPG term, the full weight function used in the weak statement is given by

$$W = N + \tau\left(\frac{\partial N}{\partial x}A_v Q_v^{-1} + \frac{\partial N}{\partial y}B_v Q_v^{-1}\right) \qquad \text{(13-1)}$$

The subscript on the matrices indicates a relationship to the primitive variable set. I considered three main variations to the terms in the weight function. First, Q_v^{-1} may be based on either the physical Jacobian between primitive and conserved variables or an artificial mass matrix selected for preconditioning. Second, if the artificial mass matrix is used, appropriate parameters are needed to form the preconditioning matrix. Third, a robust calculation is needed for the scalar time parameter, τ, to avoid under or over diffusion. These will be connected in the parametric studies and the trends are fairly clear.

First, the artificial preconditioning matrix produced better results than the physical Jacobian associated with the change of variables regardless of τ or the preconditioning parameters selected. This is not to say anything would work for a preconditioning matrix but the range of acceptable parameters outlined in [Merkle, 1995] all worked better than the physical Jacobian matrix. Without elaborating on the different τ definitions (A, E, or N) just yet, Table 13-1 shows the time (in seconds) required for convergence when the preconditioning matrix and the physical Jacobian were compared head to head for two different test cases.

In Table 13-1, $inv(LV)$ indicates the inverse of the preconditioning matrix is used in the weight term and $inv(QV)$ indicates the inverse of the physical Jacobian is used. The nozzle flow conditions

Table 13-1: Physical vs. Preconditioned Mass

		Jacobian	
Tau	Case	inv(LV)	inv(QV)
A	Nozzle	1866	***
E	Nozzle	2251	***
N	SRAND	132	162

*** did not converge

were in the 250 (inlet) to 550 fps (outlet) range at sea level conditions and are described in Figure 12-10. SRAND is a 4×4 node coupon with walls (Figure 12-3) where a random number generator sets the initial conditions. The two SRAND cases used the same initial conditions. The results in Table 13-1 show the preconditioning matrix led to faster convergence in the coupon test and was necessary for convergence in the nozzle tests. Three different τ parameters are shown, all with the same outcome.

Next, various parameters for the preconditioning matrix were tested. [Merkle, 1995] identifies four parameters to be considered: ρ_p', ρ_T', H_p', and H_T'. In physical terms, these are the partial derivatives of the density and enthalpy with respect to the primitive variables, pressure and temperature. The trick is to replace the physical partial derivatives with values selected to improve the eigenvalue condition number of the local flow.

The nozzle tests were forgiving and all combinations worked...but not in the SRAND tests. Table 13-2 shows the how the enthalpy based parameters fared with $\rho_p' = 1/V_{ref}^2$ and $\rho_T' = 0$. It shows time to convergence in seconds for different values of H_T' and H_p' for the test case SRAND.

The H_p' parameter is defined by $H_p' = 1 - \rho H_p$ so the best result is actually the physical one since $H_p = 0$ for an ideal gas. Surprisingly, H_T' was intolerant so the physical case worked the best here as well.

The other two preconditioning parameters varied were $\rho_p' = \frac{1}{V_{ref}^2}$ and $\rho_T' = K_T \rho_T$. The reference velocity starts with the local particle velocity at each node (variation A.)

Table 13-2: Preconditioning Parameters

	Hp=0	Hp=1
Ht=Cp	***	132
Ht=R	-	***

*** did not converge

$$V_{ref} \text{ case } A = \sqrt{u^2 + v^2} \tag{13-2}$$

In variation B, a Courant number based velocity is included similar to that shown in [Venkateswaran and Merkle, 1995].

$$V_{ref} \text{ case } B = \sqrt{\left(\frac{\ell_x}{\pi \Delta t^*}\right)^2 + \left(\frac{\ell_y}{\pi \Delta t^*}\right)^2 + u^2 + v^2} \tag{13-3}$$

where ℓ_x, ℓ_y are reference lengths and Δt^* is the pseudo time step. Meanwhile, the term K_T is used to scale the physical partial derivative of the density with respect to temperature. The two extremes of $K_T = 0$ and $K_T = 1$ were tested and provided the results shown in Table 13-3.

Setting $\rho_p' = 1/V_{ref}^2$ where $V_{ref}^2 = u_{local}^2 + v_{local}^2$ is a standard approach with roots all the way back to Chorin's artificial compressibility [Chorin, 1967], but $K_T = 0$ is a choice to simplify the matrix inversion. As it turns out, $K_T = 1$ provides the fastest convergence but it did not have a strong impact. Table 13-3 gives the time to convergence in seconds for the basic nozzle test case with two different versions (A and N) of the τ parameter.

I assumed the four preconditioning parameters were independent but that's really not the case. The TWS/SUPG term multiplies the conservative form of the flux gradients on the right hand side and the linearized flux gradient on the left hand side creating the four products

$$\tau \frac{\partial}{\partial x} A_v \Gamma_v^{-1} A_v \frac{\partial}{\partial x} \begin{Bmatrix} p \\ u \\ v \\ T \end{Bmatrix}, \tau \frac{\partial}{\partial x} A_v \Gamma_v^{-1} B_v \frac{\partial}{\partial y} \begin{Bmatrix} p \\ u \\ v \\ T \end{Bmatrix}, \tau \frac{\partial}{\partial y} B_v \Gamma_v^{-1} A_v \frac{\partial}{\partial x} \begin{Bmatrix} p \\ u \\ v \\ T \end{Bmatrix}, \tau \frac{\partial}{\partial y} B_v \Gamma_v^{-1} B_v \frac{\partial}{\partial y} \begin{Bmatrix} p \\ u \\ v \\ T \end{Bmatrix} \tag{13-4}$$

Table 13-3: Preconditioning Parameters for Nozzle Test Case

Tau	Kt	Vref	
		A	B
N	0	6497	8007
A	0	1866	-
A	1	1711	-

Each contains the term $\Gamma_v^{-1}A_v$ or $\Gamma_v^{-1}B_v$. Expanding the first of these gives

$$\Gamma_v^{-1}A_v = \begin{pmatrix} \frac{u\left[\rho_T'\left(1-\rho H_p\right)+\rho\rho_p H_T'\right]}{d} & \frac{\rho^2 H_T'}{d} & 0 & \frac{\rho u\left(\rho_T H_T' - \rho_T' H_T\right)}{d} \\ 1/\rho & u & 0 & 0 \\ 0 & 0 & u & 0 \\ \frac{u\left[\rho_p H_p' - \rho_p'\left(1-\rho H_p\right)\right]}{d} & \frac{\rho H_p'}{d} & 0 & \frac{u\left(\rho\rho_p' H_T - \rho_T H_p'\right)}{d} \end{pmatrix} \tag{13-5}$$

where $d = \rho\rho_p' H_T' + \rho_T' H_p'$. Multiplying this by the x-derivative of the primitive variables leads to

$$continuity = \frac{u\left[\rho_T'\left(1-\rho H_p\right)+\rho\rho_p H_T'\right]}{d}\frac{\partial p}{\partial x} + \frac{\rho^2 H_T'}{d}\frac{\partial u}{\partial x} + \frac{\rho u\left(\rho_T H_T' - \rho_T' H_T\right)}{d}\frac{\partial T}{\partial x} \tag{13-6}$$

$$energy\ eqn = \frac{u\left[\rho_p H_p' - \rho_p'\left(1-\rho H_p\right)\right]}{d}\frac{\partial p}{\partial x} + \frac{\rho H_p'}{d}\frac{\partial u}{\partial x} + \frac{u\left(\rho\rho_p' H_T - \rho_T H_p'\right)}{d}\frac{\partial T}{\partial x} \tag{13-7}$$

for the continuity and energy equations, respectively. The momentum equations do not contain preconditioning parameters so they are excluded from this analysis.

First, $d \neq 0$ so $\rho\rho_p' H_T' \neq -\rho_T' H_p'$. Substituting for ρ and ρ_T' in $d = \rho\rho_p' H_T' + \rho_T' H_p'$

$$\frac{P}{RT}\rho_p' H_T' \neq -\frac{-K_T P}{RT^2}H_p' \tag{13-8}$$

or

$$\rho_p' H_T' T \neq K_T H_p' \tag{13-9}$$

Setting any of the parameters (ρ_p', ρ_T', H_p', or H_T') equal to zero always satisfies the constraint $d \neq 0$ as long as one is not set to zero on the opposite side of the equation. Substituting the parameters selected previously, $\frac{C_p T}{u^2+v^2} \neq 1$ is another constraint. The flow would have to be supersonic to violate the constraint and the preconditioning is typically turned off as the flow reaches transonic speeds so there is no real danger of violating it.

Rearranging the continuity equation **(13-6)** gives

$$
\begin{aligned}
continuity \times d &= u\rho_T' \frac{\partial p}{\partial x} - u\rho_T' \rho H_p \frac{\partial p}{\partial x} + u\rho\rho_p H_T' \frac{\partial p}{\partial x} + \rho^2 H_T' \frac{\partial u}{\partial x} + \rho u \rho_T H_T' \frac{\partial T}{\partial x} \\
&\quad - \rho u \rho_T' H_T \frac{\partial T}{\partial x} \\
&= \left[u\rho_T' \frac{\partial p}{\partial x} - \rho u \rho_T' H_T \frac{\partial T}{\partial x} \right] + \left[u\rho\rho_p H_T' \frac{\partial p}{\partial x} + \rho^2 H_T' \frac{\partial u}{\partial x} + \rho u \rho_T H_T' \frac{\partial T}{\partial x} \right] \\
&\quad - u\rho_T' \rho H_p \frac{\partial p}{\partial x} \\
&= u\rho_T' \left[\frac{\partial p}{\partial x} - \rho H_T \frac{\partial T}{\partial x} \right] + \rho u H_T' \left[\rho_p \frac{\partial p}{\partial x} + \rho \frac{1}{u} \frac{\partial u}{\partial x} + \rho_T \frac{\partial T}{\partial x} \right] - u\rho_T' \rho H_p \frac{\partial p}{\partial x}
\end{aligned}
\tag{13-10}
$$

Substituting for ρ, ρ_p, ρ_T, H_T, and H_p gives

$$
\begin{aligned}
continuity \times d &= u\rho_T' \left[\frac{p}{p} \frac{\partial p}{\partial x} - \frac{p}{RT} C_p \frac{\partial T}{\partial x} \right] + \rho u H_T' \left[\frac{\rho}{p} \frac{\partial p}{\partial x} + \rho \frac{1}{u} \frac{\partial u}{\partial x} - \frac{\rho}{T} \frac{\partial T}{\partial x} \right] \\
&= pu \frac{-K_T \rho}{T} \frac{R}{R} \left[\frac{1}{p} \frac{\partial p}{\partial x} - \frac{C_p}{R} \frac{1}{T} \frac{\partial T}{\partial x} \right] + \rho^2 u H_T' \left[\frac{1}{p} \frac{\partial p}{\partial x} + \frac{1}{u} \frac{\partial u}{\partial x} - \frac{1}{T} \frac{\partial T}{\partial x} \right] \\
&= \rho^2 u K_T \left[C_p \frac{1}{T} \frac{\partial T}{\partial x} - R \frac{1}{p} \frac{\partial p}{\partial x} \right] + \rho^2 u H_T' \left[\frac{1}{p} \frac{\partial p}{\partial x} + \frac{1}{u} \frac{\partial u}{\partial x} - \frac{1}{T} \frac{\partial T}{\partial x} \right]
\end{aligned}
\tag{13-11}
$$

The first term on the right hand side is recognized as the enthalpy variation of the entropy (Gibb's) equation from thermodynamics where $ds = C_p \frac{dT}{T} - R \frac{dp}{p}$. K_T directly scales its influence in the continuity equation. Since the continuity residual must ultimately go to zero, this equates to enforcing $ds = 0$ as long as $K_T \neq 0$. K_T doesn't appear anywhere else so it's sole purpose amounts to scaling the entropy condition. It strikes me as odd that it should appear in the continuity equation rather than the energy equation.

The second term on the right hand side of equation **(13-11)** is the derivative of the mass flux where $d(\rho u) = \rho u \left(\frac{dp}{p} + \frac{du}{u} - \frac{dT}{T} \right)$. The continuity equation can now be written

$$
continuity \times d = \rho \left[K_T \rho u \frac{\partial s}{\partial x} + H_T' \frac{\partial (\rho u)}{\partial x} \right] \tag{13-12}
$$

Similar analysis shows the energy equation form to be

$$energy \times d = H_p' \frac{\partial(\rho u)}{\partial x} - \rho_p' u \frac{\partial p}{\partial x} \qquad \text{(13-13)}$$

It appears the four preconditioning parameters could be optimized for best influence over entropy, mass conservation, and work. The entropy and work were identified as areas of concern in Chapter 12.

The final influence factor in the TWS/SUPG term is the intrinsic time scale, τ. It evolved quite a bit over the course of the investigation, providing significant improvements in performance. The various parametric studies were performed at different times so τ and other parameters may be different between tables. Comparisons within a table are identified with a letter A through N. They are not in chronological order in Figure 13-1 so here's a brief summary.

Early on, it was unclear whether the time scale should be localized to the element or reflect some sort of global time scale. Literature investigations and some misguided one-dimensional Von Neuman analyses produced various combinations and assorted terms with π in the length scale. Eventually, type N came to the front as the "best of breed." This was typically coupled with too many Krylov vector restarts so the convergence times were very long. There was no requirement for the numerous restarts; I just didn't know any better.

Type N was used for most of the early part of the effort and it wasn't until much later, after I had started a multidimensional Von Neuman analysis of the Taylor Weak Statement derivation, that the idea of using a wave number vector came out. This gave me a basis to analyze the different approaches and develop specific, targeted improvements. Many of the parameters reported in the literature have a solid basis but they are not as good as they can be. There were quite a few more parameters tested but none are worth mentioning. Several of the ones shown in the figure only vary by a constant factor. Other factors, such as multiplying or dividing by 10, were tested just to check the influence of scaling but the ones listed had some logic behind them at the time. A version of type A was also tested where the acoustic velocity, c, was replaced by the reference velocity from the preconditioning parameter. This was an attempt to simulate the preconditioned eigenvalues. It is not really the correct form of the preconditioned eigenvalues but was tested anyway. Types J and L contain the correct preconditioned eigenvalues. It turns out $A_v \Gamma_v^{-1}$ has the same eigenvalues as $\Gamma_v^{-1} A_v$.

$$A:\quad k_{direction} = \left\{ {}^{1}\!/_{2\Delta x}, {}^{1}\!/_{2\Delta y} \right\},\quad k_{magnitude} = \frac{\sqrt{\Delta x^2 + \Delta y^2}}{2\Delta x \Delta y},\quad \tau = \frac{1}{\vec{u}\cdot\vec{k} + c\|k\|}$$

$$B:\quad k_{direction} = \{u, v\},\quad k_{magnitude} = \frac{2}{\sqrt{\Delta x^2 + \Delta y^2}},\quad \tau = \frac{1}{\vec{u}\cdot\vec{k} + c\|k\|}$$

$$C:\quad k_{direction} = \{u, v\},\quad k_{magnitude} = \frac{\pi}{\sqrt{\Delta x^2 + \Delta y^2}},\quad \tau = \frac{1}{\vec{u}\cdot\vec{k} + c\|k\|}$$

$$D:\quad k_{direction} = \{u, v\},\quad k_{magnitude} = \frac{1}{2\sqrt{\Delta x^2 + \Delta y^2}},\quad \tau = \frac{1}{\vec{u}\cdot\vec{k} + c\|k\|}$$

$$E:\quad \vec{k} = \{2\Delta x, 2\Delta y\},\quad k_{magnitude} = \frac{\sqrt{\Delta x^2 + \Delta y^2}}{2\Delta x \Delta y},\quad \tau = \frac{1}{\vec{u}\cdot\vec{k} + V_{ref}\|k\|}$$

$$F:\quad \min\left[\frac{\Delta x}{|u| + c}, \frac{\Delta y}{|v| + c}\right]$$

$$G:\quad \min\left[\frac{2\Delta x}{|u| + c}, \frac{2\Delta y}{|v| + c}\right]$$

$$H:\quad \max\left[\frac{2\Delta x}{|u| + c}, \frac{2\Delta y}{|v| + c}\right]$$

$$I:\quad k_{direction} = \{u, v\},\quad k_{magnitude} = \frac{1}{\sqrt{\Delta x^2 + \Delta y^2}},\quad \tau = \frac{1}{\vec{u}\cdot\vec{k} + c\|k\|}$$

$$J:\quad k_{direction} = \left\{ {}^{1}\!/_{2\Delta x}, {}^{1}\!/_{2\Delta y} \right\},\quad k_{magnitude} = \frac{\sqrt{\Delta x^2 + \Delta y^2}}{2\Delta x \Delta y},\quad \tau = \frac{1}{\vec{u}\cdot\vec{k} + \|k\| c_{preconditioned}}$$

$$K:\quad k_{direction} = \left\{\frac{\nabla\|Q\|}{\|\nabla\|Q\|\|}\right\},\quad k_{magnitude} = \max(\Delta x \cdot k1, \Delta y \cdot k2),\quad \tau = \frac{1}{\vec{u}\cdot\vec{k} + c\|k\|}$$

$$L:\quad k_{direction} = \left\{\frac{\nabla\|Q\|}{\|\nabla\|Q\|\|}\right\},\quad k_{magnitude} = \max(\Delta x \cdot k1, \Delta y \cdot k2),\quad \tau = \frac{1}{\vec{u}\cdot\vec{k} + \|k\| c_{preconditioned}}$$

$$M:\quad \tau = \frac{\ell_{reference}}{2\left(\|\vec{u}\| + c\right)},\quad \ell_{reference} = \frac{\sqrt{\left(x_{\max} - x_{\min}\right)^2 + \left(y_{\max} - y_{\min}\right)^2}}{\pi}$$

$$N:\quad \tau = \frac{\ell_{reference}}{2\left(\|\vec{u}\| + c\right)},\quad \ell_{reference} = \sqrt{\Delta x^2 + \Delta y^2}$$

Figure 13-1: Tau Forms

Table 13-4 shows the head to head comparison of the various methods. All the nozzle cases used the same number of Krylov vectors (40) with 5 restarts and an Eisenstat-Walker starting goal of 0.5. Type N was used extensively early in the effort and was a logical choice. The speedup with the better choices is obvious.

Type B succeeded Type N as the workhorse. It uses a reference length of twice the maximum diagonal of an element but projects it in the direction of the velocity vector. This was previously explained in the theoretical analysis (see Chapter 5). Type A was developed later and is the clear choice. It targets 2Δ waves and correctly combines the components of the velocity vector in the possible wave directions. It's variant using the preconditioned acoustic speed is a close second.

Type K shows the term recommended by [Aliabadi and Tezduyar, 1993]. Type L is a variant of K using the preconditioned acoustic velocity.

Table 13-4: Convergence Times for Tau Cases

	Kt		
	0	0	1
Tau	SRAND	Nozzle	Nozzle
A	-	1866	1711
B	-	2222	-
C	-	2382	-
D	-	2475	-
E	-	2251	-
F	-	2470	-
G	-	1995	-
H	-	2455	-
I	-	2324	-
J	-	-	1832
K	-	-	2571
L	-	-	2114
M	24*	-	-
N	13*	4552	-

* nonlinear iterations, others in second

Table 13-4 also shows an improvement stemming from changing the preconditioning parameter K_T from 0 to 1. It didn't fix the problem with the entropy or enthalpy described at length in Chapter 12 but it did slightly improve the convergence time (see Tables 13-3 and 13-4) for the same accuracy. There is probably more to be found here given the form of equation **(13-12)** but it was not investigated any further in this effort.

A shock capturing term was not developed for the primitive variable set of equations. It has been discussed throughout but would likely require a focused effort based on the conditions already mentioned.

Newton Time Stepping with Time Derivative Preconditioning

The results have already shown the need for the preconditioning matrix (inverse) in the TWS/SUPG term and the proper time scale (τ parameter) to use. The theoretical development shows the time scale is independent of the time step (physical or pseudo time) used in the Newton statement.

All the fluid flow cases tested indicate a benefit associated with including a pseudo time term in the Newton statement. The cases are admittedly simple and all use basically the same flow conditions so the benefit admittedly may not be general.

Figure 13-2 shows a short series of tests on the SRAND test case using local time stepping. The grid is uniform so cell size and aspect ratio is not an issue. However, a random number generator set the

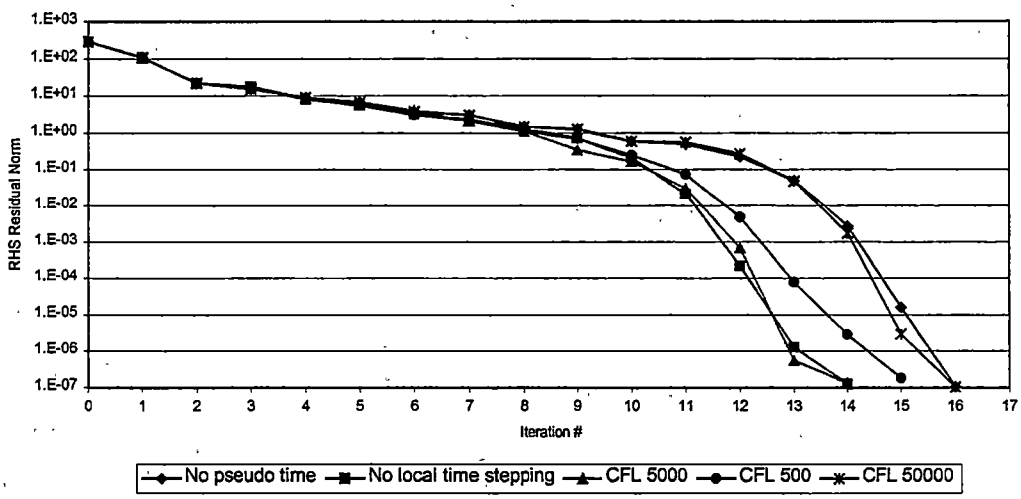


Figure 13-2: Impact of Pseudo Time Stepping on SRAND

initial conditions so velocities may vary by hundreds of feet per second. The larger CFL numbers tend to reflect larger Δt step sizes. Figure 13-2 shows that as Δt becomes larger, it weakens the influence of the time derivative preconditioning and mirrors the case where no pseudo time stepping is used. Likewise, smaller CFL numbers represent smaller time steps and can make the pseudo time term dominate, slowing convergence. The case with CFL=5000 closely reflects the case with a fixed time step size and shows a good balance between the convective terms, the viscous damping terms, and the inertia (mass). The difference is not overly significant here due to the strong nonlinearities associated with the initial conditions.

Figure 13-2 shows the benefit of including pseudo time stepping rather than ignoring it. The test case was small and GMRES was not restarted so the linear solve associated with each nonlinear step could provide an exact solution if needed. Table 13-5 shows the outcome of a series of cases with the preconditioning mass matrix turned on or off for different TWS/SUPG time scales and different preconditioning parameters. The "Jacobian" refers only to the mass matrix used as the preconditioning matrix in the time term. The nonphysical matrix (LV) was used in the TWS/SUPG term for all cases. The table reflects time to convergence in seconds and a pseudo time step size of $\Delta t = 1$ was used.

The tests show that including the pseudo time stepping term is about 40% faster than leaving it out. Using the physical Jacobian as the mass matrix did not give as much benefit but time step size studies show $\Delta t = 0.1$ may be more appropriate for the physical Jacobian, QV.

Table 13-5: Impact of Time Preconditioning on Nozzle Test Case Convergence Times

Nozzle		Tau=B	Tau=A	
		Kt		
Pseudo T	Jacobian	0	0	1
On	LV	2222	1866	1711
On	QV	-	-	2454*
Off	none	3988	-	2642

The pseudo time step size studies are shown in Table 13-6 for a range of time step sizes, τ parameters, and flow tests. Reading left to right essentially shows an evolution as τ moved from type N to type B, then A. The cases with type N used 18 Krylov restarts so the times are longer. The trends are the same though.

An important distinction between Table 13-5 and Table 13-6 is the treatment of the TWS/SUPG term. In Table 13-5, all cases used LV in the damping. In Table 13-6, the mass matrix and the TWS/SUPG term used the same form. If the mass was QV, QV^{-1} was in the damping term. The type N time scale was apparently more tolerant of this.

Adding an artificial mass matrix turns the Newton statement into a quasi-Newton method. It's also worth repeating (see Chapter 3) that a multiplicative preconditioner operates on both sides of the linear equation statement $Ax = b$. The multiplicative matrix preconditioner is commonly described by

$$MAx = Mb \tag{13-14}$$

By the same logic, the additive preconditioner should be given by

$$[Q + A]x = Qx + b \tag{13-15}$$

To show this, equate the two left hand sides to get

$$\begin{aligned} [Q + A]x &= MAx \\ M &= QA^{-1} + I \end{aligned} \tag{13-16}$$

Table 13-6: Pseudo Time Step Size Studies

	Tau=N				Tau=B		Tau=A	
	SLANTPP		Nozzle					
Delta T'	QV	LV	QV	LV	QV	LV	QV	LV
0.01	8*	10*	***	-	-	-	-	-
0.1	6*	7*	6867	***	-	-	-	-
1	6*	6*	13370	6497	***	2222	***	1711
10	-	-	-	15437	-	-	-	-
100	6*	6*	-	-	-	-	-	-
1000	-	-	12608	17493	-	-	-	-
infinite	-	-	-	-	-	3988	-	2642

*nonlinear iterations, others in seconds ***did not converge

Substituting back into the multiplicative preconditioner gives

$$MAx = Mb$$
$$[Q + A]x = QA^{-1}b + b \tag{13-17}$$

but $A^{-1}b = x$ so

$$[Q + A]x = Qx + b \tag{13-18}$$

appears to be appropriate with a preconditioning term on the right hand side as well as the left hand side. This is at odds with the concept of an artificial transient where the time transient is only on the left hand side (Chapter 3).

$$Q^* \frac{\partial x}{\partial t} + Ax = b \tag{13-19}$$

or

$$[Q + A]x = b \tag{13-20}$$

where Q is a discrete operator for $Q^* \frac{\partial}{\partial t}$.

Table 13-7 reflects the convergence times in seconds resulting from tests with the time derivative preconditioning mass matrix on the right hand side as equation **(13-18)** suggests rather than omitting it as in equation **(13-20)**. All cases used the type B time scale with the LV matrix. The time step size shown was the same on both sides of the equation. Also, since the transient is artificial, it was tested by adding it to both sides and subtracting it from both sides.

The more physical description of adding the time transient to both sides worked better than subtracting it, but excluding it from the right hand side was significantly better still. It appears

Table 13-7: Pseudo Time on Right Hand Side

Delta T'	How applied to RHS		
	+	no RHS	-
0.1	***	-	***
1	3344	2222	4658
10	3054	-	4124

***did not converge

$$[Q+A]x = b \tag{13-21}$$

is the more correct form for the matrix given. Results may be different if different logic is used to select the preconditioning matrix.

Residual Scaling as a Preconditioner

The residual scaling described in Chapter 12 acts as a multiplicative preconditioner, $MAx = Mb$, where the preconditioning matrix is diagonal with the scaling terms on the diagonal. Assuming the equations are arranged *continuity, momentum,* then *energy*; the scaling terms are applied as

$$M(Q+A)x = Mb \text{ where } M = \begin{bmatrix} 1\times10^{3} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2\times10^{-4} \end{bmatrix} \tag{13-22}$$

The same approach is used for any manipulation to the residuals such as spot scaling residuals to emphasize particular variables in specific regions of the flow, zeroing out residuals, rotating them to control the search directions, etc. These were all tested to some minimal degree but most of the effort was not controlled enough to draw conclusions. The actual application wasn't a matrix multiplication but it had the same effect.

Table 13-8 shows the impact of changing the scaling for the energy equation in the nozzle test case. This was motivated by observations of the bubble in the total enthalpy profiles at convergence (described at length in Chapter 12) and was intended to add more emphasis to the energy equation. As it turns out, it slowed convergence and had no impact on the total enthalpy profile.

Table 13-8: Energy Residual Scaling Effect on Nozzle Test Case Convergence Times

Scale	Time
2.00E-04	2671
1.00E-02	4668

A case that was tested a few times in a controlled manner was scaling local residuals. This was an attempt to emphasis the residual at boundary nodes to improve the accuracy along the wall and because of numerous observations that the boundary residuals were not converging at the same rate as in the core flow region (Chapter 12). Table 13-9 shows wall residual amplification to be surprising ineffective and actually slowed convergence. The test results (seconds to convergence) lead to a configuration with pseudo time "on" but no wall residual scaling. Many variations could be tested but it is important not to create a residual value that will overly dominate the modified Gram Schmidt process within GMRES.

Mass Matrix Variations

Returning to the artificial mass matrix, two nuances were discovered that also contributed to faster convergence. First, the mass matrix was originally treated as a legitimate term in the fluid flow equation. Finite element rigor led to the conclusion it should be multiplied by the full weight function (including the TWS/SUPG term) from the weak statement. In fact, the motivation behind putting the inverse mass matrix in the TWS/SUPG term stemmed from this product. This most likely remains true for a physical transient but numerical tests show convergence to be faster without it. This can be shown simply by

$$\left[N+\tau\left(\frac{\partial N}{\partial x}A_v\Gamma_v^{-1}+\frac{\partial N}{\partial y}B_v\Gamma_v^{-1}\right)\right]\times\left[\Gamma_v\frac{N^T}{\Delta t}\right] \tag{13-23}$$

versus

$$[N]\times\left[\Gamma_v\frac{N^T}{\Delta t}\right] \tag{13-24}$$

Table 13-9: Impact of Amplifying Wall Residuals on Nozzle Test Case Convergence Times

Nozzle		Pseudo Time	
Tau	Amp Wall	ON	OFF
B	yes	2671	4535
B	no	2222	3988
A	no	1711	2642

The second case shows a simplified weight function acting on the pseudo time term versus the full weight function. Table 13-10 shows the results of an early test of the two approaches and shows a clear improvement by simplifying the term. The convergence times were long as 18 GMRES restarts were used with a time scale parameter of type N.

Moving a little further in the direction of simplicity, "lumping" the mass matrix also provided a small benefit. The mass matrix is typically lumped in explicit methods to form a diagonal matrix, greatly simplifing direct inversion. It is also known to change the eigenvalues relative to the consistent matrix and improves stability in explicit methods. Since the problem statement is based on implicit solution techniques, the benefit was unexpected.

The approach in the current effort is to form the consistent mass matrix and then lump it using an absolute value row sum. Table 13-11 shows the moderate benefits associated with lumping.

A final observation somewhat unrelated to the preconditioning and damping terms deals with the initial conditions. In the nozzle test case, the initial conditions were set as a uniform flow suddenly introduced to the nozzle walls. Flow characteristics dictate the pressure should be set at the nozzle exit with the other flow parameters set at the inlet for subsonic flow. If the uniform flow initial conditions were set to match the downstream pressure, this amounted to the incoming flow filling the nozzle and

Table 13-10: Effect of Simplifying Weight Function for Mass Matrix on Nozzle Convergence Time

SUPG	Time
Yes	6497
No	5609

Table 13-11: Impact of Mass Matrix Lumping on Nozzle Test Case Convergence Times

	Tau	
Nozzle	B	A
Consistent	2222	1872
Lumped	2151	1711

pressurizing it in the upstream region. The characteristics for this were moving from upstream to downstream. If the uniform initial conditions were set using the anticipated upstream pressure, the startup was more like a shock tube problem where a sudden decrease in the pressure at the exit had to propagate upstream. The second scenario was used throughout the effort but Table 13-12 shows the moderate benefit of at least starting with the information working along the faster characteristics pointing in the same direction.

Summary

The test results shown support the concept of preconditioning the continuum flow before mapping to a discrete space. The TWS/SUPG derivation, analysis, and test results show the approach works for the compressible Euler equations based on primitive variables. They also validate the form of the time scale parameter, τ. From Chapter 12, there is more work to be done with developing and adding an appropriate shock capturing term. This should improve accuracy as well as speed convergence even further.

The time derivative preconditioner is easily implemented with a GMRES solver and inclusion of an appropriate term has proven to speed convergence without impacting accuracy. Selection of preconditioning parameters should be investigated further to develop an optimum set.

Table 13-12: Impact of Initial Conditions on Nozzle Test Case Convergence Times

P initial	Time
720	2222
629	1923

CHAPTER 14

ANALYSIS OF STRUCTURAL SOLVER RESULTS

The structural solver implemented in the code uses linear triangle, bilinear quadrilateral, and 8-node biquadratic quadrilateral elements. The equation set is based on the virtual work form of the general elastic (Navier) equations. Distributed external forces (such as pressure) are integrated external to the structural solver and resultant nodal force values are given as inputs. The total Lagrangian methodology is used so all integrals are taken over the initial configuration. The stiffness matrix may be linear or nonlinear. In the nonlinear case, the second Piola-Kirchoff stress is used with its finite strain conjugate, the Green/St. Venant strain (see Chapter 7). Only steady state (in physical time) cases were tested.

Implementation and Validation

Structural solver implementation is relatively straightforward and is based heavily on the methodology outlined in [Zienkiewicz and Taylor, Vol 1 and Vol 2]. Accuracy was very good for all converged solutions but solution speeds and rates of convergence were very problem and element dependent. The numerical solver was implemented using the matrix-free Newton/GMRES methodology.

The suite of validation test cases consisted mostly of long and short beams and blocks in tension, compression, and shear. Various combinations of element arrangements were tested and oriented along and at angles to the coordinate axes. The simple cases served as patch tests (see [Zienkiewicz and Taylor, Vol 1] for a detailed description) for the elements and coding validation for linear and non-linear stiffness. All the beams and blocks were two-dimensional.

Numerical results along the neutral axis matched analytic results for small deflections using both linear and nonlinear stiffness algorithms for all element types. Linear stiffness algorithms were accurate to the precision required by the termination criteria. Nonlinear stiffness algorithms matched linear results to the limit of the linear assumption. Figure 14-1 and Figure 14-2 show the basic configurations for the tension and shear test cases. The actual dimensions and element configurations are given in Table 14-1, Cases 1-8. Convergence was very fast and the results matched small deflection theory.



Figure 14-1: Finite Element Representation of a Beam in Tension

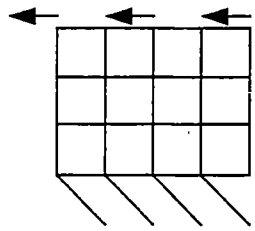


Figure 14-2: Finite Element Representation of a Block in Shear

Table 14-1 : Structural Coding Validation Test Cases

Case	Shape	Load	Dimensions	Elements	Element Sizes	Type Elements
1	bar	tension	0.2 x 6	4 x 3	equal	4-node quad
2	bar	tension	10 x 1	20 x 2	unequal	4-node quad
3	bar	tension	10 x 1	1 x 1	equal	4-node quad
4	bar	tension	10 x 1	5 x 1	equal	8-node quad
5	block	shear	1 x 1	4 x 3	equal	4-node quad
6	block	shear	0.5 x 1	1 x 2	equal	4-node quad
7	block	shear	1 x 1	1 x 4	equal	4-node quad
8	block	shear	1 x 1	4 x 1	equal	4-node quad
9	bar	side pressure load	10 x 4	20 x 2	unequal	4-node quad
10	bar	side pressure load	10 x 1	20 x 2	equal	4-node quad
11	bar	side pressure load	10 x 1	20 x 2	unequal	4-node quad
12	bar	side pressure load	10 x 1	40 x 4	unequal	4-node quad
13	bar	side pressure load	10 x 1	5 x 1	equal	8-node quad

For large loads leading to large, nonlinear deflections, element type played an important role in both accuracy and rate of convergence. An example is the idealized slender beam with a point load at the tip shown in Figure 14-3. Table 14-1, Cases 9-13 shows the configurations tested using both large and small loads with linear and nonlinear stiffness assumptions.

Figure 14-4 shows the linear stiffness results matched linear theory for all element types as long as the slender beam's deflected curvature did not become too large. If it did, the linear elements added artificial stiffness to the structure as shown by the 4-node quad linear (4-node brick type element with linear stiffness model) point at $\frac{PL^2}{EI} = 2$. Biquadratic elements (8-node quad linear) were able to follow the curvature with ease and produced very accurate results when compared with linear theory, even for large deflections.

Figure 14-4 shows that the nonlinear stiffness algorithm also produced very accurate results for the validation cases when matched with biquadratic elements. Bilinear elements "locked" (see Chapter 7) in large deflection cases as predicted in the patch test studies of [Zienkiewicz and Taylor, Vol 1 and Vol 2].

Distributed (pressure type) fixed and follower loads were also tested. The configuration shown in Figure 14-5 duplicates a large displacement case tested by [Bathe, Ramm, and Wilson, 1975] for a cantilever beam with a distributed load. Table 14-1, Cases 10-13 shows the two-dimensional cases tested. As in the tests with point loads, the linear solution converged very quickly. Constructions with biquadratic elements matched linear theory very well as shown in Figure 14-6. Bilinear elements could not follow the curvature and created artificial stiffness, even as the number of elements increased and the distribution clustered around high curvature regions. Calculated deformations using finite element techniques and nonlinear stiffness models also matched analytic results very well for biquadratic elements. Bilinear elements with nonlinear stiffness models locked and never produced a significant reduction to the residual.

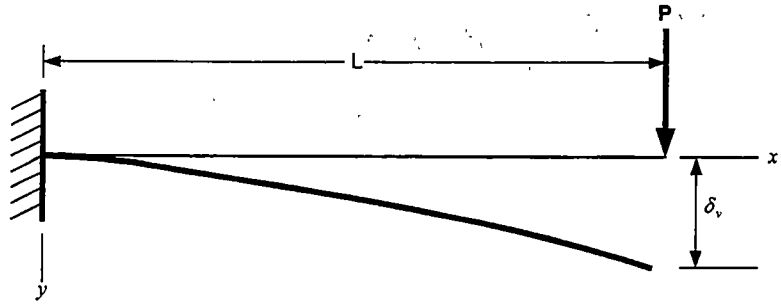


Figure 14-3: Slender beam with point load

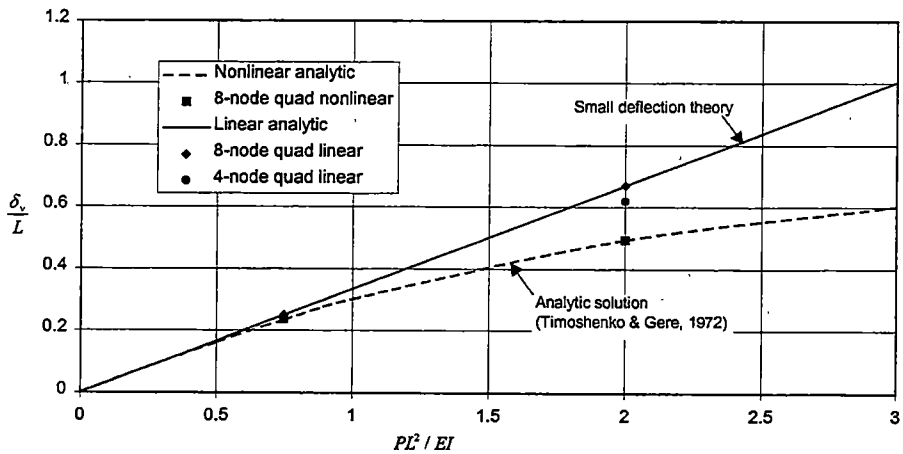
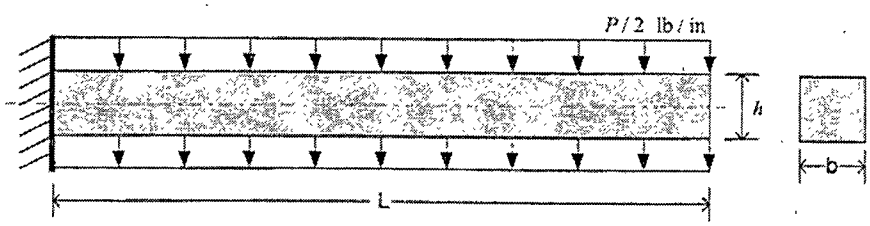


Figure 14-4: Results of slender beam with point load



$L = 10\ in$ $E = 1.2x10^4\ lb/in^2$

$h = 1\ in$ $\upsilon = 0.2$

$b = 1\ in$ $\rho = 10^{-6}\ lb\,sec^2/in^4$

Figure 14-5: Slender beam with distributed load

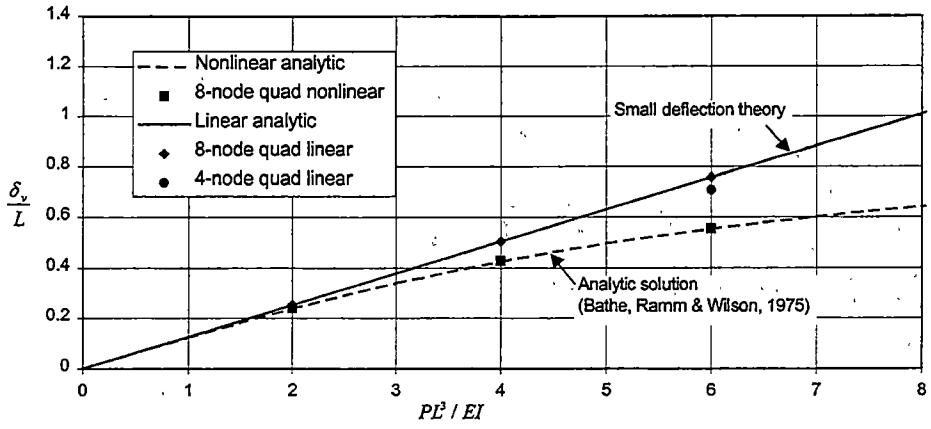


Figure 14-6: Results of slender beam with distributed load

Summary

The test cases show the steady state finite element structural solver described in Chapter 7 was coded properly and produced accurate results when compared with analytic solutions. Both linear and nonlinear stiffness models were incorporated with bilinear, biquadratic and (not shown) linear triangular element shapes. The matrix-free Newton/GMRES solver worked well with the finite element structural equation set.

A Preconditioned Krylov Subspace Approach to a Tightly Coupled Aeromechanical System

A Dissertation Presented
for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Kevin Kilpatrick
August 2001

Volume 2

CHAPTER 15

ANALYSIS OF ADAPTIVE DYNAMIC RELAXATION RESULTS

Tests with the Newton/GMRES solver applied to the steady state finite element structural equation set showed a large number of Krylov vectors (sometimes needing the full rank of the left hand side matrix) were required for each linear solution step due to numerical stiffness in the left hand side matrix. An adaptive dynamic relaxation (ADR) method was implemented on the left hand side of the Newton statement to reduce the number of Krylov vectors required. The ADR terms were only used on the left hand side and were intended to act as an additive preconditioner, similar in action and implementation to the time derivative preconditioner used in fluid flow.

Analysis

Only relatively simple steady state cases were tested and the results showed the method to be extremely successful at reducing the number Krylov vectors required for the GMRES (inner) iterations but at an unacceptable cost to nonlinear (outer) iterations. Optimal ADR was extremely effective at removing numerical stiffness from the left hand side matrix and the GMRES solver typically converged in less than four iterations for a system with 120 degrees of freedom. Convergence of the left hand side typically required more than 55 iterations without ADR and often did not converge if GMRES was restarted. However, the artificial mass and damping error terms on the left hand side dominated the converged solution increment. This led to very small displacement increments and drove excessive outer iterations of the Newton statement. Instead of expediting the solution process, it sub-optimized the process for a rapid GMRES inner iteration solution at the expense of many more outer iteration iterations. In some cases, the solution increment did not pass the right hand side residual reduction criteria and the solution process failed. The deformation increments for each Newton step seemed to slowly "ooze" the total solution forward.

Figure 15-1 shows the overall baseline (non ADR) system convergence for a 1×10 cantilever bar with a distributed side load. The line shows the rapid reduction of the right hand side residual at each outer step of the iterative Newton process. This performance is typical for systems where linear stiffness models

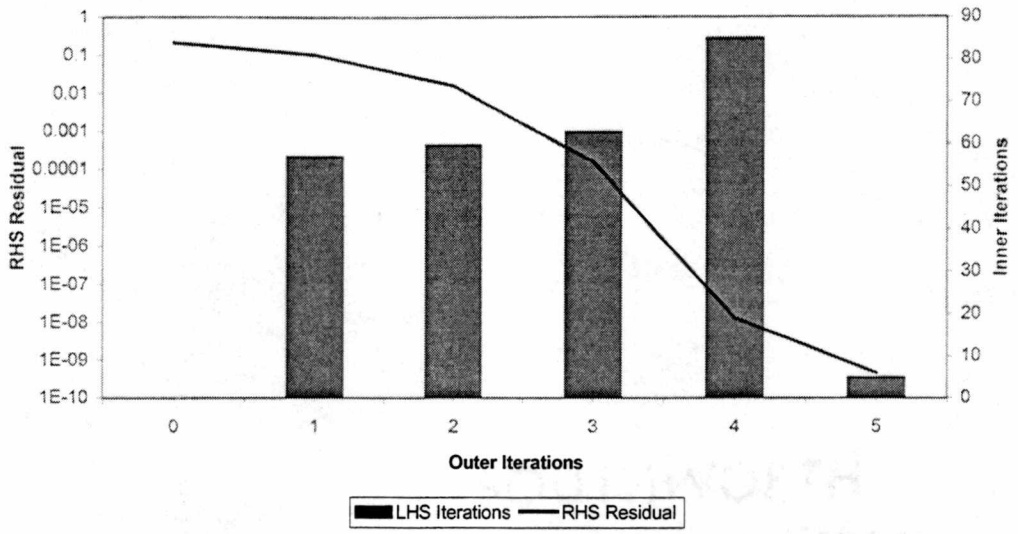


Figure 15-1: Baseline System Convergence

are relevant. In this particular case, the number of Krylov vectors allowed for the left hand side (inner) iteration is equal to the total number of degrees of freedom of the system (120). This is only possible for small problems. The columns on the chart show the total number of Krylov vectors required to meet convergence requirements set by the Eisenstat-Walker forcing function. The solution started with a residual reduction goal of 0.5. It is evident that all the steps converged in less than 120 iterations. The Eisenstat-Walker algorithm makes the convergence criteria more aggressive with each outer iteration so more and more inner iterations were required as the solution progressed. The last step shows the part of the solution logic that avoids over-solving the left hand side as the right hand side termination criteria comes into effect.

For large systems, it would not be possible or advisable to include enough Krylov vectors for each degree of freedom. A practical limit seems to be around 100 vectors as they tend to lose orthogonality and the QR algorithm normally used to provide the least squares solution after the GMRES iterations also starts to waiver. This is where restarted GMRES comes to the rescue. However, there are no general convergence theories for restarted GMRES and success is very dependent on the number of vectors included before restart and the number of restarts allowed. Figures 15-2 and 15-3 show the same baseline problem (no ADR) with 40 Krylov vectors and eight restarts allowed (nine total LHS solves per RHS step). The solution process was arbitrarily stopped after 10 outer iterations. It is readily evident that many more

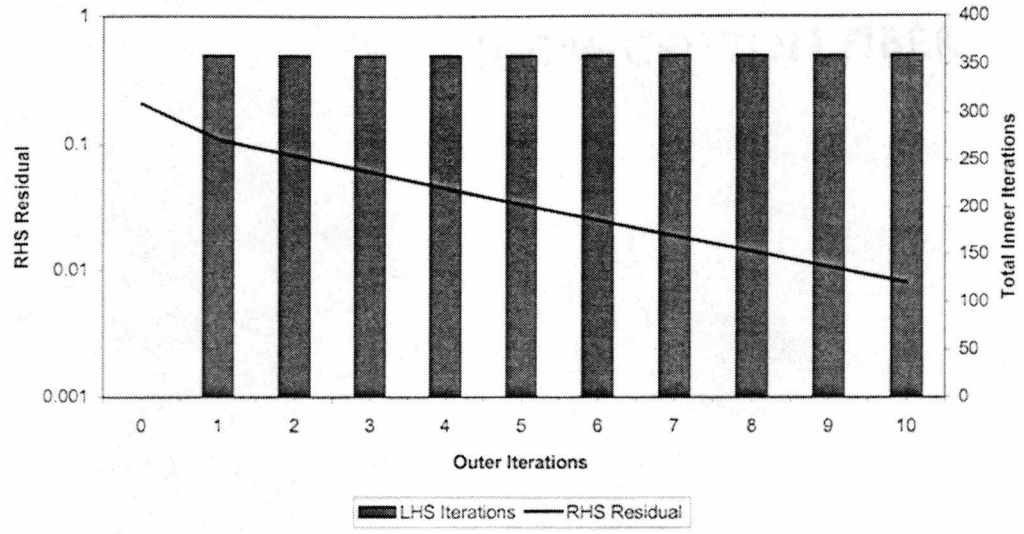


Figure 15-2: Baseline w/ Restarted GMRES(40)

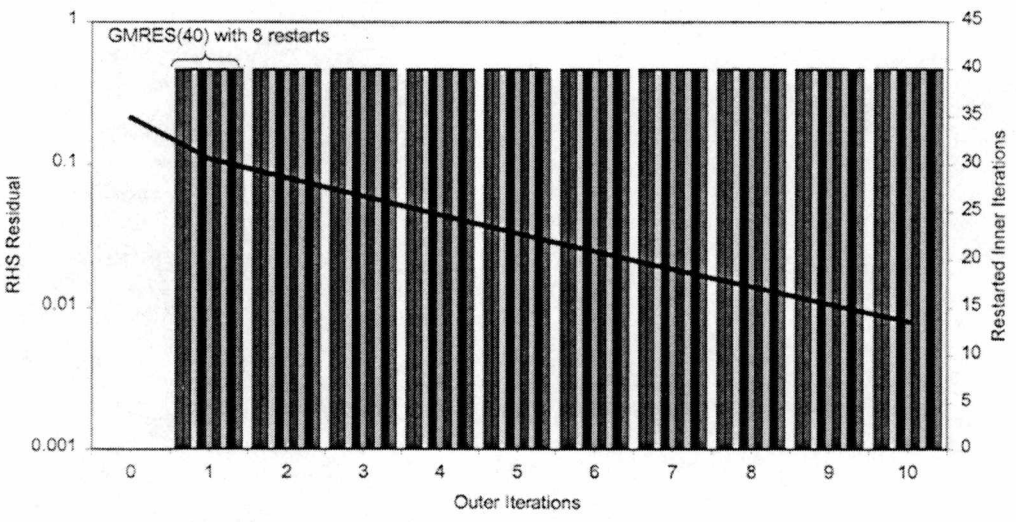


Figure 15-3: GMRES Restarts

total Krylov vectors are eventually used for the restarted approach. In this case, the full limit of 360 was used to provide the inner solution for each Newton step. It is also evident the right hand side convergence is lacking when compared to the first case (Figure 15-1). Restarted GMRES has only reduced the right hand side residual by around two orders of magnitude in 10 steps whereas the full GMRES provided over nine orders of magnitude reduction in just five steps. The use of 40 Krylov vectors is actually very generous given most cases found in the literature only allowed 20 vectors before restarting. Limiting the number of restarts is the equivalent of not allowing for the full solution of the left hand side. In this case the limited restarts were detrimental but the fluid flow analysis showed (Table 11-2) it can actually produce a significant benefit. The key point is that restarted GMRES cannot be counted on to provide a converged solution. Figure 15-4 will shed some light on this and the motivation behind the ADR approach.

The successful portion of the ADR algorithm comes from its extreme acceleration of the convergence of the left hand side iterative process. If the GMRES process always converges after only four or five iterations, there is no need to gamble with restarted GMRES. Figure 15-4 shows a comparison of the left hand side convergence rates for the first step of an iterative Newton solution. The baseline case (no ADR, full GMRES) converged with 57 Krylov vectors. The ADR case using the optimal parameters derived in Chapter 8 converged with only two Krylov vectors. The residual reduction goal for the step was 0.5 so each line in Figure 15-4 stops after it cuts the initial left hand side residual in half. The sharp drop on the right side of the baseline convergence line shows the impact of moderate numerical stiffness. Each new Krylov vector is only gradually chipping away at the residual. Each new vector adds a new dimension to the Krylov subspace but the desired solution vector is not within or close to this subspace. Once the subspace starts to grow in the general direction of the solution vector, the residual reduces quickly.

There must be a balance between the number of Krylov vectors allowed and the residual reduction goal. The shape of the baseline convergence path suggests a more aggressive goal (say 0.1) could have been satisfied with only one or two more Krylov vectors. However, different cases showed the solution process could plateau again without warning. Either way, this is not known ahead of time. The restarted GMRES case stopped its initial pass after 40 vectors. This is in the plateau region of the convergence path. The restarted process finds the best solution within the subspace spanned by the 40 Krylov vectors and uses that as the starting place for the next 40 iterations. This intermediate solution is not in the general direction

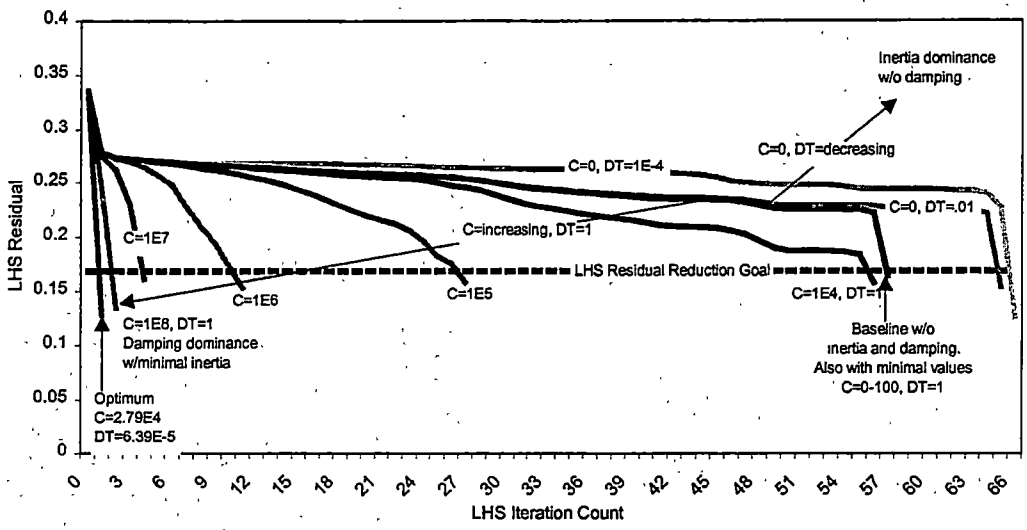


Figure 15-4: Comparison of ADR Parameters

of the solution and now, the restart begins to build a new orthogonal subspace around that vector. The next 40 vectors may move quickly in the direction of the solution or they may rebuild a slight variation to the previous subspace. The full GMRES case indicates a good solution is only 17 iterations away from the 40th vector. However, the restarted case doesn't start with the 40th vector and, even if it did, it doesn't have the other 39 vectors to force the solution into new directions. Again, there is no theory to guide restarted GMRES.

Figure 15-4 shows the ADR solution with the optimum parameters converged in just two iterations. The other convergence paths show the effects of varying the strengths of the inertial and damping terms in the pseudo time equation. Strong damping seems to be the dominant mechanism whereas high inertia actually degrades the process. These will be examined further.

The damping model used in the pseudo time equation set on the left hand side is simple mass proportional damping. M does not appear without Δt and the two are dependent. The second order (in time) set of equations is written as a first order system as

$$\underbrace{\begin{bmatrix} \frac{cM}{\Delta t} & \frac{M}{\Delta t} \\ \frac{I}{\Delta t} & -I \end{bmatrix}}_{\substack{error \\ term}} \Delta q + \begin{bmatrix} \frac{\partial K}{\partial q} & 0 \\ 0 & 0 \end{bmatrix} \Delta q = -\begin{Bmatrix} K(q)+F \\ 0 \end{Bmatrix} \tag{15-1}$$

where

$$\Delta q = \begin{Bmatrix} \Delta \vec{u} \\ \Delta \vec{v} \end{Bmatrix} = \begin{Bmatrix} \Delta deformation \\ \Delta velocity \end{Bmatrix} \tag{15-2}$$

I arbitrarily set $M = 1$ and used a lumped mass matrix. Some earlier versions of dynamic relaxation just took the maximum value of the diagonal of the stiffness matrix as the mass. Δt (actually $\frac{1}{\Delta t}$) and C now act as scaling terms for the strength of the inertial and damping terms in the equation set. Since a lumped mass matrix is used, both inertia and damping are diagonal matrices. The full effect is not just a simple addition to the diagonal of the tangent stiffness matrix though. Figure 15-4 shows $\Delta t = 1 \times 10^{-4}$ (inertial term weight of 1×10^{8}) actually degrades the left hand side solution while an equivalent weight ($C = 1 \times 10^{8}$ and $\Delta t = 1$) on the damping greatly improves convergence. The optimum combination provides a weight on the inertial term of $\frac{M}{\Delta t^2} = \frac{1}{6.39 \times 10^{-5}} = 2.45 \times 10^{8}$ and on the damping term of $\frac{CM}{\Delta t} = \frac{2.79 \times 10^{4} \times 1}{6.39 \times 10^{-5}} = 4.37 \times 10^{8}$. The trends in Figure 15-4 suggest the damping term is actually the dominant influence and very strong damping alone produces nearly the same results.

The damping dominance is actually good news from an implementation standpoint. If the full benefit can be achieved without the inertial terms, the system given in equation **(15-1)** can be reduced to a simple first order system given by

$$\left(\underbrace{\frac{cM}{\Delta t}}_{\substack{error \\ term}} + \frac{\partial K}{\partial q} \right) \Delta displacement = -\left(K(q) + F \right) \tag{15-3}$$

The left hand side of equation **(15-3)** takes the form of a diffusion equation and essentially diffuses applied forces through the structure. This effect is seen in the deformation examples of figures 6-10 where the top and bottom move on the first step but the neutral axis doesn't.

In some of my early efforts, I tried to develop an inertial term mass matrix to improve the condition without damping. In simple terms, these efforts took the form

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \frac{\partial^2 \{u\}}{\partial t^2} + \begin{bmatrix} K_{xx} & K_{xy} \\ K_{yx} & K_{yy} \end{bmatrix} \{u\} = \{0\} \qquad \textbf{(15-4)}$$

and tried to determine the best values for a, b, c and d. This is the same basic approach used by [Turkel, 1987] for developing a fluid flow preconditioner. As it turns out, the diagonal parameters scale the bending and shear eigenvalues equally so they have no affect on the ratio. Meanwhile, the off-diagonal parameters do not provide scaling and simply rotate the eigenvalues around a circle in the imaginary plane. Upon reflection, this was to be expected.

The dismal failure part of the ADR method is the extremely small deformation increment it produces. [Underwood, 1983] noted the number of iterations required for convergence could be quite large for explicit dynamic relaxation methods. This also turns out to be the case for the implicit version. Figure 15-5 tracks the lateral deflection increment of a node near the tip at the end of the first Newton (outer) iteration. It shows how aggressive each combination of parameters is at moving towards the converged solution. The horizontal axis tracks the strength of the damping term while the vertical axis shows the lateral deformation. At convergence, the comparison node deflects 8.185×10^{-3} so all the cases on the chart are less than the converged deflection, as they should be. The upper left point shows the baseline case without ADR and a full complement of Krylov vectors is the most aggressive. It produces an initial deformation increment in the same order of magnitude as the converged solution. If the number of Krylov vectors is capped at three to match the better ADR requirements, the increment drops to 1.3×10^{-8} at the lower left corner. The optimum inner convergence point produces the smallest increment for the outer iteration. It is interesting to note the linear log-log relationship between the damping coefficient and the lateral deflection but I haven't found any practical value. The naïve conclusion would seem to be that

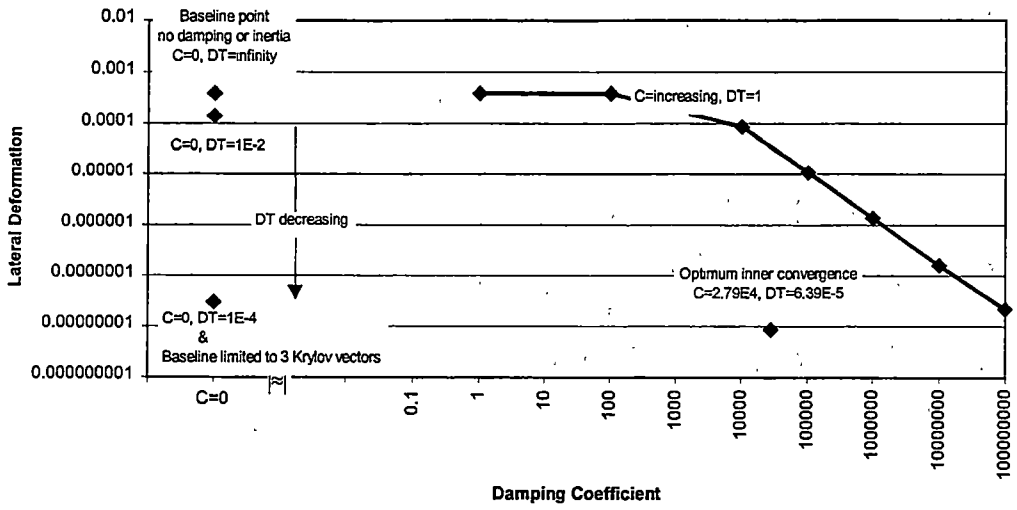


Figure 15-5: Deflections Near the Tip

less Krylov vectors lead to a smaller solution increment. This does not hold, though, as a single Krylov vector could produce the full solution increment for a linear deflection if it pointed in the direction of the solution. A good starting vector should lead to faster convergence, as should an operator that aggressively forces new search directions.

The Krylov solution process for the linear (inner) solve is non-stationary. However, the Newton (outer) step is still a nonlinear version of a standard fixed-point iteration. The ADR term works very well for accelerating convergence of the inner iteration but produces too small of an increment, driving excessive outer iterations. A reasonable next step is to see if the inner solution might be over-relaxed to produce a larger displacement increment. The Newton statement has the basic form

$$Ax = b \tag{15-5}$$

or

$$F'\left(x^n\right)\Delta x^n = -F\left(x^n\right) \tag{15-6}$$

Now,

$$\Delta x^n = -F'\left(x^n\right)^{-1} F\left(x^n\right) \tag{15-7}$$

and

$$x^{n+1} = x^n + \Delta x^n \tag{15-8}$$

The baseline case shows x^* (the converged solution) should have an order of magnitude of 1×10^{-3} for the neutral axis deflection at the tip of the beam. The first step of the non-ADR case creates this size increment on the first step and adjusts to full convergence with decreasing increments. The optimum ADR case produces (Figure 15-5) an initial increment around 1×10^{-9}. A large over-relaxation parameter

$$x^{n+1} = x^n + 10^5 \times \Delta x^n \tag{15-9}$$

$\left(10^5\right)$ would make the ADR step and the baseline step about the same size. If the solution vector from the ADR method were in the same direction as the baseline vector, the simple over-relaxation of the step would provide the needed deformation increment.

Unfortunately, that's not the case. Figure 15-6 shows a magnified view of the lateral movement of the baseline case for the deformed beam after a single Newton step. The deformation is appropriately distributed across the length and thickness of the beam with the largest deformation at the tip. The optimum ADR case in Figure 15-7 shows a different pattern. The deformation pattern shows displacement at the surface along the length of the beam but not at the neutral axis. The maximum deflection is near the tip but not at it. This pattern will be dissected further in Figure 15-8 and Figure 15-9.

Figure 15-8 shows the pattern produced with an inertial term the same as the optimum ADR but no damping. The deformation is extremely magnified for visualization but shows a pattern completely unrelated to the deformation expected from a uniformly distributed load along the top and bottom surfaces of the beam. In the case of inertial dominance, the equation set has the form of a wave equation with a low fundamental frequency. The best analogy is one of a beam (or noodle) with extreme mass and minimal stiffness subjected to a sudden load. From a modal perspective, the baseline case in Figure 15-6 shows a deformation consistent with the expected first (fundamental) mode for the beam. The case with inertial dominance seems to reflect the expected third bending mode.

Minimal inertia and very large damping produce the magnified pattern in Figure 15-9. As in the optimal ADR case, the main deflection is along the top and bottom surfaces but the maximum deflection is not at the tip. The centerline of the beam deflects some but is no longer at the centerline of the deformed beam. In the damping dominated case, the governing equations take on the form of a diffusion equation.

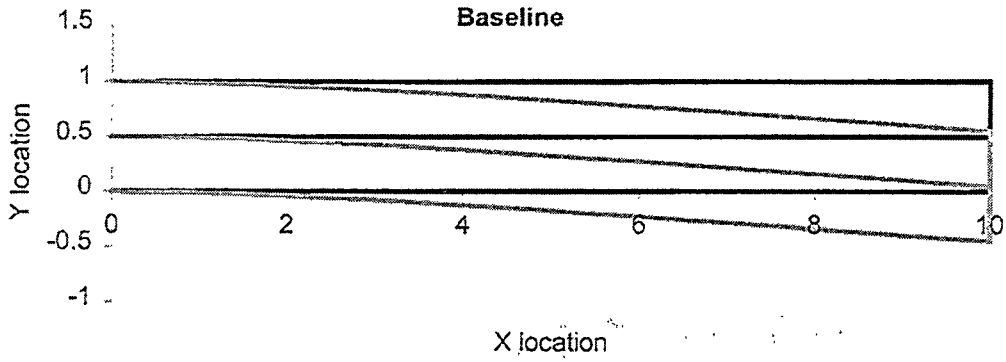


Figure 15-6: Baseline Deformation

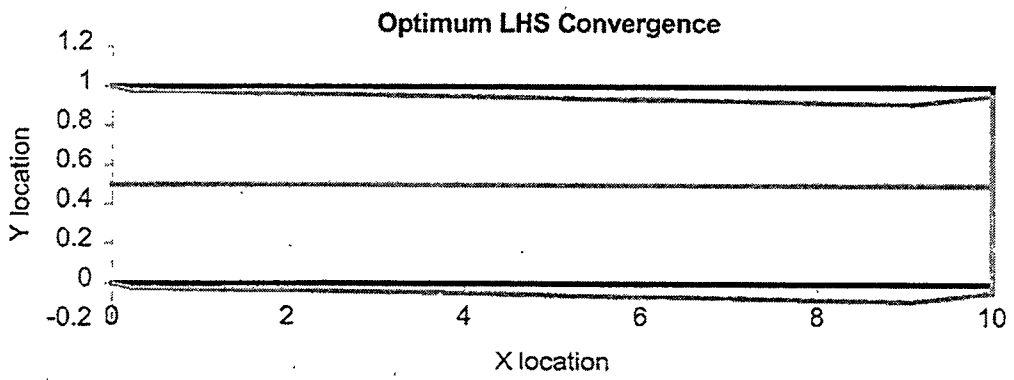


Figure 15-7: Optimum ADR Deformation

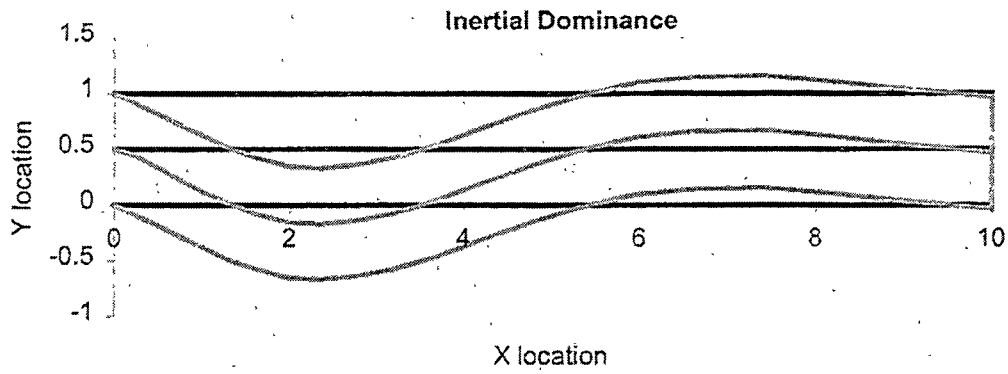


Figure 15-8: Inertial Dominated Deformation

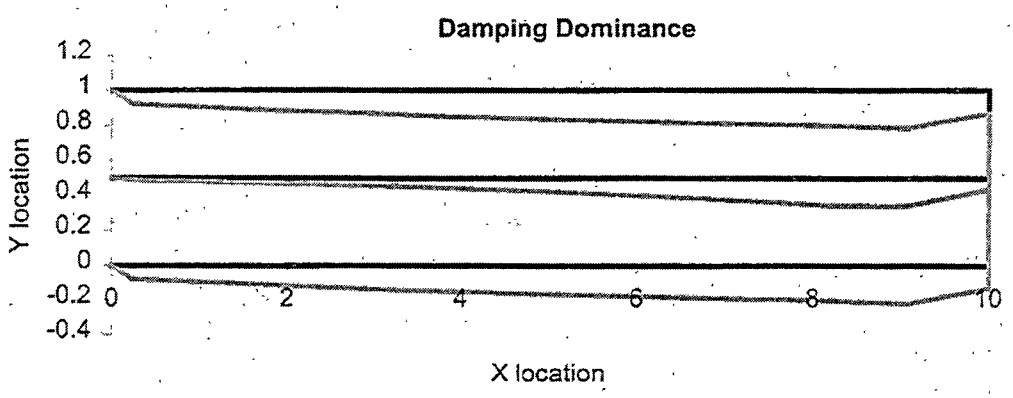


Figure 15-9: Damping Dominated Deformation

This is consistent with the pattern in Figure 15-9. The equation set diffuses the applied load through the interior of the beam with a directional rate equal to $\frac{stiffness}{damping}$.

A final check is the baseline case without ADR but the number of Krylov vectors are limited (Figure 15-10) to the same number as needed for the ADR case (three). No restarting is allowed. The deflection is now very small, $\mathcal{O}(10^{-8})$, compared to the full baseline case and shows the same basic second mode pattern as the damping dominated case. Severely limiting the number of Krylov vectors tends to destroy the convergence properties of the Newton step. The mode shape may indicate a cluster of eigenvalues where the fundamental mode isn't clearly separated from the others. From a physical standpoint with this geometry, it is possibly an indicator of the right hand side residual exciting a higher order mode and the energy hasn't yet cascaded down past the second mode to the first. Again, the left hand side solution could be found in just a few steps with a good starting vector.

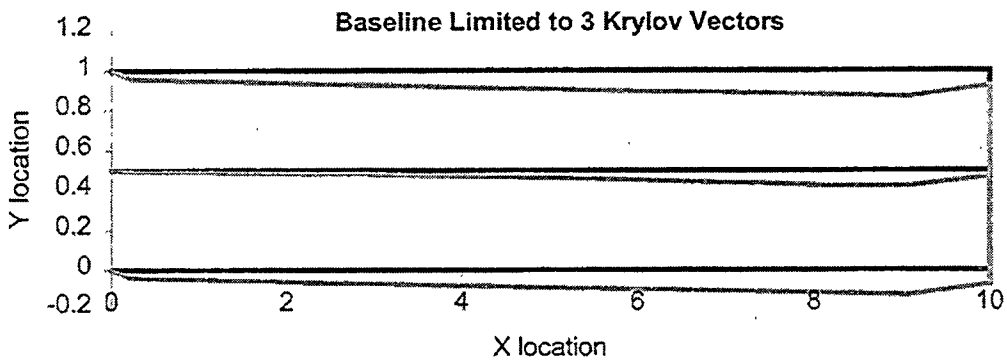


Figure 15-10: Baseline with Limited Krylov Vectors

Summary

The various shapes in figures 6-10 show the deformation vectors are not in the same direction as the baseline case. Over-relaxation is unlikely to speed convergence of the outer iterations and this held true in numerical experiments. The optimum and damping dominated cases also show the governing equations taking on the characteristics of diffusion equations.

In summary, the inclusion of ADR terms on the left hand side dramatically accelerated convergence of the inner iteration by reducing numerical stiffnes but did so at an unacceptable price to the outer iteration. ADR makes sense as an approach for explicit solvers but is too costly for implicit ones. None of the ADR cases were followed to full convergence of the outer iteration, as this would have taken at least 10^5 steps.

CHAPTER 16

ANALYSIS OF COUPLED FLUID-STRUCTURE SOLVER

Background

[Farhat, et.al., 1995] present a well founded strategy for solving the three field fluid-structure-grid problem based on partitioning the full system into it's three components and iterating between them to eliminate lags. They point out "the numerical solution of [the three field problem] via a fully coupled monolithic scheme is computationally challenging and software-wise unmanageable."[1]

I would have to agree with the first part but the second is mostly solved by the matrix-free GMRES approach. Unfortunately, there appears to be a conservation law in computational engineering known as "conservation of misery." The residual statement of the coupled field problem is relatively straightforward using the element by element assembly procedure prevalent in applications of the finite element method. The matrix-free GMRES approach eliminates the need for one to be concerned about bandwidth and coupled terms in the left hand side matrix because the residual assembly procedure accounts for these. The misery comes from two places: the preconditioning required to solve the left hand side matrix (the structure additive preconditioning didn't perform up to expectations); and the numerous disk read/write operations required as each residual vector calls the appropriate data and equation set from memory. These both turned out to be fatal flaws. The first may eventually be overcome by more clever preconditioning schemes. The second is cause for concern as, during testing, it literally (physically) destroyed the hard drive on my personal computer on the first attempt at the nozzle flow problem with a flexible wall.

[Bendiksen, 1991] presented the results of a successful monolithic approach to the coupled field problem using explicit methods. He highlighted the phase accuracy as one of the significant benefits. Since the same phase accuracy can be obtained by performing sub-iterations of the partitioned problem, the

[1] C. Farhat, M. Lesoinne, P.S. Chen, and S. Lanteri, "AIAA-95-1290-CP, Parallel Heterogeneous Algorithms for the Solution of Three-Dimensional Transient Coupled Aeroelastic Problems." Proceedings of the AIAA/ASME/ASCE/AHS/ASC 36th Structures, Structural Dynamics, and Materials Conference and AIAA/ASME Adaptive Structures Forum. 10-13 April 1995. New Orleans, LA., p1147.

advantages of the monolithic approach do not outweigh the numerous disadvantages. [Park and Felippa, 1983] provide a very thorough analysis and motivation for partitioning large systems and this research project has led me to strongly support the partitioned approach for practical applications.

Tightly Coupled Fluid-Structure-Grid Solution Results

The fully coupled fluid-structure-grid interaction test case was an 8-foot long subsonic fluid nozzle with an area ratio of 2. The walls were made of 1.5-inch thick mild steel and supported only at the inlet and exit. This was expected to be a simple initial test with a steady state pressure gradient along the wall and small deflections. The fluid and structure used the same wall discretization so this avoided the issues associated with interpolating variables at the fluid-structure interface (Figure 16-1).

Chapter 15 already showed the failure of ADR as a preconditioner so it was not used in any of the coupled fluid-structure-grid calculations. Chapter 14 also pointed out the need to carry the full complement of Krylov vectors for the thin structural problems. The structure in the coupled case had 88 degrees of freedom after accounting for boundary conditions so the coupled problem was allowed 88 Krylov vectors before restarting. Eleven restarts were allowed but the portion of the residual vector representing the solid would theoretically converge before restarting if taken in isolation. Time derivative preconditioning was used in the fluid equations to speed convergence. Chapter 13 showed the fluid converged more quickly with 88 Krylov vectors but 40 was adequate. This highlights the first problem associated with the

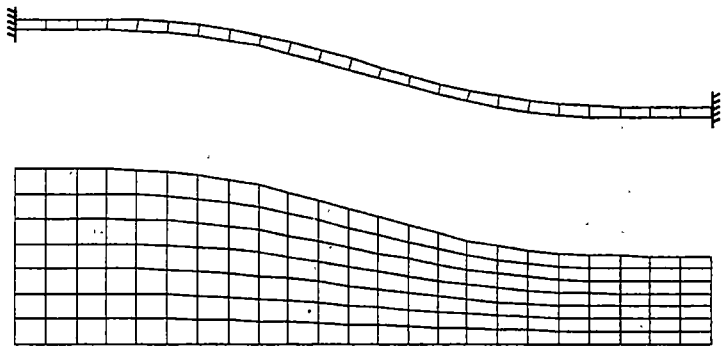


Figure 16-1: Coupled Fluid-Structure Geometry

monolithic approach: the solver must satisfy the worst case needs for an individual domain in the coupled system and apply it to the every domain in the system. This drives a great deal of additional work in the solution process by forcing the solver to perform unnecessary calculations.

The second issue with the coupled problem has already been discussed: the residual assembly process drives multiple read/write operations to a storage device for every vector. As the number of Krylov vectors increased so did the number of read/writes. Before the convergence stagnated, it drove over 34,000 such read/write operations to the hard drive. Most of these were associated with the relatively fast linear residuals on the left hand side so the read/write operations were nearly continuous and paced the calculation. Figure 16-2 shows how this influenced the overall performance of the computer. The first bar shows the average number of floating point operations per second (FLOPS) the system was able to generate over about 50 *fluid only* nozzle test cases with 40 Krylov vectors and anywhere from 0 to 18 GMRES restarts. (Higher FLOPS mean faster calculations.) The results were very similar for other test cases with more and less degrees of freedom. The second bar shows a single *fluid only* nozzle test case with 88 Krylov vectors. This shows better performance because more floating-point operations were accomplished (88 vectors worth) once the data was loaded. The third bar shows the drop in performance for the coupled case. Although it had 88 Krylov vectors, the read/write operations slowed the overall performance.

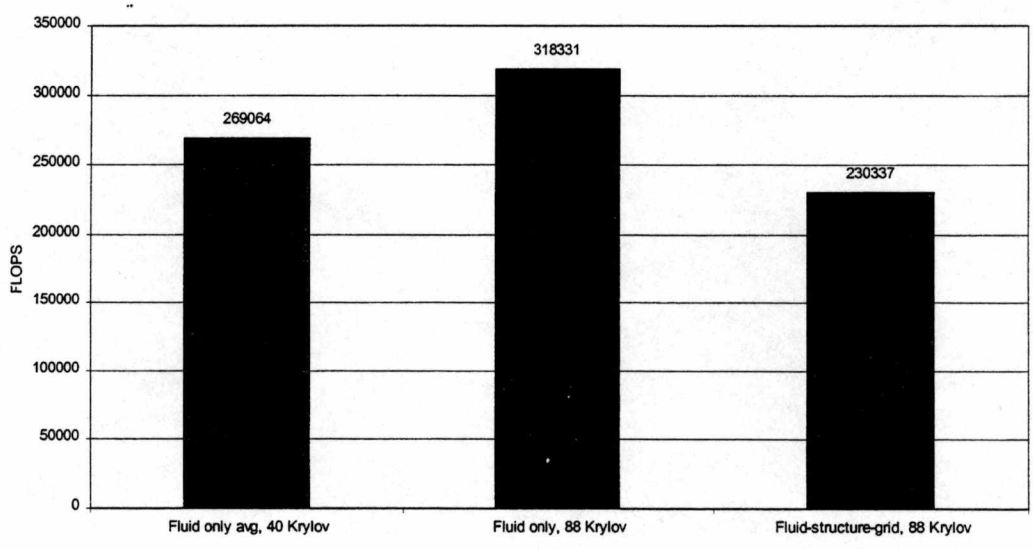


Figure 16-2: Computer Performance

The third finding has to do with the convergence of the coupled problem. I probably should have started with a converged fluid flow before introducing the fluid-structure interactions but I didn't. I started with uniform flow initial conditions as in all the other nozzle test cases. I hoped the initial convergence would be no worse than *fluid only* calculations since more than enough Krylov vectors were allowed to satisfy the structure, even without preconditioning. Also, the structure is relatively strong compared to the expected pressure load so deformations were not expected to radically alter the geometry. However, the tightly coupled fluid-structure-grid problem did not converge. After over 7 hours of calculations and 12 nonlinear Newton steps, the coupled residual was only reduced by a single order of magnitude. The exact same case was completed in just less than 59 minutes when the fields were decoupled and subiterated.

Table 16- 1 provides a summary of the convergence results. The partitioned approach is the clear winner by all counts, even if the monolithic case had converged. The accuracy of the decoupled case is the same as for the individual domains.

Summary

The software for a coupled fluid-structure-grid system is really not much more complex than the software for the three fields taken separately using the matrix-free GMRES algorithm and element-by-element assembly procedures. The codes were not modified internally for the coupled or uncoupled approach so existing codes could work in either approach.

Table 16- 1: Fluid-Structure-Grid Convergence

Coupling	Domain	Domain Iteration	Nonlinear Iterations	Time	FLOPS	Residual
Tight	All	-	12	25431	5.86E+09	3.64E+02
Loose	Fluid	1	7	-	-	-
	Structure	1	3	-	-	-
	Grid	1	5	-	-	7.53E+01
	Fluid	2	4	-	-	-
	Structure	2	3	-	-	-
	Grid	2	5	-	-	4.05E-02
	Fluid	3	0	-	-	-
	Structure	3	2	-	-	-
	Grid	3	4	3533	1.03E+09	1.36E-12

The performance results for the monolithic case were not what I had hoped. In spite of some success with the time derivative preconditioner in the fluid flow equations, I was never able to precondition the individual domains to an acceptable level so that GMRES restarts would not be needed. Without being able to control the individual domains, there was little hope of adequately preconditioning the tightly coupled system. The numerical difficulty associated with solving the ill-conditioned, coupled system is enormous. Even if some future technology eliminates the need for mechanical disk drives, the additional work associated with oversolving some domains with too many Krylov vectors while others are lagging negates any benefit from a simultaneous solution. Subcycling between the domains removes any phase errors and there are numerous examples in the literature documenting the accuracy of the approach for transient problems.

CHAPTER 17

CONCLUSIONS

The fluid-structure-grid problem has many facets and draws on several different engineering and mathematical disciplines. In the author's opinion, there isn't a single silver bullet to eliminate the complexities or difficulties associated with a tightly coupled fluid-structure system but rather a shotgun blast with a thousand silver BB's. The relevant disciplines have been explored in depth with some retracing of prior developments and others venturing into new ground. All the concepts outlined in the approach were implemented into a computer code and tested in numerical experiments and simple applications.

Many new ideas have been presented. Some ideas worked well and others failed. These have been discussed at length by category in the previous chapters and will be summarized below. However, some of the significant contributions include:

- Proved the benefits of using an additive preconditioner to speed convergence of a globally convergent Newton iteration using a GMRES solver for the inner solution iterations
- Provided direct comparison and the benefits of pseudo time stepping versus pure Newton iterations for both fluid flow and structural equations
- Showed the correct form for additive preconditioners, implementing as a pseudo time step rather than a multiplicative preconditioner in fluid flow calculations
- Developed, documented and implemented various residual scaling strategies to overcome numerical stiffness created by residual orders of magnitude for fluid flow equations, structural equations, and coupled fluid-structure systems
- Implemented and documented the impact and influence of the number of Krylov vectors and restarts allowed in GMRES for both fluid flow and structural problems
- Implemented and documented the benefits of halting GMRES prior to full convergence of the inner iteration to speed the total solution process
- Implemented and documented the benefits of the Eisenstat-Walker forcing function for nonlinear iterations and developed an optimum starting condition for both fluid flow and structural equations

- Developed and implemented a TWS/SUPG formulation for a finite element approach to inviscid fluid flow using (dimensional) primitive variables
  - Developed and documented the theoretical basis for the time scale parameter, τ, and demonstrated significant performance improvements over the current state of the art
  - Developed the correct form for the TWS/SUPG damping term and demonstrated its impact on both stability and consistency
  - Developed and implemented the correct form of the weight function to use in a finite element weak statement formulation of the inviscid fluid flow equations in primitive variables
  - Documented parameters and incorporated time derivative preconditioning into the TWS/SUPG fluid flow equation with pseudo time stepping
- Developed both monolithic and partitioned approaches to solving coupled fluid-structure problems and implemented a matrix-free solution process while providing a direct comparison of convergence results
  - Documented shortfalls of the monolithic solution process for coupled fluid-structure problems in both disk read/write operations and oversolving of converged domains
  - Developed and implemented the theoretical basis for preconditioning structural equations using an implicit form of the Adaptive Dynamic Relaxation technique
  - Proved the benefits of additive preconditioning by dramatically speeding convergence of the inner solution of a thin structure, diagnosed active terms, and documented adverse impact on the outer iteration

The overarching strategy for preconditioning by adding a pseudo time term to the left hand side appears to have merit but there is still a great deal of room for improvement.

Implementation in the MATLAB computational environment turned out to be a great experience. Code development was straightforward and MATLAB's basis as a matrix oriented language eliminated some rather tedious coding. It was relatively fast and allowed for interactive program development and execution. Flow variable and residual surface plots provided insight into convergence evolution of the numerical methods and points toward areas for improvement. It was also possible to develop and test algorithms on a personal computer. Although limited in speed and memory, I was able to develop and test the algorithms and perform some rather complex calculations for small but relevant problems.

Numerical Solver

The GMRES solver should come with a warning label. It was relatively robust when sufficient Krylov vectors could be kept to solve the linear problem without restarting. Restarts were unpredictable and always degraded performance. However, more restarts were not necessarily better and a lot of time was spent doing unnecessary work. In general, the concept of partially solving the left hand side of a linear equation set to a desired degree of accuracy is great. It enables the solution process to be tailored so the amount of computational effort is always the minimum required to meet the established goals.

The magnitude of the individual terms in the residuals should be scaled to a common order. This is very similar to pivoting in direct solvers and applies for the same basic reasons. The problem lies with the modified Gram Schmidt process used in the current version of GMRES but may be more forgiving if the Householder reflection method of orthogonalization were used instead.

The globally convergent Newton statement worked well with the parabolic line search but nothing ever survived to convergence after two step length reductions. Eight reductions were allowed throughout the effort but two would have been adequate.

The Eisenstat-Walker algorithm drove faster convergence times than a fixed value for the forcing function. There are other algorithms available but none were tested in this effort. It appears that having a variable forcing function is better then having one that is fixed.

The matrix-free implementation of the left hand side matrix-vector product worked well. It was simple to code and allowed for easy manipulation of the left hand side operations. It also eliminated concerns about node numbering, matrix bandwidth and storage.

Fluid Flow

The fluid flow solver results are encouraging. The primitive variable form of the compressible Euler equations worked fairly well in terms of both accuracy and convergence times. A nonlinear shock capturing term should be added and will likely lead to accuracy comparable to any fluid flow code available. The code developed in this effort was able to accommodate a mixture of triangle and

quadrilateral elements and any order of node numbering so localized grid refinement was possible. This could be extended to solution adaptive grid refinement.

The TWS/SUPG damping term worked well when coupled with the time derivative preconditioning matrix and the appropriate time scale parameter. The convergence was stable and there was never a sign of a CFL type constraint. The global Newton statements accepted relatively harsh startup conditions without any instabilities.

An analysis of the time scale parameter consolidated the findings of all the parameters reported in the literature. The theoretical development allowed a term to be developed and implemented that provided faster convergence than any found in the literature for the given set of equations. This development should continue with a shock capturing term.

The time derivative precondition approach showed promise. Convergence times were always faster with an appropriate mass matrix than with the pure Newton statement, all other things being equal. It didn't provide convergence fast enough to eliminate the need for GMRES restarts but maybe that will be possible in the future.

The results for the ALE approach to the moving grid problem were not presented. The coding worked but the test cases never created a flow field deformation large enough to distort the grid so the results with grid motion were indistinguishable from those without. The ALE approach has been proven many times so there was nothing new to report.

Structural Solver

The structural solver worked as planned and was accurate for both linear and nonlinear stiffness matrices over relatively large deformations when compared to classic test cases. A biquadratic quadrilateral element was required to accurately model large deformations. Greatly increasing the number of linear elements introduced artificial stiffness and could not duplicate the accuracy of the biquadratic element.

Adaptive Dynamic Relaxation was ineffective as a matrix preconditioner. It worked extremely well in reducing the number of GMRES iterations but introduced such a large error in the solution transient

that convergence was destroyed. Any future effort in this area should consider a first order pseudo time term rather than the standard approach of using a second order term.

Locking in thin structures was a recurring issue. The problem is well documented and practical workarounds already exist. The symptoms and the workarounds are extremely similar to those used to resolve problems with pressure-velocity decoupling in compressible flow codes in low speed flows. The fluid flow time derivative preconditioning matrix effectively eliminates these problems so there may be a way to generate a similar effect in solids. The time scales associated with the aspect ratio of the physical geometry may be a limiting issue though.

Grid Motion

Grid motion never became an issue and was not rigorously tested since it used the exact same algorithms and routines as the structural solver. The "eyeball" test showed grid points were reasonably well distributed for the few cases tested.

Coupled Fluid-Structure-Grid

The monolithic fluid-structure-grid approach did not work but for different reasons than anticipated. The software aspects of coupling the fields was relatively straightforward using the element-by-element assembly procedure and the matrix-free implementation of the left hand side avoided the problems with a large Jacobian matrix with complicated cross derivative terms. I was never able to assess the time derivative preconditioning of the coupled problem because it did not work for the field equations associated with the structure (and thus the grid).

The monolithic approach also drove a great deal of unnecessary calculations as the structural equations were solved many times more than theoretically needed for an exact solution. Forming the residuals drove numerous read/write operations to the hard drive as each set of field equations was recalled from memory. This actually paced the calculation and drove such a long and rapid cycling that it actually physically destroyed the hard drive on the computer. Since the solution had also stagnated, the effort was abandoned.

A partitioned approach to the coupled problem with subiterations performed on each set of field equations converged rapidly with each field as accurate as if it had been calculated in isolation. A great deal of time, money and effort has been invested in developing single domain codes and these should not be put aside. The partitioned approach with subcycling is the best approach from the perspective of code reuse, ease of implementation, solution time and accuracy.

BIBLIOGRAPHY

BIBLIOGRAPHY

Aliabadi, S.K. and Tezduyar, T.E., "Space-Time Finite Element Computation of Compressible Flows involving Moving Boundaries and Interfaces." Computer Methods in Applied Mechanics and Engineering, Vol. 107, Nos. 1-2 (1993), 209-223.

Alonso, J., Martinelli, L., and Jameson, A., "Multigrid Unsteady Navier-Stokes Calculations with Aeroelastic Applications." AIAA-95-0048, 33rd Aerospace Sciences Meeting and Exhibit, 9-12 January 1995. Reno, NV.

Alonso, J.J., and Jameson, A., "Fully-Implicit Time-Marching Aeroelastic Solutions." AIAA-94-0056, 32nd Aerospace Sciences Meeting and Exhibit, 10-13 January 1994. Reno, NV.

Aris, R., Vectors, Tensors, and the Basic Equations of Fluid Mechanics. Dover Publications, Inc., New York, 1962.

Axelsson, Owe. Iterative Solution Methods. Cambridge University Press, Cambridge, 1994.

Baker, A.J. and Kim, J.W. "A Taylor Weak-Statement Algorithm for Hyperbolic Conservation Laws." International Journal for Numerical Methods in Fluids, Volume 7, No. 5 (1987), 489-520.

Baker, A.J. Finite Element Computational Fluid Mechanics. Hemisphere Publishing Corporation, Washington, 1983.

Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and Van der Vorst, H. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. Society of Industrial and Applied Mathematics, Philadelphia, PA, 1993.

Bathe, K.J., Ramm, E., Wilson, E.L. "Finite Element Formulations for Large Deformation Dynamic Analysis." International Journal for Numerical Methods in Engineering. Vol 9, Number 2, 1975.

Batina, J.T., "Unsteady Euler Airfoil Solutions using Unstructured Dynamic Meshes." AIAA-89-0115, 27th Aerospace Sciences Meeting and Exhibit, 11-14 January 1989. Reno, NV.

Baum, J. Luo, H., Lohner, R., Yang, C., Pelossone, D., and Charman, C. "A Coupled Fluid/Structure Modeling of Shock Interaction with a Truck." AIAA-96-0795, 34th Aerospace Sciences Meeting & Exhibit. 15-18 January 1996. Reno, NV.

Belytschko, T. and Hsieh, B.J. "Non-Linear Transient Finite Element Analysis with Convected Coordinates." International Journal for Numerical Methods in Engineering. Vol 7, Number 3, 1973.

Belytschko, T. and Bachrach, W.E. "Efficient Implementation of Quadrilaterals with High Coarse-Mesh Accuracy." Computer Methods in Applied Mechanics and Engineering, Vol. 54, No. 3 (1986), 279-301.

Bendiksen, Oddvar O. "A New Approach to Computational Aeroelasticity." AIAA-91-0939-CP, American Institute of Aeronautics and Astronautics paper, 1991.

Benek, J.A., Steger, J.L., Dougherty, F.C., and Buning, P.G. Chimera: A Grid-Embedding Technique. AEDC-TR-85-64, Arnold Engineering Development Center, Arnold Air Force Station, Tennessee, 1986.

Bisplinghoff, R.L., Ashley, H., and Halfman, R.L. Aeroelasticity. Addison-Wesley Publishers, Reading, MA, 1955.

Bisplinghoff, R.L., and Ashley, H. Principles of Aeroelasticity. John Wiley & Sons, New York, 1962.

Biswas, R. and Strawn, R. "A New Procedure for Dynamic Adaption of Three-Dimensional Unstructured Grids." AIAA-93-0672, 31st Aerospace Sciences Meeting & Exhibit, 11-14 January 1993, Reno, NV.

Blanco, M., and Zingg, D.W. "A Fast Solver for the Euler Equations on Unstructured Grids Using a Newton-GMRES Method." AIAA-97-0331, 35th Aerospace Sciences Meeting & Exhibit. 6-10 January 1997. Reno, NV.

Brekhovskikh, L.M., and Goncharov, V. Wave Phenomena: Mechanics of Continua and Wave Dynamics. Second Edition, Springer Verlag, New York, 1994.

Brillouin, L. Wave Propagation in Periodic Structures. Second Edition, Dover Publications, New York, 1953.

Brillouin, L. Wave Propagation and Group Velocity. Academic Press, New York, 1960.

Brooks, A.N. and Hughes, T.J.R., "Streamline-Upwind /Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equation." Computer Methods in Applied Mechanics and Engineering, Volume 32 (1982), 199-259.

Buelow, P.E.O., Venkateswaran, S., and Merkle, C.L. "Grid Aspect Ratio Effects on the Convergence of Upwind Schemes." AIAA-95-0565, 33rd Aerospace Sciences Meeting and Exhibit, 9-12 January 1995, Reno, NV.

Byun, C. and Guruswamy, G.P. "Wing-Body Aeroelasticity Using Finite-Difference Fluid/Finite-Element Structural Equations on Parallel Computers." AIAA-94-1487, AIAA/ASME/ASCE/AHS/ASC 35th Structures, Structural Dynamics, and Materials Conference. 18-20 April 1994. Hilton Head, SC.

Cai, X.C., Keyes, D.E., and Venkatakrishnan, V. "Newton-Krylov-Schwarz: An Implicit Solver for CFD." Proceedings of the Eighth International Conference on Domain Decomposition Methods. Ed. R. Glowinski, et al. John Wiley & Sons, New York, 1997, pp. 387-400.

Cebral, J.R., and Lohner, R. "Fluid-Structure Coupling: Extensions and Improvements." AIAA-97-0858, 35th Aerospace Sciences Meeting & Exhibit. 6-10 January 1997. Reno, NV.

Cescotto, S., Frey, F., Fonder, G. "Total and Updated Lagrangian Descriptions in Nonlinear Structural Analysis: A Unified Approach." Energy Methods in Finite Element Analysis. Ed. R. Glowinski, E.Y. Rodin, O.C. Zienkiewicz. John Wiley & Sons, 1979, pp. 283-296.

Choi, Y.H., and Merkle, C.L. "The Application of Preconditioning in Viscous Flows." Journal of Computation Physics, Volume 105, Number 2, 1993.

Conte, S.D. and de Boor, C. "Elementary Numberical Analysis, An Algorithmic Approach." 3rd edition, McGraw-Hill, New York, 1980.

Chorin, A.J. "A Numerical Method for Solving Incompressible Viscous Flow Problems." Journal of Computational Physics, Volume 2, 1967.

Craig, R.R., Jr. Structural Dynamics. John Wiley & Sons, New York, 1981.

Donea, J. Giuliani, S., and Halleux, J.P. "An Arbitrary Lagrangian-Eulerian Finite Element Method for Transient Dynamic Fluid-Structure Interactions." Computer Methods in Applied Mechanics and Engineering., Vol. 33, No. 1-3 (1982), 689-723.

Donea, Jean. "Arbitrary Lagrangian-Eulerian Finite Element Methods." Computational Methods for Transient Analysis. Ed. T. Belytschko and T.J.R. Hughes. Elsevier Science Publishers, 1983, pp. 473-516.

Donea, Jean. "A Taylor-Galerkin Method for Convective Transport Problems." International Journal for Numerical Methods in Engineering, Vol. 20, No. 1 (1984), 101-119.

Donea, J. and Lamain, L.G. "A Modified Representation of Transverse Shear in C^0 Quadrilateral Plate Elements." Computer Methods in Applied Mechanics and Engineering., Vol. 63, No. 2 (1987), 183-207.

Edwards, W.S., Tuckerman, L.S., Friesner, R.A., and Sorensen, D.C. "Krylov Methods for the Incompressible Navier-Stokes Equations." Journal of Computational Physics, Volume 110, Number 1, 1994, pp. 82-102.

Farhat, C., Lesoinne, M., Chen, P.S. and Lanteri, S. "AIAA-95-1290-CP, Parallel Heterogeneous Algorithms for the Solution of Three-Dimensional Transient Coupled Aeroelastic Problems." Proceedings of the AIAA/ASME/ASCE/AHS/ASC 36th Structures, Structural Dynamics, and Materials Conference and AIAA/ASME Adaptive Structures Forum. 10-13 April 1995. New Orleans, LA.

Farhat, C. and Lin, Y.T. "Transient Aeroelastic Computations Using Multiple Moving Frames of Reference." AIAA-90-3053, 8th Applied Aerodynamics Conference, 20-22 August 1990, Portland, OR.

Fung, Y.C. An Introduction to the Theory of Aeroelasticity. John Wiley & Sons, London, 1955.

Godfrey, A.G., Walters, R.W., and van Leer, B. "Preconditioning for the Navier-Stokes Equations with Finite-Rate Chemistry." AIAA-93-0535, 31st Aerospace Sciences Meeting & Exhibit, 11-14 January 1993, Reno, NV.

Golub, G., and Van Loan, C. Matrix Computations. Second Edition, Johns Hopkins University Press, Baltimore, 1989.

Gropp, W.D., Keyes, D. E., McInnes, L.C. and Tidriri, M.D. "Parallel Implicit PDE Computations: Algorithms and Software." Proceedings of ``Parallel CFD '97", Editors A. Ecer, et al., 1997, pp. 333-344.

Guillot, D.M., and Friedmann, P.P. "A Fundamental Aeroservoelastic Study Combining Unsteady CFD with Adaptive Control." AIAA-94-1721, Dynamic Specialists Conference, 21-22 April 1994, Hilton Head, SC.

Guruswamy, G.P. "Integrated Approach for Active Coupling of Structures and Fluids." AIAA Journal, Volume 27, Number 6 (1989), 788-793.

Guruswamy, G.P. and Byun, C., "Fluid-Structure Interactions Using Navier-Stokes Flow Equations Coupled with Shell Finite Element Structures." AIAA-93-3087, 1993.

Hajela, P. Recent Trends in Aeroelasticity, Structures, and Structural Dynamics. Editor Prabhat Hajela, University of Florida Press, 1986.

Hibbitt, H.D., Marcal, P.V., and Rice, J.R. "A Finite Element Formulation for Problems of Large Strain and Large Displacement." International Journal of Solids and Structures. Volume 6, 1970.

Hirsch, Charles., Fundamentals of Numerical Discretization. Vol. I of Numerical Computation of Internal and External Flows. John Wiley & Sons, 1988.

Hirsch, Charles., Computational Methods for Inviscid and Viscous Flows. Vol. II of Numerical Computation of Internal and External Flows. John Wiley & Sons, 1990.

Hirt, C.W., Amsden, A.A., and Cook, J.L. "An Arbitrary Lagrangian-Eulerian Computing Method for All Flow Speeds." Journal of Computational Physics. Vol. 14, No. 1 (1974), 227-253.

Hughes, T.J.R., Franca, L.P., and Mallet, M., "A New Finite Element Formulation for Computational Fluid Dynamics: I. Symmetric Forms of the Compressible Euler and Navier-Stokes Equations and the Second Law of Thermodynamics." Computer Methods in Applied Mechanics and Engineering, Vol. 54, No. 2 (1986), 223-234.

Hughes, T.J.R., Mallet, Michel, and Mizukami, Akira, "A New Finite Element Formulation for Computational Fluid Dynamics: II. Beyond SUPG." Computer Methods in Applied Mechanics and Engineering, Vol. 54, No. 3 (1986), 341-355.

Hughes, T.J.R. and Mallet, Michel, "A New Finite Element Formulation for Computational Fluid Dynamics: III. The Generalized Streamline Operator for Multidimensional Advective-Diffusive Systems." Computer Methods in Applied Mechanics and Engineering, Vol. 58, No. 3 (1986), 305-328.

Hughes, T.J.R. and Mallet, Michel, "A New Finite Element Formulation for Computational Fluid Dynamics: IV. A Discontinuity-Capturing Operator for Multidimensional Systems." Computer Methods in Applied Mechanics and Engineering, Vol. 58, No. 3 (1986), 329-336.

Hughes, T.J.R., Franca, L.P., and Mallet, M., "A New Finite Element Formulation for Computational Fluid Dynamics: VI. Convergence Analysis of the Generalized SUPG Formulation for Linear Time-Dependent Multidimensional Advective-Diffusive Systems." Computer Methods in Applied Mechanics and Engineering, Vol. 63, No. 13 (1987), 97-112.

Hughes, T.J.R. "Recent Progress in the Development and Understanding of SUPG Methods with Special Reference to the Compressible Euler and Navier-Stokes Equations." Finite Elements in Fluids - Volume 7. Ed. R.H. Gallagher, R. Glowinski, P.M. Gresho, J.T. Oden and O.C. Zienkiewicz. John Wiley & Sons, Ltd, 1987, pp. 273-287.

Hughes, T.J.R. "Finite Element Methods for Fluids." AGARD Report 787. Special Course on Unstructured Grid Methods for Advection Dominated Flows. 1992, Chapter 2.

Idelsohn, S., Nigro, N., Storti, M., and Buscaglia, G., "A Petrov-Galerkin Formulation for Advection-Reaction-Diffusion Problems." Computer Methods in Applied Mechanics and Engineering, Vol. 136, Nos. 1-2 (1996), 27-46.

Ingard, K.U. Fundamentals of Waves and Oscillations. Cambridge University Press, Cambridge, 1988.

Irons, B.M. Applications of a theorem on eigenvalues to finite element problems. CR/132/70. University of Wales, Department of Civil Engineering, Swansea, 1970.

Johnson, Claes, "Streamline Diffusion Finite Element Method for Compressible and Incompressible Fluid Flow." Finite Elements in Fluids - Volume 8. Ed. T.J. Chung. Hemisphere Publishing, 1992, pp. 75-96.

Janus, J.M. and Whitfield, D.L. "Counterrotating Prop-Fan Simulations which Feature a Relative-Motion Multiblock Grid Decomposition Enabling Arbitrary Time Steps." AIAA-90-0687, 28th Aerospace Science Meeting, 8-11 January 1990.

Kania, L. "An Algebraic Adaptive Grid Generation Scheme for Analysis of Complex Fluid Flows." AIAA-96-0030, 34th Aerospace Sciences Meeting & Exhibit. 15-18 January 1996. Reno, NV.

Kaushik, D.K., Keyes, D.E., and Smith, B.F. "Newton-Krylov-Schwarz Methods for Aerodynamics Problems: Compressible and Incompressible Flows on Unstructured Grids." Proceedings of the 11th International Conference On Domain Decomposition Methods, Editors C.-H. Lai, et al, 1998.

Kelly, C.T. "Iterative Methods for Linear and Nonlinear Equations." Society of Industrial and Applied Mathematics, Philadelphia, 1995.

Keyes, D.E. "How Scalable is Domain Decomposition in Practice?" Proceedings of the 11th International Conference On Domain Decomposition Methods, Editors C.-H. Lai, et al, 1998.

Kinght, N.F., Jr., Macy, S.C., and McCleary, S.L. "Assessment of Structural Analysis Technology for Elastic Shell Collapse Problems." Computational Structural Mechanics and Multidisciplinary Optimization, Winter Annual Meeting of the American Society of Mechanical Engineers, 10-15 December 1989, San Francisco, CA.

LeBeau, G.J., Ray, S.E., Aliabadi, S.K., and Tezduyar, T.E., "SUPG Finite Element Computation of Compressible Flows with the Entropy and Conservation Variables Formulations." Computer Methods in Applied Mechanics and Engineering, Vol. 104, No. 3 (1993), 397-422.

Lesoinne, M. and Farhat, C. "Geometric Conservation Laws for Flow Problems with Moving Boundaries and Deformable Meshes, and their Impact on Aeroelastic Computations." Computer Methods in Applied Mechanics and Engineering, Vol. 134, Nos. 1-2 (1996), 71-90.

Lesoinne, M. and Farhat, C. "A Numerical Method for Solving Aeroelastic Eigenproblems in all Flight Regimes." AIAA-97-0647, 35th Aerospace Sciences Meeting & Exhibit, 6-10 January 1997, Reno, NV.

Lohner, R. "Finite Element Methods in CFD: Grid Generation, Adaptivity and Parallelization." AGARD Report 787. Special Course on Unstructured Grid Methods for Advection Dominated Flows. 1992, Chapter 8.

MacMurdy, D.E., Guruswamy, G.P., and Kapania, R.K. "Static Aeroelastic Analysis of Wings Using Euler/Navier-Stokes Equations Coupled with Improved Wing-Box Finite Element Structures." AIAA-94-1587, AIAA/ASME/ASCE/AHS/ASC 35th Structures, Structural Dynamics, and Materials Conference. 18-20 April 1994. Hilton Head, SC.

Malvern, L.E. Introduction to the Mechanics of a Continuous Medium. Englewood Cliffs, New Jersey: Prentice-Hall, 1969.

MathWorks, "MATLAB Reference Guide." The MathWorks, Inc. 1992.

MathWorks, "MATLAB User's Guide." The MathWorks, Inc. 1992.

McHugh, P.R., Knoll, D.A., and Keyes, D.E. "Schwarz-Preconditioned Newton-Krylov Algorithm for Low Speed Combustion Problems." AIAA-96-0911, 34th Aerospace Sciences Meeting & Exhibit. 15-18 January 1996. Reno, NV.

McNamara, J.F., and Marcal, P.V. "Incremental Stiffness Method for Finite Element Analysis of the Nonlinear Dynamic Problem." Numerical and Computer Methods in Structural Mechanics. Editors S.J. Fenves, N. Perrone, A.R. Robinson, and W.C. Schnobrich., Academic Press, New York, 1973.

Merkle, Charles L. "Preconditioning Methods for Viscous Flow Calculations." Computational Fluid Dynamics Review 1995, Ed. M. Hafez and K. Oshima, Chichester, England.: John Wiley & Sons, (1995), pp. 419-436.

Merkle, C.L., and Choi, Y.H. "Computation of Low-Speed Flow with Heat Addition." AIAA Journal. Vol 25, Number 5, 1987.

Merkle, C.L., and Choi, Y.H. "Computation of Low-Speed Compressible Flows with Time-Marching Procedures." International Journal for Numerical Methods in Engineering. Vol 25, Number 2, 1988.

Mestreau, E. and Lohner, R. "Airbag Simulation Using Fluid/Structure Coupling." AIAA-96-0798, 34th Aerospace Sciences Meeting & Exhibit. 15-18 January 1996. Reno, NV.

McNamara, John F., and Marcal, Pedro V. "Incremental Stiffness Method for Finite Element Analysis of the Nonlinear Dynamic Problem." Numerical and Computer Methods in Structural Mechanics, Ed. S.J. Fenves, N. Perrone, A.R. Robinson, W.C. Schnobrich, New York: Academic Press, (1973), pp. 353-376.

Noble, B. and Daniel, J. Applied Linear Algebra. Third Edition, Prentice-Hall, Englewood Cliffs, NJ, 1988.

Oakley, D.R. and Knight, N.F. "Adaptive Dynamic Relaxation Algorithm for Non-Linear Hyperelastic Structures, Part I. Formulation." Computer Methods in Applied Mechanics and Engineering, Volume 126, Numbers 1-2, (1995), 67-89.

Park, K.C. and Felippa, C.A. "Partitioned Analysis of Coupled Systems." Computational Methods for Transient Analysis, Volume 1 in Computational Methods in Mechanics, Editors Belytschko, T., and Hughes, T.J.R., Elsevier Science Publishers (1983), 157-219.

Pracht, William E. "Calculating Three-Dimensional Fluid Flows at All Speeds with an Eulerian-Lagrangian Computing Mesh." Journal of Computational Physics. Vol. 17 (1975), 132-159.

Pueyo, A., and Zingg, D.W. "Progress in Newton-Krylov Methods for Aerodynamic Calculations." AIAA-97-0877, 35th Aerospace Sciences Meeting & Exhibit. 6-10 January 1997. Reno, NV.

Ray, S.E., Wren, G.P., and Tezduyar, T.E. "Simulations of Compressible Fluid-Elastic Solid Interactions." AIAA-97-10872, 35th Aerospace Sciences Meeting & Exhibit. 6-10 January 1997. Reno, NV.

Raymond, W.H., and Garder, A. "Selective Damping in a Galerkin Method for Solving Wave Problems with Variable Grids." Monthly Weather Review, Volume 104 (1976), 1583-1590.

Redwood, M. Mechanical Waveguides. Pergamon Press, New York, 1960.

Reed, C.L., and Anderson, D.A. "Application of Low Speed Preconditioning to the Compressible Navier-Stokes Equations." AIAA-97-0873, 35th Aerospace Sciences Meeting & Exhibit. 6-10 January 1997. Reno, NV.

Saad, Y., and Schultz, M. "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems." SIAM Journal of Scientific and Statistical Computations, Volume 7, No. 3 (1886), 856-869.

Saleem, M., Pulliam, T. and Cheer, A.Y. "Acceleration of Convergence and Spectrum Transformation of Implicit Finite Difference Operators Associated with Navier-Stokes Equations." Journal of Computational Physics, Volume 104, No. 1 (1993), 1-13.

Sheng, C., Chen, J.P., Taylor, L., Jiang, M.Y., and Whitfield, D., "Unsteady Multigrid Method for Simulating 3-D Incompressible Navier-Stokes Flows on Dynamic Relative Motion Grids." AIAA-97-0446, 35th Aerospace Sciences Meeting & Exhibit. 6-10 January 1997. Reno, NV.

Shuen, J.S., Chen, K.H., and Choi, Y. "A Time-Accurate Algorithm for Chemical Non-Equilibrium Viscous Flows at All Speeds." AIAA-92-3639, AIAA/SAE/ASME/ASEE 28th Joint Propulsion Conference and Exhibit, 6-8 July 1992, Nashville, TN.

Smith, M.J., Cesnik, C.E.S., Hodges, D.H., and Moran, K.J., "An Evaluation of Computational Algorithms to Interface Between CFD and CSD Methodologies." AIAA-96-1400, 37th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference. 15-17 April 1996, Salt Lake City, UT.
Soni, B.K., and Yang, Jiann-Cherng. "General Purpose Adaptive Grid Generation System." AIAA-92-0664, 30th Aerospace Sciences Meeting & Exhibit. 6-9 January 1992. Reno, NV.

Stolarski, H. and Belytschko, T. "Shear and Membrane Locking in Curved C^0 Elements." Computer Methods in Applied Mechanics and Engineering, Vol. 41, No. 3 (1983), 279-296.

Stolarski, H. and Belytschko, T. "On the Equivalence of Mode Decomposition and Mixed Finite Elements based on the Hellinger-Reissner Principle, Part I: Theory." Computer Methods in Applied Mechanics and Engineering, Vol. 58, No. 3 (1986), 249-263.

Stolarski, H. and Belytschko, T. "On the Equivalence of Mode Decomposition and Mixed Finite Elements based on the Hellinger-Reissner Principle, Part II: Applications." Computer Methods in Applied Mechanics and Engineering, Vol. 58, No. 3 (1986), 265-284.

Strang, G. Introduction to Applied Mathematics. Wellesley-Cambridge Press, Wellesley, MA, 1986.

Strang, G. and Fix, G.J. An Analysis of the Finite Element Method. Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

Stricklin, J.A., Von Riesemann, W.A., Tillerson, J.R., and Haisler, W.E. "Static Geometric and Material Nonlinear Analysis." Advances in Computational Methods in Structural Mechanics and Design. Editors J.T. Oden, R.W. Clough, and Y. Yamamoto, University of Alabama-Huntsville Press (1972), Huntsville, AL.

Tamura, Y. and Fujii, K. "Conservation Law for Moving and Transformed Grids." AIAA-93-3365, 11th AIAA Computational Fluid Dynamics Conference, 6-9 July 1993. Orlando, FL.

Taylor, A.E., and Mann, W.R. Advanced Calculus. 3rd edition, John Wiley & Sons: New York, 1983.

Tezduyar, T.E., and Park, Y.J., "Discontinuity-Capturing Finite Element Formulations for Nonlinear Convection-Diffusion-Reaction Equations." Computer Methods in Applied Mechanics and Engineering, Vol. 59, No. 3 (1986), 307-325.

Tezduyar, T.E., Park, Y.J., and Deans, H.A., "Finite Element Procedures for Time-Dependent Convection-Diffusion-Reaction Systems." Finite Elements in Fluids - Volume 7. Ed. R.H. Gallagher, R. Glowinski, P.M. Gresho, J.T. Oden and O.C. Zienkiewicz. John Wiley & Sons, Ltd, 1987, pp. 25-45.

Tezduyar, T.E., Liou, J., Ganjoo, D.K., Behr, M., and Glowinski, R., "Unsteady Incompressible Flow Computations with the Finite Element Method." Finite Elements in Fluids - Volume 8. Ed. T.J. Chung. Hemisphere Publishing, 1992, pp. 177-209.

Tezduyar, T.E., "Finite Element Computation of Unsteady Incompressible Flows involving Moving Boundaries and Interfaces and Iterative Solution Strategies." AGARD Report 787. Special Course on Unstructured Grid Methods for Advection Dominated Flows. 1992, pp. 3.1-3.25.

Tezduyar, T.E., Behr, M., and Liou, J., "A New Strategy for Finite Element Computations Involving Moving Boundaries and Interfaces - The Deforming -Spatial-Domain/Space-Time Procedure: I. The

Concept and the Preliminary Numerical Tests." Computer Methods in Applied Mechanics and Engineering, Vol. 94, No. 3 (1992), 339-351.

Thomas, P.D., and Lombard, C.K. "Geometric Conservation Laws and Its Application to Flow Computations on Moving Grids." AIAA Journal, Volume 17, Number 10, 1978, pp. 1030-1037.

Thompson, Joe F. "A Survey of Dynamically-Adaptive Grids in the Numerical Solution of Partial Differential Equations." Applied Numerical Mathematics, Vol. 1, No. 1 (January 1985), 3-27.

Tidriri, M.D., "Krylov and Schwarz Methods for Compressible Flows." ICASE Research Quarterly, Volume 4, Number 3, 1995.

Timoshenko, S.P. and Gere, J.M. Mechanics of Materials. Litton Educational Publishing (1972), Monterey, CA.

Turkel, E. "Preconditioned Methods for Solving the Incompressible and Low Speed Compressible Equations." Journal of Computational Physics, Volume 72, Number 2, 1987, pp. 277-298.

Turkel, E. and Arnone, A. "Pseudo-Compressibility Methods for the Incompressible Flow Equations." AIAA-93-3329, 11th AIAA Computational Fluid Dynamics Conference, 6-9 July 1993, Orlando, FL.

Underwood, P. "Dynamic Relaxation." Computational Methods for Transient Analysis, Volume 1 in Computational Methods in Mechanics, Editors Belytschko, T., and Hughes, T.J.R., Elsevier Science Publishers (1983), 245-265.

Venkatakrishnan, V. "Newton Solution of Inviscid and Viscous Problems." AIAA Journal, Volume 27, Number (1989), 885-891.

Venkateswaran, S. and Merkle, Charles L. "AIAA 95-0078; Dual Time-Stepping and Preconditioning for Unsteady Computations." Presented at 33rd Aerospace Sciences Meeting of the American Institute of Aeronautics and Astronautics. 9-12 January 1995. Reno, Nevada.

Wood, W.L. Practical Time-stepping Schemes. Oxford University Press, New York, 1990.

Wriggers, P. and Reese, S. "A Note on Enhanced Strain Methods for Large Deformations." Computer Methods in Applied Mechanics and Engineering, Vol. 135, Nos. 3-4 (1996), 201-209.

Yamada, Y. "Incremental Formulation for Problems with Geometric and Material Nonlinearities." Advances in Computational Methods in Structural Mechanics and Design. Editors J.T. Oden, R.W. Clough, and Y. Yamamoto, University of Alabama-Huntsville Press (1972), Huntsville, AL.

Yang, C. and Lohner, R. "Numerical Simulation of a Maneuvering Missile Using a Loose Fluid-Structure Coupling Algorithm." AIAA-97-0408, 35th Aerospace Sciences Meeting & Exhibit, 6-10 January 1997, Reno, Nevada.

Zienkiewicz, O.C. and Taylor, R.L. The Finite Element Method. Fourth Edition, Volume 1, McGraw-Hill Book Company, London, 1989.

Zienkiewicz, O.C. and Taylor, R.L. The Finite Element Method. Fourth Edition, Volume 2, McGraw-Hill Book Company, London, 1991.

Zori, L., and Rajagopalan, G. "The Development of an Euler/Navier-Stokes Flow Solver for all Speeds on 2D Unstructured Meshes." AIAA-97-0332, 35th Aerospace Sciences Meeting & Exhibit. 6-10 January 1997. Reno, NV.

APPENDICES

APPENDIX A

EIGENVALUES

APPENDIX A

EIGENVALUES

Rayleigh Quotient

(From [Noble and Daniel, pp 417-418].) The Rayleigh quotient of a hermitian matrix A is the function ξ defined for $v \neq 0$ by

$$\xi = \frac{v^T A v}{v^T v} \tag{A-1}$$

This minimizes $\|(A - \xi I)v\|_2$. If A is $p \times p$ and has eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_p$ and an associated set of orthonormal eigenvectors then $\lambda_1 \leq \xi \leq \lambda_p$ for all $v \neq 0$. λ_1 is the minimum value of ξ for $v \neq 0$, and $\xi = \lambda_1$ if and only if v is an eigenvector associated with λ_1. λ_p is the maximum value of ξ for $v \neq 0$, and $\xi = \lambda_p$ if and only if v is an eigenvector associated with λ_p. Since $\xi = \lambda_1$ if and only if v is an eigenvector associated with λ_1, it is can be shown that ξ will be near λ_1 when v is near an eigenvector associated with λ_1. A version of the Rayleigh quotient for non-symmetric matrices derives from premultiplying $Av = \lambda v$ with an arbitrary vector y^T to get $y^T Av = y^T \lambda v$. Rearranging gives

$$\lambda = \frac{y^T A v}{y^T v} \tag{A-2}$$

Rayleigh Quotient Iteration

(From [Golub and Van Loan, pp. 440-441].) The Rayleigh quotient has numerous practical applications including determining the eigenvalue associated with an approximate eigenvector. Conversely, the eigenvector associated with an approximate eigenvalue comes from $(A - \lambda I)v = b$ as

$b \to 0$. The Rayleigh quotient iteration combines these to quickly determine an approximate eigenvalue and eigenvector from an initial guess.

$$
\begin{aligned}
&\text{given } v_0, \|v_0\| = 1 \\
&\text{for } k = 0,1,\ldots \\
&\quad \mu_k = \frac{v_k^T A v_k}{v_k^T v_k} \\
&\quad \text{Solve } (A - \mu_k I) z_{k+1} = v_k \text{ for } z_{k+1} \\
&\quad v_{k+1} = \frac{z_{k+1}}{\|z_{k+1}\|_2} \\
&\text{end}
\end{aligned}
\tag{A-3}
$$

The Rayleigh quotient iteration converges globally and achieves a cubic rate of convergence.

Eigenvalue Bounds

(From [Irons, 1970] and [Wood, pp 10-13].) Start with the general form of eigenvalue-eigenvector problems stated as

$$Ax = \lambda x \tag{A-4}$$

If A is not positive definite, a multiple of the identity may be added to make it so [Strang, 1986].

$$(nI + A)x = \mu x \tag{A-5}$$

The eigenvectors will be the same but the eigenvalues will be shifted. Alternatively, a negative shift may be implemented as

$$(I - hA)x = \mu x \tag{A-6}$$

where $h > 0$ is a scalar. Again, A and $I - hA$ share the same set of eigenvectors and the eigenvalues are related by

$$\lambda_i = \frac{(1 - \mu_i)}{h} \tag{A-7}$$

The concept is the same if $n = \frac{1}{h}$

Partial differential equations stated as a first order in time with a mass and a stiffness matrix, may be written as

$$Kv = \omega Mv \text{ where } \lambda = \omega \tag{A-8}$$

Second order problems with zero damping may be similarly written as

$$Kv = \omega^2 Mv \text{ where } \lambda = \omega^2 \tag{A-9}$$

If K and M are symmetric (or hermitian) and M is positive definite, the eigenvalue bounds are given by the generalized Rayleigh quotient

$$\xi = \frac{v^T K v}{v^T M v} \tag{A-10}$$

In finite elements, the mass and stiffness matrices are usually calculated element by element and then assembled into a global mass and stiffness matrix. This process can be written as

$$\xi = \frac{\sum_j v_j{}^T K_j^{(e)} v_j}{\sum_j v_j{}^T M_j^{(e)} v_j} \tag{A-11}$$

where $K_j^{(e)}$ and $M_j^{(e)}$ are the stiffness and mass matrices for the jth element and v_j is the corresponding component of the vector v. Then if $\xi_j^{(e)}$ is the Rayleigh quotient for the jth element,

$$\xi_j^{(e)} = \frac{v_j{}^T K_j^{(e)} v_j}{v_j{}^T M_j^{(e)} v_j} \rightarrow \xi_1^{(e)} = \frac{v_1{}^T K_1^{(e)} v_1}{v_1{}^T M_1^{(e)} v_1}, \xi_2^{(e)} = \frac{v_2{}^T K_2^{(e)} v_2}{v_2{}^T M_2^{(e)} v_2}, \text{ etc.} \tag{A-12}$$

ξ can be rewritten as a weighted mean in $\xi_j^{(e)}$ using positive weights

$$\xi = \frac{\sum_j \alpha_j \xi_j^{(e)}}{\sum_j \alpha_j} \text{ and } \alpha_j = v_j^T M_j^{(e)} v_j > 0 \tag{A-13}$$

since the mass matrix for each element is positive definite. Since the Rayleigh quotient, ξ, is a weighted mean of the element Rayleigh quotients, $\xi_j^{(e)}$ with positive weights, it is bounded above and below by the element quotients.

$$\xi_{\min}^{(e)} \le \xi \le \xi_{\max}^{(e)} \tag{A-14}$$

This describes the bounds on the eigenvalues of the global problem from the Rayleigh quotients of the elements.

Lumped and Consistent Mass Matrices

It is possible to estimate the difference in the global bounds for consistent and lumped mass matrices using the same approach. The eigenvalues of an individual element are bounded by the ratio of eigenvalues from the spectrum of the mass and stiffness matrices.

$$\xi_j^{(e)} \le \frac{\max\left\{v_j{}^T K_j^{(e)} v_j\right\}}{\min\left\{v_j{}^T M_j^{(e)} v_j\right\}} \text{ where } \max\left\{v_j{}^T K_j^{(e)} v_j\right\} \text{and } \min\left\{v_j{}^T M_j^{(e)} v_j\right\} \tag{A-15}$$

represent the maximum and minimum eigenvalues from the spectrum of the elements stiffness and mass matrices, respectively.

Using natural linear elements in one and two dimensions, the distributed elemental mass matrices are

$$M_D^{(e)} = \frac{h_j}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \text{ and } M_D^{(e)} = \frac{A_j}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \tag{A-16}$$

where h_j and A_j are measures of the length and area of the elements. The associated eigenvalues spectrums are $\left\{\frac{h_j}{2}, \frac{h_j}{6}\right\}$ and $\left\{\frac{A_j}{12}, \frac{A_j}{12}, \frac{A_j}{3}\right\}$. With lumped mass matrices,

$$M_L^{(e)} = \frac{h_j}{2}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \text{ and } M_L^{(e)} = \frac{A_j}{3}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A-17}$$

and the associated eigenvalue spectrums are $\left\{\frac{h_j}{2}, \frac{h_j}{2}\right\}$ and $\left\{\frac{A_j}{3}, \frac{A_j}{3}, \frac{A_j}{3}\right\}$.

For a given stiffness matrix spectrum, element eigenvalues are reduced in the ratios of 3:1 and 4:1 for the one and two dimensional elements first order in time. For systems second order in time, these are $\sqrt{3}$:1 and 2:1.

This gives an idea of the magnitude of the effects produced by lumping mass matrices. Lumping scales the eigenvalues of the elements in the given ratios which, in turn, scales the bounds on the global eigenvalues. However, if lumping is applied for all elements in a system, all elements are scaled equally. While this scales the global spectrum of eigenvalues, it does not necessarily change the condition.

Gershgorin Circle Theorem

(From [Noble and Daniel, pp 317-318].) Each (real or complex) eigenvalue λ of $p \times p$ matrix A with $\langle A \rangle_{ij} = a_{ij}$ satisfies at least one of the inequalities

$$|\lambda - a_{ii}| \le r_i \quad \text{where} \quad r_i = \sum_{\substack{j=1 \\ j \ne i}}^{p} |a_{ij}| \; (i = 1, \ldots, p) \tag{A-18}$$

i.e. each eigenvalue lies in at least one of the discs with center $a_{ii} = \langle A \rangle_{ii}$ and radius r_i. If the union of n of the discs is disjoint from the remainder, then there are precisely n eigenvalues of A in that cluster.

Condition

(From [Noble and Daniel, pp 317-318] and [Golub and Van Loan, pp 341-349].) The general notion of the condition applies to both physical and discrete problems. It is a measure of how the "answer" changes in comparison to changes in the "data." The "data" may come from physical properties, geometry,

measurements, models, etc and have inherent values and inaccuracies. If small changes in the data produce small changes in the answer, the problem is well-conditioned. If small changes in the data can sometimes produce large changes in the answer, the problem is ill-conditioned.

There are differing but closely related measures of condition that come into play. The primary definition for condition of a matrix is

$$\kappa(A) = \|A\| \|A^{-1}\| \tag{A-19}$$

for any matrix norm. The matrix, A, may be coefficients in a continuous linearized system of equations or the discrete representation of the system.

The 2-norm, $\|\cdot\|_2$, equals the maximum singular value of A = (maximum eigenvalue of $A^T A)^{1/2}$. Thus, if the matrix is symmetric, the condition may also be written

$$\kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}} \tag{A-20}$$

Many authors use this definition (or approximation) for the condition outside the limits of matrix symmetry.

Matrix condition (actually its reciprocal) measures how far a matrix is from the nearest singular matrix. A well-conditioned matrix is far from a singular matrix and and ill-conditioned matrix may be very near to a singular matrix. If a matrix has a condition of 10^8, a singular matrix is within about 10^{-8}.

Other measures used to indicate the condition are more specialized to specific applications. For discrete fluid flow problems, the ratio of CFL numbers gives

$$\kappa = \frac{CFL_{\max}}{CFL_{\min}} = \frac{(\lambda \Delta t / \Delta x)_{\max}}{(\lambda \Delta t / \Delta x)_{\min}}. \tag{A-21}$$

This reduces to the definition based on eigenvalues if the time step and grid size are constant. However, fluid matrices are seldom symmetric.

For physical problems, the condition may be inferred from the various time scales involved in the problem. Without a rigorous definition, the condition is related to

$$\kappa = \frac{t_{\max}}{t_{\min}}. \tag{A-22}$$

Fast moving events or thin physical dimensions have small time scales while slow moving events or large physical dimensions may have large time scales. If the eigenvalue is viewed as the speed with which a signal or event is communicated in the medium, then time scale is the physical dimension divided by the velocity.

The condition provides insight into possible sensitivity to perturbations. For example, a simple disturbance to the system $Ax = b$ leads to $\|b\| = \|Ax\| \le \lambda_{\max}\|x\|$ and $\|\delta b\| = \|A\delta x\| \ge \lambda_{\min}\|\delta x\|$. Rearranging gives

$$\frac{\|\delta x\|}{\|x\|} \le \frac{\lambda_{\max}}{\lambda_{\min}}\frac{\|\delta b\|}{\|b\|} = \kappa\frac{\|\delta b\|}{\|b\|} \tag{A-23}$$

by the previous definition. Changes in the data vector b can (but not necessarily) lead to large changes in the solution. Perturbations may be extended to changes in the coefficient matrix as well through the concepts of eigenvalue and eigenvector condition.

The Bauer-Fike theorem for eigenvalue condition supposes $p \times p$ A has a linearly independent set of eigenvectors $x_1,\ldots,x_p$ with associated eigenvalues λ_i, and that P is the nonsingular matrix of eigenvectors $P = [x_1 \cdots x_p]$. If λ is an eigenvalue of the perturbed matrix $A + \delta A$, then λ satisfies at least on of the inequalities

$$|\lambda - \lambda_i| \le \left\| P \left\| P^{-1} \right\| \delta A \right\| \text{ for } i = 1,\ldots,p \tag{A-24}$$

If P is unitary or orthogonal (as it would be for A symmetric) so that $\left\| P \left\| P^{-1} \right\| \right\| = 1$, then λ satisfies at least one of $|\lambda - \lambda_i| \le \|\delta A\|$.

This shows that the condition (sensitivity to perturbations) of the eigenvalues of A depends directly on the condition $\kappa = \left\| P \left\| P^{-1} \right\| \right\|$ of the matrix P of eigenvectors of A. A normal matrix ($A^T A = AA^T$) cannot have extreme eigenvalue sensitivity while a non-normal matrix can have well-conditioned, ill-conditioned or a mixture of well-conditioned and ill-conditioned eigenvalues.

The Bauer-Fike theorem does not really address the sensitivity of a particular eigenvalue. Suppose λ is a simple eigenvalue of A and that x and y are the normalized right and left eigenvectors of A associated with λ. Golub and Van Loan show that λ moves by an amount less than or equal to the inverse of $|y^T x|$. The reciprocal of

$$s(\lambda) = |y^T x| \tag{A-25}$$

is the condition of the simple eigenvalue λ. If the disturbed matrix is written as $(A + \varepsilon \cdot \delta A)x(\varepsilon) = \lambda(\varepsilon)x(\varepsilon)$ where $\|\delta A\|_2 = 1, \|x(\varepsilon)\|_2 = 1, \lambda(0) = 0$ and $x(0) = 0$ then an order ε perturbation may perturb an eigenvalue λ by an amount $\varepsilon / s(\lambda)$. Note that $s(\lambda)$ is actually the cosine of the angle between the left and right eigenvectors so a small value implies provides ill-conditioning and implies A is near a matrix with a repeated eigenvalue at λ.

Sensitivity of repeated eigenvalues is more difficult and much less specific. In general, $O(\varepsilon)$ perturbations in A can result in $O\left(\varepsilon^{1/p}\right)$ perturbations in λ if λ is a defective eigenvalue associated with a p-dimensional Jordan block. (A defective eigenvalue is an eigenvalue repeated P times with less than P associated independent eigenvectors.)

Eigenvectors may also be sensitive to changes in A and eigenvector condition follows the same general analysis as eigenvalues. For A with distinct eigenvalues, a perturbation to A moves an eigenvector as

$$x_k(\varepsilon) \approx x_k + \sum_{\substack{i=1 \\ i \neq k}}^{n} \frac{y_i^T \delta A x_k}{(\lambda_k - \lambda_i) y_i^T x_i} x_i + O(\varepsilon^2). \tag{A-26}$$

In effect, the condition of an eigenvector depends on the eigenvalue sensitivity and the separation of the associated eigenvalue from the other eigenvalues.

Matrices with distinct and well-separated eigenvalues are expected to have well-conditioned eigenvectors but matrices with repeated or clustered eigenvalues may have ill-conditioned eigenvectors.

A final form of condition measures the sensitivity of the invariant subspace of a matrix. An invariant subspace ς_0 of A is a subspace of V^p or $\mathcal{R}^p$ for which Ax is in ς_0 whenever x is in ς_0. The eigenvectors of A form an invariant subspace. The analysis above shows that as the eigenvalues begin to cluster, the eigenvectors begin to merge and become less determinant. Golub and Van Loan point out that a collection of sensitive eigenvalues can still define an insensitive invariant subspace if the cluster of eigenvalues is well separated from other eigenvalues or other eigenvalue clusters.

The measure of distance between two matrices A_{11} and A_{22} is the smallest singular value of the linear transformation $X \rightarrow A_{11}X - XA_{22}$. The transformation is nonsingular if and only if the intersection of the eigenvalue spectrums of A_{11} and A_{22} is the null set $\lambda(A_{11}) \cap \lambda(A_{22}) = \varnothing$. The separation between A_{11} and A_{22} is defined by

$$\text{sep}(A_{11}, A_{22}) = \min_{X \neq 0} \frac{\|A_{11}X - XA_{22}\|_F}{\|X\|_F} \tag{A-27}$$

where $\|\cdot\|_F$ is the Frobenius norm. If A_{11} and A_{22} represent blocks on the diagonal of A from a Schur decomposition and Q_1 represents the invariant subspace range of A_{11}, the reciprocal of the separation can be thought of as a condition number that measures the sensitivity of the range of Q_1 as an invariant subspace.

$$\kappa = \frac{\|\delta A\|_2}{\text{sep}(A_{11}, A_{22})} \text{ where } A = Q^T B Q = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} \tag{A-28}$$

is a Schur decomposition of B.

Condition of Finite Element Basis

(From [Strang and Fix, pp 205-207].) At the heart of the finite element approach rests the basis elements φ_j. Although the finite element solution is independent of the basis chosen roundoff error in the

computed solution and stability do depend on the basis. The linear independence of the basis elements is quantified by the inner products of the various basis elements

$$M_{jk} = \left(\varphi_j, \varphi_k\right) = \int_{\Omega} \varphi_j \varphi_k d\Omega \tag{A-29}$$

The M_{jk} make up the elements of the *Gram matrix,* M. This is essentially the mass matrix. The condition number based on eigenvalues is appropriate since M is symmetric and positive definite; $\kappa(M) = \dfrac{\lambda_{\max}(M)}{\lambda_{\min}(M)}$. On a regular mesh, the finite element basis functions are uniformly linearly independent, $\kappa(M) \leq$ constant, and the eigenvalues of M are all of the same order. With irregular elements, the various basis elements may be greatly different in size and the condition will increase but remain bounded. The condition may be improved by rescaling the elements using a diagonal matrix D so that $\kappa = \kappa\left(DMD^T\right)$ is well conditioned. However, [Strang and Fix, 1973] point out that when the condition number (due to irregular elements) is improved by rescaling, the numerical difficulties cured by rescaling were never likely to propagate throughout the whole problem. In effect, rescaling cures local difficulties and may improve the accuracy at a particular node but does not affect any global measure. The standard rule in scaling a sparse, positive definite matrix is to make all the diagonal elements equal.

Condition of Discrete vs. Continuous Problems

(From [Strang and Fix, pp 209-223].) Ultimately, the condition number of a numerical approximation is based on the governing physics (specifically the smallest eigenvalue and order of the governing equations) of the continuous problem and the finite element discretization chosen. Strang and Fix state that, for each variational problem and each choice of finite element, there is a constant c such that

$$\kappa \leq c h_{\min}^{-2m} \tag{A-30}$$

where $h_{\min}$ is a measure of the smallest element and $2m$ is the order of the equation. The constant depends on the inverse of the smallest eigenvalue of the continuous problem and it increases if the

geometry of the discretized elements becomes degenerate. If the physical problem is poorly conditioned, that must be reflected in the associated finite element approximation. Since the order of the finite element basis polynomial does not come into play, an implication is to improve condition by increasing the size of h and maintain accuracy by increasing the degree of the polynomials in the trial functions.

The fundamental (smallest) eigenvalue for a given continuous problem is often the most important. Strang and Fix prove the approximation in the discrete problem always equals or exceeds the fundamental eigenvalue in the continuous problem. For structures, this means the discrete structure is always stiffer than the continuous one. In fact, every eigenvalue in the continuous problem is approximated from above by its discrete analog so $\lambda_i \le \lambda_i^h$ for all i.

Perturbed Matrix Inverse

(From [Golub and Van Loan, p 51] and [Noble and Daniel, pp 129-132].) Suppose a given matrix A has inverse A^{-1}, the *Sherman-Morrison-Woodbury* formula provides a description of the inverse of a matrix summation $\left(A+UV^T\right)$ where UV^T is not limited to a small perturbation. The components U and V may be *n*-by-*k* if A is *k*-by-*k* and real.

$$\left(A+UV^T\right)^{-1} = A^{-1} - A^{-1}U\left(I+V^TA^{-1}U\right)^{-1}V^TA^{-1} \qquad \textbf{(A-31)}$$

If U and V are *k*-by-*k* and V is the identity matrix I, manipulation yields a variety of useful forms.

$$
\begin{aligned}
(A+U)^{-1} &= A^{-1} - A^{-1}U\left(I + A^{-1}U\right)^{-1}A^{-1} \\
&= A^{-1} - A^{-1}U(A+U)^{-1} \\
&= A^{-1}\left(I - U(A+U)^{-1}\right) \\
&= A^{-1}\left(I - \left(I + AU^{-1}\right)^{-1}\right) \\
&= A^{-1}\left(\left(I + AU^{-1}\right)\left(I + AU^{-1}\right)^{-1} - \left(I + AU^{-1}\right)^{-1}\right) \\
&= A^{-1}\left(\left(I + AU^{-1}\right) - I\right)\left(I + AU^{-1}\right)^{-1}
\end{aligned}
\qquad \text{(A-32)}
$$

$$
(A+U)^{-1} = A^{-1}\left\{\left(AU^{-1}\right)\left(I + AU^{-1}\right)^{-1}\right\} = A^{-1}P^{-1} \text{ where } P^{-1} = \left(AU^{-1}\right)\left(I + AU^{-1}\right)^{-1}
$$

$$
\text{note: } (PA)^{-1} = A^{-1}P^{-1}
$$

In effect, the additive matrix U may be restated as a left preconditioning matrix for A. Also,

$$
\begin{aligned}
(A+U)^{-1} &= A^{-1} - A^{-1}U\left(I + A^{-1}U\right)^{-1}A^{-1} \\
&= A^{-1} - \left(\left(I + A^{-1}U\right)U^{-1}A\right)^{-1}A^{-1} \\
&= \left(I - \left(\left(I + A^{-1}U\right)U^{-1}A\right)^{-1}\right)A^{-1} \\
&= \left(I - \left(U^{-1}A + I\right)^{-1}\right)A^{-1} \\
&= \left(\left(U^{-1}A + I\right)\left(U^{-1}A + I\right)^{-1} - \left(U^{-1}A + I\right)^{-1}\right)A^{-1} \\
&= \left(\left(U^{-1}A + I\right) - I\right)\left(U^{-1}A + I\right)^{-1}A^{-1}
\end{aligned}
\qquad \text{(A-33)}
$$

$$
(A+U)^{-1} = \left(U^{-1}A\right)\left(U^{-1}A + I\right)^{-1}A^{-1} = P^{-1}A^{-1} \text{ where } P^{-1} = \left(U^{-1}A\right)\left(U^{-1}A + I\right)^{-1}
$$

so the additive matrix may also be restated as a right preconditioning matrix for A.

The *Sherman-Morrison-Woodbury* formula may also be used as a technique to solve large systems of equations related to coupled field systems. For example, a fluid-structure-grid system contains a small number of large, almost independent subsystems with a small number of interconnections between the systems.

$$A' = \begin{matrix} \\ 1000< \\ 100< \\ 1000< \end{matrix} \begin{matrix} \overset{1000}{\wedge} & \overset{100}{\wedge} & \overset{1000}{\wedge} \\ \end{matrix} \begin{bmatrix} F & 0 & 0 \\ 0 & S & 0 \\ 0 & 0 & G \end{bmatrix} + \begin{matrix} \overset{50}{\wedge} & \overset{950}{\wedge} & \overset{50}{\wedge} & \overset{50}{\wedge} & \overset{250}{\wedge} & \overset{500}{\wedge} & \overset{250}{\wedge} \\ \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & W & 0 \\ P & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & B & 0 & 0 & 0 & 0 \end{bmatrix} \tag{A-34}$$

In this case,

$$A = \begin{matrix} \\ 1000< \\ 100< \\ 1000< \end{matrix} \begin{matrix} \overset{1000}{\wedge} & \overset{100}{\wedge} & \overset{1000}{\wedge} \\ \end{matrix} \begin{bmatrix} F & 0 & 0 \\ 0 & S & 0 \\ 0 & 0 & G \end{bmatrix}, \quad U = \begin{matrix} \\ 1000< \\ 100< \\ 1000< \end{matrix} \begin{matrix} \overset{50}{\wedge} & \overset{50}{\wedge} & \overset{500}{\wedge} \\ \end{matrix} \begin{bmatrix} 0 & 0 & W \\ P & 0 & 0 \\ 0 & B & 0 \end{bmatrix} \text{ and } V^T = \begin{matrix} \\ 1000< \\ 100< \\ 1000< \end{matrix} \begin{matrix} \overset{50}{\wedge} & \overset{950}{\wedge} & \overset{50}{\wedge} & \overset{50}{\wedge} & \overset{250}{\wedge} & \overset{500}{\wedge} & \overset{250}{\wedge} \\ \end{matrix} \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & I & 0 \end{bmatrix}.$$ The A matrix contains all the information for the internal relationships for the individual fluid, structure, and grid systems. The U matrix contains the coupling information that describes the interactions between the systems. Solving the fully coupled system by direct elimination would take around $k^3/3 = (1000+100+1000)^3/3 \approx 1.1\times10^{10}$ operations. However, solving using the *Sherman-Morrison-Woodbury* formula leads to

$$k_1^3/3 + k_2^3/3 + k_3^3/3 + k^2 + nk^2 + n^2k + n^3/3 = \\ 1000^3/3 + 100^3/3 + 1000^3/3 + 2200^2 + 600\times2200^2 + 600^2\times2200 + 600^3/3 = 5.7\times10^9 \tag{A-35}$$

operations. This represents about a 50% saving over the direct method.

APPENDIX B

GRAM SCHMIDT PROCESS

APPENDIX B

GRAM SCHMIDT PROCESS

(From [Noble and Daniel, pp 229-233].) Let $B = \{u_1, u_2, \ldots, u_q\}$ be an orthogonal spanning set for a given subspace. Any vector w may be expressed as a linear combination of the vectors as

$$w = \alpha_1 u_1 + \cdots + \alpha_q u_q \text{ where } \alpha_q = \frac{(u_i, w)}{(u_i, u_i)}. \quad \text{(B-1)}$$

The traditional and modified Gram Schmidt processes are designed to form an orthogonal set of basis vectors given a spanning set of vectors for a given subspace. The traditional method unfolds as follows.

Traditional Gram Schmidt

Given a spanning set $v_1, \ldots, v_q$, define $u_1 = v_1$.

For $2 \le i \le q$, define

$$u_i = v_i - \alpha_{1,i} u_1 - \cdots - \alpha_{i-1,i} u_{i-1} \text{ where } \alpha_{j,i} = \frac{(u_j, v_i)}{(u_j, u_j)} \text{ if } u_j \neq 0 \text{ and } \alpha_{j,i} = 0 \text{ if } u_j = 0 \quad \text{(B-2)}$$

This forms and orthogonal set of vectors $u_1, \ldots, u_q$ from a given spanning set $v_1, \ldots, v_q$. Traditional Gram Schmidt works through a series of orthogonal projections. The working vector is first projected onto the space defined by the orthogonal basis vectors. This piece is subtracted from the working vector so the remainder defines a new vector orthogonal to all the previous vectors. The process considers the vectors one at a time. Numerical experiments prove the method to be very sensitive to computer round off and the modified Gram Schmidt process is used in practical applications.

The modified Gram Schmidt is similar in concept to the traditional version. In the traditional process, at the *i*th step only the vector v_i is altered: it is changed to u_i and is orthogonal to all the previous u_j. In the modified process, at the *i*th step, all the remaining v_i are altered so as to be made orthogonal to the recently computed u vector. The modified Gram Schmidt process unfolds as follows.

Modified Gram Schmidt

Given a spanning set $v_1,\ldots,v_q$, define $v_1^0,\ldots,v_q^0$ as $v_1^0 = v_1,\ldots,v_q^0 = v_q$.

At the *i*th step, for $i = 1,\ldots,q$

define $u_i = v_i^{i-1}$ **(B-3)**

define $v_j^i = v_j^{i-1} - \dfrac{u_i\left(u_i, v_j^{i-1}\right)}{\left(u_i,u_i\right)}$ for $j = i = 1,\ldots,q$. **(B-4)**

This produces the same results as the traditional process in perfect arithmetic. It produces much better results in practical applications.

A couple of points are worth noting. First, zeros in v_i may be filled in during orthogonalization. Second, by definition the inner products $\left(u_i,u_i\right)$ and $\left(u_i,v_j\right)$ use a root sum square type of process

$$\left(u_i,u_i\right) = \left(\sum_k u_{ik}^2\right)^{1/2} \quad \textbf{(B-5)}$$

or

$$\left(u_i,v_j\right) = \left(\sum_k u_{ik}v_{jk}\right)^{1/2} \quad \textbf{(B-6)}$$

If an element of u_i or v_j is significantly greater than the other elements, it can dominate the inner product. This is especially true of the $\left(u_i,u_i\right)$ product.

APPENDIX C

CODE ARCHITECTURE

APPENDIX C

CODE ARCHITECTURE

The computer code is written using the MATLAB desktop computer coding environment. It attempts to use matrix and vector operations whereever possible to increase coding efficiency and speed computations. The structure is such that code is either common to all routines or it may be reused.

The hierarchy is best illustrated. Figure C-1 shows the commonality of the partial differential equation solver and how it is instigated using a top level (*master*) routine. The particular problem geometry and governing equations are called or coded into the left and right hand side residual functions *f_lhs* and *f_rhs*. These two routines may be exactly the same code or they may be totally different. Any preconditioning is performed in these residual statements.

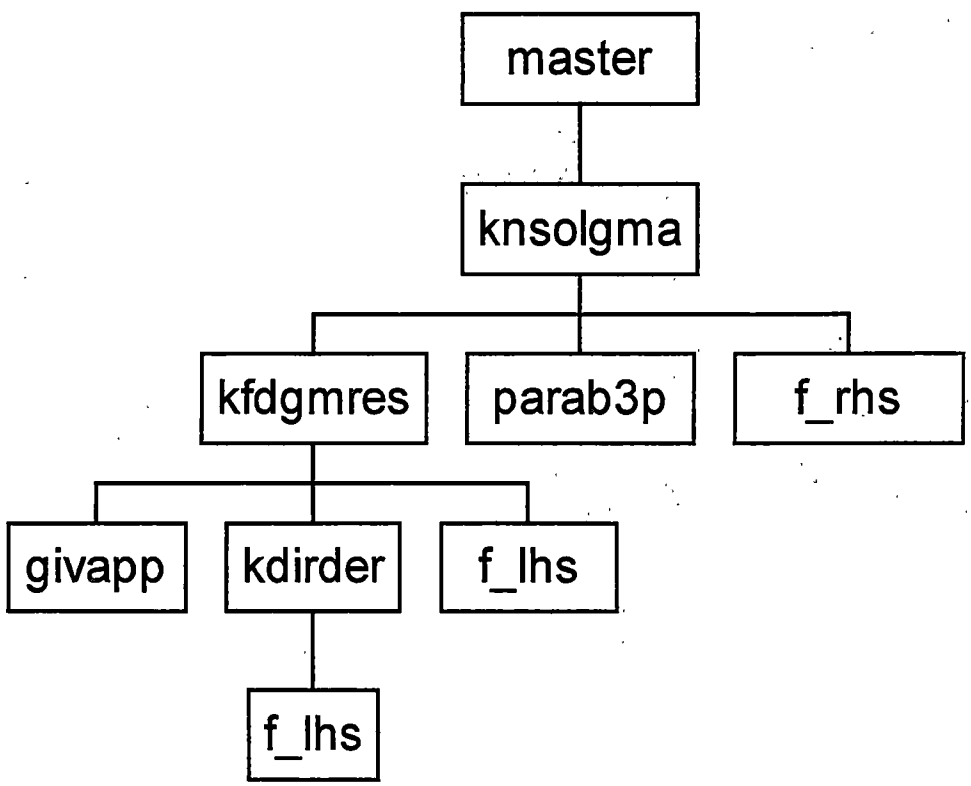


Figure C-1: Code Hierarchy

The particulars of the routines reflected in Figure C-1 are detailed in Appendices D through H. However, the *master* routine typically sets the numerical iteration parameters, creates the workspace, initializes relevant vectors and matrices, and launches the numerical solver. It performs these tasks iteratively in the loosely coupled solution process described in Appendix H.

The left and right hand side functions are also hierarchical. At the top level is an organizing routine (see Figure C-2) that typically calls subroutines to perform the various boundary condition, integration, assembly, and scaling tasks. A Gauss integration routine is used on the right hand side to perform element-by-element integration of the conservation laws coded into another subroutine. At the lowest level are the generic shape functions associated with the finite element method. In this implementation, the right hand side creates and stores element integration matrices to be recalled in left hand side calculations.

The left hand side function (see Figure C-3) takes the current solution increment created by the GMRES solver and adjusts it for boundary conditions. The left hand side residual associated with the current solution increment is formed element-by-element from the recalled integration matrices and assembled into a single vector. The residual elements associated with Dirichlet boundary conditions are zeroed out to remove the degrees of freedom.

The tightly coupled solution process described in Appendix H adds one more level to the process for the right and left hand side functions. Instead of calculating the residual for a single domain using variables passed to it from a higher level routine, it steps through each of the domains separately by opening a domain start file and then calling the single domain residual function shown in Figures C-2 and C-3. This interim level calls upon the exact same coding as the single domain or loosely coupled approach but combines the residuals into a single composite vector. On the left hand side, this equates to the domain coupling being updated with each iteration of the GMRES solver.

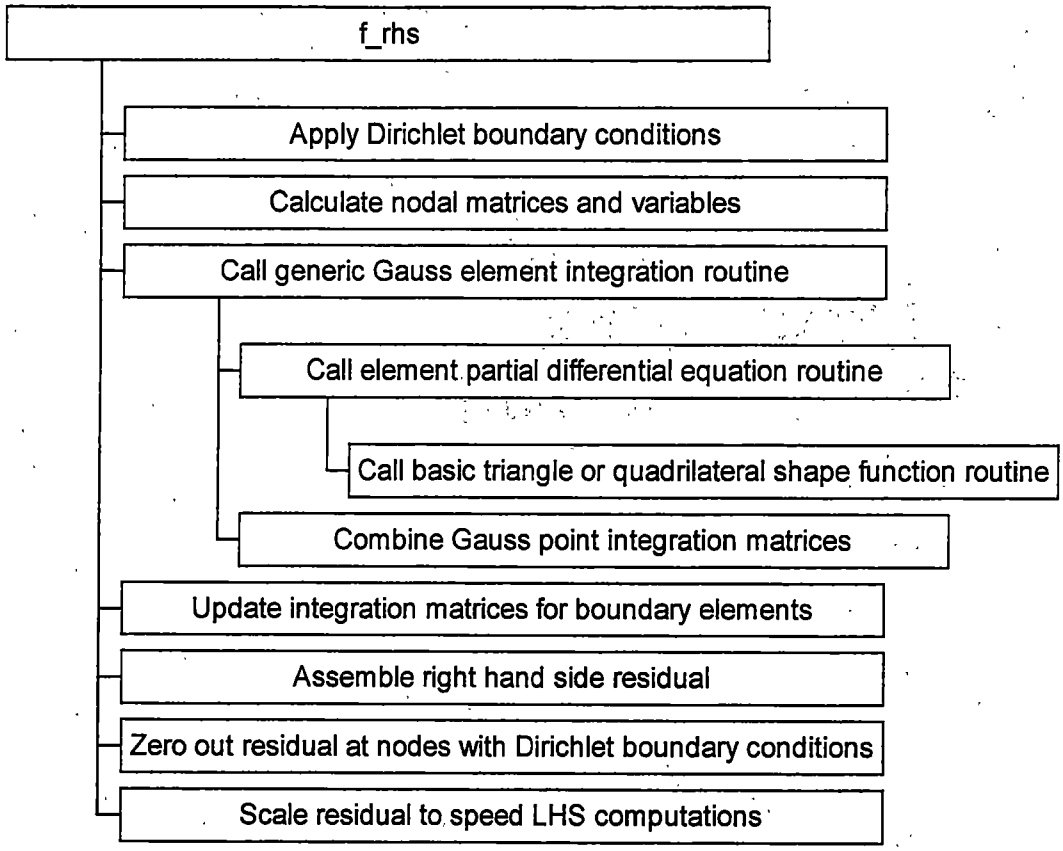


Figure C-2: Right Hand Side Residual Hierarchy

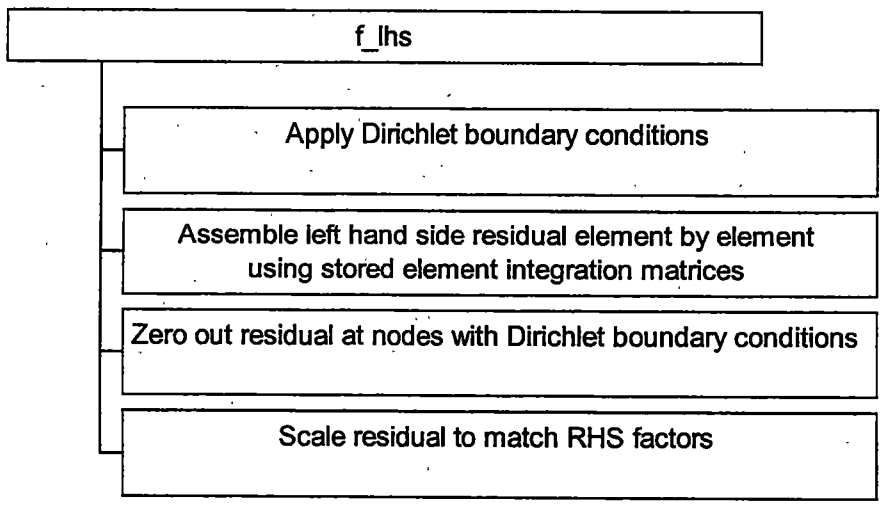


Figure C-3: Left Hand Side Residual Hierarchy

APPENDIX D

NUMERICAL SOLVER ROUTINES

APPENDIX D

NUMERICAL SOLVER ROUTINES

The numerical solver routines are generic to nonsymmetric (or symmetric), nonlinear partial differential equations and implement the algorithms outlined in Chapters 3 and 11. The conservation laws or differential equations are independently described in separate routines and proved a residual statement to the numerical solver for resolution. Nonlinear equations are linearized in a globally convergent Newton statement. The right hand side residual represents the full nonlinear equation set while left hand side residuals used in the GMRES algorithm may be the full nonlinear statement or a linearized version. The left hand side Jacobian matrix-vector product is approximated with a finite difference directional derivative.

The routines are summarized below with the interactions shown in Figures D-1 and D-2.

function knsolgma

- Newton-GMRES globally convergent solver for f(x) = 0
- Uses Eisenstat-Walker forcing term
- Uses parabolic line search via three point interpolation

function kfdgmres - GMRES linear equation solver for use in Newton-GMRES solver

function kdirder - Finite difference directional derivative; approximates [f'(x)]dx

function karnoldi - Arnoldi process for implementing an orthonormal basis for the Krylov subspace using a finite difference method to estimate the action of the matrix/vector product. Used for estimating extreme eigenvalues.

function givapp - Apply a sequence of k Givens rotations, used within GMRES codes

function parab3p - Apply three-point safeguarded parabolic model for a line search.

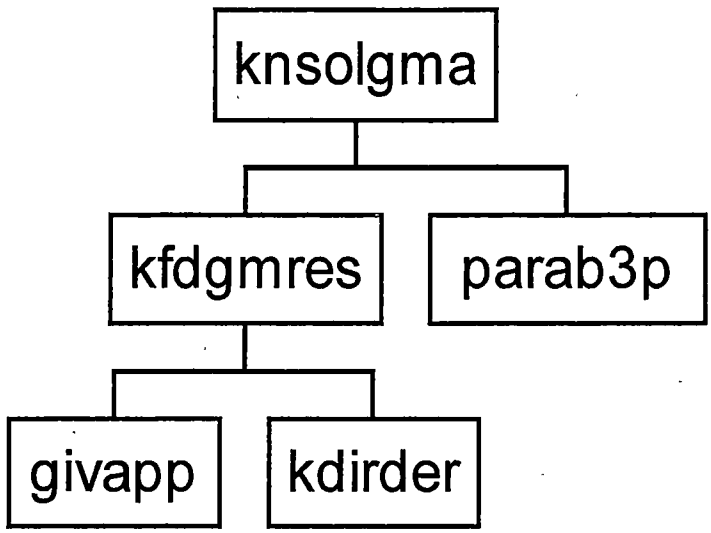


Figure D-1: KNSOLGMA Code Hierarchy

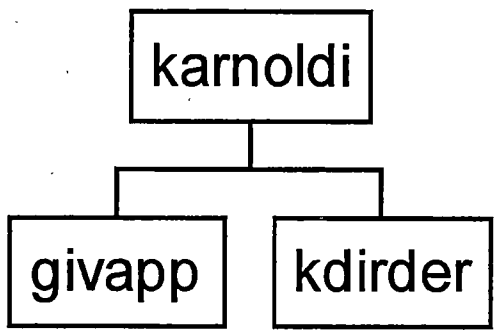


Figure D-2: KARNOLDI Code Hierarchy

KNSOLGMA

```
function [sol, it_hist, ierr, outstat] = knsolgma(f_lhs,f_rhs,
tol,parms,x,xpm1,...
P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,...
P13,P14,P15,P16,P17,P18,P19,P20,...
P21,P22,P23,P24,P25,P26,P27,P28,P29,P30)


% function [sol, it_hist, ierr, outstat] =
knsolgma(f_lhs,f_rhs,tol,parms,x,xpm1,...
%  P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,...
%  P13,P14,P15,P16,P17,P18,P19,P20,...
%  P21,P22,P23,P24,P25,P26,P27,P28,P29,P30)
%
% Newton-GMRES globally convergent solver for f(x) = 0
% Uses Eisenstat-Walker forcing term
% Uses parabolic line search via three point interpolation
%
% Original:  C. T. Kelley, July 1, 1994
% Modified by Kevin Kilpatrick, 19 Oct 96, to include more input
parameters
%                                14 Dec 96, more parameters
%
% INPUTS:
% f_lhs = left hand side function NAME
% f_rhs = right hand side function NAME
% tol = [atol, rtol, dof] relative/absolute error tolerances for
%       the nonlinear iteration and number of degrees of freedom at
each node.
%       Try [1E-6, 1E-6, #], or (dx)^2
% parms = [maxit, lmaxit, etamax, lmeth, restart_limit]
%          maxit = maxmium number of nonlinear iterations, default = 40
%          lmaxit = maximum number of inner iterations, default = 40
%          |etamax| = Maximum error tolerance for residual in inner
%                iteration. The inner iteration terminates when the
%                relative linear residual is smaller than
%                eta*| F(x_c) |. eta is determined by the modified
%                Eisenstat-Walker formula if etamax > 0. If etamax < 0,
%                then eta = |etamax| for the entire iteration.
%                default: etamax=.9, however, setting this smaller
%                may give significant speedups.  Try etamax = .1
%          lmeth = choice of linear iterative method
%                1 (GMRES), 2 GMRES(m),
%                3 (BICGSTAB), 4 (TFQMR)
%                default = 1 (GMRES, no restarts)
%            currently hardwired to 1
%
%          restart_limit = max number of restarts for GMRES if
%                   lmeth = 2
%                  default=20
% x   = initial iterate
%
% OUTPUTS:
```

```
% sol = solution
% it_hist(maxit,3) = scaled L2 norms of nonlinear residuals
%            for the iteration, number function evaluations,
%            and number of steplength reductions
% ierr = error indicator
%      = 0 upon successful termination
%      = 1 if either after maxit iterations the termination criterion
%        is not satsified or the ratio of successive nonlinear
residuals
%        exceeds 1. In this latter case, the iteration is terminated.
%      = 2 failure in the line search. The iteration is terminated if
%        too many steplength reductions are taken.  Currently not an
option.
% outstat= [iteration count, residual norm, inner iteration count,
%           residual norm ratio, number of Armillo step length
reductions]
%
% INTERNAL PARAMETERS:
%       debug = turns on/off iteration statistics display as
%               the iteration progresses
%
%       alpha = 1.d-4, parameter to measure sufficient decrease
%
%       sigma0 = .1, sigma1=.5, safeguarding bounds for the linesearch
%
%       maxarm = 5, maximum number of steplength reductions before
%                   failure is reported
%
% Requires kfdgmres.m and parab3p.m
%

global nfti
nfti=zeros(51,1);

% Set the debug parameter, 1 turns display on, otherwise off
% initialize it_hist, ierr, and set the default iteration parameters
debug=0; %1;                                          %xxxx
gamma=.9;
ierr = 0; maxit=40; lmaxit=40; it_histx=zeros(maxit,3); etamax=.9;

% Set default alpha, sigma0, sigma1, maxarm, and restart_limit
alpha = 1.d-4; sigma0=.1; sigma1=.5; maxarm = 8;
lmeth=1; restart_limit=20;

% Initialize parameters for the iterative methods
maxit=parms(1); lmaxit=parms(2); etamax=parms(3);
gmparms=[abs(etamax), lmaxit];

if length(parms) == 4
  lmeth = parms(4);            % Only GMRES and GMRES(m) supported
elseif length(parms) >= 5
  lmeth = parms(4);            % Only GMRES and GMRES(m) supported
  restart_limit= parms(5);
end
if length(tol)<3
  dofnode = 1;
```

```
else
  dofnode = tol(3);
end
rtol=tol(2); atol=tol(1); n = length(x)*dofnode; fnrm=1; itc=0;

% Set up arguments for function call
args = '';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% MOD HERE
for j = 1:nargin-6
  args = [args,',P',int2str(j)];
end

% Evaluate initial iterate, find the norm of the residual
% and compute the stop tolerance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% MOD HERE
f0=eval([f_rhs,'(x,xpm1',args,')']);
fnrm=norm(f0)/sqrt(n);

% Book keeping
it_histx(itc+1,1)=fnrm; it_histx(itc+1,2)=0; it_histx(itc+1,3)=0;
fnrmo=1;
stop_tol=atol + rtol*fnrm;
outstat(itc+1, :) = [itc fnrm 0 0 0];


%                                MAIN ITERATION LOOP

while(fnrm > stop_tol & itc < maxit)

% Keep track of the ratio (rat = fnrm/frnmo) of successive
% residual norms and the iteration counter (itc)

    rat=fnrm/fnrmo;
    fnrmo=fnrm;
    itc=itc+1;

% Linear iterative method.  Note only GMRES and GMRES(m) are currently
% supported.

    x0 = zeros(size(x));
%%%    x0 = ones(size(x));
%%%   x0 = x-xpm1;
%%%   x0=x;
    kinn = 0;
    if lmeth == 1 | lmeth == 2

% Compute the step using a GMRES routine especially designed
% for this purpose

      [step, errstep, inner_it_count]= ...
           eval(['kfdgmres(f0,f_lhs,gmparms,x0,x,xpm1',args,')']);

%trial overlaxation parameter
%step=step*1E5;
```

```
% Restart at most restart_limit times
      while        inner_it_count == lmaxit & ...
            errstep(inner_it_count) > gmparms(1)*norm(f0) & ...
                                kinn < restart_limit
        kinn = kinn + 1;
        x0 = step;
        [step, errstep, inner_it_count]= ...
             eval(['kfdgmres(f0,f_lhs,gmparms,x0,x,xpm1',args,')']);
      end

      inner_it_count = inner_it_count + kinn*lmaxit;
    end

inner_f_evals = 0;                          % More book keeping

% PLOTTING STUFF - lots of diagnostic looks
global RHSR RHSR_CTR
RHSR_CTR = itc;
figure(2)
ptc=round((((itc-1)/4)-fix((itc-1)/4))*4)+1
%ends=[6,7,10,11,22,23,26,27,38,39,42,43,54,55,58,59];
subplot(4,2,2*ptc), plot(step);
grid on
if ptc == 1
  subplot(4,2,2), title('STEP');
end
hold on;
%plot(ends,zeros(16,1),'r+');
hold off;
subplot(4,2,2*ptc-1), plot(RHSR);
grid on
subplot(4,2,2*ptc-1), ylabel(['ITC ',int2str(itc)])
if ptc == 1
  subplot(4,2,1), title('RHSR');
end
hold on;
%plot(ends,zeros(16,1),'r+');
hold off;
uicontrol('Style','pushbutton','Callback','zoom','Units',...
          'normalized','Position',[.45,0,.1,.05],'String','ZOOM');
drawnow;


%   The PARABOLIC LINE SEARCH starts here.

    xpm1 = x;
    xold=x;
    lambda=1; lamm=1; lamc=lambda; iarm=0;
    xt = x + lambda*step;
    ft=eval([f_rhs,'(xt,xpm1',args,')']);

    nft=norm(ft); nf0=norm(f0); ff0=nf0*nf0; ffc=nft*nft; ffm=nft*nft;
    nfti(iarm+1)=nft;

%   ARMIJO loop, iarm
```

```
    while nft >= (1 - alpha*lambda) * nf0;

%   apply the three point parabolic model

        if iarm == 0
            lambda=sigma1*lambda;
        else
            lambda=parab3p(lamc, lamm, ff0, ffc, ffm);
        end

% update x; keep the books on lambda

        xt=x+lambda*step;
        lamm=lamc;
        lamc=lambda;

% keep the books on the function norms
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% MOD HERE
        ft=eval([f_rhs,'(xt,xpm1',args,')']);

% flag for armijo step length reductions
iarm

        nft=norm(ft);
        nfti(iarm+1) = nft;
        ffm=ffc;
        ffc=nft*nft;
        iarm=iarm+1;
        if iarm > maxarm
            disp(' Armijo failure, too many reductions ');
            ierr=2;
            disp(outstat)
            it_hist=it_histx(1:itc+1,:);
            sol=xold;
            return;
        end
    end                                     % end of Armijo 'while' loop
    x=xt;
    f0=ft;

%   end of line search

    fnrm=norm(f0)/sqrt(n);
    it_histx(itc+1,1)=fnrm;

%   How many function evaluations did this iteration require?

    it_histx(itc+1,2)=it_histx(itc,2)+inner_f_evals+iarm+1;
    if(itc == 1) it_histx(itc+1,2) = it_histx(itc+1,2)+1; end;
    it_histx(itc+1,3)=iarm;

    rat=fnrm/fnrmo;

%   adjust eta as per Eisenstat-Walker
    if etamax > 0
```

```
        etaold=gmparms(1);
        etanew=gamma*rat*rat;
        if gamma*etaold*etaold > .1
            etanew=max(etanew,gamma*etaold*etaold);
        end
        gmparms(1)=min([etanew,etamax]);
        gmparms(1)=max(gmparms(1),.5*stop_tol/fnrm);        %xxxx
    end

    outstat(itc+1, :) = [itc fnrm inner_it_count rat iarm];

    if debug==1
       disp(outstat(itc+1,:));                              %xxxx
    end
% end while
end

sol=x;

it_hist=it_histx(1:itc+1,:);
if debug==1
    disp(outstat);                                          %xxxx
    it_hist=it_histx(1:itc+1,:);
end

% on failure, set the error flag

if fnrm > stop_tol
    ierr = 1;
end
```

KFDGMRES

```
function [dx, error, total_iters] =
kfdgmres(f0_rhs,f_lhs,params,w,xc,xold,...
P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,...
P13,P14,P15,P16,P17,P18,P19,P20,...
P21,P22,P23,P24,P25,P26,P27,P28,P29,P30)
% function [dx, error, total_iters] =
kfdgmres(f0_rhs,f_lhs,params,w,xc,xold,...
% P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,...
% P13,P14,P15,P16,P17,P18,P19,P20,...
% P21,P22,P23,P24,P25,P26,P27,P28,P29,P30)
%
% GMRES linear equation solver for use in Newton-GMRES solver
%
% Original: C. T. Kelley, July 24, 1994
% Modified: Kevin Kilpatrick, 19 Oct 96
%                             14 Dec 96, added more parameters
%
% INPUT:
%  f0_rhs = right hand side function evaluated at the current point
```

```
%  f_lhs = nonlinear function NAME for the left hand side function
%      the format for f is  function fx = f(x,P1..Pn)
%      Note that for Newton-GMRES we incorporate any
%         preconditioning into the function routine.
% params = two dimensional vector to control iteration
%          params(1) = relative residual reduction factor
%          params(2) = max number of iterations
%          params(3) (Optional) = reorthogonalization method
%                 1 -- Brown/Hindmarsh condition (default)
%                 2 -- Never reorthogonalize (not recommended)
%                 3 -- Always reorthogonalize (not cheap!)
% w = initial iterate. w=0 is a valid default. This
%     is a reasonable choice unless restarted GMRES
%     will be used as the linear solver.  Previous value of dx may
%     be useful for time stepping problems.
% xc = current state vector
% P1..P30 = input parameters for function, f_lhs
%
% OUTPUT:
% dx     = solution
% error = vector of residual norms for the history of the iteration
% total_iters = number of iterations
%
% Requires givapp.m, dirder.m

% initialization
errtol=params(1);
kmax=params(2);
reorth=1;
if length(params) == 3
    reorth=params(3);
end

% right side of linear equation for the step is -f0_rhs

b = - f0_rhs;
n = length(b);

% set up arguments for function call
args = '';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% MOD HERE
for j = 1:nargin-6
  args = [args,',P',int2str(j)];
end

% Use input vector as initial iterate for Newton step, calculate
% the LHS function at the current value, and perform the
% first iteration

dx = w;
f0_lhs = zeros(size(xc));
r = zeros(size(xc));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% MOD HERE
f0_lhs = eval([f_lhs,'(xc,xold',args,')']);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% MOD HERE
r = - eval(['kdirder(xc, xold, dx, f_lhs, f0_lhs',args,')']) - f0_rhs;

% initialize memory and iteration parameters
h=zeros(kmax);
v=zeros(n,kmax);
c=zeros(kmax+1,1);
s=zeros(kmax+1,1);

rho=norm(r);
g=rho*eye(kmax+1,1);
errtol=errtol*norm(b);
error=[];

% test for termination on entry

error=[error,rho];
total_iters=0;
if(rho < errtol)
 disp(' early termination ')
  return
end

% initialize the rest of the iteration parameters

v(:,1)=r/rho;
beta=rho;
k=0;

% plotting data initiation
rhos=zeros(kmax,2);
rhos(k+1,:)=[k,rho];

% begin                              GMRES ITERATION

while((rho > errtol) & (k < kmax))
    k=k+1;

% call directional derivative function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% MOD HERE
    v(:,k+1) = eval(['kdirder(xc,xold,v(:,k),f_lhs,f0_lhs',args,')']);

% v(:,k) was normalized before going into the directional derivative.
Internally,
% the algorithm will scale it to match norm(xc).  However,
norm(v(:,k))=1 is not
% necessarily what I want.  In fact, it scales the entire result.

    normav=norm(v(:,k+1));

% Make the current vector, V, orthogonal to previous vectors
% using modified Gram-Schmidt
```

```
    for j=1:k
        h(j,k)=v(:,k+1)'*v(:,j);
        v(:,k+1)=v(:,k+1)-h(j,k)*v(:,j);
    end

    h(k+1,k)=norm(v(:,k+1));
    normav2=h(k+1,k);

% reorthogonalize?

if  (reorth == 1 & normav + .001*normav2 == normav) | reorth ==  3
    for j=1:k
      hr=v(:,k+1)'*v(:,j);
      h(j,k)=h(j,k)+hr;
      v(:,k+1)=v(:,k+1)-hr*v(:,j);
    end
    h(k+1,k)=norm(v(:,k+1));
end

%   watch out for happy breakdown

    if(h(k+1,k) ~= 0)
      v(:,k+1)=v(:,k+1)/h(k+1,k);
    end

%   Form and store the information for the new Givens rotation

    if k > 1
      h(1:k,k)=givapp(c(1:k-1),s(1:k-1),h(1:k,k),k-1);
    end

%   Don't divide by zero if solution has been found

    nu=norm(h(k:k+1,k));
    if nu~=0
      c(k)=h(k,k)/nu;
      s(k)=-h(k+1,k)/nu;
      h(k,k)=c(k)*h(k,k)-s(k)*h(k+1,k);
      h(k+1,k)=0;
      g(k:k+1)=givapp(c(k),s(k),g(k:k+1),1);
    end

% Update the residual norm

    rho=abs(g(k+1));
    error=[error,rho];

    rhos(k+1,:)=[k,rho];      % plotting data

% end while, end GMRES ITERATION
end

% At this point either k > kmax or rho < errtol.
% It's time to compute dx and leave.
```

```
y=h(1:k,1:k)\g(1:k);
total_iters=k;
dx = dx + v(1:n,1:k)*y;


figure(1)
hold on
plot(rhos(1:k+1,1),rhos(1:k+1,2),'*')
drawnow
```

KDIRDER

```
function z =
kdirder(x,xold,w,f,f0,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,...
P13,P14,P15,P16,P17,P18,P19,P20,...
P21,P22,P23,P24,P25,P26,P27,P28,P29,P30)
%
%function z =
kdirder(x,xold,w,f,f0,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,...
% P13,P14,P15,P16,P17,P18,P19,P20,...
% P21,P22,P23,P24,P25,P26,P27,P28,P29,P30)
%
% Finite difference directional derivative
% Approximate f'(x) w
%
% Original: C. T. Kelley, November 25, 1993
% Modified: Kevin Kilpatrick, 19 Oct 96, to include more parameters
%           for use in function evaluation
%                              14 Dec 96, added still more paremeters
%
% INPUTS:
% x = current state vector about which to form evaluation
% w = direction in which to form derivative
% f = function name
% f0 = f(x), in nonlinear iterations
%      f(x) has usually been computed before the call to dirder
%
% OUTPUTS:
% z = vector representing f'(x) dx

n=length(x);

% set up arguments for function calls
args = '';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% MOD HERE
for j = 1:nargin-5
  args=[args,',P',int2str(j)];
end

% Hardwired difference increment
epsnew=1.d-7;
```

```
% scale the step
if norm(w) == 0
  z=zeros(n,1);
  return
end
epsnew = epsnew/norm(w);
if norm(x) > 0
    epsnew=epsnew*norm(x);
end

% del and f1 could share the same space if storage
% is more important than clarity

del=x+epsnew*w;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% MOD HERE
f1=eval([f,'(del,xold',args,')']);
z = (f1 - f0)/epsnew;
```

KARNOLDI

```
function [maxee,minee,minabs,error,total_iters] =
karnoldi(f_name,params,w,xc,xold,...
P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,...
P13,P14,P15,P16,P17,P18,P19,P20,...
P21,P22,P23,P24,P25,P26,P27,P28,P29,P30)
% function [maxee,minee,minabs,error,total_iters] =
% karnoldi(f_name,params,w,xc,xold,...
% P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,...
% P13,P14,P15,P16,P17,P18,P19,P20,...
% P21,P22,P23,P24,P25,P26,P27,P28,P29,P30)
%
% Arnoldi process for implementing an orthonormal basis for the Krylov
subspace
% using a finite difference method to estimate the action of the
matrix/vector
% product.  Used for estimating the extreme eigenvalues.
%
%
% Created: Kevin Kilpatrick, 1 Jan 99
%  - based on original coding for FDGMRES by C. T. Kelley, July 24,
1994
%
%
% INPUT:
%  f_name = nonlinear function NAME for the matrix vector product
function
%      the format for f is  function fx = f(x,P1..Pn)
%      Note that any preconditioning is incorporated into the function
routine.
% params = three term vector to control iteration
```

```
%          params(1) = relative residual reduction factor
%          params(2) = max number of iterations
%          params(3) (Optional) = reorthogonalization method
%                 1 -- Brown/Hindmarsh condition (default)
%                 2 -- Never reorthogonalize (not recommended)
%                 3 -- Always reorthogonalize (not cheap!)
% w = initial iterate. w={1}/norm(w) is a valid default.
% xc = current state vector
% xold = another state vector,
% P1..P30 = input parameters for function, f_name
%
% OUTPUT:
% maxee = maximum eigenvalue estimate
% minee = minimum eigenvalue estimate
% error = vector of residual norms for the history of the iteration
% total_iters = number of iterations
%
% Requires givapp.m, dirder.m

% initialization
errtol=params(1);
kmax=params(2);
reorth=1;
if length(params) == 3
    reorth=params(3);
end

% set up arguments for function call
args = '';

for j = 1:nargin-5
  args = [args,',P',int2str(j)];
end

% Use input vector as initial vector around which to build
% Krylov subspace

dx = w;
f0 = zeros(size(xc));
r = zeros(size(xc));

% perform initial function evaluation...f_name(xc)
% will dither xc later and evaluate f_name(xc+eps*dx) and form
% directional derivative to evaluate [df_name/dxc]*dx as the
% matrix*vector product

f0 = eval([f_name,'(xc, xold',args,')']);

b = f0;
n = length(b);

% form directional derivative to simulate matrix vector product
r = eval(['kdirder(xc, xold, dx, f_name, f0',args,')']);

% initialize memory and iteration parameters
h=zeros(kmax);
```

```
v=zeros(n,kmax);
c=zeros(kmax+1,1);
s=zeros(kmax+1,1);

% initialize iteration values
rho=norm(r);
g=rho*eye(kmax+1,1);
errtol=errtol*norm(b);
error=[];

% test for termination on entry
error=[error,rho];
total_iters=0;
if(rho < errtol)
  disp(' early termination ')
  return
end

% initialize the rest of the iteration parameters
v(:,1)=r/rho;
beta=rho;
k=0;

% begin                              ARNOLDI ITERATION

while((rho > errtol) & (k < kmax))
    k=k+1;

% call directional derivative function
    v(:,k+1) = eval(['kdirder(xc,xold,v(:,k),f_name,f0',args,')']);

    normav=norm(v(:,k+1));

% Make the current vector, V, orthogonal to previous vectors
% using modified Gram-Schmidt
    for j=1:k
        h(j,k)=v(:,k+1)'*v(:,j);
        v(:,k+1)=v(:,k+1)-h(j,k)*v(:,j);
    end

    h(k+1,k)=norm(v(:,k+1));
    normav2=h(k+1,k);

% reorthogonalize?
if  (reorth == 1 & normav + .001*normav2 == normav) | reorth ==  3
    for j=1:k
      hr=v(:,k+1)'*v(:,j);
      h(j,k)=h(j,k)+hr;
      v(:,k+1)=v(:,k+1)-hr*v(:,j);
    end
    h(k+1,k)=norm(v(:,k+1));
end

%   watch out for happy breakdown
    if(h(k+1,k) ~= 0)
      v(:,k+1)=v(:,k+1)/h(k+1,k);
```

```
    end

%   Form and store the information for the new Givens rotation
    if k > 1
      h(1:k,k)=givapp(c(1:k-1),s(1:k-1),h(1:k,k),k-1);
    end

%   Don't divide by zero if solution has been found
    nu=norm(h(k:k+1,k));
    if nu~=0
      c(k)=h(k,k)/nu;
      s(k)=-h(k+1,k)/nu;
      h(k,k)=c(k)*h(k,k)-s(k)*h(k+1,k);
      h(k+1,k)=0;
      g(k:k+1)=givapp(c(k),s(k),g(k:k+1),1);
    end

% Update the residual norm

    rho=abs(g(k+1));
    error=[error,rho];

% end while, end ARNOLDI ITERATION
end

% At this point either k > kmax or rho < errtol.
% It's time to compute the eigenvalues of h

total_iters=k;
for i=1:total_iters
  TE=sort(eig(h(1:i,1:i)))';
  minee(i)=min(TE);
  maxee(i)=max(TE);
  minabs(i)=min(abs(TE));
end
%min(minee)
%max(maxee)
%min(minabs)
```

GIVAPP

```
function vrot=givapp(c,s,vin,k)
%  Apply a sequence of k Givens rotations, used within gmres codes
%
%  C. T. Kelley, July 10, 1994
%
% This code comes with no guarantee or warranty of any kind.
%
%  function vrot=givapp(c, s, vin, k)
%
vrot=vin;
for i=1:k
```

```
    w1=c(i)*vrot(i)-s(i)*vrot(i+1);
    w2=s(i)*vrot(i)+c(i)*vrot(i+1);
    vrot(i:i+1)=[w1,w2];
end
```

PARAB3P

```
function lambdap = parab3p(lambdac, lambdam, ff0, ffc, ffm)
% Apply three-point safeguarded parabolic model for a line search.
%
% C. T. Kelley, June 29, 1994
%
% This code comes with no guarantee or warranty of any kind.
%
% function lambdap = parab3p(lambdac, lambdam, ff0, ffc, ffm)
%
% input:
%       lambdac = current steplength
%       lambdam = previous steplength
%       ff0 = value of \| F(x_c) \|^2
%       ffc = value of \| F(x_c + \lambdac d) \|^2
%       ffm = value of \| F(x_c + \lambdam d) \|^2
%
% output:
%       lambdap = new value of lambda given parabolic model
%
% internal parameters:
%       sigma0 = .1, sigma1=.5, safeguarding bounds for the linesearch
%

%
% set internal parameters
%
sigma0=.1; sigma1=.5;
%
% compute coefficients of interpolation polynomial
%
% p(lambda) = ff0 + (c1 lambda + c2 lambda^2)/d1
%
% d1 = (lambdac - lambdam)*lambdac*lambdam < 0
%      so if c2 > 0 we have negative curvature and default to
%      lambdap = sigam1 * lambda
%
c2 = lambdam*(ffc-ff0)-lambdac*(ffm-ff0);
if c2 >= 0
    lambdap = sigma1*lambdac; return
end
c1=lambdac*lambdac*(ffm-ff0)-lambdam*lambdam*(ffc-ff0);
lambdap=-c1*.5/c2;
if (lambdap < sigma0*lambdac) lambdap=sigma0*lambdac; end
if (lambdap > sigma1*lambdac) lambdap=sigma1*lambdac; end
```

APPENDIX E

FLUID FLOW ROUTINES

APPENDIX E

FLUID FLOW ROUTINES

The fluid flow routines implement the SUPG/TWS algorithms developed in Chapters 4-6 and 12-13. The routines follow the hierarchical structure described in Appendix C. Figures E-1 through E-3 are specific implementations of the methodology illustrated in Figures C-1 through C-3. The coding is original and attempts to take advantage of MATLAB's intrinsic functions.

The blocks shaded in Figure E-1 are specific fluid flow routines. The blocks not shaded are generic and did not change. Figures E-2 and E-3 expand the hierarchical structure for the right and left hand side residual functions. All the functions in the diagram were accomplished. Shaded blocks show where a specific subroutine was called while the tasks in the unshaded blocks were performed in the parent block. The routines are summarized and listed in full detail.

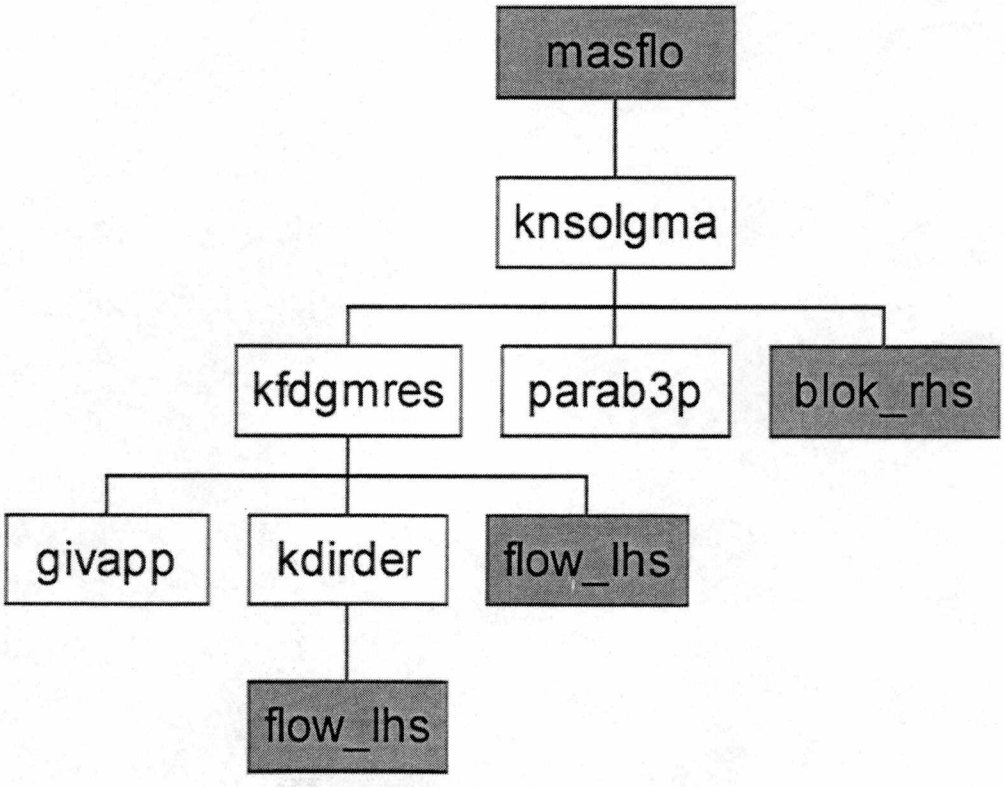


Figure E-1: Fluid Flow Master Hierarchy

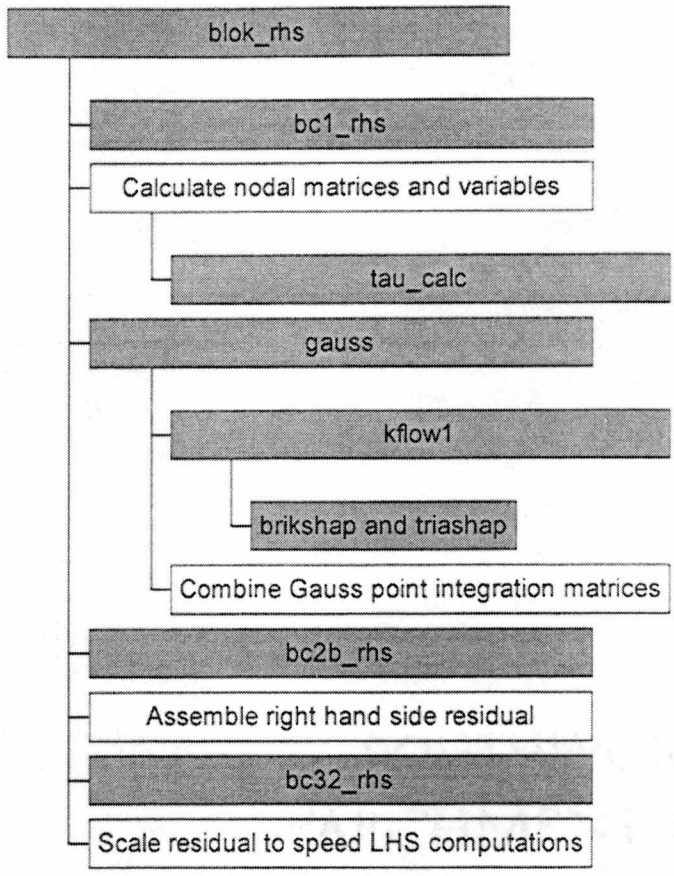


Figure E-2: Right Hand Side Residual of Fluid Flow

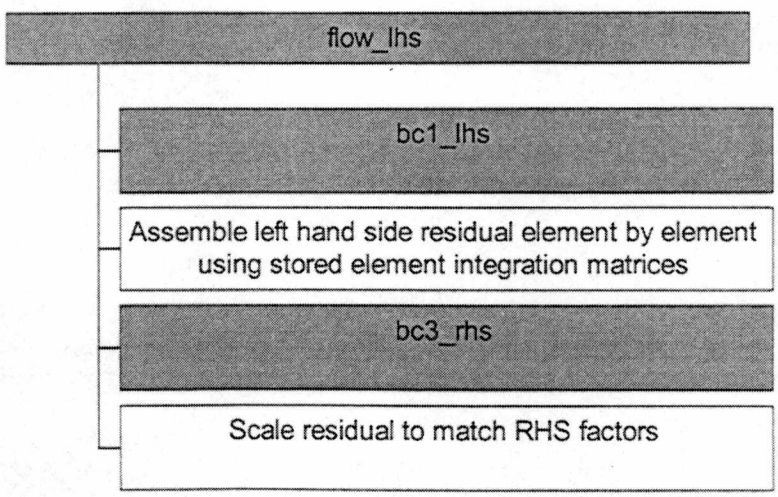


Figure E-3: Left Hand Side Residual of Fluid Flow

script masflo - Master routine for single domain fluid flow calculations

function blok_rhs
- Driver routine for RHS fluid flow calculations
- Forms a variety of nodal variables and calculates nodal matrices for later use in the element by element calculations
- Nodal calculations are done here to avoid the duplication inherent to element by element calculations that share a node
- Evaluates the flowfield by calling a gauss quadrature routine to integrate the variables over the elements using an SUPG formulation
- Stores the integrated element matrices as global variables for rapid use on the LHS calculations
- Assembles the element contributions to give the RHS residual

function bc1_rhs
- Forms nodal tangent vectors for all boundary nodes according to right hand rule
- Determines inflow/outflow conditions based on supersonic/subsonic normal flow and directions of the characteristics. Then sets appropriate Dirichlet boundary conditions for primitive variables
- Updates the current vector of primitive nodal variables
- BC2_RHS adds line integration contributions along the boundaries for subsonic inflow and outflow boundary conditions
- Residuals at Dirichlet boundary nodes are manipulated in BC3_RHS after assembly

function bc2b_rhs
- Integrate along boundary edges for subsonic inflow/outflow boundaries and to adjust zero flux terms for pressure along solid boundaries
- Incorporate contribution into the various integration matrices for RHS and LHS residual calculations

function bc32_rhs
- Modifies fully assembled residuals at Dirichlet boundary nodes by setting appropriate components to zero. Used in both LHS and RHS

function tau_calc - Calculates the SUPG parameter TAU

function gauss - Performs gauss quadrature integration over a regular interval

function kflow1
- Evaluates the flowfield at a point within an element using a pseudo-time preconditioned TWS/SUPG finite element method. Intended for use in gauss quadrature
- Uses Green-Gauss theorem to transform flux gradient term to volume plus surface integrals. Surface integrals are not evaluated due to known cancellation at assembly
- Complete algorithm for elements not touching a boundary of any type
- Impermeable wall boundaries must be corrected for pressure (BC2B_RHS)
- Inflow/outflow boundaries must be corrected for surface flux (BC2B_RHS)

function brikshap - Evaluates various aspects of the linear quadralateral shape function

function triashap - Evaluates various aspects of the linear triangle shape function using natural (area) coordinates

function flow_lhs - Driver routine for LHS fluid flow calculations

function bc1_lhs - Enforces Dirichlet and impermeable wall boundary conditions on LHS residual using RHS definitions of inflow/outflow and tangent vectors

MASFLO

```
% Master script program for fluid flow
%
% created 5 Mar 97
%
% initialize variables
% XN, YN = original X and Y location
% ELMAT  = nodes assigned to elements


% MP     = fluid properties
% BCTYPE = vector of nodal definition of boundary condition type
%          type 0 = not on a boundary
%          type 1 = inflow/outflow
%          type 2 = impermeable wall
% BC1    =  matrix of type 1 boundary conditions
%           [node number, P, U, V, T]
% DOFNODE = degrees of freedom per node (currently 4)
% NN =      order of gauss integration (1,2, or 3)
%
% global variables
% MFLOW =   flat file containing summation of element integration
matrices for LHS
% MOUT  =   matrix of nodal RH rule tangent vectors and boundary type
information
%           [node number, dx, dy, type (1 - 54), index into BC1, DX,
DY]
%           BC conditions 1=supersonic inflow,  2=subsonic inflow
%                         3=supersonic outflow, 4=subsonic outflow
%                         5=impermeable wall,
%                         51=wall+super in,     52=wall+sub in
%                         53=wall+super out,    54=wall+sub out
%           [dx,dy] = wall tangent and in/out tangent
%           [DX,DY] = in/out tangent if [dx,dy] has wall tangent for a
corner,
%                     otherwise original direction of velocity vector
% GFLOW =   flat file containing summation of element integration
matrices for LHS
%           grid motion in pseudo time

global MFLOW MOUT GFLOW

eval(['load ',name]);
disp('any changes')
keyboard

% set some Newton (outer) and GMRES (inner) iteration parameters
% [absolute error tolerance, relative error tolerance, degrees of
freedom/node]
TOL=[1E-6,1E-6,5];
%TOL=[1E-9,1E-9,4];
LXN = length(XN);
NBC1 = length(find(BCTYPE == 1));
```

```
NBC2 = length(find(BCTYPE == 2));

% Max number of GMRES iterations min(hard limit,DOF, rank) where
% type 2 BC loose 1 DOF and in/out BC loose 2 DOF on average
LIN_ITS = min([40,DOFNODE*LXN,DOFNODE*LXN-NBC2-2*NBC1]);
%LIN_ITS = 88;

% [max Newton steps, max linear iterations, etamax, linear soln method,
max restarts]
% 20 Newton steps
% 40 GMRES iterations per try
% .9 etamax
% 2 = Restarted GMRES(LIN_ITS)
% 1 = max number of GMRES restarts -> linear its max/Newton step
PARMS=[20,LIN_ITS,.5,2,5];
%PARMS=[50,LIN_ITS,.5,2,5];

T_TOL=[1E-6,1E-6];      % [absolute time error tolerance, relative
error tolerance]
T_STEP=1;               % max time steps

DEL_T = 0;              % physical time step size, 0 = steady state
DEL_TAU = 1             % pseudo time step size
%DEL_TAU=0
%disp(['no pseudo time'])

flops(0);
tic

%  Initialize matrices for element by element storage
[NEL,LARGEST]=size(ELMAT);
MFLOW=zeros(NEL,(4*LARGEST)^2);     % LHS element integration matrix
LBC = length(find(BCTYPE));         % number of boundary nodes
MOUT = zeros(LBC,7);                % [node number,dx,dy,type,index
into BC1,DX,DY]
MOUT(:,1)=find(BCTYPE);             % load boundary node numbers
GFLOW = zeros(NEL,2*DOFNODE*LARGEST^2);   % Fluid flow correction for
grid motion

% Create space space for time solution vectors
QP  = zeros(LXN*DOFNODE,1);         % current solution initial guess
QP1 = zeros(LXN*DOFNODE,1);         % most recent solution vector
QP2  = zeros(LXN*DOFNODE,1);        % next most recent solution vector
SOLN = zeros(LXN*DOFNODE,1);        % current solution vector
T_HISTORY=zeros(4*LXN,T_STEP+1);

FNRM=1;
STOP_TOL=T_TOL(1) + T_TOL(2)*FNRM;
ITC = 0;

QP1 = reshape(QINITIAL,LXN*DOFNODE,1);    % load initial conditions
SOLN = reshape(QINITIAL,LXN*DOFNODE,1);

while(FNRM > STOP_TOL & ITC < T_STEP)
  ITC=ITC+1;
```

```
  QP2 = QP1;
  QP1 = SOLN;
  QP = SOLN;

% Set up figure windows
%  figure(1)
%  clg
%  set(1,'Position',[2 35 447 380]);
%  figure(2)
%  clg
%  set(2,'Position',[457 283 560 420]);
%  figure(3)
%  clg
%  set(3,'Position',[2 455 400 250]);

  figure(1)
  clg
  set(1,'Position',[2 30 447 340]);
  figure(2)
  clg
  set(2,'Position',[456 414 371 310]);
  figure(3)
  clg
  set(3,'Position',[2 415 447 308]);
  figure(4)
  clg
  set(4,'Position',[456 30 370 339]);

  [SOLN, HISTORY, IERROR, STATS] =
knsolgma('FLOW_LHS','BLOK_RHS',TOL,PARMS,QP,QP,...

XN,YN,UN,VN,QP1,QP2,DEL_T,DEL_TAU,MP,ELMAT,BCTYPE,BC1,MOUT_PTR,DOFNODE,
NN);

disp([STATS(:,1),zeros(size(STATS(:,2))),STATS(:,3:5)])
format long
disp(STATS(:,2))
format short

% update SOLN for inflow/outflow BC values
  SOLN = reshape(SOLN,LXN,DOFNODE);
  SOLN = bc1_lhs(SOLN,BC1);
  SOLN = reshape(SOLN,LXN*DOFNODE,1);

  T_HISTORY(:,ITC+1)= SOLN;

%  FNRM = norm(SOLN-QP1)/sqrt(length(SOLN))
  FNRM = norm(SOLN)/sqrt(length(SOLN))

end   % time stepping

toc
flops
figure(1)
clg
semilogy(HISTORY(:,1))
```

BLOK_RHS

```
function R =
blok_rhs(QP,QPM1,XN,YN,UN,VN,QP1,QP2,DEL_T,DEL_TAU,MP,ELMAT,BCTYPE,BC1,
MOUT_PTR,DOFNODE,NN)
% function R =
blok_rhs(QP,QPM1,XN,YN,UN,VN,QP1,QP2,DEL_T,DEL_TAU,MP,ELMAT,BCTYPE,BC1,
MOUT_PTR,DOFNODE,NN)
%
% 17 Feb 97, original
%  4 Mar 97, matrix and flux calculations moved inside element loop
% 27 Mar 97, completed characteristic boundary conditions
%  1 Sep 97, updated to new algorithm and boundary conditions
%
% Driver routine for RHS fluid flow calculations.
%
% Forms a variety of nodal variables and calculates nodal matrices for
% later use in the element by element calculations. The nodal
calculations
% are done here to avoid the duplication inherent to element by element
% calculations that share a node.
%
% Evaluates the flowfield by calling a gauss quadrature routine to
integrate
% the variables over the elements using an SUPG formulation.
%
% Stores the integrated element matrices as global variables for rapid
use on
% the LHS calculations
%
% Assembles the element contributions to give the RHS residual
%
% QP  =     vector of primitive nodal variables [P; U; V; T]
% QPM1 =    QP at last (p-1) iteration
% XN =      vector of current X locations of nodes
% YN =      vector of current Y locations of nodes
% UN =      vector of X velocity of nodes
% VN =      vector of Y velocity of nodes
% QP1 =     vector of primitive nodal variables [P; U; V; T]
% QP2 =     vector of primitive nodal variables [P; U; V; T]
% DEL_T =   physical time step size
% DEL_TAU = pseudo time step size
% MP =      thermodynamic properties of fluid, MP = [R, CP, GAMMA]
% ELMAT =   matrix listing the nodes assigned to each element
% BCTYPE =  vector of nodal definition of boundary condition type
%           not necessarily the most efficient implementation but it
forces
%           a single definition of boundary state for nodes
%           type 0 = not on a boundary
%           type 1 = inflow/outflow
%           type 5 = impermeable wall
%           type 6 = combination wall and inflow/outflow (i.e. a corner
node)
%           type 21= definitely inflow
```

```
%           type 22= definitely outflow
%           type 26= combination wall and definitely inflow
%           type 27= combination wall and definitely outflow
% BC1     =  matrix of type 1, 6, 21, 22, 26 & 27 boundary conditions
%           [node number, P, U, V, T]
% MOUT_PTR =nodal vector giving index for boundary nodes in MOUT matrix
% DOFNODE = degrees of freedom per node (currently 4)
% NN =      order of gauss integration (1,2, or 3)
%
% global variables
% MFLOW =   flat file containing summation of element integration
matrices for LHS
% MOUT  =   matrix of nodal RH rule tangent vectors and boundary type
information
%           [node number, dx, dy, type (1 - 54), index into BC1, DX,
DY]
%           BC conditions 1=supersonic inflow,  2=subsonic inflow
%                         3=supersonic outflow, 4=subsonic outflow
%                         5=impermeable wall,
%                         51=wall+super in,     52=wall+sub in
%                         53=wall+super out,    54=wall+sub out
%           [dx,dy] = wall tangent and in/out tangent
%           [DX,DY] = in/out tangent if [dx,dy] has wall tangent for a
corner,
%                     otherwise original direction of velocity vector
% GFLOW =   flat file containing summation of element integration
matrices for LHS
%           grid motion in pseudo time

global MFLOW MOUT GFLOW

% initialize
NNODES = length(XN);          % number of nodes
[NEL,VERTICES] = size(ELMAT); % VERTICES is the max number of nodes in
any element

% reformat into a matrix shape with columns of primitive variables
QP = reshape(QP,NNODES,DOFNODE);
QPM1 = reshape(QPM1,NNODES,DOFNODE);
QP1 = reshape(QP1,NNODES,DOFNODE);
QP2 = reshape(QP2,NNODES,DOFNODE);

% Safeguard against negative pressure or temperature
QP(:,1)=max(1,abs(QP(:,1)));
QP(:,4)=max(1,abs(QP(:,4)));

% Calculate some useful constants
CONST1 = MP(3)*((MP(3)-1)*(MP(2)+MP(1))^2+MP(1)^2);   % gamma((gamma-
1)(Cp+R)^2+R^2)
CONST2 = -MP(3)/MP(2);                                % -gamma/Cp
CONST3 = MP(3)/(MP(2)*MP(1));                         % gamma/(Cp*R)
CONST4 = MP(3)/MP(1);                                 % gamma/R
CONST5 = (MP(3)*MP(1))/(4*MP(2)^2);                   %
(gamma*R)/(4*Cp^2)
CONST6 = MP(3)*MP(1);                                 % gamma*R
CONST7 = (MP(3)-1)/MP(3);                             % (gamma-1)/gamma
```

```
CONST8 = MP(3)-1;                                          % gamma*R/Cp

LX = ((max(XN) - min(XN))/pi);                             % X reference
dimension
LY = ((max(YN) - min(YN))/pi);                             % Y reference
dimension
LREF = norm([LX,LY]);                                      % reference
length

R = zeros(NNODES*DOFNODE,1);

% VERTICES = determines the shape of the region (1 = rectangle, 2 =
triangle)
if VERTICES == 3                % triangle
  FLAT   = zeros(7,12^2);
  QP_MAT = zeros(12);
  QC_MAT = zeros(12);
  F_MAT  = zeros(12);
  G_MAT  = zeros(12);

elseif VERTICES == 4            % rectangle
  FLAT   = zeros(7,16^2);
  QP_MAT = zeros(16);
  QC_MAT = zeros(16);
  F_MAT  = zeros(16);
  G_MAT  = zeros(16);
end
                                                           % plotting stuff
figure(3)
clg
if MOUT(1,4) ~= 0
  step=QP-QPM1;
end
quiver(XN,YN,QP(:,2)-QPM1(:,2),QP(:,3)-QPM1(:,3))
%plot(XN,YN,'y.')
drawnow

%************************************ BOUNDARY CONDITION ROUTINE 1 **
% Finds boundary node tangents and enforces Dirichlet inflow/outflow
% boundary conditions
%**********************************************************************
MOUT = zeros(size(MOUT));
MOUT(:,1)=find(BCTYPE);

QP = bc1_rhs(QP,QPM1,XN,YN,UN,VN,MP,ELMAT,BCTYPE,BC1,MOUT_PTR,DOFNODE);
QPM1=bc1_lhs(QPM1,BC1);

%**********************************************************************
% Set up blocking instructions
%**********************************************************************
BLOCKSIZE=23;                                              % # of elements
in block
if NEL > BLOCKSIZE
  NBLOCKS = fix(NEL/BLOCKSIZE);                            % # of full
blocks
```

```
  LASTBLOCK = NEL - BLOCKSIZE * NBLOCKS;                % # of elements
in last block
  if LASTBLOCK >= 1
    BLOCKMAT = BLOCKSIZE * ones(NBLOCKS+1,1);           % BLOCKMAT is a
vector with the
    BLOCKMAT(NBLOCKS+1) = LASTBLOCK;                    % number of
elements in each
  else                                                  % block
    BLOCKMAT = BLOCKSIZE * ones(NBLOCKS,1);
  end
else
  BLOCKMAT = NEL;
  NBLOCKS = 1;
end

%***************************** START BLOCK CALCULATIONS
******************
%
%*************************************************************************
***

RB_ELEMENT = 0;
for ib = 1:length(BLOCKMAT)
  LB_ELEMENT = RB_ELEMENT + 1;                          % first element
in the block
  RB_ELEMENT = LB_ELEMENT + BLOCKMAT(ib) - 1;           % last element in
the block
  [m,n] = size(ELMAT(LB_ELEMENT:RB_ELEMENT,:));

% some really fancy sorting
  LIST = reshape(ELMAT(LB_ELEMENT:RB_ELEMENT,:),m*n,1);  % make list of
all nodes in elements
  LIST = sort(LIST);                                    % sort into
ascending order
  BLOCK_PTR = zeros(NNODES,1);                          % make a pointing
vector
  ibc = ones(size(LIST));                               % initialize
temporary subscripting vector
  ibc(2:m*n) = LIST(1:m*n-1) ~= LIST(2:m*n);            % identify
duplicate node numbers
  LIST = LIST(ibc);                                     % cull out
duplicate node numbers
  LIST = LIST(find(LIST));                              % get rid of any
zeros xxx
  BNODES = length(LIST);                                % how many unique
nodes?
  BLOCK_PTR(LIST) = 1:1:BNODES;                         % form index into
node block storage

%*************************************************************************
% Initialize maximum element workspace
%*************************************************************************
PVEC = zeros(BNODES,1);       % element nodal pressure vector
UVEC = zeros(BNODES,1);       % element nodal X velocity vector
VVEC = zeros(BNODES,1);       % element nodal Y velocity vector
TVEC = zeros(BNODES,1);       % element nodal temperature vector
```

```
TAU = zeros(BNODES,1);          % element nodal time scale parameter
vector

PS_PVEC = zeros(BNODES,1);      % element nodal delta pressure vector
PS_UVEC = zeros(BNODES,1);      % element nodal delta X velocity vector
PS_VVEC = zeros(BNODES,1);      % element nodal delta Y velocity vector
PS_TVEC = zeros(BNODES,1);      % element nodal delta temperature vector

QC = zeros(BNODES,4);           % transpose of conserved variables at
time n
QCM1 = zeros(BNODES,4);         % transpose of conserved variables at
time n-1

% space for element by element system matrices
FT = zeros(BNODES,4);           % transpose of X flux vector for all
nodes
GT = zeros(BNODES,4);           % transpose of Y flux vector for all
nodes
QV = zeros(BNODES,16);          % flat file of conservative variable
transformation matrix
QVI = zeros(BNODES,16);         % inverse of QV
LV = zeros(BNODES,16);          % flat file of steady preconditioning
matrix
LVI = zeros(BNODES,16);         % inverse of LV
AV = zeros(BNODES,16);          % flat file of X flux jacobian for all
nodes
BV = zeros(BNODES,16);          % flat file of Y flux jacobian for all
nodes
AVLI = zeros(BNODES,16);        % flat file of AV*LVI for all nodes
BVLI = zeros(BNODES,16);        % flat file of BV*LVI for all nodes

% these require interactions with shape functions
VR2 = zeros(BNODES,1);          % nodal velocity magnitude squared
C2 = zeros(BNODES,1);           % nodal acoustic speed squared
LVR = zeros(BNODES,1);          % nodal reference length/velocity

% temporary variables
HH    = zeros(BNODES,1);        % nodal static enthalpy
HO    = zeros(BNODES,1);        % nodal total enthalpy
TI    = zeros(BNODES,1);        % nodal 1/temperature
RHOP  = zeros(BNODES,1);        % nodal pressure derivative of density
RHO   = zeros(BNODES,1);        % nodal density
RHOT  = zeros(BNODES,1);        % nodal temperature derivative of density
RHOU  = zeros(BNODES,1);        % nodal X momentum (rho U)
RHOV  = zeros(BNODES,1);        % nodal Y momentum (rho V)
RHOI  = zeros(BNODES,1);        % nodal 1/density

%*********************************************************************
% Start calculating the variables for all nodes in the block
%*********************************************************************
PVEC = QP(LIST,1);                % current element nodal pressure
vector
UVEC = QP(LIST,2);                % current element nodal X velocity
vector
VVEC = QP(LIST,3);                % current element nodal Y velocity
vector
```

```
TVEC = QP(LIST,4);                    % current element nodal temperature
vector

PS_PVEC = PVEC - QPM1(LIST,1);        % delta QP(pressure) in pseudo time
PS_UVEC = UVEC - QPM1(LIST,2);        % delta QP(u velocity) in pseudo
time
PS_VVEC = VVEC - QPM1(LIST,3);        % delta QP(v velocity) in pseudo
time
PS_TVEC = TVEC - QPM1(LIST,4);        % delta QP(temperature) in pseudo
time


% calculate the working variables using MP = [R, CP, GAMMA]
C2 = (MP(3) * MP(1)) * TVEC;                % c^2 = gamma*R*t
VR2 = UVEC.^2 + VVEC.^2;                    % vr^2 = u^2 + v^2
VP2 = VR2;                                  % + (LREF/DEL_TAU)^2;
LVR = 1/2./(sqrt(VR2) + sqrt(C2));          % lvr = 1/2(|u|+c)
%LVR = 2./(sqrt(VR2) + sqrt(C2));           % lvr = 2/(|u|+c)
%LVR = 1./(sqrt(VR2) + sqrt(C2));           % lvr = 1/(|u|+c)
%LVR = 1/pi./(sqrt(VR2) + sqrt(C2));              % lvr = 1/pi(|u|+c)
 %LVR = LREF/2./(sqrt(VR2) + sqrt(C2));     % lvr = lref/2(|u|+c)
 %LVR = LREF ./ (sqrt(VR2)*pi);             % lvr = lref/(pi*vr)
 %LVR = 1/2./(sqrt(VR2) + sqrt(VR2));
 %pk = 5;                                   % pk = polynomial constant
 %LVR = LREF*(VR2 + C2/pk)./(pk*(sqrt(C2).*VR2 + sqrt(VR2).*C2));
 %LVR = LREF ./ sqrt(C2);                   % lvr = lref / c;
 %LVR = 1./ sqrt(C2);                       % lvr = 1 / c;
%xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
%TI = 1 ./ TVEC;                           % ti = 1/T
RHOP = 1/MP(1)./TVEC;                       % rhop = 1/(R*t)
RHO = PVEC .* RHOP;                         % rho = p/(R*t)
RHOT = -RHO ./ TVEC;                        % rhot = -p/(R*t^2)
RHOU = RHO .* UVEC;                         % rhou = rho*u
RHOV = RHO .* VVEC;                         % rhov = rho*v
RHOI = 1./RHO;                              % rhoi = 1/rho
HH = MP(2) * TVEC;                          % h = Cp*T
HO_REF=MP(2)*535.2;
EO_REF=(MP(2)-MP(1))*535.2;
%HH = PVEC./RHO/CONST7;
HO = HH + .5 * VR2 - HO_REF;                % ht = Cp*T + (u^2 + v^2)/2
EE = (MP(2)-MP(1))* TVEC ;                  % ee = Cv*T
EO = EE + .5*VR2 - EO_REF;                  % et = Cv*T + (u^2 + v^2)/2

% calculate the FLUX VECTORS at the nodal locations

% X mass flux      = density U
% XX momentum flux = density U U + P
% XY momentum flux = density U V
% X energy flux    = density U Ho (total specific enthalpy)
FT(:,1) = RHOU;
FT(:,2) = RHOU .* UVEC + PVEC;
FT(:,3) = RHOU .* VVEC;
FT(:,4) = RHOU .* HO;

% Y mass flux      = density V
% YX momentum flux = density U V
```

```
% YY momentum flux = density V V + P
% Y energy flux     = density V (total specific enthalpy)
GT(:,1) = RHOV;
GT(:,2) = RHOV .* UVEC;
GT(:,3) = RHOV .* VVEC + PVEC;
GT(:,4) = RHOV .* HO;

% calculate the CONSERVED VARIABLE TRANSFORMATION JACOBIAN at the nodal
locations
% as flat files

% QV(node) = [q11, q21, q31, q41, 0, q22, 0, q24, 0, 0, q33, q43, q14,
q24, q34, q44]
% QV(node) = [1/(R*t), u/(R*t)      , v/(R*t)      ,
(Cp*t+(u^2+v^2)/2)/(R*t)-1
%             0       , p/(R*t)     , 0            , u*p/(R*t)
%             0       , 0           , p/(R*t)      , v*p/(R*t)
%          -p/(R*t^2), -u*p/(R*t^2), -v*p/(R*t^2), -
p*(Cp*t+(u^2+v^2)/2)/(R*t^2)+p*Cp/(R*t)]

ZRO = zeros(BNODES,1);
WON = ones(BNODES,1);

QV = [RHOP   , UVEC.*RHOP, VVEC.*RHOP, HO.*RHOP-1          , ...
      ZRO    , RHO       , ZRO       , RHOU               , ...
      ZRO    , ZRO       , RHO       , RHOV               , ...
      RHOT   , UVEC.*RHOT, VVEC.*RHOT, HO.*RHOT+MP(2)*RHO];

% alternate version modified to make HO constant
% QV = [RHOP   , UVEC.*RHOP, VVEC.*RHOP, HO.*RHOP-1          , ...
%       ZRO    , RHO       , ZRO       , ZRO                , ...
%       ZRO    , ZRO       , RHO       , ZRO                , ...
%       RHOT   , UVEC.*RHOT, VVEC.*RHOT, HO.*RHOT           ];

% Inverse of conserved variable transformation jacobian (QV)
QVI=[CONST8*VR2/2, -UVEC.*RHOI, -VVEC.*RHOI, (-TVEC -
CONST2*VR2/2).*RHOI,...
     -CONST8*UVEC, RHOI       , ZRO        , CONST2*UVEC.*RHOI,...
     -CONST8*VVEC, ZRO        , RHOI       , CONST2*VVEC.*RHOI,...
     CONST8*WON  , ZRO        , ZRO        , -CONST2*RHOI];

% Merkle preconditioning matrix  - changed VR to VP
% Kp=1, Kt=0, Hp=1, Ht=Cp
%HT=MP(2)*ones(size(RHO));;          %Cp
%good%HT=MP(2);
%HT=(MP(2)-MP(1))*ones(size(RHO));  %Cv
%HT=VR2./TVEC/2;
%LV = [1./VP2, UVEC./VP2, VVEC./VP2, EO./VP2    ,...
%      ZRO   , RHO      , ZRO      , RHOU       ,...
%      ZRO   , ZRO      , RHO      , RHOV       ,...
%      ZRO   , ZRO      , ZRO      , HT.*RHO];

%good%LV = [1./VP2, UVEC./VP2, VVEC./VP2,(HO./VP2)-1,...
%good%      ZRO   , RHO      , ZRO      , RHOU       ,...
%good%      ZRO   , ZRO      , RHO      , RHOV       ,...
%good%      ZRO   , ZRO      , ZRO      , HT*RHO];
```

```
% Inverse of Merkle preconditioning matrix
%LVI = [VP2  , -UVEC./RHO, -VVEC./RHO, (-EE+VR2/2)./(RHO.*HT),...
%       ZRO  , 1./RHO    , ZRO       ,     -UVEC./(RHO.*HT),...
%       ZRO  , ZRO       , 1./RHO    ,     -VVEC./(RHO.*HT),...
%       ZRO  , ZRO       , ZRO       ,         1./(RHO.*HT)];
%good%LVI = [VP2  , -UVEC./RHO, -VVEC./RHO, (-HO+VR2+VP2)./(RHO*HT),...
%good%       ZRO  , 1./RHO    , ZRO       ,     -UVEC./(RHO*HT),...
%good%       ZRO  , ZRO       , 1./RHO    ,     -VVEC./(RHO*HT),...
%good%       ZRO  , ZRO       , ZRO       ,         1./(RHO*HT)];

% Alternative Merkle preconditioning matrix
% Kp=1, Kt=0, Hp=0, Ht=Cp
%LV = [1./VR2, UVEC./VR2, VVEC./VR2,(HO./VR2)-1 ,...
%      ZRO   , RHO       , ZRO      , RHOU       ,...
%      ZRO   , ZRO       , RHO      , RHOV       ,...
%      ZRO   , ZRO       , ZRO      , MP(2)*RHO];
%LVI = [VR2  , -UVEC./RHO, -VVEC./RHO, (-HO+2*VR2)./(RHO*MP(2)),...
%       ZRO  , 1./RHO    , ZRO       ,     -UVEC./(RHO*MP(2)),...
%       ZRO  , ZRO       , 1./RHO    ,     -VVEC./(RHO*MP(2)),...
%       ZRO  , ZRO       , ZRO       ,         1./(RHO*MP(2))];
% strategy: need rho(kt) included to enforce entropy condition
% may or may not need ht but definitely need one or the other
% if ht is dropped, need to include hp to enforce mass conservation
RHOPP=1./VR2;      % 1/vref^2
RHOTP = 1*RHOT;    % kt=1
HTP = MP(2)*WON;   % ht=Cp
HPP = WON;         % hp=1
DD = RHO.*HTP.*RHOPP+HPP.*RHOTP;

LV = [RHOPP , UVEC.*RHOPP, VVEC.*RHOPP, HO.*RHOPP-HPP      , ...
      ZRO   , RHO        , ZRO        , RHOU              , ...
      ZRO   , ZRO        , RHO        , RHOV              , ...
      RHOT  , UVEC.*RHOT , VVEC.*RHOT , HO.*RHOT+HTP.*RHO];

% Inverse of conserved variable transformation jacobian (QV)
LVI=[(RHO.*HTP+(HO-VR2).*RHOTP)./DD, -UVEC./RHO , -VVEC./RHO, (HPP+(-
HO+VR2).*RHOPP)./DD, ...
      UVEC.*RHOTP./DD              , 1./RHO     , ZRO       , -
UVEC.*RHOPP./DD          , ...
      VVEC.*RHOTP./DD              , ZRO        , 1./RHO    , -
VVEC.*RHOPP./DD          , ...
      -RHOTP./DD                   , ZRO        , ZRO       , RHOPP./DD
];

%LV = QV;
%LV(:,1:4) = [1./VR2, UVEC./VR2, VVEC./VR2,(HO./VR2)];
%%temp=VR2./(MP(2)*TVEC-VR2);
%temp=VR2./(MP(2)*TVEC);
%LVI=QVI;
%LVI(:,[1,5,9,13])=[VR2.*temp/2, -UVEC.*temp, -VVEC.*temp, temp];
%%temp1=MP(2)*TVEC-VR2;
%%temp=WON./(RHOT.*temp1);
%temp=WON./(RHO*MP(2));
%%LVI(:,[4,8,12,16])=[(temp1-VR2/2).*temp,UVEC.*temp,VVEC.*temp,-temp];
```

```
%LVI(:,[4,8,12,16])=[(-MP(2)*TVEC+VR2/2).*temp,-UVEC.*temp,-
VVEC.*temp,temp];

% Calculate the primitive variable FLUX JACOBIANS at the nodal
locations as flat files

% AV(node) = [a11, a21, a31, a41, a12, a22, a32, a42, a13, a23, a33,
a43, a14, a24, a34, a44]
% (1/(R*t))*[u       , R*t+u^2                        , u*v            ,
u*(Cp*t+(u^2+v^2)/2)
%           p        , 2*p*u                          , p*v            ,
p*(Cp*t+u^2+(u^2+v^2)/2)
%           0        , 0                              , p*u            ,
p*u*v
%           -(p*u)/t, -(p*u^2)/t                      , -(p*u*v)/t     ,
-(p*u*(u^2+v^2))/(2*t)     ]

AV = [RHOP.*UVEC, RHOP.*(UVEC.^2)+1   , RHOP.*UVEC.*VVEC,
RHOP.*UVEC.*HO    ,...
      RHO       , 2*RHOU              , RHOV           ,
RHO.*(HO+UVEC.^2) ,...
      ZRO       , ZRO                 , RHOU           , RHOU.*VVEC
,...
      RHOT.*UVEC, RHOT.*(UVEC.^2)     , RHOT.*UVEC.*VVEC,
UVEC.*QV(:,16)    ];

% BV(node) = [b11, b21, b31, b41, b12, b22, b32, b42, b13, b23, b33,
b43, b14, b24, b34, b44]
% (1/(R*t))*[v       , u*v                          , R*t+v^2
, v*(Cp*t+(u^2+v^2)/2)
%           0        , p*v                          , 0
, p*u*v
%           p        , p*u                          , 2*p*v
, p*(Cp*t+v^2+(u^2+v^2)/2)
%           -(p*v)/t , -(p*v*u)/t                   , -(p*v^2)/t
, -(p*v*(u^2+v^2))/(2*t)     ]

BV = [RHOP.*VVEC, RHOP.*UVEC.*VVEC , RHOP.*(VVEC.^2)+1        ,
RHOP.*VVEC.*HO    ,...
      ZRO       , RHOV             , ZRO                    ,
RHOV.*UVEC        ,...
      RHO       , RHOU             , 2*RHOV                 ,
RHO.*(HO+VVEC.^2) ,...
      RHOT.*VVEC, RHOT.*UVEC.*VVEC , RHOT.*(VVEC.^2)        ,
VVEC.*QV(:,16)    ];

% alternative version modified to make HO constant
%AV = [RHOP.*UVEC, RHOP.*UVEC.^2+1   , RHOP.*UVEC.*VVEC, RHOP.*UVEC.*HO
,...
%      RHO       , 2*RHOU            , RHOV           , RHO.*HO
,...
%      ZRO       , ZRO               , RHOU           , ZRO
,...
%      RHOT.*UVEC, RHOT.*UVEC.^2     ; RHOT.*UVEC.*VVEC, UVEC.*RHOT.*HO
];
% alternative version modified to make HO constant
```

```
%BV = [RHOP.*VVEC, AV(:,3)     , RHOP.*VVEC.^2+1          , RHOP.*VVEC.*HO
,...
%      ZRO         , RHOV       , ZRO                      , AV(:,12)
,...
%      RHO         , RHOU       , 2*RHOV                   , RHO.*HO
,...
%      RHOT.*VVEC, AV(:,15)     , RHOT.*VVEC.^2            , VVEC.*RHOT.*HO
];

% Calculated the CONSERVED VARIABLES
% density     = p/(Rt)
% X momentum = (pu)/(Rt)
% Y momentum = (pv)/(Rt)
% energy      = (p/(Rt))*(CpT+(u^2+v^2)/2)-p
% Note: Currently looks at steady state only.  Need to update here and
in KFLOW
%       for transient capability.

                    % Calculated the CONSERVED VARIABLES at time n
%  QC(:,1) = QP1(LIST,1) ./ (MP(1) * QP1(LIST,4));
%  QC(:,2) = QC(:,1) .* QP1(LIST,2);
%  QC(:,3) = QC(:,1) .* QP1(LIST,3);
%  QC(:,4) = QC(:,1) .* (MP(2)*QP1(LIST,4) + ...
%            .5*(QP1(LIST,2).^2 + QP1(LIST,3).^2)) - QP1(LIST,1);

QC(:,1) = RHO;
QC(:,2) = RHOU;
QC(:,3) = RHOV;
QC(:,4) = RHO.*HO-PVEC;

if DEL_T ~= 0
                    % Calculated the CONSERVED VARIABLES at time n-1
  QCM1(:,1) = QP2(LIST,1) ./ (MP(1) * QP2(LIST,4));
  QCM1(:,2) = QCM1(:,1) .* QP2(LIST,2);
  QCM1(:,3) = QCM1(:,1) .* QP2(LIST,3);
  QCM1(:,4) = QCM1(:,1) .* (MP(2)*QP2(LIST,4) + ...
            .5*(QP2(LIST,2).^2 + QP2(LIST,3).^2)) - QP2(LIST,1);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% form AV*LVI and BV*LVI           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:BNODES
  TEMP = reshape(LVI(i,:),4,4);
  TEMP2 = reshape(AV(i,:),4,4);
%  TEMP2(2,1)=TEMP2(2,1)-1;          %take out the pressure element
  AVLI(i,:)=reshape(TEMP2*TEMP,1,16); % multiply LV inverse
  TEMP2 = reshape(BV(i,:),4,4);
%  TEMP2(3,1)=TEMP2(3,1)-1;          %take out the pressure element
  BVLI(i,:)=reshape(TEMP2*TEMP,1,16); % multiply LV inverse
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%***************************** RESIDUAL FORMATION
************************
```

```
% Form the RESIDUAL vector using local vector operations and an element
by
% element approach
%*************************************************************************
***

for j = LB_ELEMENT:RB_ELEMENT

% check the shape of each element and set up its needed workspace size
  INDEX=find(ELMAT(j,:));  % INDEX contains the non-zero node numbers
for the element
  VERTICES = length(INDEX);

  if VERTICES == 3              % Triangle
    REGION = 2;
    LLL = 12;
    MMM = 144;
    GGG = LLL*(2*3);
    NNN = 72;
  elseif VERTICES == 4          % Quadralateral
    REGION = 1;
    LLL = 16;
    MMM = 256;
    GGG = LLL*(2*4);
    NNN = 128;
  end

SLIST = ELMAT(j,INDEX);              % index into variables stored in
data file
ELIST = BLOCK_PTR(SLIST);            % index into variables calculated
in block

% FLAT: Build flat matrix of element integration arrays for each
element
% Row 1 = sum of RHS matrices for Q primitive (shock capture+)
% Row 2 = sum of RHS matrices for Q conserved (transient physical time)
% Row 3 = sum of LHS matrices for pseudo time terms
% Row 4 = sum of RHS matrices for F (X flux vector) operations
% Row 5 = sum of RHS matrices for G (Y flux vector) operations
% Row 6 = sum of global LHS matrices for Q operations
% Row 7 = sum of global LHS matrices for pseudo time moving grid
operations

% calculate the element SUPG parameter TAU at each of the element nodes
[TAU, TSHOCK] = TAU_CALC(XN(SLIST),YN(SLIST),...
                PVEC(ELIST),UVEC(ELIST),VVEC(ELIST),TVEC(ELIST),...
                C2(ELIST),VR2(ELIST),LVR(ELIST),DEL_TAU);

% ********************* CALL THE GAUSS INTEGRATION PROCEDURE **********
% KFLOW1 contains element integration routines based on shape functions
% and nodal values calculated above
%*************************************************************************

  FLAT = gauss('kflow1',NN,REGION,XN(SLIST),YN(SLIST), ...
                PVEC(ELIST), UVEC(ELIST), VVEC(ELIST), TVEC(ELIST), ...
                FT(ELIST,:), GT(ELIST,:), QV(ELIST,:), LV(ELIST,:), ...
```

```
                LVI(ELIST,:), AV(ELIST,:), BV(ELIST,:), ...
                LVR(ELIST), C2(ELIST), VR2(ELIST), DEL_T,
DEL_TAU,DOFNODE,...
                QC(ELIST,:), TAU, AVLI(ELIST,:), BVLI(ELIST,:));

%********************** BOUNDARY CONDITION ROUTINE 2 ******************
% Integrate along boundary edges for subsonic inflow/outflow boundaries
% and to adjust zero flux terms for pressure along solid boundaries.
% Incorporate contribution into the various integration matrices for
% RHS and LHS residual calculations.
%************************************************************************
  if length(find(BCTYPE(SLIST) == 1  | ...
                 BCTYPE(SLIST) == 5  | ...
                 BCTYPE(SLIST) == 6  | ...
                 BCTYPE(SLIST) == 21 | ...
                 BCTYPE(SLIST) == 22 | ...
                 BCTYPE(SLIST) == 26 | ...
                 BCTYPE(SLIST) == 27)) > 0  | j==26

  FLAT = bc2b_rhs(FLAT,XN(SLIST),YN(SLIST),AV(ELIST,:),BV(ELIST,:),...
              BCTYPE(SLIST),DOFNODE,QC(ELIST,:),...
              TAU,AVLI(ELIST,:),BVLI(ELIST,:));

  end

  QP_MAT = reshape(FLAT(1,1:MMM),LLL,LLL);      % matrix for current Q
primitive (shock capture+)
  QC_MAT = reshape(FLAT(2,1:MMM),LLL,LLL);      % matrix for past Q
conserved (transient physical time)
  QPV_MAT = reshape(FLAT(3,1:MMM),LLL,LLL);     % matrix for (LHS)
pseudo time terms
  F_MAT  = reshape(FLAT(4,1:MMM),LLL,LLL);      % matrix for X flux
  G_MAT  = reshape(FLAT(5,1:MMM),LLL,LLL);      % matrix for Y flux
  MFLOW(j,1:MMM) = FLAT(6,1:MMM);               % matrix for LHS
  GFLOW(j,1:GGG) = FLAT(7,1:GGG);               % matrix for LHS grid
motion

                                                % Index into residual
vector
  SHUFFLE= [SLIST, SLIST+NNODES, SLIST+2*NNODES, SLIST+3*NNODES];

  if DEL_T == 0                % form RHS residual for STEADY STATE
fluid flow equations

    R(SHUFFLE)= R(SHUFFLE) + ...
                (QP_MAT*[PVEC(ELIST); UVEC(ELIST); VVEC(ELIST);
TVEC(ELIST)] + ...
                 F_MAT*[FT(ELIST,1); FT(ELIST,2); FT(ELIST,3);
FT(ELIST,4)] + ...
                 G_MAT*[GT(ELIST,1); GT(ELIST,2); GT(ELIST,3);
GT(ELIST,4)]);
%                  +...
%
QPV_MAT*[PS_PVEC(ELIST);PS_UVEC(ELIST);PS_VVEC(ELIST);PS_TVEC(ELIST)];
```

```
  else                                % form RHS residual for TRANSIENT
fluid flow equations
    % NEED TO UPDATE
    R(SHUFFLE)= R(SHUFFLE) + QP_MAT*[PVEC; UVEC; VVEC; TVEC] + ...
                QC_MAT*(reshape(QCM1,LLL,1)-
4*reshape(QC,LLL,1))/(2*DEL_T) + ...
                F_MAT*[FT(:,1); FT(:,2); FT(:,3); FT(:,4)] + ...
                G_MAT*[GT(:,1); GT(:,2); GT(:,3); GT(:,4)] + ...
                QPV_MAT*[PS_PVEC; PS_UVEC; PS_VVEC; PS_TVEC];
  end

end   % End of RESIDUAL FORMATION loop (for j=LB_ELEMENT:RB_ELEMENT)

end   % End of BLOCKING CALCULATIONS loop (for ib = 1:length(BLOCKMAT))

%********************** BOUNDARY CONDITION ROUTINE 3 ******************
% Update fully assembled residual vector to zero out residual at nodal
% variables with Dirichlet boundary conditions.
%************************************************************************

%% R = bc3_rhs(R,NNODES,DOFNODE,1);
R = bc32_rhs(R,QP,BC1,NNODES,DOFNODE,1);

      % Scale the residual to make all perturbations to all equations
have
      % roughly the same order of magnitude.  Makes the Gram-Schmidt
      % orthogonalization step in GMRES more accurate.  Must be used in
LHS
      % residual as well

global FLUID_SCALE
global RHSR

FLUID_SCALE=[1E-3;1;1;5E3;0];   % last term turns on boundary scaling
%FLUID_SCALE=[1;1;1;1;0];
%FLUID_SCALE=[1E-3;1;1;5E2];
%FLUID_SCALE = FLUID_SCALE * 5E-9;        % scale to structure
temp=length(R)/4;
R(1:temp)=R(1:temp)/FLUID_SCALE(1);
R(temp+1:2*temp)=R(temp+1:2*temp)/FLUID_SCALE(2);
R(2*temp+1:3*temp)=R(2*temp+1:3*temp)/FLUID_SCALE(3);
R(3*temp+1:4*temp)=R(3*temp+1:4*temp)/FLUID_SCALE(4);

% add emphasis to boundary residuals
if FLUID_SCALE(5)==1
  INDEX=find(BCTYPE>0);
  R=reshape(R,NNODES,4);
  R(INDEX,:)=2*R(INDEX,:);    % double all boundary residuals
  INDEX=find(BCTYPE>20);
  R(INDEX,:)=2*R(INDEX,:);    % double corner residuals again
  R=reshape(R,NNODES*4,1);
end
RHSR = R;

                        % Diagnostic plotting routines
% calculate and plot total enthalpy
```

```
E=MP(2)*(QP(:,4)-530)+.5*(QP(:,2).^2+QP(:,3).^2);
figure(3)
clg
if length(E) == 16
  surfc(reshape(E,4,4)'), view(37.5,30)
elseif length(E) == 96
  surfc(reshape(E,24,4)'), view(37.5,30)
elseif length(E) == 192
  surfc(reshape(E,24,8)'), view(37.5,30)
elseif length(E) == 1760
  surfc(reshape(E,110,16)'), view(37.5,30)
elseif length(E) > 192
  surfc(reshape(E(1:192),24,8)'), view(37.5,30)
elseif length(E) == 1600
  surfc(reshape(E,80,20)'), view(37.5,30)
end
drawnow

% calculate and plot entropy
S=(MP(3)/(MP(3)-1))*log(QP(:,4)/530)-log(QP(:,1)/629.2411);
figure(4)
clg
if length(S) == 16
  surfc(reshape(S,4,4)'), view(-37.5,30)
elseif length(S) == 96
  surfc(reshape(S,24,4)'), view(-37.5,30)
elseif length(S) == 192
  surfc(reshape(S,24,8)'), view(-37.5,30)
elseif length(S) == 1760
  surfc(reshape(S,110,16)'), view(-37.5,30)
elseif length(S) > 192
  surfc(reshape(S(1:192),24,8)'), view(-37.5,30)
elseif length(S) == 1600
  surfc(reshape(S,80,20)'), view(-37.5,30)
end
drawnow
% prepare for LHS GMRES residual plots
figure(1)
clg;

                                % create restart file

global RHSR_CTR
if RHSR_CTR > 0
  CTR = RHSR_CTR;
else
  CTR = 1;
end
[m,n]=size(R);
eval(['cd c:\matlab\restarts']);
fname=(['ITC',int2str(CTR)]);
fid=fopen(fname,'w+');
fprintf(fid,'%g     %g\n',[R,reshape(QP,m,n)]');
fclose(fid);
eval(['cd c:\matlab']);
```

BC1_RHS

```
function QP =
bc1_rhs(QP,QPM1,XN,YN,UN,VN,MP,ELMAT,BCTYPE,BC1,MOUT_PTR,DOFNODE)
% function QP =
bc1_rhs(QP,QPM1,XN,YN,UN,VN,MP,ELMAT,BCTYPE,BC1,MOUT_PTR,DOFNODE)
%
% Forms nodal tangent vectors for all boundary nodes according to RH
rule
% Determines inflow/outflow conditions based on supersonic/subsonic
normal flow
%   and directions of the characteristics.  Then sets appropriate
Dirichlet
%   boundary conditions for primitive variables.
%
% Updates the current vector of primitive nodal variables
%
% BC2_RHS adds line integration contributions along the boundaries for
subsonic
% inflow and outflow boundary conditions
% Residuals at Dirichlet boundary nodes are manipulated in BC3_RHS
after assembly
%
% QP  =     vector of primitive nodal variables [P; U; V; T]
% QPM1 =    QP at last (p-1) iteration
% XN =      vector of current X locations of nodes
% YN =      vector of current Y locations of nodes
% UN =      vector of X velocity of nodes
% VN =      vector of Y velocity of nodes
% MP =      thermodynamic properties of fluid, MP = [R, CP, GAMMA]
% ELMAT =   matrix listing the nodes assigned to each element
% BCTYPE =  vector of nodal definition of boundary condition type
%           not necessarily the most efficient implementation but it
forces
%           a single definition of boundary state for nodes
%           type 0 = not on a boundary
%           type 1 = inflow/outflow
%           type 5 = impermeable wall
%           type 6 = combination wall and inflow/outflow (i.e. a corner
node)
%           type 21= definitely inflow
%           type 22= definitely outflow
%           type 26= combination wall and definitely inflow
%           type 27= combination wall and definitely outflow
% BC1    =  matrix of type 1, 6, 21, 22, 26 & 27 boundary conditions
%           [node number, P, U, V, T]
% MOUT_PTR =
% DOFNODE = degrees of freedom per node (currently 4)
%
% global variables
% MOUT  =   matrix of nodal RH rule tangent vectors and boundary type
information
%           [node number, dx, dy, type (1 - 54), index into BC1, DX,
DY]
```

```
%           BC conditions 1=supersonic inflow,  2=subsonic inflow
%                         3=supersonic outflow, 4=subsonic outflow
%                         5=impermeable wall,
%                         51=wall+super in,     52=wall+sub in
%                         53=wall+super out,    54=wall+sub out
%           [dx,dy] = wall tangent and in/out tangent
%           [DX,DY] = in/out tangent if [dx,dy] has wall tangent for a
corner,
%                     otherwise original direction of velocity vector


global MOUT

% initialize
NNODES = length(XN);           % number of nodes
[NEL,VERTICES] = size(ELMAT); % VERTICES is the max number of nodes in
any element

% gamma*R
CONST6 = MP(3)*MP(1);

% VERTICES = determines the shape of the region (1 = rectangle, 2 =
triangle)
if VERTICES == 3               % triangle
  FLAT   = zeros(5,12^2);
  QP_MAT = zeros(12);
  QC_MAT = zeros(12);
  F_MAT  = zeros(12);
  G_MAT  = zeros(12);

elseif VERTICES == 4           % rectangle
  FLAT   = zeros(5,16^2);
  QP_MAT = zeros(16);
  QC_MAT = zeros(16);
  F_MAT  = zeros(16);
  G_MAT  = zeros(16);
end

% Step through the elements
for j = 1:NEL

% check the shape of each element and set up its needed workspace size
  INDEX=find(ELMAT(j,:));  % INDEX contains the non-zero node numbers
for the element
  VERTICES = length(INDEX);

  if VERTICES == 3             % Triangle
    REGION = 2;
    LLL = 12;
    MMM = 144;
  elseif VERTICES == 4         % Quadralateral
    REGION = 1;
    LLL = 16;
    MMM = 256;
  end
```

```
% Is this element exposed to a BOUNDARY on more than one node?
  nonzero=find(BCTYPE(ELMAT(j,INDEX)));

% Calculate the right hand rule tangent vector on each side of the
element
  if length(nonzero) > 1;

% Get the proper cyclic order of the nodes
    for left = 1:VERTICES
      if left < VERTICES          % kicks out if left = VERTICES
        right = left + 1;
      else
        right = 1;
      end

      % Form and assemble TANGENT vectors [DX,DY] in RH rule direction
      % Normal vectors [DY,-DX] will be OUT of region
      % Entire side must be exposed to boundary since tangents are
formed at
      % the midpoints and assembled at the nodes

      if (BCTYPE(ELMAT(j,left)) ~= 0) & (BCTYPE(ELMAT(j,right)) ~= 0)
        DX = XN(ELMAT(j,right))-XN(ELMAT(j,left));
        DY = YN(ELMAT(j,right))-YN(ELMAT(j,left));
        DN = norm([DX,DY]);                                          %
normalize
        MOUT_LEFT = MOUT_PTR(ELMAT(j,left));
        MOUT_RIGHT = MOUT_PTR(ELMAT(j,right));

        if ((BCTYPE(ELMAT(j,left)) == 6)  |  ...
            (BCTYPE(ELMAT(j,left)) == 26) |  ...
            (BCTYPE(ELMAT(j,left)) == 27)) &  ...
           ((BCTYPE(ELMAT(j,right)) == 1) |  ...
            (BCTYPE(ELMAT(j,right)) == 21)|  ...
            (BCTYPE(ELMAT(j,right)) == 22))
          MOUT(MOUT_LEFT,6:7) = MOUT(MOUT_LEFT,6:7) + [DX,DY]/DN; %
separate corner tangents
          MOUT(MOUT_LEFT,4) = 6;                                     %
mark it
        else
          MOUT(MOUT_LEFT,2:3) = MOUT(MOUT_LEFT,2:3) + [DX,DY]/DN; %
assemble at nodes
        end

        if ((BCTYPE(ELMAT(j,right)) == 6)  |  ...
            (BCTYPE(ELMAT(j,right)) == 26) |  ...
            (BCTYPE(ELMAT(j,right)) == 27)) &  ...
           ((BCTYPE(ELMAT(j,left)) == 1)   |  ...
            (BCTYPE(ELMAT(j,left)) == 21)  |  ...
            (BCTYPE(ELMAT(j,left)) == 22))
          MOUT(MOUT_RIGHT,6:7) = MOUT(MOUT_RIGHT,6:7) + [DX,DY]/DN;
      % separate corner tangents
          MOUT(MOUT_RIGHT,4) = 6;
      % mark it
        else
```

```
          MOUT(MOUT_RIGHT,2:3) = MOUT(MOUT_RIGHT,2:3) + [DX,DY]/DN;
      % assemble at nodes
        end

      end
    end
  end              % will need to renormalize nodal tangent vectors
after element assembly
end

% renormalize tangent vectors at BOUNDARY nodes
NBC = length(MOUT(:,1));
for j=1:NBC
  MOUT(j,2:3) = MOUT(j,2:3)/norm(MOUT(j,2:3));
  if MOUT(j,4) == 6 | MOUT(j,4) > 50                                    %
corner node
    MOUT(j,6:7) = MOUT(j,6:7)/norm(MOUT(j,6:7));
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% apply INFLOW/OUTFLOW
boundary conditions
INDEX = BC1(:,1)';                                  % create a list of all
in/out node numbers
NBC1 = length(INDEX);                               % this should also be
the length of BC1

% Step through the inflow/outflow nodes
for j=1:NBC1                                        % QP=[P;U;V;T];
  QPL = QP(INDEX(j),:)';                            % current [P; U; V; T]
for node
%  QPL1 = QPM1(INDEX(j),:)';                        % time n-1 [P; U; V; T]
for node
  MOUT_INDEX = MOUT_PTR(INDEX(j));

  if BCTYPE(INDEX(j),1) == 6  | ...
     BCTYPE(INDEX(j),1) == 26 | ...
     BCTYPE(INDEX(j),1) == 27
    TAN_VEC=MOUT(MOUT_PTR(INDEX(j)),6:7)';          % nodal tangent vector
[dx;dy]
  else
    TAN_VEC=MOUT(MOUT_PTR(INDEX(j)),2:3)';          % nodal tangent vector
[dx;dy]
  end
  NORM_VEC = [-TAN_VEC(2);TAN_VEC(1)];              % nodal inward normal
vector [-dy;dx]
  VIN = QPL(2:3)'*NORM_VEC;                         % nodal inward normal
velocity component

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% INFLOW
  if (VIN >= 0 | BCTYPE(INDEX(j),1) == 21 | BCTYPE(INDEX(j),1) == 26)
...
              & BCTYPE(INDEX(j),1) ~= 22 & BCTYPE(INDEX(j),1) ~= 27

    if VIN^2-CONST6*QPL(4) > 0                      % SUPERSONIC RELATIVE
INFLOW (1)
```

```
      if BCTYPE(INDEX(j),1) == 1 | BCTYPE(INDEX(j),1) == 21
        MOUT(MOUT_INDEX,4) = 1;                  % remember BC type
      else
        MOUT(MOUT_INDEX,4) = 51;                 % remember BC type
      end
      MOUT(MOUT_INDEX,5) = j;                    % remember index into
BC1
      QP(INDEX(j),:) = BC1(j,2:5);               % set all boundary
values

    else                                         % SUBSONIC RELATIVE
INFLOW (2)
      if BCTYPE(INDEX(j),1) == 1 | BCTYPE(INDEX(j),1) == 21
        MOUT(MOUT_INDEX,4) = 2;                  % remember BC type
      else
        MOUT(MOUT_INDEX,4) = 52;                 % remember BC type
      end
      MOUT(MOUT_INDEX,5) = j;                    % remember index into
BC1
      QP(INDEX(j),2:4) = BC1(j,3:5);             % set [U,V,T] boundary
values
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% OUTFLOW
  else
    if VIN^2-CONST6*QPL(4) > 0                   % SUPERSONIC RELATIVE
OUTFLOW (3)
      if BCTYPE(INDEX(j),1) == 1 | BCTYPE(INDEX(j),1) == 22
        MOUT(MOUT_INDEX,4) = 3;                  % remember BC type
      else
        MOUT(MOUT_INDEX,4) = 53;                 % remember BC type
      end
      MOUT(MOUT_INDEX,5) = j;                    % remember index into
BC1
                                                 % set no boundary
values

    else                                         % SUBSONIC RELATIVE
OUTFLOW (4)
      if BCTYPE(INDEX(j),1) == 1 | BCTYPE(INDEX(j),1) == 22
        MOUT(MOUT_INDEX,4) = 4;                  % remember BC type
      else
        MOUT(MOUT_INDEX,4) = 54;                 % remember BC type

      end
      MOUT(MOUT_INDEX,5) = j;                    % remember index into
BC1
      QP(INDEX(j),1) = BC1(j,2);                 % set [P] boundary
value
    end
  end
end

% apply TYPE 5, 6, 26, & 27 BOUNDARY CONDITIONS (impermeable wall)
INDEX = find(BCTYPE == 5 | BCTYPE == 6 | BCTYPE == 26 | BCTYPE == 27);
NBC2 = length(INDEX);
```

```
for j=1:NBC2

  MOUT_INDEX = MOUT_PTR(INDEX(j));              % index for node in
MOUT
  if MOUT(MOUT_INDEX,4) < 50                    % check to make sure
its not a corner node
    MOUT(MOUT_INDEX,4) = 5;                     % remember BC type
  end
end
```

BC2B_RHS

```
function FLAT =
bc2b_rhs(FLAT,XN,YN,AV,BV,BCTYPE,DOFNODE,QC,TAU,TAVLI,TBVLI)
%
% function FLAT =
bc2b_rhs(FLAT,XN,YN,AV,BV,BCTYPE,DOFNODE,QC,TAU,TAVLI,TBVLI)
%
% Adds line integration contributions along the boundaries for subsonic
%   inflow and outflow boundary conditions.  Contributions are used to
update the
%   RHS matrices and stored in MFLOW for the LHS calculations.  Expects
input data
%   to be for a single element.
%
% BC1_RHS:
% Forms nodal tangent vectors for all boundary nodes according to RH
rule
% Enforces impermeable wall flow tangency conditions by rotating
velocity vector
% Determines inflow/outflow conditions based on supersonic/subsonic
normal flow
%   and directions of the characteristics.  Then sets appropriate
Dirichlet
%   boundary conditions for primitive variables.
%
% BC3_RHS:
% Modifies fully assembled RESIDUALS at Dirichlet boundary nodes by
setting
%   appropriate components to zero.  Used in both LHS and RHS.
%
% Variables:
% FLAT =    flattened matrix of the various integration matrices needed
for both
%           the RHS and LHS residual formation
% XN =      vector of current X locations of nodes
% YN =      vector of current Y locations of nodes
% AV =      flat matrix (row = node) of the X flux jacobian
% BV =      flat matrix (row = node) of the Y flux jacobian
% BCTYPE =  vector of nodal definition of boundary condition type
```

```
%           not necessarily the most efficient implementation but it
forces
%           a single definition of boundary state for nodes
%           type 0 = not on a boundary
%           type 1 = inflow/outflow
%           type 5 = impermeable wall
%           type 6 = combination wall and inflow/outflow (i.e. a corner
node)
%           type 21= definitely inflow
%           type 22= definitely outflow
%           type 26= combination wall and definitely inflow
%           type 27= combination wall and definitely outflow
% NNODES =  number of nodes in global domain
% DOFNODE = degrees of freedom per node (currently 4)
% QC =      Matrix of conserved variables [rho, rho*U, rho*V, rho*Ho-
P].  Used as
%           LHS source terms during grid motion.
%

NNODES = length(XN);
NDOF = DOFNODE*NNODES;
MMM = NDOF^2;
GGG = NDOF*NNODES*2;

% form TAU*AV*LVI and TAU*BV*LVI for all the nodes in the element
for i=1:NNODES
  TAVLI(i,:) = TAU(i) * TAVLI(i,:);
  TBVLI(i,:) = TAU(i) * TBVLI(i,:);
end

% Put the integration matrices in a useful shape
QP_MAT = reshape(FLAT(1,1:MMM),NDOF,NDOF);      % matrix for primitive
variables
F_MAT  = reshape(FLAT(4,1:MMM),NDOF,NDOF);      % matrix for X flux
G_MAT  = reshape(FLAT(5,1:MMM),NDOF,NDOF);      % matrix for Y flux
LHS_MAT = reshape(FLAT(6,1:MMM),NDOF,NDOF);     % matrix for LHS
GRID_MAT = reshape(FLAT(7,1:GGG),NDOF, NNODES*2);     % matrix for LHS
grid motion

% Create matrices to represent the edge flux contribution [2 1]* LE *
n_hat (component)
%                                                          [1 2]   6
% The edge length and outward normal component will be accounted for
later
% Matrices are used to work one node at a time
EDGE     = [2,1;1,2]/6;                    % use with pressure terms
EDGEMAT1 = kron(eye(DOFNODE),EDGE);        % use with F, G
EDGEMAT2 = kron(ones(DOFNODE),EDGE);       % use with AV, BV
EDGEMAT3 = kron(ones(DOFNODE,1),EDGE);     % use with grid motion

% Step through the edges one at a time.  Currently assumes linear edges

for i = 1:NNODES
  NODE = i;
  if NODE == 1
    NODEM = NNODES;
```

```
  else
    NODEM = NODE - 1;
  end

  % Index for the rows of the node in question
  RINDEX = kron([0;1;2;3],[1;1]) * NNODES +
kron(ones(4,1),[NODEM;NODE]);
  RXP_INDEX = 1*NNODES + [NODEM;NODE];    % Index for the row with the
X pressure flux
  RYP_INDEX = 2*NNODES + [NODEM;NODE];    % Index for the row with the
Y pressure flux

% Is the primary node on an inflow/outflow boundary?  If so, proceed.
  if (BCTYPE(NODE) == 1  | ...
      BCTYPE(NODE) == 6  | ...
      BCTYPE(NODE) == 21 | ...
      BCTYPE(NODE) == 22 | ...
      BCTYPE(NODE) == 26 | ...
      BCTYPE(NODE) == 27)

% Is the preceding node on an inflow/outflow boundary?  If so, add in
the edge flux
% contribution
    if (BCTYPE(NODEM) == 1  | ...
        BCTYPE(NODEM) == 6  | ...
        BCTYPE(NODEM) == 21 | ...
        BCTYPE(NODEM) == 22 | ...
        BCTYPE(NODEM) == 26 | ...
        BCTYPE(NODEM) == 27)

      % Outward normal = [DY, -DX]/LE
      DX = XN(NODE)-XN(NODEM);
      DY = YN(NODE)-YN(NODEM);
      LE = norm([DX,DY]);

      CINDEX = RINDEX';
      AV_NODE = reshape(AV(NODE,:),DOFNODE,DOFNODE);
      AV_NODEM = reshape(AV(NODEM,:),DOFNODE,DOFNODE);
      BV_NODE = reshape(BV(NODE,:),DOFNODE,DOFNODE);
      BV_NODEM = reshape(BV(NODEM,:),DOFNODE,DOFNODE);

      % Update the matrices with this edge's contribution
      F_MAT(RINDEX,CINDEX) = F_MAT(RINDEX,CINDEX) + EDGEMAT1 * DY;
      G_MAT(RINDEX,CINDEX) = G_MAT(RINDEX,CINDEX) + EDGEMAT1 * (-DX);
      LHS_MAT(RINDEX,CINDEX) = LHS_MAT(RINDEX,CINDEX) + ...
            kron(reshape([AV_NODEM;AV_NODE],DOFNODE,2*DOFNODE),[1;1]).*
EDGEMAT2 * DY + ...
            kron(reshape([BV_NODEM;BV_NODE],DOFNODE,2*DOFNODE),[1;1]).*
EDGEMAT2 * (-DX);

% shock capture
      EDGE_DNDX = kron(eye(4),[-1,1])*DY/LE;                    %
(4x8) normal
      EDGE_DNDY = kron(eye(4),[-1,1])*(-DX)/LE;                 %
(4x8) normal
```

```
      BIG_N = kron(eye(4,4),[1,1]/2);                                    %
(4x8) for use with flux

% take shock capture out of edge terms
%       delta=mean(TAU)/100;
      delta=0;
      QP_MAT(RINDEX,CINDEX) = QP_MAT(RINDEX,CINDEX) -
delta*BIG_N'*(EDGE_DNDX+EDGE_DNDY);
      LHS_MAT(RINDEX,CINDEX) = LHS_MAT(RINDEX,CINDEX) -
delta*BIG_N'*(EDGE_DNDX+EDGE_DNDY);

      % no further flux changes needed for grid motion
    end
end

% Is the primary node on a solid wall boundary?  If so, proceed.
    if (BCTYPE(NODE) == 5  | ...
        BCTYPE(NODE) == 6  | ...
        BCTYPE(NODE) == 26 | ...
        BCTYPE(NODE) == 27)

% Is the preceding node on a solid wall boundary?  If so, add in the
pressure
% force contribution to the surface on the primary node.  The pressure
contribution
% to the other node will be accounted for when it becomes the primary
node.
    if (BCTYPE(NODEM) == 5  | ...
        BCTYPE(NODEM) == 6  | ...
        BCTYPE(NODEM) == 26 | ...
        BCTYPE(NODEM) == 27)

      % Outward normal = [DY, -DX]/LE
      DX = XN(NODE)-XN(NODEM);
      DY = YN(NODE)-YN(NODEM);
      LE = norm([DX,DY]);

      CINDEX = RINDEX';

% form TAU*DN/DX*AV*LVI and TAU*DN/DY*BV*LVI
      EDGE_DNDXAVLI = reshape(TAVLI([NODEM,NODE],:),4*2,4) ...%
                      .* (kron(ones(4,4),[-1;1])*DX/LE);          %
(8x4) tangent
      EDGE_DNDYBVLI = reshape(TBVLI([NODEM,NODE],:),4*2,4) ...
                      .* (kron(ones(4,4),[-1;1])*DY/LE);          %
(8x4) tangent
%     EDGE_DNDXAVLI = kron(eye(4),[-1;1])*(DX/LE)* ...
%                     reshape(mean(TAVLI([NODEM,NODE],:)),4,4);
      % (8x4) tangent
%      EDGE_DNDYBVLI = kron(eye(4),[-1;1])*(DY/LE)* ...
%                     reshape(mean(TBVLI([NODEM,NODE],:)),4,4); %
(8x4) tangent

      SUPG = (EDGE_DNDXAVLI + EDGE_DNDYBVLI)';  % (4x8) transpose later
      BIG_N = kron(eye(4,4),[1,1]/2);           % (4x8) for use with F,
G
```

```
      BIG_NP = zeros(4,2*4);                          % (4x8) start building
pressure contribution
      BIG_NP(2:3,1:2)=[DY,DY;(-DX),(-DX)]/LE/2; % (4x8) add in edge
normal vector components
      EDGE_FLUX = SUPG'*BIG_N;                        % (8x8) for use with
flux terms

% take out wall flux created by SUPG terms and add pressure back in
      F_MAT(RINDEX,CINDEX) = F_MAT(RINDEX,CINDEX) - EDGE_FLUX * DY/LE;
      G_MAT(RINDEX,CINDEX) = G_MAT(RINDEX,CINDEX) - EDGE_FLUX * (-
DX)/LE;
      QP_MAT(RINDEX,CINDEX) = QP_MAT(RINDEX,CINDEX) + SUPG' * BIG_NP;

% Create the nodal matrices for the LHS and remove the pressure force
terms
AVMP_NODE = reshape(AV(NODE,:),DOFNODE,DOFNODE);
AVMP_NODE(2,1)=AVMP_NODE(2,1)-1;
AVMP_NODEM = reshape(AV(NODEM,:),DOFNODE,DOFNODE);
AVMP_NODEM(2,1)=AVMP_NODEM(2,1)-1;
BVMP_NODE = reshape(BV(NODE,:),DOFNODE,DOFNODE);
BVMP_NODE(3,1)=BVMP_NODE(3,1)-1;
BVMP_NODEM = reshape(BV(NODEM,:),DOFNODE,DOFNODE);
BVMP_NODEM(3,1)=BVMP_NODEM(3,1)-1;

% take out wall flux created by SUPG terms
LHS_MAT(RINDEX,CINDEX) = LHS_MAT(RINDEX,CINDEX) - SUPG' * ...

(reshape([AVMP_NODEM;AVMP_NODE],DOFNODE,2*DOFNODE) * DY/LE/2 ...
                          +
reshape([BVMP_NODEM;BVMP_NODE],DOFNODE,2*DOFNODE) * (-DX)/LE/2);

% form DNDX*nx and DNDY*ny for shock capture removal
      EDGE_DNDX = kron(eye(4),[-1,1])*DY/LE;                    %
(4x8) normal
      EDGE_DNDY = kron(eye(4),[-1,1])*(-DX)/LE;                 %
(4x8) normal

% take shock capture out of wall terms
%     delta=mean(TAU)/100;
     delta=0;
     QP_MAT(RINDEX,CINDEX) = QP_MAT(RINDEX,CINDEX) -
delta*BIG_N'*(EDGE_DNDX+EDGE_DNDY);
     LHS_MAT(RINDEX,CINDEX) = LHS_MAT(RINDEX,CINDEX) -
delta*BIG_N'*(EDGE_DNDX+EDGE_DNDY);

      CINDEX = [NODEM,NODE];

      % Update the matrices with this edge's pressure force
contribution
      QP_MAT(RXP_INDEX,CINDEX) = QP_MAT(RXP_INDEX,CINDEX) + EDGE * DY;
      QP_MAT(RYP_INDEX,CINDEX) = QP_MAT(RYP_INDEX,CINDEX) + EDGE * (-
DX);
      LHS_MAT(RXP_INDEX,CINDEX) = LHS_MAT(RXP_INDEX,CINDEX) + EDGE *
DY;
      LHS_MAT(RYP_INDEX,CINDEX) = LHS_MAT(RYP_INDEX,CINDEX) + EDGE * (-
DX);
```

```
%       GRID_MAT(RINDEX, [CINDEX, CINDEX + NNODES]) = ...
%           GRID_MAT(RINDEX, [CINDEX, CINDEX + NNODES]) + ...
%           [kron(QC(CINDEX,1:DOFNODE)',[1;1]).*EDGEMAT3*DY, ...
%            kron(QC(CINDEX,1:DOFNODE)',[1;1]).*EDGEMAT3*(-DX)];
    end
  end
end                              % looping through edges

% reshape matrices to flat array
FLAT([1,4:6],1:MMM)= [reshape(QP_MAT,1,MMM); ...
                      reshape(F_MAT,1,MMM); ...
                      reshape(G_MAT,1,MMM); ...
                      reshape(LHS_MAT,1,MMM)];
```

BC32_RHS

```
function R = bc32_rhs(R,QP,BC1,NNODES,DOFNODE,NFLAG)
%
% function R = bc32_rhs(R,QP,BC1,NNODES,DOFNODE,NFLAG)
%
% Modifies fully assembled RESIDUALS at Dirichlet boundary nodes by
setting
%   appropriate components to zero.  Used in both LHS and RHS.
%
% BC1_RHS:
% Forms nodal tangent vectors for all boundary nodes according to RH
rule
%* Enforces impermeable wall flow tangency conditions by rotating
velocity vector
% Determines inflow/outflow conditions based on supersonic/subsonic
normal flow
%   and directions of the characteristics.  Then sets appropriate
Dirichlet
%   boundary conditions for primitive variables.
%
% BC2_RHS:
% Adds line integration contributions along the boundaries for subsonic
%   inflow and outflow boundary conditions
%
% Variables:
% R =       right hand side residual
% NNODES =  number of nodes in global domain
% DOFNODE = degrees of freedom per node (currently 4)
% NFLAG  =  flag for zeroing out one of the residuals at wall boundary
nodes
%           NFLAG = 1 ... Do it,   NFLAG = 0 ... Don't do it
%           - use if rigorously enforcing wall BC during LHS solve
%           - use or not if using wall penalty during LHS solve
%
% global variables
```

```
% MOUT  =   matrix of nodal RH rule tangent vectors and boundary type
information
%          [node number, dx, dy, type (1 - 54), index into BC1, DX,
DY]
%          BC conditions 1=supersonic inflow,  2=subsonic inflow
%                        3=supersonic outflow, 4=subsonic outflow
%                        5=impermeable wall,
%                        51=wall+super in,     52=wall+sub in
%                        53=wall+super out,    54=wall+sub out
%          [dx,dy] = wall tangent and in/out tangent
%          [DX,DY] = in/out tangent if [dx,dy] has wall tangent for a
corner,
%                    otherwise original direction of velocity vector

global MOUT

% initialize
NBC = length(MOUT(:,1));

for j=1:NBC                                  % step through the boundary
elements
  INDEX = NNODES*[0:1:DOFNODE-1]+MOUT(j,1);

  if MOUT(j,4) == 5                                                    %
impermeable wall
    if NFLAG == 1
      if abs(MOUT(j,2)) > abs(MOUT(j,3))                               % remove a
DOF by making
%        R(INDEX(3)) = 0;                                              % an
element of the residual zero
      else
%        R(INDEX(2)) = 0;
      end
%         R(INDEX(2:3))= norm(R(INDEX(2:3)))*MOUT(j,8:9)';  % rotate
onto original velocity
    end
%%      R(INDEX)=R(INDEX)*2;                                           % amplify
wall boundary

  elseif MOUT(j,4) == 1 | MOUT(j,4) == 51                              %
supersonic inflow
    R(INDEX) = [0;0;0;0];                                              % Dirichlet
[P,U,V,T]
%%      R(INDEX)=QP(INDEX(1),1:4)' - BC1(MOUT(j,5),2:5)';

  elseif MOUT(j,4) == 2 | MOUT(j,4) == 52                              % subsonic
inflow
                                                                       % [P]
calculated
    R(INDEX(2:4)) = [0;0;0];                                           % Dirichlet
[U,V,T]
%%R(INDEX(2:4))=QP(INDEX(1),2:4)' - BC1(MOUT(j,5),3:5)';

  elseif MOUT(j,4) == 3 | MOUT(j,4) == 53                              %
supersonic outflow
```

```
    if MOUT(j,4) == 53                                          %
wall/outflow corner
      if NFLAG == 1
        if abs(MOUT(j,2)) > abs(MOUT(j,3))
%          R(INDEX(3)) = 0;
        else
%          R(INDEX(2)) = 0;
        end
%         R(INDEX(2:3))= norm(R(INDEX(2:3)))*MOUT(j,8:9)';  % rotate
onto original velocity
      end
%%      R(INDEX)=R(INDEX)*2;                                  % amplify
wall boundary
    end
    % do nothing                                              % [P,U,V,T]
calculated

  elseif MOUT(j,4) == 4 | MOUT(j,4) == 54                     % subsonic
outflow
    if MOUT(j,4) == 54                                        %
wall/outflow corner
      if NFLAG == 1
        if abs(MOUT(j,2)) > abs(MOUT(j,3))
%          R(INDEX(3)) = 0;
        else
%          R(INDEX(2)) = 0;
        end
%         R(INDEX(2:3))= norm(R(INDEX(2:3)))*MOUT(j,8:9)';  % rotate
onto original velocity
      end
%%      R(INDEX)=R(INDEX)*2;                                  % amplify
wall boundary
    end
    R(INDEX(1)) = 0;                                          % Dirichlet
[P]
%%      R(INDEX(1))=QP(INDEX(1),1)' - BC1(MOUT(j,5),2)';
                                                              % [U,V,T]
calculated
  end
end
```

TAU_CALC

```
function [TSUPG,TSHOCK] =
TAU_CALC(XN,YN,PVEC,UVEC,VVEC,TVEC,C2,VR2,LVR,DEL_TAU)
%
%function [TSUPG,TSHOCK] =
TAU_CALC(XN,YN,PVEC,UVEC,VVEC,TVEC,C2,VR2,LVR,DEL_TAU)
%
%Calculates the SUPG parameter TAU
%
% Variables:
```

```
% XN =       vector of current X locations of nodes
% YN =       vector of current Y locations of nodes
% AV =       flat matrix (row = node) of the X flux jacobian
% BV =       flat matrix (row = node) of the Y flux jacobian
% LVI =      flat matrix (row = node) of the inverse time preconditioner
% PVEC =     nodal pressure vector
% UVEC =     nodal velocity in X direction
% VVEC =     nodal velocity in Y direction
% TVEC =     nodal temperature vector
% C2 =       nodal acoustic velocity squared vector
% VR2 =      nodal reference velocity vector
% LVR =      nodal reference dimension
% DEL_TAU = pseudo time step size
%
% Output:
% TSUPG = nodal time scale parameter for SUPG term.
% TSHOCK = nodal time scale parameter for the shock capturing.  Still
needs delta

NNODES = length(XN);
TX = max(XN)-min(XN);
TY = max(YN)-min(YN);
M2=VR2./C2;

SKIP=0;
if SKIP == 0

  KKK=norm([TX,TY])/(2*TX*TY); % great version
%  VEL_DOT_K = abs(UVEC/2/TX)+abs(VVEC/2/TY);
%  TAU=1./(VEL_DOT_K.*(1+M2)/2 + sqrt((VEL_DOT_K.*(1-M2)/2).^2 +
C2*KKK^2));

  TAU=1./(abs(UVEC/2/TX)+abs(VVEC/2/TY)+KKK*sqrt(C2));      % best
version
%  TAU=1./(abs(UVEC/2/TX)+abs(VVEC/2/TY)+KKK*sqrt(VR2));

%  TAU=max(2*TX./(abs(UVEC)+sqrt(C2)),2*TY./(abs(VVEC)+sqrt(C2)));
%  TAU=min(2*TX./(abs(UVEC)+sqrt(C2)),2*TY./(abs(VVEC)+sqrt(C2)));
%  TAU=min(TX./(abs(UVEC)+sqrt(C2)),TY./(abs(VVEC)+sqrt(C2)));

  TSUPG = TAU;
  TSHOCK = zeros(size(TAU));

else                          % calculate gradient based version

  SHAPE = length(XN);
  if SHAPE == 4   % quadralateral
    TYPE = 1;
   elseif SHAPE == 3  % triangle
    TYPE = 2;
  end

  QNORM2 = (PVEC.^2+UVEC.^2+VVEC.^2+TVEC.^2);

  [DNIDXJ] = gauss('KB11BL',1,TYPE,XN,YN);
```

```
  BETA = DNIDXJ'*QNORM2;
  if norm(BETA)~=0
    BETA = BETA/norm(BETA);
  end

  LTERM=max([TX;TY].*abs(BETA));
  KKK=1;
  VEL_DOT_K = abs([UVEC,VVEC]*BETA);
%  TAU = 1./(VEL_DOT_K.*(1+M2)/2 + sqrt((VEL_DOT_K.*(1-M2)/2).^2 +
C2*KKK^2));
  TAU = 1./(VEL_DOT_K+sqrt(C2));

  TSUPG = LTERM*TAU;
  TSHOCK = TAU.^2;

end

end
```

GAUSS

```
function [M, MM] = gauss(fun,N,T,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,...
                         P11,P12,P13,P14,P15,P16,P17,P18,P19,P20,...
                         P21,P22,P23,P24,P25,P26,P27,P28,P29,P30)

%  function [M, MM] = gauss(fun,N,T,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,...
%                           P11,P12,P13,P14,P15,P16,P17,P18,P19,P20,...
%                           P21,P22,P23,P24,P25,P26,P27,P28,P29,P30)
%
%  Performs gauss quadrature integration over a regular interval
%  fun = string variable of the name of the function to be integrated
%  N   = integer (1, 2, 3 for linear, quadratic, cubic) describing the
%        order of the integration (see Pgauss.m for details)
%  T   = integer (1=rectangle, 2=triangle, 3=line) describing
%        the shape of the region over which to integrate (see Pgauss.m)
%  P1-P30 = parameters to be passed to the function being integrated.
The
%        program checks for the number of parameters and passes them to
the
%        function as FUN(ZETA,P1,P2,...).
%  M   = the output variable equal to the sum of products of the
function
%        evaluated at the gauss points times the gauss weight
%  MM  = optional second output variable equal to the sum of products
of the
%        function evaluated at the gauss points times the gauss weight

% get gauss points and weights
[ZETA,WT]=pgauss(N,T);

% set up arguments for function call
args1 = '(ZETA(';
```

```
args2 = ',:)';
for j = 1:nargin-3
   args2 = [args2,',P',int2str(j)];
end
args2 = [args2,')'];

% evaluate function at gauss points, multiply by weights, and sum
for j = 1:length(WT)
  args = [args1,int2str(j),args2];
  if nargout == 1
    y = eval([fun,args])*WT(j);
  elseif nargout == 2
    [y,z] = eval([fun,args]);
     y = y * WT(j);
     z = z * WT(j);
  end
  if j == 1
    M = zeros(size(y));
  end
  M = M + y;
  if nargout == 2
    if j == 1
      MM = zeros(size(z));
    end
    MM = MM + z;
  end
end
```

KFLOW1

```
function MFLAT = kflow1(ZETA,XN,YN,PVEC,UVEC,VVEC,TVEC,...
                        FT,GT,QV,LV,LVI,AV,BV,LVR,C2,VR2,...
                        DEL_T,DEL_TAU,DOFNODE,QC,TAU,AVLI,BVLI)
%function MFLAT = kflow1(ZETA,XN,YN,PVEC,UVEC,VVEC,TVEC,...
%                        FT,GT,QV,LV,LVI,AVTRANS,BVTRANS,...
%                        DNDXAVLI,DNDYBVLI,TAU,C2,VR2,...
%                        DEL_T,DEL_TAU,DOFNODE,QC,TAU,AVLI,BVLI)
%
%  Kevin Kilpatrick,  9 Jan 98
%                    21 Oct 98 - updated to include LHS grid motion
terms
%
%  Evaluates the flowfield at a point within an element using a
%  pseudo-time preconditioned TWS/SUPG finite element method.
%  Intended for use in gauss quadrature.
%
%  Uses Green-Gauss theorem to transform flux gradient term to volume
%  plus surface integrals.  Surface integrals are not evaluated due to
%  known cancellation at assembly.
%
%  Complete algorithm for elements not touching a boundary of any type.
```

```
%  Impermeable wall boundaries must be corrected for pressure
(BC2_RHS).
%  Inflow/outflow boundaries must be corrected for surface flux
(BC2_RHS).
%
%  Variables:
%  ZETA =    row vector of the current position in local coordinates
%  XN =      column vector of the global X position of the element nodes
%  YN =      column vector of the global Y position of the element nodes
%  PVEC =    column vector of the nodal pressures
%  UVEC =    column vector of the nodal X velocities
%  VVEC =    column vector of the nodal Y velocities
%  TVEC =    column vector of the nodal temperatures
%  FT =      matrix (row = node)of the nodal X flux variables
[FP,FU,FV,FT]
%  GT =      matrix (row = node)of the nodal Y flux variables
[GP,GU,GV,GT]
%  QV =      flat matrix (row = node) of the conserved/primitive
jacobian
%  LV =      flat matrix (row = node) of the preconditioning matrix
%  LVI =     flat matrix (row = node) of the preconditioning matrix
inverse
%  AV =      flat matrix (row = node) of the X flux jacobian
%  BV =      flat matrix (row = node) of the Y flux jacobian
%  LVR =     column vector of the reference length/nodal velocities
%  C2 =      column vector of the nodal acoustic velocities squared
%  DEL_T =   physical time step size (0 if steady state)
%  DEL_TAU = pseudo time step size
%  DOFNODE = degrees of freedom per node (4)
%  QC =      Matrix of conserved variables [rho, rho*U, rho*V, rho*Ho-
P].  Used as
%            LHS source terms during grid motion.
%  TAU =     nodal SUPG time scale parameter
%  AVLI =    nodal combination AV*LVI
%  BVLI =    nodal combination BV*LVI
%
%  MFLAT =   flat (row relates to vector multiplier) matrix of element
integration
%            matrices at the current location
%            [QP_MAT                  RHS term multiplying current Q
primitive (shock capture)
%             QC_MAT                  RHS term multiplying past Q
conservative
%             QPV_MAT                 RHS term multiplying pseudo time Q
primitive
%             F_MAT                   RHS term multiplying X flux
variables
%             G_MAT                   RHS term multiplying Y flux
variables
%             QP_LHS                  total LHS matrix multiplying
Delta_Q primitive
%             GRID]*AE;               LHS matrix multiplying pseudo time
grid motion terms

NNODES = length(XN);
NDOF = DOFNODE*NNODES;
```

```
MMM = NDOF^2;
GGG = NDOF*NNODES*2;

% NNODES = determines the shape of the region (4 = rectangle, 3 =
triangle)
if NNODES == 3                % Triangle
 DNIDZJ = zeros(3);
 DZJDXK = zeros(3,2);
 DXJDZI = zeros(2,3);
 NI     = zeros(3,1);
 DNIDXK = zeros(3,2);
 ZIP    = zeros(3,1);
 [DZJDXK, DNIDZJ, NI, DXJDZI, AE] = triashap(ZETA,XN,YN);

elseif NNODES == 4            % Rectangle
 DNIDZJ = zeros(4,2);
 DZJDXK = zeros(2,2);
 DXJDZI = zeros(2,2);
 NI     = zeros(4,1);
 DNIDXK = zeros(4,2);
 ZIP    = zeros(4,1);
 [DZJDXK, DNIDZJ, NI, DXJDZI, AE] = brikshap(ZETA,XN,YN);
end

% calculate a few more variables of convenience
 DNIDXK = DNIDZJ*DZJDXK;

%%%%%%%%%%%%%%%%%%%%%%%%
[m,n]  = size(AV);
DNDXAVLI = zeros(m,n);
DNDYBVLI = zeros(m,n);

% form TAU*AV*LVI and TAU*BV*LVI
 for i = 1:NNODES
  DNDXAVLI(i,:)=TAU(i)*AVLI(i,:);
  DNDYBVLI(i,:)=TAU(i)*BVLI(i,:);
 end

% form TAU*DN/DX*AV*LVI and TAU*DN/DY*BV*LVI
DNDXAVLI=reshape(DNDXAVLI,4*m,4) .* kron(ones(4,4),DNIDXK(:,1)); %
multiply
DNDYBVLI=reshape(DNDYBVLI,4*m,4) .* kron(ones(4,4),DNIDXK(:,2)); %
multiply

AVTRANS = zeros(m,n);
BVTRANS = zeros(m,n);
NAV = zeros(m,n);
NBV = zeros(m,n);
DNDXAV = zeros(m,n);
DNDYBV = zeros(m,n);

% transpose matrix blocks
for i = 1:NNODES
  AVTRANS(i,:) = reshape(reshape(AV(i,:),4,4)',1,16);          %
transpose AV
```

```
  BVTRANS(i,:) = reshape(reshape(BV(i,:),4,4)',1,16);                    %
transpose BV
  QV(i,:)      = reshape(reshape(QV(i,:),4,4)',1,16);                    %
transpose QV
  LV(i,:)      = reshape(reshape(LV(i,:),4,4)',1,16);                    %
transpose LV
end

% form interpolated NI*AV and DN/DX*AV
NAV = reshape(AVTRANS,4*m,4)' .* kron(ones(4,4),NI');                    %
transpose & multiply
DNDXAV = reshape(AVTRANS,4*m,4)' .* kron(ones(4,4),DNIDXK(:,1)');        %
transpose & multiply

% form interpolated NI*AV and DN/DY*BV
NBV = reshape(BVTRANS,4*m,4)' .* kron(ones(4,4),NI');                    %
transpose & multiply
DNDYBV = reshape(BVTRANS,4*m,4)' .* kron(ones(4,4),DNIDXK(:,2)');        %
transpose & multiply

% form interpolated NI*QV and NI*LV
QV     = reshape(QV,4*m,4)' .* kron(ones(4,4),NI');                      %
transpose & multiply
LV     = reshape(LV,4*m,4)' .* kron(ones(4,4),NI');                      %
transpose & multiply

% terms used for grid motion
NQC = [kron(NI,ones(1,DOFNODE)).*QC]';                                   %
multiply and transpose
DNDXQC = [kron(DNIDXK(:,1),ones(1,DOFNODE)).*QC]';                       %
multiply and transpose
DNDYQC = [kron(DNIDXK(:,2),ones(1,DOFNODE)).*QC]';                       %
multiply and transpose

%%%%%%%%%%%%%%%%%%

% form BIG matrix interpolation functions
BIG_N = kron(eye(4),NI');                  % transpose later
BIG_DNDX = kron(eye(4),DNIDXK(:,1)');      % transpose later
BIG_DNDY = kron(eye(4),DNIDXK(:,2)');      % transpose later
SUPG = (DNDXAVLI+DNDYBVLI)';               % transpose later
TEST = BIG_N + SUPG;                       % transpose later

BIG_DXN = BIG_DNDX' * BIG_N;               % dN/dX * N'
BIG_DYN = BIG_DNDY' * BIG_N;               % dN/dY * N'
BIG_DXNAV = BIG_DNDX'* NAV;                % dN/dX * NAV
BIG_DYNBV = BIG_DNDY'* NBV;                % dN/dY * NBV

% LHS and RHS physical time and shock capture terms
% need definition for DELTA before putting shock capture back in
if DEL_T == 0                                                % STEADY STATE
version
%  QP_MAT = ...                                              % 1993 Tezduyar
version
%  kron(DELTA*eye(4),DNIDXK(:,1)*DNIDXK(:,1)') + ...  % QP shock
capture
```

```
%  kron(DELTA*eye(4),DNIDXK(:,2)*DNIDXK(:,2)');      % QP shock
capture
  QC_MAT = zeros(NDOF);
  QP_MAT = zeros(NDOF);                              % drop shock
capture for now
%  delta = mean(TAU)/100;
  delta=0;
  QP_MAT = delta*(BIG_DNDX' * BIG_DNDX + BIG_DNDY' * BIG_DNDY);

else                          %needs work            % TRANSIENT
version
  QP_MAT = ...                                             % 1993
Tezduyar version
  (3/(2*DEL_T)) * kron(QV,NI) + ...                        % QP
physical time
  (3/(2*DEL_T)) * kron(AV_TAU*QV,DNIDXK(:,1)) + ...        % QP
physical time
  (3/(2*DEL_T)) * kron(BV_TAU*QV,DNIDXK(:,2)) + ...        % QP
physical time
  kron(DELTA*eye(4),DNIDXK(:,1)*DNIDXK(:,1)') + ...        % QP shock
capture
  kron(DELTA*eye(4),DNIDXK(:,2)*DNIDXK(:,2)');             % QP shock
capture
  QC_MAT = ...
  kron(eye(4),NI*NI') + ...                          % QCN and QCN-1
physical time
  kron(AV_TAU,DNIDXK(:,1)*NI') + ...                 % QCN and QCN-1
physical time
  kron(BV_TAU,DNIDXK(:,2)*NI');                      % QCN and QCN-1
physical time
end

% LHS and RHS pseudo time terms
QPV_MAT = zeros(NDOF);                                     % No pseudo
time terms
if DEL_TAU ~= 0
  %QPV_MAT = TEST' * (LV / DEL_TAU);                       % All test
function terms
  %QPV_MAT = BIG_N' * (QV / DEL_TAU);                      % QV mass
matrix
  QPV_MAT = BIG_N' * (LV / DEL_TAU);                       % No SUPG
terms
  QPV_MAT=diag(sum(abs(QPV_MAT')));                        % make it
diagonal
end

% RHS use only (interpolate nodal flux functions)

F_MAT =  -BIG_DXN + SUPG'*BIG_DNDX;              % F x flux
G_MAT =  -BIG_DYN + SUPG'*BIG_DNDY;              % G y flux

% LHS use only - form and sum element integration matrices
QP_LHS =  QPV_MAT - BIG_DXNAV - BIG_DYNBV ...
                        + SUPG'*(DNDXAV+DNDYBV)+ QP_MAT;

% LHS use only - form and sum grid motion matrices
```

```
GRID_MAT = [BIG_DNDX'*NQC - SUPG'*DNDXQC, BIG_DNDY'*NQC -
SUPG'*DNDYQC];

% reshape matrices to flat array
MFLAT = zeros(7,MMM);
MFLAT(1:6,1:MMM) = [reshape(QP_MAT,1,MMM); ...
                    reshape(QC_MAT,1,MMM); ...
                    reshape(QPV_MAT,1,MMM); ...
                    reshape(F_MAT,1,MMM); ...
                    reshape(G_MAT,1,MMM); ...
                    reshape(QP_LHS,1,MMM)]*AE;
MFLAT(7,1:GGG) =    reshape(GRID_MAT,1,GGG)*AE;
```

BRIKSHAP

```
function [DZIDXJ, DNIDZJ, NI, DXJDZI, AE]=brikshap(ZETA,XN,YN)
%  function [DZIDXJ, DNIDZJ, NI, DXJDZI, AE]=brikshap(ZETA,XN,YN)
%
%  Evaluates various aspects of the LINEAR QUADRALATERAL shape function
%
%  Calculates the 2x2 GLOBAL to LOCAL jacobian DZIDXJ      (dZi / dXj)
%  Calculates the 4x2 local shape derivative matrix DNIDZJ (dNi / dZj)
%  Calculates the 4x1 local shape function                 (Ni)
%  Calculates the 2x2 LOCAL to GLOBAL jacobian DXJDZI      (dXj / dZi)
%  Calculates the scalar GLOBAL to LOCAL area ratio of the element (AE)
%
%  ZETA is a row vector of the current position in local coordinates
%  XN is a column vector of the global X position of the element nodes
%  YN is a column vector of the global Y position of the element nodes

% initialize space
DZIDXJ = zeros(2,2);
DXJDZI = zeros(2,2);
NI     = zeros(4,1);
DNIDZJ = zeros(4,2);

DNDZ1 = zeros(4,1);
DNDZ2 = zeros(4,1);

% create some common terms
MZ = ones(1,2) - ZETA;
PZ = ones(1,2) + ZETA;

% local shape matrix
NI = [MZ(1)*MZ(2); PZ(1)*MZ(2); PZ(1)*PZ(2); MZ(1)*PZ(2)]/4;

% local shape derivative matrix
DNIDZJ(:,1) = [-MZ(2), MZ(2), PZ(2), -PZ(2)]'/4;
DNIDZJ(:,2) = [-MZ(1), -PZ(1), PZ(1), MZ(1)]'/4;

% form LOCAL to GLOBAL jacobian matrix
DXJDZI = [XN'*DNIDZJ;
```

```
          YN'*DNIDZJ];

% form GLOBAL to LOCAL jacobian matrix using adjoint/det formula
AE = det(DXJDZI);
DZIDXJ = [DXJDZI(2,2), -DXJDZI(1,2);
          -DXJDZI(2,1), DXJDZI(1,1)]/AE;
AE = abs(AE);
```

TRIASHAP

```
function [DZIDXJ, DNIDZJ, NI, DXJDZI, AE]=triashap(ZETA,XN,YN)
%  function [DZIDXJ, DNIDZJ, NI, DXJDZI, AE]=triashap(ZETA,XN,YN)
%
%  Evaluates various aspects of the LINEAR TRIANGLE shape function
%  using natural (area) coordinates
%
%  Calculates the 3x2 GLOBAL to LOCAL jacobian DZIDXJ      (dZi / dXj)
%  Calculates the 3x3 local shape derivative matrix DNIDZJ (dNi / dZj)
%  Calculates the 3x1 local shape function                 (Ni)
%  Calculates the 2x3 LOCAL to GLOBAL jacobian DXJDZI      (dXj / dZi)
%  Calculates the scalar GLOBAL to LOCAL area ratio of the element (AE)
%
%  ZETA is a vector of the position in local coordinates
%  XN is a vector of the global X position of the element nodes
%  YN is a vector of the global Y position of the element nodes

% initialize space
DZIDXJ = zeros(3,2);
DXJDZI = zeros(3,3);
NI     = zeros(3,1);

% local shape matrix
NI = ZETA';

% local shape derivative matrix
DNIDZJ = eye(3,3);

% form LOCAL to GLOBAL jacobian matrix
DXJDZI = [XN';YN';ones(1,3)];
TWO_AE = det(DXJDZI);
AE     = abs(TWO_AE/2);
DXJDZI = DXJDZI(1:2,:);

% form GLOBAL to LOCAL jacobian matrix
DZIDXJ = [YN(2)-YN(3), XN(3)-XN(2);
          YN(3)-YN(1), XN(1)-XN(3);
          YN(1)-YN(2), XN(2)-XN(1)]/TWO_AE;
```

FLOW_LHS

```
function R =
flow_lhs(QP,QPM1,XN,YN,WX,WY,QP1,QP2,DEL_T,DEL_TAU,MP,ELMAT,BCTYPE,BC1,
MOUT_PTR,DOFNODE,NN)
% function R =
flow_lhs(QP,QPM1,XN,YN,WX,WY,QP1,QP2,DEL_T,DEL_TAU,MP,ELMAT,BCTYPE,BC1,
MOUT_PTR,DOFNODE,NN)
%
% 17 Feb 97, original
%  4 Mar 97, matrix and flux calculations moved inside element loop
%  7 Apr 97, completed characteristic boundary conditions
%  8 Sep 97, updated with new boundary condition routines
%
% Driver routine for LHS fluid flow calculations.
%
% QP  =      vector of primitive nodal variables [P; U; V; T]
% XN =       vector of current X locations of nodes
% YN =       vector of current Y locations of nodes
% WX =       vector of X velocity of nodes
% WY =       vector of Y velocity of nodes
% QP1 =      vector of primitive nodal variables [P; U; V; T]
% QP2 =      vector of primitive nodal variables [P; U; V; T]
% DEL_T =    physical time step size
% DEL_TAU = pseudo time step size
% MP =       thermodynamic properties of fluid, MP = [R, CP, GAMMA]
% ELMAT =    matrix listing the nodes assigned to each element
% BCTYPE =   vector of nodal definition of boundary condition type
%            not necessarily the most efficient implementation but it
forces
%            a single definition of boundary state for nodes
%            type 0 = not on a boundary
%            type 1 = inflow/outflow
%            type 2 = impermeable wall
% BC1     =  matrix of type 1 boundary conditions
%            [node number, P, U, V, T]
% DOFNODE = degrees of freedom per node (currently 4)
% NN =       order of gauss integration (1,2, or 3)
%
% global variables
% MFLOW =    flat file containing summation of element integration
matices for LHS
% MOUT  =    matrix of nodal outward normal vectors and boundary type
information
%            [node number, dx, dy, type (0 1 2 3 4 5), index into BC1,
c, rho]
%            BC conditions 1=supersonic inflow,  2=subsonic inflow
%                          3=supersonic outflow, 4=subsonic outflow
%                          5=impermeable wall
% GFLOW =    flat file containing summation of element integration
matrices for LHS
%            grid motion in pseudo time

global MFLOW MOUT GFLOW
```

```
% initialize
NNODES = length(XN);          % number of nodes
[NEL,VERTICES] = size(ELMAT); % VERTICES is the max number of nodes in
any element

R = zeros(NNODES*DOFNODE,1);

% reformat into a matrix shape with columns of primitive variables
QP = reshape(QP,NNODES,DOFNODE);

% Safeguard against negative pressure or temperature
QP(:,1)=max(1,abs(QP(:,1)));
QP(:,4)=max(1,abs(QP(:,4)));

% initialize maximum element workspace
PVEC = zeros(VERTICES,1);     % element nodal pressure vector
UVEC = zeros(VERTICES,1);     % element nodal X velocity vector
VVEC = zeros(VERTICES,1);     % element nodal Y velocity vector
TVEC = zeros(VERTICES,1);     % element nodal temperature vector
WXVEC = zeros(VERTICES,1);    % element nodal X grid velocity
WYVEC = zeros(VERTICES,1);    % element nodal Y grid velocity

QP_MAT = zeros(VERTICES*DOFNODE);              % element integration
matrix
GRID_MAT = zeros(VERTICES*DOFNODE,VERTICES*2); % grid motion matrix

%************************************ BOUNDARY CONDITION ROUTINE ****
% enforce BOUNDARY CONDITIONS using RHS definitions of inflow/outflow
% and tangent vectors
%*********************************************************************

 QP = bc1_lhs(QP,BC1);

% Form the residual vector using local vector operations and an element
by
% element approach

for j = 1:NEL

% check the shape of each element and set up its needed workspace size

  INDEX=find(ELMAT(j,:));  % INDEX contains the non-zero node numbers
for the element
  VERTICES = length(INDEX);

  if VERTICES == 3            % Triangle
    REGION = 2;
    LLL = 12;
    MMM = 144;
    GLL = 2*3;
    GGG = LLL*GLL;
  elseif VERTICES == 4        % Quadralateral
    REGION = 1;
    LLL = 16;
    MMM = 256;
```

```
    GLL = 2*4;
    GGG = LLL*GLL;
  end

  PVEC = QP(ELMAT(j,INDEX),1);            % current element nodal
pressure vector
  UVEC = QP(ELMAT(j,INDEX),2);            % current element nodal X
velocity vector
  VVEC = QP(ELMAT(j,INDEX),3);            % current element nodal Y
velocity vector
  TVEC = QP(ELMAT(j,INDEX),4);            % current element nodal
temperature vector
  WXVEC = WX(ELMAT(j,INDEX),1);           % current element nodal X
grid velocity
  WYVEC = WY(ELMAT(j,INDEX),1);           % current element nodal Y
grid velocity

  QP_MAT = reshape(MFLOW(j,1:MMM),LLL,LLL);     % matrix for current Q
primitive
  GRID_MAT = reshape(GFLOW(j,1:GGG),LLL,GLL);   % matrix for grid
motion
  SHUFFLE=[ELMAT(j,INDEX),ELMAT(j,INDEX)+NNODES,...
           ELMAT(j,INDEX)+2*NNODES,ELMAT(j,INDEX)+3*NNODES];

% form LHS residual for fluid flow equations

  R(SHUFFLE)= R(SHUFFLE) + QP_MAT*[PVEC; UVEC; VVEC; TVEC] + ...
              GRID_MAT*[WXVEC; WYVEC];
end

%************************************ BOUNDARY CONDITION ROUTINE ****
% Update fully assembled residual to reflect Dirichlet boundary
% conditions.  Routine is common to RHS and LHS.
% Relies on action of DIRDER to form delta terms for boundary
conditions
%**********************************************************************

R = bc32_rhs(R,QP,BC1,NNODES,DOFNODE,1);

% apply residual scaling
global FLUID_SCALE
temp=length(R)/4;
R(1:temp)=R(1:temp)/FLUID_SCALE(1);
R(temp+1:2*temp)=R(temp+1:2*temp)/FLUID_SCALE(2);
R(2*temp+1:3*temp)=R(2*temp+1:3*temp)/FLUID_SCALE(3);
R(3*temp+1:4*temp)=R(3*temp+1:4*temp)/FLUID_SCALE(4);

% add emphasis to boundary residuals
if FLUID_SCALE(5)==1
  INDEX=find(BCTYPE>0);
  R=reshape(R,NNODES,4);
  R(INDEX,:)=2*R(INDEX,:);    % double all boundary residuals
  INDEX=find(BCTYPE>20);
  R(INDEX,:)=2*R(INDEX,:);    % double corner residuals again
  R=reshape(R,NNODES*4,1);
end
```

BC1_LHS

```
function QP = bc1_lhs(QP,BC1)
%
% function QP = bc1_lhs(QP,BC1)
%
% 8 Sep 97, original
%
% Enforces Dirichlet and impermeable wall BOUNDARY CONDITIONS on LHS
residual
% using RHS definitions of inflow/outflow and tangent vectors
%
% Variables:
% QP  =     matrix of primitive nodal variables [P, U, V, T]
% BC1    =  matrix of type 1 and type 6 boundary conditions
%           [node number, P, U, V, T]
%
% global variables
% MOUT  =   matrix of nodal RH rule tangent vectors and boundary type
information
%           [node number, dx, dy, type (1 - 54), index into BC1, DX,
DY]
%           BC conditions 1=supersonic inflow,  2=subsonic inflow
%                         3=supersonic outflow, 4=subsonic outflow
%                         5=impermeable wall,
%                         51=wall+super in,     52=wall+sub in
%                         53=wall+super out,    54=wall+sub out
%           [dx,dy] = wall tangent and in/out tangent
%           [DX,DY] = in/out tangent if [dx,dy] has wall tangent for a
corner,
%                     otherwise original direction of velocity vector

global MOUT

NBC=length(MOUT(:,1));
for j = 1:NBC

  if (MOUT(j,4) == 5)          % apply TYPE 2 BOUNDARY CONDITIONS
(impermeable wall)
%     TAN_VEC=MOUT(j,2:3);                          % nodal tangent vector
%     UV = QP(MOUT(j,1),2:3)';                      % [U, V] for node
%     QP(MOUT(j,1),2:3) = sign(TAN_VEC*UV)*norm(UV) * TAN_VEC;      %
take tangent vector component of [U,V]
%     QP(MOUT(j,1),2:3)= UV'-(TAN_VEC*UV) * TAN_VEC;                %
normal component of velocity at wall

                              % apply TYPE 1 BOUNDARY CONDITIONS
(characteristic inflow/outflow)
  elseif MOUT(j,4) == 1                         % supersonic relative
inflow (1)
    QP(MOUT(j,1),:) = BC1(MOUT(j,5),2:5);       % set all boundary
values
```

```
  elseif MOUT(j,4) == 51                          % supersonic relative
inflow (1) w/ wall
    QP(MOUT(j,1),:) = BC1(MOUT(j,5),2:5);         % set all boundary
values
%     TAN_VEC=MOUT(j,2:3);                        % nodal tangent vector
%     UV = QP(MOUT(j,1),2:3)';                    % [U, V] for node
%     QP(MOUT(j,1),2:3) = sign(TAN_VEC*UV)*norm(UV) * TAN_VEC;     %
take tangent vector component of [U,V]
%     QP(MOUT(j,1),2:3)= UV'-(TAN_VEC*UV) * TAN_VEC;               %
normal component of velocity at wall

  elseif MOUT(j,4) == 2                         % subsonic relative inflow
(2)
    QP(MOUT(j,1),2:4) = BC1(MOUT(j,5),3:5);     % set [U,V,T] boundary
values

  elseif MOUT(j,4) == 52                        % subsonic relative
inflow (2) w/ wall
    QP(MOUT(j,1),2:4) = BC1(MOUT(j,5),3:5);     % set [U,V,T] boundary
values
%     TAN_VEC=MOUT(j,2:3);                        % nodal tangent vector
%     UV = QP(MOUT(j,1),2:3)';                  % [U, V] for node
%     QP(MOUT(j,1),2:3) = sign(TAN_VEC*UV)*norm(UV) * TAN_VEC;     %
take tangent vector component of [U,V]
%     QP(MOUT(j,1),2:3)= UV'-(TAN_VEC*UV) * TAN_VEC;               %
normal component of velocity at wall

  elseif MOUT(j,4) == 3                         % supersonic relative outflow
(3)
                                                % set no boundary
values

  elseif MOUT(j,4) == 53                        % supersonic relative
outflow (3) w/ wall
                                                % set no boundary
values

%     TAN_VEC=MOUT(j,2:3);                        % nodal tangent vector
%     UV = QP(MOUT(j,1),2:3)';                    % [U, V] for node
%     QP(MOUT(j,1),2:3) = sign(TAN_VEC*UV)*norm(UV) * TAN_VEC;     %
take tangent vector component of [U,V]
%     QP(MOUT(j,1),2:3)= UV'-(TAN_VEC*UV) * TAN_VEC;               %
normal component of velocity at wall

  elseif MOUT(j,4) == 4                         % subsonic relative outflow
(4)
    QP(MOUT(j,1),1) = BC1(MOUT(j,5),2);         % set [P] boundary
value

  elseif MOUT(j,4) == 54                        % subsonic relative
outflow (4) w/ wall
    QP(MOUT(j,1),1) = BC1(MOUT(j,5),2);         % set [P] boundary
value
%     TAN_VEC=MOUT(j,2:3);                        % nodal tangent vector
%     UV = QP(MOUT(j,1),2:3)';                    % [U, V] for node
```

```
%     QP(MOUT(j,1),2:3) = sign(TAN_VEC*UV)*norm(UV) * TAN_VEC;      %
take tangent vector component of [U,V]
%     QP(MOUT(j,1),2:3)= UV'-(TAN_VEC*UV) * TAN_VEC;                %
normal component of velocity at wall
  end
end
```

APPENDIX F

SOLID ROUTINES

APPENDIX F

SOLID ROUTINES

The structural solver (solid) routines implement the linear and nonlinear stiffness algorithms outlined in Chapters 7 and 14 for a two-dimensional continuum solid. They include the implicit Adaptive Dynamic Relaxation (ADR) methods derived in Chapters 8 and 15. The coding is original and was developed in the MATLAB environment.

The code structure closely follows the architecture illustrated in Figures C-1 through C-3 and Figure D-2. Figure F-1 illustrates the specific names and relationships of the scripts and subroutines. The shaded blocks designate algorithms specific to the structural solver while unshaded blocks show generic routines that did not change. Major elements the right and left hand side residual algorithms are further illustrated in Figures F-2 through F-3.

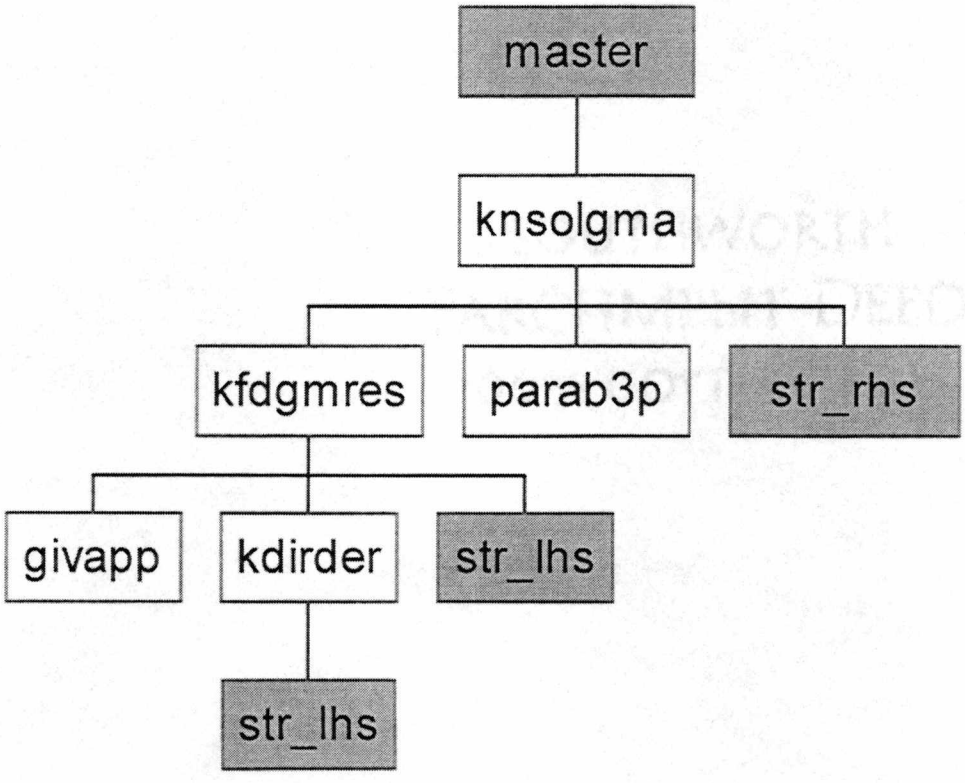


Figure F-1: Structural Solver Master Hierarchy

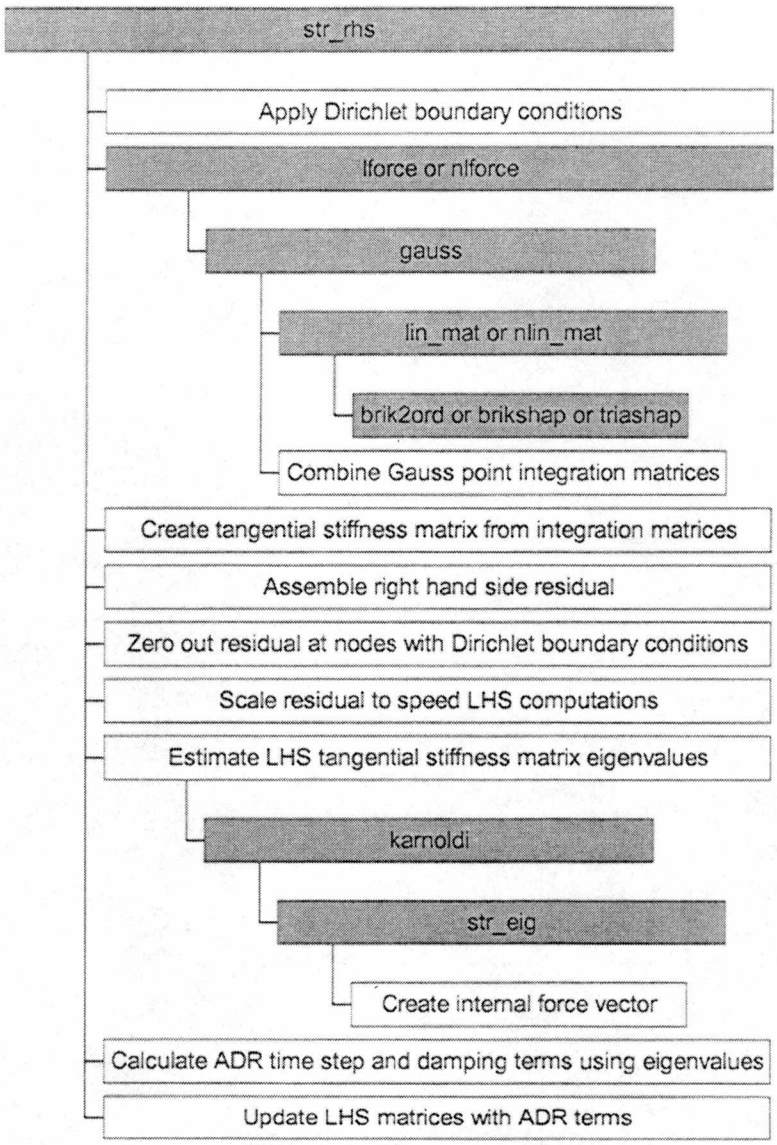


Figure F-2: Right Hand Side Structural Deformation Residual

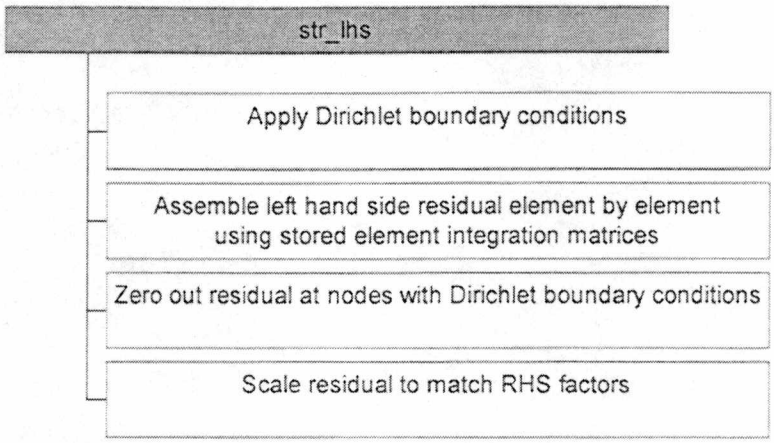


Figure F-3: Left Hand Side Structural Deformation Residual

The shaded blocks in Figures F-2 and F-3 indicate where a specific subroutine is called to perform a function. The unshaded blocks show a task to be performed in the parent block.

The right hand side residual statement in Figure F-2 follows the same general logic as outlined Figure C-2. It has an option to calculate the internal force vector using either a linear or nonlinear stiffness matrix. Large (nonlinear) deformations are possible even with a linear stiffness. At the lowest level of indenture, the option is available to use biquadratric, bilinear, or triangular elements. The right hand side residual routine adds an additional step to estimate the extreme eigenvalues in the tangent stiffness matrix using the Arnoldi method described in Chapter 8 and Appendix D. The left hand side residual statement in Figure F-3 follows the generic logic in Figure C-3 exactly.

script master - Master routine for single domain continuum solid deformation calculations

function str_rhs
- Combines different routines to form the static or dynamic structural equations
- Output is the residual (want R = 0 at equilibrium)
- Damping is selected to provide proportional mass damping

function lforce - Calculates the internal forces at the nodes based on linear elastic material properties

function nlforce
- Calculates the LHS and RHS stiffness matrices based on non-linear deformations and linear elastic material properties with consistent or lumped mass matrices
- Mass matrices may be used with physical and/or pseudo time

function lin_mat
- Forms the tensor function Transpose(B)*B for the linear stiffness matrix using linear quadralateral (brick) or triangle shape functions

- Similar to the function (dNi/dXj) * transpose(dNi/dXj) * Area but also forms the cross elements needed in the stiffness calculations
- Forms the pseudo time mass matrix for LHS operations and breaks pseudo time motion into a first order system of equations in DX, DY, VX, and VY
- Intended for use in gauss quadrature

function nlin_mat
- Calculates the matrix for forces at the nodes based on non-linear deformations and linear elastic material properties
- Intended for use in gauss quadrature

function brik2ord - Evaluates various aspects of the serendipity biquadratic shape function

function str_eig - Combines different routines to quickly form the static structural equations for use in estimating the eigenvalues of the tangent stiffness matrix

function str_lhs
- Combines different routines to form the residual of the incremental static or dynamic structural equations
- Treats time and pseudo-time as a system of first order differential equations

MASTER

```
% Master script program
%
% last modified 22 Dec 96 to include MB200 as a global value
%
% initialize variables
% XN, YN = original X and Y location
% ELMAT  = nodes assigned to elements
% MP     = material properties
% NN     = Gauss integration order
% DOFNODE= number of degrees of freedom per node
% PTX,PTY= external nodal loads in X and Y direction
% BCX,BCY= displacement boundary conditions in X and Y directions
% DX, DY = initial displacement vectors in X and Y directions
%
% load rod   % single element in tension
% load bar   % long and skinny in tension
% load beam  % long and skinny with side load
%
% eval(['load ',name]);

global STIFF
eval(['load ',name]);
DOFNODE = 4;
%DOFNODE = 2;     %xxxx

% set some Newton (outer) and GMRES (inner) iteration parameters
TOL=[1E-6,1E-6];        % [absolute error tolerance, relative error
tolerance]
[MX,NX] = size(BCX);
```

```
[MY,NY] = size(BCY);
NNODES = length(XN);

% Max number of GMRES iterations min(384, rank)
LIN_ITS = min(96,DOFNODE*NNODES-MX-MY);

%PARMS=[20,LIN_ITS,.9]; % [max Newton steps, max GMRES its/Newton step,
etamax]
% [max Newton steps, max linear iterations, etamax, linear soln method,
max restarts]
% 10 Newton steps
% 40 GMRES iterations per try
% .5 etamax
% 2 = Restarted GMRES(LIN_ITS)
% 4 = max number of GMRES restarts -> linear its max/Newton step
%PARMS=[10,LIN_ITS,.5,2,8];  %xxxx
%PARMS=[10,40,.5,2,8];
%PARMS=[10,LIN_ITS,.5,1,0];
PARMS=[80,LIN_ITS,.5,1,0];
%PARMS=[80,2,.5,1,0];

T_TOL=[1E-9,1E-9,2];  % [absolute time error tolerance, relative error
tolerance,DOF per node]
T_STEP=5;              % max time steps
                       % 1=linear, 2=nonlinear
LIN = [1,0; 1,0; 1,0; 1,0; 1,0; 1,0; 1,0; 1,0; 1,0; 1,0];
DEL_T = 0;               % physical time step size
DEL_TAU = 1;             % pseudo time step size
%DEL_TAU = 0;    %xxxx
% DEL_T = (5.73E-1)/3;

%LIN(:,1) = 2*LIN(:,1);                % non-linear

T_HISTORY=zeros(DOFNODE*NNODES,T_STEP+1);

%load temp6 % DX DY
 DX = zeros(size(XN));
 DY = zeros(size(YN));
%DX=DXI;
%DY=DYI;
 DX(BCX(:,1))=BCX(:,2);
 DY(BCY(:,1))=BCY(:,2);

 if DOFNODE == 4
   VX = zeros(size(XN));
   VY = zeros(size(YN));
   VX(BCX(:,1))=BCX(:,3);
   VY(BCY(:,1))=BCY(:,3);
 end

flops(0);
tic

%  Initialize matrices for element by element storage
[NEL,LARGEST]=size(ELMAT);
```

```
% STIFF = [c*M/del_tau+KO+KL+KS , M/del_tau]
%         [I/del_tau            , -I       ]
%STIFF=zeros(NEL,(4*LARGEST)^2);
STIFF=zeros(NEL,(DOFNODE*LARGEST)^2);

% decide how many load increments to use
NL = 1;
SCALE=ones(NL,1);
for i = 1:NL
  SCALE(i)=i/NL;
end

FNRM=1;
STOP_TOL=T_TOL(1) + T_TOL(2)*FNRM;
ITC = 0;

if DOFNODE == 4
  SOLN = [DX;DY;VX;VY];
else
  SOLN = [DX;DY];
end

T_STEP=NL;                    % match to NL for load increments

while(FNRM > STOP_TOL & ITC < T_STEP)
  ITC=ITC+1;
  NODE_VEC = SOLN;

  figure(1)
  clg
  set(1,'Position',[2 30 447 340]);
  figure(2)
  clg
  set(2,'Position',[456 414 371 310]);
  figure(3)
  clg
  set(3,'Position',[2 415 447 308]);
  figure(4)
  clg
  set(4,'Position',[456 30 370 339]);

  [SOLN, HISTORY, IERROR, STATS] =
knsolgma('STR_LHS','STR_RHS',T_TOL,PARMS,NODE_VEC,...

NODE_VEC,XN,YN,ELMAT,PTX,PTY,SCALE(ITC),BCX,BCY,MP,DEL_T,DEL_TAU, ...
                      NN,DOFNODE,LIN(ITC,:));
%  [SOLN, HISTORY, IERROR, STATS] =
knsolgma('STR_RHS','STR_RHS',T_TOL,PARMS,NODE_VEC,...
%
NODE_VEC,XN,YN,ELMAT,PTX,PTY,SCALE(ITC),BCX,BCY,MP,DEL_T,DEL_TAU, ...
%                       NN,DOFNODE,LIN(ITC,:));

disp([STATS(:,1),zeros(size(STATS(:,2))),STATS(:,3:5)])
format long
disp(STATS(:,2))
format short
```

```
FNRM = norm(SOLN-NODE_VEC)/sqrt(length(SOLN))

% put solution SOLN back into input for next step
T_HISTORY(:,ITC+1)=SOLN;

end

DX = SOLN(1:NNODES);
DY = SOLN(NNODES+1:2*NNODES);
if DOFNODE == 4
  VX = SOLN(2*NNODES+1:3*NNODES);
  VY = SOLN(3*NNODES+1:4*NNODES);
end

toc
flops
figure(1)
semilogy(HISTORY(:,1))
```

STR_RHS

```
function R =
str_rhs(NODE_VEC,dummy,XN,YN,ELMAT,PTX,PTY,SCALE,BCX,BCY,MP,DEL_T,DEL_T
AU,NN, ...
                    DOFNODE,LIN,XM1,VM1)

% function R =
str_rhs(NODE_VEC,dummy,XN,YN,ELMAT,PTX,PTY,SCALE,BCX,BCY,MP,DEL_T,DEL_T
AU,NN, ...
%                   DOFNODE,LIN,XM1,VM1)
%
% 15 Dec 96 - created
% 22 Dec 96 - added call to initialize PTIME
% 12 Jan 97 - added parameter to select linear or nonlinear force
% 24 May 98 - changed overall structure to be more consistent with
fluid flow method
% 11 Nov 98 - updated to include 1st order system in time
%
% Combines different routines to form the static or dynamic structural
equations
% Output is the residual (want R = 0 at equilibrium).
%
% c*d/dt (dx;dy) + rho*d/dt (u;v) + K*(dx; dy)                = 0
%   d/dt (dx;dy)                                  -    (u;v)     = 0
%                              or
% [c, rho]*transpose{dx, dy, u, v}+[K, 0]*transpose{dx, dy, u, v}=0
% [I,  0 ]      dt                 [0,-I]
%
% c is selected to provide proportional mass damping, (alpha*rho)
%
% INPUT VARIABLES:
```

```
% NODE_VEC = nodal deformation and velocity vector (DX, DY, U, V)
% dummy = place holder for GMRES solver
% XN = X location of nodes
% YN = Y location of nodes
% ELMAT = matrix listing the nodes assigned to each element
% PTX = external point (nodal) loads in the X direction
% PTY = external point (nodal) loads in the Y direction
% SCALE = scaling factor used to gradually apply loads
% BCX = X deformation (and velocity) Dirichlet boundary conditions
[node #, X, VX]
% BCY = Y deformation (and velocity) Dirichlet boundary conditions
[node #, Y, VY]
% MP = material properties (E...lbf/sqft, Nu, density...lbm/cuft,
damping)
% DEL_T = physical time step size
% DEL_TAU = pseudo time step size
% NN = order of gauss integration
% DOFNODE = number of degrees of freedom per node
% LIN = 2 element vector of calculation choices
%       term 1: linear or non-linear selection, 1 = linear, 2 =
nonlinear
%       term 2: not used
% XM1 = [DX;DY] at t-1
% VM1 = [VX;VY] at t-1
%
% OUTPUT VARIABLES:
% R = residual of summation of external and internal forces
%

global STIFF

% initialize variables and workspace
NNODES = length(XN);
R = zeros(size(NODE_VEC));
F_APPLIED = zeros(size(NODE_VEC));
[NEL,VERTICES] = size(ELMAT); % VERTICES is the max number of nodes in
any element
NDOF = VERTICES * DOFNODE;

if VERTICES == 3              % triangle
  FLAT    = zeros(4,12^2);
  MASS_MAT = zeros(12);       % physical time consistent mass matrix
  STIFF_MAT = zeros(12);      % internal force vector matrix
(BBAR'*STRESS)

elseif VERTICES == 4          % rectangle
  FLAT    = zeros(4,16^2);
  MASS_MAT = zeros(16);
  STIFF_MAT = zeros(16);
elseif VERTICES == 8          % rectangle
  FLAT    = zeros(4,16^2);
  MASS_MAT = zeros(16);
  STIFF_MAT = zeros(16);
end

%************************************ BOUNDARY CONDITION ROUTINE 1 **
```

```
% Form deformation vectors and apply dirichlet boundary conditions
%*********************************************************************
DX = NODE_VEC(1:NNODES);
DY = NODE_VEC(NNODES+1:2*NNODES);
DX(BCX(:,1))=BCX(:,2);
DY(BCY(:,1))=BCY(:,2);

if DOFNODE == 4
  VX = NODE_VEC(2*NNODES+1:3*NNODES);
  VY = NODE_VEC(3*NNODES+1:4*NNODES);
  VX(BCX(:,1))=zeros(size(BCX(:,2)));     % zero out dirichlet boundary
velocity
  VY(BCY(:,1))=zeros(size(BCY(:,2)));     % zero out dirichlet boundary
velocity
end

if DEL_T ~= 0
  DX1 = XM1(1:NNODES);
  DY1 = XM1(NNODES+1:2*NNODES);
  DX1(BCX(:,1))=BCX(:,2);
  DY1(BCY(:,1))=BCY(:,2);
  if DOFNODE == 4
    VX1 = VM1(1:NNODES);
    VY1 = VM1(NNODES+1:2*NNODES);
    VX1(BCX(:,1))=BCX(:,3);
    VY1(BCY(:,1))=BCY(:,3);
  end
end

% scale applied nodal loads and put into a single vector

F_APPLIED(1:2*NNODES) = SCALE * [PTX; PTY];
R = -F_APPLIED;

% plot deformed structure
figure(3)
clg;
hold on
maxx=max([XN;YN]);
minx=min([XN;YN]);
maxdx=max([abs(DX);abs(DY)]);
if maxdx == 0
  maxdx = 1;
end
amp=.1*(maxx-minx)/maxdx;
amp=1;
plot(XN+amp*DX,YN+amp*DY,'y+')
axis('equal')
grid on
drawnow

%keyboard
%***************************** ELEMENT CALCULATIONS
**********************
% Form the RESIDUAL vector using local vector operations and an element
by
```

```
% element approach based on a total Lagrangian formulation
%*****************************************************************
***
for j = 1:NEL

% check the shape of each element and set up its needed workspace size
  INDEX=find(ELMAT(j,:));  % INDEX contains the non-zero node numbers
for the element
  VERTICES = length(INDEX);

  NDOF = VERTICES * DOFNODE;                % this element
  SLIST = ELMAT(j,INDEX);                   % index into variables stored
in data file

% pseudo time matrix terms
  if DEL_TAU ~= 0
%    PSEUDO_MP = 1E5*MP(3);                 %xxxx
    PSEUDO_MP = 1;
  else
    PSEUDO_MP = 0;
  end

% FLAT: flat matrix of element integration arrays for each element
% Row 1 = sum of LHS stiffness matrices for [DX;DY] operations
% Row 2 = consistent mass matrix for physical and pseudo time [DX;DY]
operations
% Row 3 = lumped mass matrix for physical and pseudo time [DX;DY]
operations
% Row 4 = matrix for RHS internal forces for [DX;DY] operations
% first order systems for physical and/or pseudo time will be built
from these matrices

if LIN(1) == 1              % calculate and sum linear forces
 FLAT = lforce(XN(SLIST),YN(SLIST),DX(SLIST),DY(SLIST),...
               MP,PSEUDO_MP,NN,DEL_T,DEL_TAU,2);

elseif LIN(1) == 2          % calculate and sum nonlinear forces
 FLAT = nlforce(XN(SLIST),YN(SLIST),DX(SLIST),DY(SLIST),...
               MP,PSEUDO_MP,NN,DEL_T,DEL_TAU,2);
end

% RHS use only
% internal force matrix = BBAR'*STRESS_MAT
% LHS use only
% KT = K0+KL+KS = tangential stiffness matrix (a.k.a. total stiffness
matrix)

% LHS tangent stiffness matrix (K0+KL+KS)
  LHS_STIFF = reshape(FLAT(1,1:(2*VERTICES)^2),2*VERTICES,2*VERTICES);
% consistent mass matrix [NI*NI',0;0,NI*NI']
  C_MASS    = reshape(FLAT(2,1:(2*VERTICES)^2),2*VERTICES,2*VERTICES);
% lumped mass matrix [NI*NI',0;0,NI*NI']
  L_MASS    = reshape(FLAT(3,1:(2*VERTICES)^2),2*VERTICES,2*VERTICES);
% internal force matrix [BB'*STRESS]
  STIFF_MAT = reshape(FLAT(4,1:(2*VERTICES)^2),2*VERTICES,2*VERTICES);
```

```
  SHUFFLE = [];

  for ii = 1:DOFNODE                                  % Index into
residual vector
      SHUFFLE = [SHUFFLE, SLIST+NNODES*(ii-1)];
  end

  if DEL_T == 0                % form RHS residual for STEADY STATE
structural equations
% NOTE FOR PSEUDO TIME WITHOUT PHYSICAL TIME:
% If STIFF_MAT is of the form [K,0;0,-I], the -I will
% cause nonzero terms in the residual if the velocity is not zero.
% Therefore, do not include any velocity contribution to the RHS
residual.

    R(SHUFFLE(1:2*VERTICES))= R(SHUFFLE(1:2*VERTICES)) ...
                + STIFF_MAT * [DX(SLIST);DY(SLIST)];

  else                          %   1st order time derivative = (Un-Un-
1)/DT
    zippo = zeros(size(STIFF_MAT));
    if length(MP)>3
      DAMP = MP(4);
    else
      DAMP = 0;
    end
    R(SHUFFLE)= R(SHUFFLE) ...
                + [STIFF_MAT, zippo;zippo,-C_MASS] ...
                  * [DX(SLIST);DY(SLIST);VX(SLIST);VY(SLIST)] ...
                + ([DAMP*C_MASS, MP(3)*C_MASS; C_MASS,zippo]/DEL_T) ...
                  * [DX(SLIST)-DX1(SLIST);DY(SLIST)-DY1(SLIST);...
                     VX(SLIST)-VX1(SLIST);VY(SLIST)-VY1(SLIST)];
  end

% set up LHS matrix

  if DEL_T~=0 & DEL_TAU~=0          % physical and pseudo time present
    % mixed physical and pseudo time not tested or developed
theoretically
                                    % pseudo time terms will be added
later
    CDT = C_MASS/DEL_T;
    STIFF(j,:) = reshape([LHS_STIFF+DAMP*CDT, MP(3)*CDT; ...
                          CDT               , -C_MASS
],1,(4*VERTICES)^2);
  elseif DEL_T==0 & DEL_TAU~=0      % pseudo time only
                                    % pseudo time terms will be added
later
    STIFF(j,:) = reshape([LHS_STIFF, C_MASS   ; ...
                          C_MASS   , -C_MASS  ],1,(4*VERTICES)^2);
  elseif DEL_T~=0 & DEL_TAU==0      % physical time only
    CDT = C_MASS/DEL_T;
    STIFF(j,:) = reshape([LHS_STIFF+DAMP*CDT, MP(3)*CDT; ...
                          CDT               , -C_MASS
],1,(4*VERTICES)^2);
  else
```

```
    STIFF(j,:) = FLAT(1,:);          % no time stepping
  end

end

%************************************ BOUNDARY CONDITION ROUTINE 2 **
% Zero out Dirichlet boundary conditions in the residual
%*********************************************************************
[MBX, NBX] = size(BCX);
[MBY, NBY] = size(BCY);
if MBX > 0
  R(BCX(:,1)) = zeros(MBX,1);              % zeros residual for DX
dirichlet BC
  if DOFNODE ~= 2
    R(BCX(:,1)+2*NNODES) = zeros(MBX,1); % zeros residual for VX
dirichlet BC
  end
end
if MBY > 0
  R(BCY(:,1)+NNODES) = zeros(MBY,1);       % zeros residual for DY
dirichlet BC
  if DOFNODE ~= 2
    R(BCY(:,1)+3*NNODES) = zeros(MBY,1);  % zeros residual for VY
dirichlet BC
  end
end

%************************************ SCALING ROUTINE ***************
% Scale the equations to make the associated residuals roughly the same
% order of magnitude for equivalent perturbations.  Greatly aids
% orthogonality in LHS modified Gram Schmidt process
%*********************************************************************
global STRUCT_SCALE

temp=length(R)/DOFNODE;
if DOFNODE == 4
%  STRUCT_SCALE=[1E5;1E5;1E-3;1E-3];
%  STRUCT_SCALE=[1E5,1E5,1E-5,1E-5];
%  STRUCT_SCALE=[1E1;1E1;1E-3;1E-3];
  STRUCT_SCALE=[1E1;1E1;1E-1;1E-1];
  R(2*temp+1:3*temp)=R(2*temp+1:3*temp)/STRUCT_SCALE(3);
  R(3*temp+1:4*temp)=R(3*temp+1:4*temp)/STRUCT_SCALE(4);
else
  STRUCT_SCALE=[1;1];
end

R(1:temp)=R(1:temp)/STRUCT_SCALE(1);
R(temp+1:2*temp)=R(temp+1:2*temp)/STRUCT_SCALE(2);

% plot the scaled residual
%figure(1)
%clg
%plot(RHSR)
%drawnow

global RHSR
```

```
RHSR=R;
norm(R)
%************************************ EIGENVALUE ROUTINE ************
% run 10 or 20 steps of an Arnoldi process to estimate the largest
% and smallest eigenvalues of (KO+KL+KS)/M.  Used to set damping
% coefficient and pseudo-time step for LHS operations.
%*********************************************************************
if DEL_TAU ~= 0

  [maxee,minee,minabs,error,total_iters] = karnoldi('str_eig',[1E-
9,20,1],...

ones(2*NNODES,1)/sqrt(2*NNODES),...

[DX;DY],[],XN,YN,ELMAT,BCX,BCY,...
                                              2,4,PSEUDO_MP);

% determine c and del_tau
  ALPHA=2;
%  C=ALPHA*sqrt(min(minee))          % calculated
%  DEL_TAU=2/sqrt(max(maxee))        % calculated
  C=1E8                                   % hard wired
  DEL_TAU=1                         % hard wired

% divide top right and lower left blocks by del_tau
% add c*upper right block/del_tau to KO+KL+KS

  [NEL,VERTICES] = size(ELMAT);     % VERTICES is the max number of
nodes in any element
  NDOF = VERTICES * DOFNODE;
  TEMP = zeros(NDOF);               % max space needed

  for j = 1:NEL

% check the shape of each element and set up its needed workspace size
    INDEX=find(ELMAT(j,:));  % INDEX contains the non-zero node numbers
for the element
    VERTICES = length(INDEX);
    V2 = 2*VERTICES;
    V4 = 4*VERTICES;
    NDOF = VERTICES * DOFNODE;                  % this element

    TEMP = zeros(NDOF);
    TEMP = reshape(STIFF(j,1:NDOF^2),NDOF,NDOF);      % unwrap element
STIFF matrix
    TEMP(1:V2,1:V2) = TEMP(1:V2,1:V2)...
                    - TEMP(V2+1:V4,V2+1:V4)*C*PSEUDO_MP/DEL_TAU;
      % KT block
    TEMP(1:V2,V2+1:V4) = - TEMP(V2+1:V4,V2+1:V4)*PSEUDO_MP/DEL_TAU;
      % Mass block
    TEMP(V2+1:V4,1:V2) = - TEMP(V2+1:V4,V2+1:V4)/DEL_TAU;
      % Identity block

    STIFF(j,1:NDOF^2) = reshape(TEMP,1,NDOF^2); % rewrap element STIFF
matrix
  end
```

```
end
figure(4)
clg
plot(R);
drawnow;

figure(1)
clg

%%%%%%%%%% form global matrix
%%%%%%
kkk=0;
if kkk==1
KT=zeros(2*NNODES);
for j = 1:NEL
  INDEX=find(ELMAT(j,:));     % INDEX contains the non-zero node
numbers for the element
  VERTICES = length(INDEX);
  NDOF = VERTICES * DOFNODE;  % NDOF for the element
  SLIST = ELMAT(j,INDEX);     % index into variables stored in data
file
  LHS_MAT = reshape(STIFF(j,1:NDOF^2),NDOF,NDOF);     % recall stored
LHS integration matrix
  SHUFFLE = [];
  for ii = 1:DOFNODE                                  % Index into
residual vector
    SHUFFLE = [SHUFFLE, SLIST+NNODES*(ii-1)];
  end
  KT(SHUFFLE,SHUFFLE)=KT(SHUFFLE,SHUFFLE)+LHS_MAT;
end
KT(BCX(:,1),:)=[];
KT(length(XN)+BCY(:,1),:)=[];
KT(:,BCX(:,1))=[];
KT(:,length(XN)+BCY(:,1))=[];
disp('Condition'); cond(KT)
tmp=eig(KT);
min(tmp)
max(tmp)
end
```

LFORCE

```
function FLAT =
lforce(XN,YN,DX,DY,MP,PSEUDO_MP,NN,DEL_T,DEL_TAU,DOFNODE);
%function FLAT =
lforce(XN,YN,DX,DY,MP,PSEUDO_MP,NN,DEL_T,DEL_TAU,DOFNODE);
%
% 14 Dec 96 - created
% 24 May 98 - update for more consistency with fluid flow method
% 13 Nov 98 - updated to include system 1st order in time
%
```

```
% Calculates the internal forces at the nodes based on linear
% elastic material properties
%
% Could be combined with NLFORCE and moved into STRUCT
%
% XN     = column vector of global X POSITION of element nodes
% YN     = column vector of global Y POSITION of element nodes
% DX     = column vector of global X DEFORMATION of element nodes (not
used)
% DY     = column vector of global Y DEFORMATION of element nodes (not
used)
% MP     = vector of material properties {E, Nu, density}.  Assumes
constant
%        material properties and plane stress case (easily modified)
% PSEUDO_MP = artificial material properties for pseudo time mass
matrix
% NN     = integer describing the order of integration to use in
%        gauss quadrature (1,2,3 for linear, quadratic, cubic)
%        NN = 2 is the normal value
% DEL_T = physical time step size
% DEL_TAU = pseudo time step size
% DOFNODE number of degrees of freedom per node
%
% FLAT: flat matrix of element integration arrays for each element
% Row 1 = sum of LHS stiffness matrices for [DX;DY] operations
% Row 2 = consistent mass mass matrix for [DX;DY] operations
% Row 3 = lumped mass mass matrix [DX;DY] operations
% Row 4 = matrix for RHS internal forces for [DX;DY] operations

% initialize
NNODES = length(XN);
NDOF = DOFNODE*NNODES;

% Constitutive equation for material properties.
% This could be moved inside the element loop if variable material
properties
% were needed
% plane stress model: D11 = E/(1-Nu^2)  D12 = Nu*D11  D33 = (D11-D12)/2
D11 = MP(1)/(1-MP(2)^2);
D12 = D11 * MP(2);
D33 = (D11 - D12)/2;

% Plane strain model= General 2-D isotropic elastic tensor
% LAMDA = Nu*E/((1+Nu)(1-2Nu)  Mu = E/2(1+Nu)
% [lamda+2mu lamda      0 ]
% [lamda     lamda+2mu  0 ]
% [0         0          mu]
LAMDA = MP(2)*MP(1)/(1+MP(2))/(1-2*MP(2));
MU = MP(1)/2/(1+MP(2));
%D11=LAMDA+2*MU;
%D12=LAMDA;
%D33=(D11-D12)/2;

if NNODES == 3                % triangle
  REGION = 2;
elseif NNODES == 4            % rectangle
```

```
  REGION = 1;
elseif NNODES == 8              % rectangle
  REGION = 1;
end

FLAT = zeros(4,NDOF^2);
W    = zeros(NDOF);
WT   = zeros(NDOF);
STIFFNESS = zeros(NDOF);
TR   = zeros(NDOF);
C_MASS = zeros(NDOF);
L_MASS = zeros(NDOF);

% Build PDE integration for each element
% FLAT: flat matrix of element integration arrays for each element
% Row 1 = sum of LHS stiffness matrices for [DX;DY] operations
% Row 2 = consistent mass mass matrix for [DX;DY] operations
% Row 3 = lumped mass mass matrix [DX;DY] operations
% Row 4 = matrix for RHS internal forces for [DX;DY] operations

FLAT =
gauss('lin_mat',NN,REGION,XN,YN,MP,PSEUDO_MP,DEL_T,DEL_TAU,DOFNODE);

W = reshape(FLAT(1,1:NDOF^2),NDOF,NDOF);              % [W]
C_MASS = reshape(FLAT(2,1:NDOF^2),NDOF,NDOF);         % consistent mass
matrix
L_MASS = reshape(FLAT(3,1:NDOF^2),NDOF,NDOF);         % lumped mass
matrix

% form correct stiffness matrix for element
% block pseudo-transpose - reverses indices on indicial notation (Wac
vs. Wca)
WT = [W(1:NNODES,1:NNODES),      W(NNODES+1:2*NNODES, 1:NNODES);
      W(1:NNODES, NNODES+1:2*NNODES),
W(NNODES+1:2*NNODES,NNODES+1:2*NNODES)];

% block matrix trace
TR = [W(1:NNODES,1:NNODES)+W(NNODES+1:2*NNODES,NNODES+1:2*NNODES),
zeros(NNODES,NNODES);
      zeros(NNODES,NNODES),
W(1:NNODES,1:NNODES)+W(NNODES+1:2*NNODES,NNODES+1:2*NNODES)];

% linear stiffness matrix
% plane strain
%STIFFNESS = LAMDA * W + MU * (WT + TR);
% plane stress
STIFFNESS = D12 * W + D33 * (WT + TR);
%STIFFNESS=[STIFFNESS(1:4,1:4),zeros(NNODES,NNODES);zeros(NNODES,NNODES
),STIFFNESS(5:8,5:8)];
%STIFFNESS=[STIFFNESS(1:4,1:4),zeros(NNODES,NNODES);zeros(NNODES,NNODES
),zeros(NNODES,NNODES)];
%STIFFNESS(5:8,5:8)=zeros(NNODES,NNODES);
%STIFFNESS = LAMDA * W + 2*MU * [W(1:4,1:4),
zeros(4,4);zeros(4,4),W(5:8,5:8)];

% FLAT: flat matrix of element integration arrays for each element
```

```
% Row 1 = sum of LHS stiffness matrices [KO+KL+KS] for [DX;DY]
operations
% Row 2 = consistent mass mass matrix for [DX;DY] operations
% Row 3 = lumped mass mass matrix [DX;DY] operations
% Row 4 = matrix for RHS internal forces for [DX;DY] operations

FLAT = zeros(4,NDOF^2);

FLAT = [reshape(STIFFNESS,1,NDOF^2); ...
        reshape(C_MASS,1,NDOF^2); ...
        reshape(L_MASS,1,NDOF^2); ...
        reshape(STIFFNESS,1,NDOF^2)];
```

NLFORCE

```
function FLAT =
nlforce(XN,YN,DX,DY,MP,PSEUDO_MP,NN,DEL_T,DEL_TAU,DOFNODE);
%function FLAT =
nlforce(XN,YN,DX,DY,MP,PSEUDO_MP,NN,DEL_T,DEL_TAU,DOFNODE);
%
% 15 Dec 96 - created
% 25 May 98 - update for more consistency with fluid flow method
% 14 Nov 98 - updated to include system 1st order in time
%
% Calculates the LHS and RHS stiffness matrices based on non-linear
% deformations and linear elastic material properties and consistent
% and lumped mass matrices.  Mass matrices may be used with physical
% and/or pseudo time.
%
% Interim routine to add possible functionality
%
% XN     = column vector of global X POSITION of element nodes
% YN     = column vector of global Y POSITION of element nodes
% DX     = column vector of global X DEFORMATION of element nodes
% DY     = column vector of global Y DEFORMATION of element nodes
% MP     = vector of material properties {E, Nu}.  Assumes constant
%        material properties and plane stress case (easily modified)
% NN     = integer describing the order of integration to use in
%        gauss quadrature (1,2,3 for linear, quadratic, cubic)
%        NN = 2 is the normal value
% DEL_T = physical time step size
% DEL_TAU = pseudo time step size
% DOFNODE number of degrees of freedom per node
%
% FLAT: flat matrix of element integration arrays for each element
% Row 1 = sum of LHS stiffness matrices for [DX;DY] operations
% Row 2 = consistent mass mass matrix for [DX;DY] operations
% Row 3 = lumped mass mass matrix [DX;DY] operations
% Row 4 = matrix for RHS internal forces for [DX;DY] operations

% initialize
NNODES = length(XN);
```

```
NDOF = DOFNODE*NNODES;

if NNODES == 3          % triangle
  REGION = 2;
elseif NNODES == 4      % bilinear rectangle
  REGION = 1;
elseif NNODES == 8      % biquadratic rectangle
  REGION = 1;
end

FLAT = zeros(4,(2*NNODES)^2);


% integrate the non-linear matrix over the element.  DOFNODE hardwired
to 2
FLAT =
gauss('nlin_mat',NN,REGION,XN,YN,DX,DY,MP,PSEUDO_MP,DEL_T,DEL_TAU,2);

% FLAT: flat matrix of element integration arrays for each element
% Row 1 = sum of LHS stiffness matrices [KO+KL+KS] for [DX;DY]
operations
% Row 2 = consistent mass mass matrix for [DX;DY] operations
% Row 3 = lumped mass mass matrix [DX;DY] operations
% Row 4 = matrix for RHS internal forces for [DX;DY] operations
```

LIN_MAT

```
function MFLAT = lin_mat(ZETA,XN,YN,MP,PSEUDO_MP,DEL_T,DEL_TAU,DOFNODE)
%function MFLAT =
lin_mat(ZETA,XN,YN,MP,PSEUDO_MP,DEL_T,DEL_TAU,DOFNODE)
%
% 14 Dec 96 - created
% 24 May 98 - update for more consistency with fluid flow method
% 13 Nov 98 - updated to include system 1st order in time
%
% Forms the tensor function Transpose(B)*B for the linear stiffness
% matrix using linear quadralateral (brick) or triangle shape
functions.
% Similar to the function (dNi/dXj) * transpose(dNi/dXj) * Area but
also
% forms the cross elements needed in the stiffness calculations.
%
% Also forms the pseudo time mass matrix for LHS operations and breaks
% pseudo time motion into a first order system of equations in DX, DY,
VX,
% and VY.
%
% The module checks ZETA to figure out which shape to use.
% Routine assumes a vector parameter (2-D) is to be distributed over
% the element.  Intended for use in gauss quadrature.
%
% ZETA = row vector of current position in local coordinates
```

```
% XN    = column vector of global X position of the element nodes
% YN    = column vector of global Y position of the element nodes
% MP     = vector of material properties {E, Nu, Rho}.  Assumes constant
%        material properties and plane stress case (easily modified)
% PSEUDO_MP = properties in pseudo time mass matrix [diag X, off XY;
off YX, diag Y]
% DEL_T = physical time step size
% DEL_TAU = pseudo time step size
% DOFNODE = number of degrees of freedom per node
%
% MFLAT: flat matrix of element integration arrays for each element
% Row 1 = sum of LHS stiffness matrices for [DX;DY] operations
% Row 2 = consistent mass mass matrix for [DX;DY] operations
% Row 3 = lumped mass mass matrix [DX;DY] operations
% Row 4 = matrix for RHS internal forces for [DX;DY] operations

NNODES = length(XN);
NDOF = NNODES*DOFNODE;
MFLAT = zeros(4,NDOF^2);
SHAPE=length(ZETA);

if SHAPE == 3                    % Triangle

  DZJDXK = zeros(3,2);
  DNIDZJ = zeros(3,3);
  NI     = zeros(3,1);
  DXKDZJ = zeros(3,2);
  DNIDXK = zeros(3,2);
  [DZJDXK, DNIDZJ, NI, DXKDZJ, AE] = triashap(ZETA, XN, YN);

elseif SHAPE == 2                % Brick

  if length(XN) == 4                % bilinear
    DZJDXK = zeros(2,2);
    DNIDZJ = zeros(4,2);
    NI     = zeros(4,1);
    DXKDZJ = zeros(2,2);
    DNIDXK = zeros(4,2);
   [DZJDXK, DNIDZJ, NI, DXKDZJ, AE] = brikshap(ZETA, XN, YN);

  elseif length(XN) == 8            % biquadratic
    DZJDXK = zeros(2,2);
    DNIDZJ = zeros(8,2);
    NI     = zeros(8,1);
    DXKDZJ = zeros(2,2);
    DNIDXK = zeros(8,2);
   [DZJDXK, DNIDZJ, NI, DXKDZJ, AE] = brik2ord(ZETA, XN, YN);
  end
end

DNIDXK = DNIDZJ*DZJDXK;

% basis for stiffness matrix
B211 = [DNIDXK(:,1);DNIDXK(:,2)] * [DNIDXK(:,1);DNIDXK(:,2)]';

% consistent and lumped mass matrices
```

```
C_MASS = kron(eye(2),NI*NI');
L_MASS = diag(sum(abs(C_MASS'))');

% MFLAT: Build flat matrix of element integration arrays for each
element
% Row 1 = sum of LHS stiffness matrices for [DX;DY] operations
% Row 2 = consistent mass mass matrix for [DX;DY] operations
% Row 3 = lumped mass mass matrix [DX;DY] operations
% Row 4 = matrix for RHS internal forces for [DX;DY] operations

MFLAT = [reshape(B211,1,NDOF^2); ...
         reshape(C_MASS,1,NDOF^2); ...
         reshape(L_MASS,1,NDOF^2); ...
         reshape(B211,1,NDOF^2)] * AE;
```

NLIN_MAT

```
function MFLAT =
nlin_mat(ZETA,XN,YN,DX,DY,MP,PSEUDO_MP,DEL_T,DEL_TAU,DOFNODE)
%function MFLAT =
nlin_mat(ZETA,XN,YN,DX,DY,MP,PSEUDO_MP,DEL_T,DEL_TAU,DOFNODE)
%
% 14 Nov 98 - updated to include system 1st order in time
%
% Calculates the matrix for forces at the nodes based on non-linear
% deformations and linear elastic material properties.  Handles both
% linear triangular and bilinear quadralateral elements
%
% BBAR' * D * BBAR * Area
% Intended for use in gauss quadrature.
%
% ZETA = row vector of current position in local coordinates
% XN   = column vector of global X initial position of the element
nodes
% YN   = column vector of global Y initial position of the element
nodes
% DX    = column vector of global X DEFORMATION of element nodes
% DY    = column vector of global Y DEFORMATION of element nodes
% MP    = vector of material properties {E, Nu, Rho}.  Assumes constant
%       material properties and plane stress case (easily modified)
% PSEUDO_MP = properties in pseudo time mass matrix [diag X, off XY;
off YX, diag Y]
% DEL_T = physical time step size
% DEL_TAU = pseudo time step size
% DOFNODE = number of degrees of freedom per node: 2 for basic system,
4 for
%           first order system in pseudo time or physical time
%
% MFLAT: flat matrix of element integration arrays for each element
% Row 1 = sum of LHS stiffness matrices for [DX;DY] operations
% Row 2 = consistent mass mass matrix for [DX;DY] operations
% Row 3 = lumped mass mass matrix [DX;DY] operations
```

```
% Row 4 = matrix for RHS internal forces for [DX;DY] operations

% initialize
NNODES = length(XN);
NDOF = NNODES*DOFNODE;
MFLAT = zeros(4,NDOF^2);
SHAPE=length(ZETA);

% Constitutive equation for material properties.
% Plane stress model: D11 = E/(1-Nu^2)  D12 = Nu*D11  D33 = (D11-D12)/2
D = zeros(3);
D(1,1) = MP(1)/(1-MP(2)^2);
D(1,2) = D(1,1) * MP(2);
D(3,3) = (D(1,1) - D(1,2))/2;
D(2,2) = D(1,1);
D(2,1) = D(1,2);

% Plane strain model= General 2-D isotropic elastic tensor
% LAMDA = Nu*E/((1+Nu)(1-2Nu)  Mu = E/2(1+Nu)
% [lamda+2mu lamda      0 ]
% [lamda     lamda+2mu  0 ]
% [0         0          mu]
LAMDA = MP(2)*MP(1)/(1+MP(2))/(1-2*MP(2));
MU = MP(1)/2/(1+MP(2));
%D = zeros(3);
%D(1,1)=LAMDA+2*MU;
%D(1,2)=LAMDA;
%D(3,3)=(D(1,1)-D(1,2))/2;
%D(2,2) = D(1,1);
%D(2,1) = D(1,2);

% SHAPE = determines the shape of the region (2 = rectangle, 3 =
triangle)
DNIDZJ = zeros(NNODES,SHAPE);
DZJDXK = zeros(SHAPE,2);
DXKDZJ = zeros(2,SHAPE);
NI     = zeros(NNODES,1);
DNIDXK = zeros(NNODES,2);
BBAR   = zeros(3,NDOF);        % nonlinear tangent strain matrix
STRAIN_LIN = zeros(3,NDOF);    % linear strain (E_L = STRAIN_LIN *
{dx;dy}')
STRAIN_NL = zeros(3,NDOF);     % nonlinear strain E_NL = STRAIN_NL *
{dx;dy}')
B1     = zeros(3,NNODES);      % matrix of convenience
B2     = zeros(3,NNODES);      % matrix of convenience
ZIP    = zeros(NNODES,1);      % zeros

if SHAPE == 3                % Triangle
 [DZJDXK, DNIDZJ, NI, DXKDZJ, AE] = triashap(ZETA,XN,YN);

elseif SHAPE == 2            % Brick

  if NNODES == 4                    % bilinear
    [DZJDXK, DNIDZJ, NI, DXKDZJ, AE] = brikshap(ZETA,XN,YN);

  elseif NNODES == 8                % biquadratic
```

```
   [DZJDXK, DNIDZJ, NI, DXKDZJ, AE] = brik2ord(ZETA, XN, YN);
  end
end

DNIDXK = DNIDZJ*DZJDXK;

% form the non-linear strain tangent matrix BBAR=B_linear + B_nonlinear
% see Zienkiewicz & Taylor, Vol 2, Chapter 8, 1991
B1 = [DNIDXK(:,1)'; ZIP'; DNIDXK(:,2)'];
B2 = [ZIP'; DNIDXK(:,2)'; DNIDXK(:,1)'];
GG = [DNIDXK(:,1)', ZIP'
      ZIP'        , DNIDXK(:,1)'
      DNIDXK(:,2)', ZIP'
      ZIP'        , DNIDXK(:,2)'];
STRAIN_LIN = [B1,B2];
STRAIN_NL  = .5*[B1*(DX), B1*(DY), B2*(DX), B2*(DY)]*GG;
FF=[DNIDXK(:,1)'*(XN+DX),DNIDXK(:,2)'*(XN+DX)
    DNIDXK(:,1)'*(YN+DY),DNIDXK(:,2)'*(YN+DY)];
EE=.5*(FF'*FF-eye(2));


%BBAR= infinitesimal + large displacement
BBAR = STRAIN_LIN    + 2*STRAIN_NL;

% form the point integral of the non-linear matrix product BBAR' * D *
BBAR
KBAR = BBAR' * D * BBAR;

%form the stress vector and internal force matrix
STRESS_MAT = D * (STRAIN_LIN + STRAIN_NL);
STRESS =  STRESS_MAT * [DX;DY];
INT_FORCE = BBAR' * STRESS_MAT;

% initial stress matrix
MM = [STRESS(1)*eye(2), STRESS(3)*eye(2); STRESS(3)*eye(2),
STRESS(2)*eye(2)];
KS = GG' * MM * GG;

% consistent and lumped mass matrices
C_MASS = kron(eye(2),NI*NI');
L_MASS = diag(sum(abs(C_MASS'))');

% MFLAT: Build flat matrix of element integration arrays for each
element
% Row 1 = sum of LHS stiffness matrices for [DX;DY] operations
% Row 2 = consistent mass mass matrix for [DX;DY] operations
% Row 3 = lumped mass mass matrix [DX;DY] operations
% Row 4 = matrix for RHS internal forces for [DX;DY] operations

MFLAT = [reshape(KBAR+KS,1,NDOF^2); ...
         reshape(C_MASS,1,NDOF^2); ...
         reshape(L_MASS,1,NDOF^2); ...
         reshape(INT_FORCE,1,NDOF^2)] * AE;
```

BRIK2ORD

```
function [DZIDXJ, DNIDZJ, NI, DXJDZI, AE]=brik2ord(ZETA,XN,YN)
%  function [DZIDXJ, DNIDZJ, NI, DXJDZI, AE]=brik2ord(ZETA,XN,YN)
%
% Evaluates various aspects of the serendipity BIQUADRATIC shape
function
%
%  Calculates the 2x2 GLOBAL to LOCAL jacobian DZIDXJ      (dZi / dXj)
%  Calculates the 8x2 local shape derivative matrix DNIDZJ (dNi / dZj)
%  Calculates the 8x1 local shape function                 (Ni)
%  Calculates the 2x2 LOCAL to GLOBAL jacobian DXJDZI      (dXj / dZi)
%  Calculates the scalar GLOBAL to LOCAL area ratio of the element (AE)
%
%  ZETA is a row vector of the current position in local coordinates
%  XN is a column vector of the global X position of the element nodes
%  YN is a column vector of the global Y position of the element nodes

% initialize space
Z1=ZETA(1);
Z2=ZETA(2);
DZIDXJ = zeros(2,2);
DXJDZI = zeros(2,2);
NI     = zeros(8,1);
DNIDZJ = zeros(8,2);

DNDZ1 = zeros(8,1);
DNDZ2 = zeros(8,1);

% create some common terms
MZ = ones(1,2) - ZETA;
PZ = ones(1,2) + ZETA;
MZ2 = ones(1,2) - ZETA.^2;
PZ2 = ones(1,2) + ZETA.^2;

% local shape matrix
NI = [MZ(1)*MZ(2)*(-Z1-Z2-1);...
      PZ(1)*MZ(2)*(Z1-Z2-1);...
      PZ(1)*PZ(2)*(Z1+Z2-1);...
      MZ(1)*PZ(2)*(-Z1+Z2-1);...
      2*MZ2(1)*MZ(2);...
      2*PZ(1)*MZ2(2);...
      2*MZ2(1)*PZ(2);...
      2*MZ(1)*MZ2(2)]/4;

% local shape derivative matrix
DNIDZJ(:,1) = ...
    [-(1/4*(1 - Z1)*(1 - Z2)) - (1*(1 - Z2)*(-1 - Z1 - Z2))/4,...
     (1*(1 + Z1)*(1 - Z2))/4 + (1*(1 - Z2)*(-1 + Z1 - Z2))/4,...
     (1*(1 + Z1)*(1 + Z2))/4 + (1*(1 + Z2)*(-1 + Z1 + Z2))/4,...
     -(1/4*(1 - Z1)*(1 + Z2)) - (1*(1 + Z2)*(-1 - Z1 + Z2))/4,...
     -(Z1*(1 - Z2)),...
     (1*(1 - Z2^2))/2,...
     -(Z1*(1 + Z2)),...
```

```
     (1*(-1 + Z2^2))/2]';
DNIDZJ(:,2) = ...
   [-(1/4)*(1 - Z1)*(1 - Z2) - 1/4*(1 - Z1)*(-1 - Z1 - Z2),...
    -(1/4)*(1 + Z1)*(1 - Z2) - 1/4*(1 + Z1)*(-1 + Z1 - Z2),...
    1/4*(1 + Z1)*(1 + Z2) + 1/4*(1 + Z1)*(-1 + Z1 + Z2),...
    1/4*(1 - Z1)*(1 + Z2) + 1/4*(1 - Z1)*(-1 - Z1 + Z2),...
    1/2*(-1 + Z1^2),...
    -(1 + Z1)*Z2,...
    1/2*(1 - Z1^2),...
    -(1 - Z1)*Z2]';

% form LOCAL to GLOBAL jacobian matrix
DXJDZI = [XN'*DNIDZJ;
          YN'*DNIDZJ];

% form GLOBAL to LOCAL jacobian matrix using adjoint/det formula
AE = det(DXJDZI);
DZIDXJ = [DXJDZI(2,2), -DXJDZI(1,2);
          -DXJDZI(2,1), DXJDZI(1,1)]/AE;
AE = abs(AE);
```

STR_EIG

```
function INT_F =
str_eig(NODE_VEC,dummy,XN,YN,ELMAT,BCX,BCY,DOFNODE,TDOFNODE,PSEUDO_MP)

% function INT_F =
str_eig(NODE_VEC,dummy,XN,YN,ELMAT,BCX,BCY,DOFNODE,TDOFNODE,PSEUDO_MP)
%
% 1 Jan 99 - original from STR_LHS
%
% Combines different routines to form the static structural equations
% Technique was not been developed for physical transient problems and
is untested
%
% K*(dx; dy)
% At this point, STIFF should be organized as
% [KO+KL+KS, anything]
% [anything, -I      ]
%
% Output is the internal force vector
%
% INPUT VARIABLES:
% NODE_VEC = nodal deformation vector (DX, DY)
% dummy = place holder for ARNOLDI iteration
% XN = X location of nodes
% YN = Y location of nodes
% ELMAT = matrix listing the nodes assigned to each element
% BCX = X deformation (and velocity) Dirichlet boundary conditions
[node #, X, VX]
% BCY = Y deformation (and velocity) Dirichlet boundary conditions
[node #, Y, VY]
```

```
% DOFNODE = number of degrees of freedom per node to use in
calculations
% TDOFNODE = total number of degrees of freedom per node
% PSEUDO_MP = pseudo mass material property
%
% OUTPUT VARIABLES:
% INT_F = summation of internal forces
%

global STIFF

% initialize variables and workspace
NNODES = length(XN);
INT_F = zeros(size(NODE_VEC));       % Space for INTERNAL FORCE vector
MASS_VEC = zeros(size(NODE_VEC));    % Space for LUMPED MASS diagonal

[NEL,VERTICES] = size(ELMAT);        % VERTICES is the max number of
nodes in any element

% element by element work space
TEMP = zeros(VERTICES*TDOFNODE);     % Set up max space for largest
element STIFF matrix
KT_MAT = zeros(VERTICES*DOFNODE);    % Set up max space for total
tangent stiffness matrix
MASS_MAT = zeros(VERTICES*DOFNODE); % Set up max space for pseudo mass
matrix
SHUFFLE = zeros(1,VERTICES*DOFNODE);% Index

%************************************ BOUNDARY CONDITION ROUTINE 1 **
% Form deformation vectors and apply dirichlet boundary conditions
%*************************************************************************
DX = NODE_VEC(1:NNODES);
DY = NODE_VEC(NNODES+1:2*NNODES);
DX(BCX(:,1))=BCX(:,2);               % set dirichlet X boundary
deformation
DY(BCY(:,1))=BCY(:,2);               % set dirichlet Y boundary
deformation

%************************** ELEMENT MASS CALCULATIONS
*******************
% Form the LUMPED MASS matrix (vector of diagonal) using local vector
% operations and an element by element approach
%*************************************************************************
***
for j = 1:NEL
% check the shape of each element and set up its needed workspace size
  INDEX=find(ELMAT(j,:));      % INDEX contains the non-zero node
numbers for the element
  VERTICES = length(INDEX);
  NDOF = VERTICES * DOFNODE;  % NDOF for the element
  TNDOF = VERTICES * TDOFNODE; % total NDOF for the element
  SLIST = ELMAT(j,INDEX);      % index into variables stored in data
file

  TEMP = zeros(TNDOF);
```

```
  TEMP = reshape(STIFF(j,1:TNDOF^2),TNDOF,TNDOF);            % recall
stored stiffness matrix
  MASS_MAT = -PSEUDO_MP * TEMP(NDOF+1:TNDOF,NDOF+1:TNDOF);  % pull out
the block of interest

  SHUFFLE = [];
  for ii = 1:DOFNODE                                        % Index into mass
vector
    SHUFFLE = [SHUFFLE, SLIST+NNODES*(ii-1)];
  end

% form diagonal of lumped mass matrix
  MASS_VEC(SHUFFLE) = MASS_VEC(SHUFFLE) + sum(abs(MASS_MAT'))';
end

%***************************** ELEMENT CALCULATIONS
**********************
% Form the INTERNAL FORCE vector using local vector operations and an
% element by element approach
%*************************************************************************
***
for j = 1:NEL

% check the shape of each element and set up its needed workspace size

  INDEX=find(ELMAT(j,:));     % INDEX contains the non-zero node
numbers for the element
  VERTICES = length(INDEX);
  NDOF = VERTICES * DOFNODE;  % NDOF for the element
  TNDOF = VERTICES * TDOFNODE; % total NDOF for the element
  SLIST = ELMAT(j,INDEX);     % index into variables stored in data
file

  TEMP = zeros(TNDOF);
  TEMP = reshape(STIFF(j,1:TNDOF^2),TNDOF,TNDOF);     % recall stored
stiffness matrix
  KT_MAT = TEMP(1:NDOF,1:NDOF);                       % pull out the
block of interest

  SHUFFLE = [];
  for ii = 1:DOFNODE                                  % Index into
internal force vector
    SHUFFLE = [SHUFFLE, SLIST+NNODES*(ii-1)];
  end

% form internal force vector for structural equations
  INT_F(SHUFFLE)= INT_F(SHUFFLE) + ...
                  KT_MAT * ([DX(SLIST);DY(SLIST)]./MASS_VEC(SHUFFLE));
end
```

STR_LHS

```
function R =
str_lhs(NODE_VEC,dummy,XN,YN,ELMAT,PTX,PTY,SCALE,BCX,BCY,MP,DEL_T,DEL_T
AU,NN, ...
                    DOFNODE,LIN,XM1,VM1)

% function R =
str_lhs(NODE_VEC,dummy,XN,YN,ELMAT,PTX,PTY,SCALE,BCX,BCY,MP,DEL_T,DEL_T
AU,NN, ...
%                   DOFNODE,LIN,XM1,VM1)
%
% 15 Dec 96 - updated
% 22 Dec 96 - added call to initialize PTIME
% 12 Jan 97 - added parameter to select linear or nonlinear force
% 24 May 98 - changed overall structure to be more consistent with
fluid flow method
% 13 Nov 98 - updated to include 1st order system in time
%
% Combines different routines to form the static or dynamic structural
equations
% Treats time and pseudo-time as a system of first order differential
equations
%
% c*d/dt (dx;dy) + rho*d/dt (u;v) + K*(dx; dy)              = 0
%   d/dt (dx;dy)                                - (u;v)     = 0
%
% [c, rho]*transpose{dx, dy, u, v}+[K, 0]*transpose{dx, dy, u, v}=0
% [I,  0 ]      dt                 [0,-I]
%
% Output is the residual (want R = 0 at equilibrium).
%
% INPUT VARIABLES:
% NODE_VEC = nodal deformation vector (DX, DY)
% dummy = place holder for GMRES solver
% XN = X location of nodes
% YN = Y location of nodes
% ELMAT = matrix listing the nodes assigned to each element
% PTX = external point (nodal) loads in the X direction
% PTY = external point (nodal) loads in the Y direction
% SCALE = scaling factor used to gradually apply loads
% BCX = X deformation (and velocity) Dirichlet boundary conditions
[node #, X, VX]
% BCY = Y deformation (and velocity) Dirichlet boundary conditions
[node #, Y, VY]
% MP = material properties (E, Nu, density)                    (not
used on LHS)
% DEL_T = physical time step size                              (not
used on LHS)
% DEL_TAU = pseudo time step size                              (not
used on LHS)
% NN = order of gauss integration                              (not
used on LHS)
```

```
% DOFNODE = number of degrees of freedom per node
% LIN = 2 element vector of calculation choices                    (not
used on LHS)
%       term 1: linear or non-linear selection, 1 = linear, 2 =
nonlinear
%       term 2: pseudo time step, 0 = do not use, 0 < DT include
% XM1 = [DX;DY] at t-1                                             (not
used on LHS)
% VM1 = [VX;VY] at t-1                                             (not
used on LHS)
%
% OUTPUT VARIABLES:
% R = residual of summation of external and internal forces
%

global STIFF

% initialize variables and workspace
NNODES = length(XN);
R = zeros(size(NODE_VEC));          % Space for RESIDUAL vector
F_APPLIED = zeros(size(NODE_VEC));  % Space for APPLIED LOAD vector
[NEL,VERTICES] = size(ELMAT);       % VERTICES is the max number of
nodes in any element

LHS_MAT = zeros(VERTICES*DOFNODE);  % Set up max space for LHS jacobian
matrix
SHUFFLE = zeros(1,VERTICES*DOFNODE);% Index

%************************************ BOUNDARY CONDITION ROUTINE 1 **
% Form deformation vectors and apply dirichlet boundary conditions
%*********************************************************************
DX = NODE_VEC(1:NNODES);
DY = NODE_VEC(NNODES+1:2*NNODES);
DX(BCX(:,1))=BCX(:,2);              % set dirichlet X boundary
deformation
DY(BCY(:,1))=BCY(:,2);              % set dirichlet Y boundary
deformation
if DOFNODE == 4
  VX = NODE_VEC(2*NNODES+1:3*NNODES);
  VY = NODE_VEC(3*NNODES+1:4*NNODES);
  VX(BCX(:,1))=BCX(:,3);                % set dirichlet X boundary
velocity
  VY(BCY(:,1))=BCY(:,3);                % set dirichlet Y boundary
velocity
end

if DEL_T ~= 0
  DX1 = XM1(1:NNODES);
  DY1 = XM1(NNODES+1:2*NNODES);
  DX1(BCX(:,1))=BCX(:,2);
  DY1(BCY(:,1))=BCY(:,2);
  VX1 = VM1(1:NNODES);
  VY1 = VM1(NNODES+1:2*NNODES);
  VX1(BCX(:,1))=BCX(:,3);
  VY1(BCY(:,1))=BCY(:,3);
end
```

```
% scale applied nodal loads and put into a single vector
F_APPLIED(1:2*NNODES) = SCALE * [PTX; PTY];
R = -F_APPLIED;

%***************************** ELEMENT CALCULATIONS
**********************
% Form the RESIDUAL vector using local vector operations and an element
by
% element approach
%*************************************************************************
***
for j = 1:NEL

% check the shape of each element and set up its needed workspace size

  INDEX=find(ELMAT(j,:));      % INDEX contains the non-zero node
numbers for the element
  VERTICES = length(INDEX);
  NDOF = VERTICES * DOFNODE;  % NDOF for the element
  SLIST = ELMAT(j,INDEX);      % index into variables stored in data
file

  LHS_MAT = reshape(STIFF(j,1:NDOF^2),NDOF,NDOF);      % recall stored
LHS integration matrix

  SHUFFLE = [];
  for ii = 1:DOFNODE                                    % Index into
residual vector
    SHUFFLE = [SHUFFLE, SLIST+NNODES*(ii-1)];
  end

% form LHS residual for structural equations
% static and physical transient terms included in LHS_MAT
  if DOFNODE == 4
    R(SHUFFLE)= R(SHUFFLE) ...
                + LHS_MAT *
[DX(SLIST);DY(SLIST);VX(SLIST);VY(SLIST)];
  else
    R(SHUFFLE)= R(SHUFFLE) ...
                + LHS_MAT * [DX(SLIST);DY(SLIST)];
  end
end


%*********************************** BOUNDARY CONDITION ROUTINE 2 **
% Zero out Dirichlet boundary conditions in the residual
%*************************************************************************
[MBX, NBX] = size(BCX);
[MBY, NBY] = size(BCY);
if MBX > 0
  R(BCX(:,1)) = zeros(MBX,1);              % Zero out X boundary
position residual
  if DOFNODE == 4
    R(BCX(:,1)+2*NNODES) = zeros(MBX,1);  % Zero out X boundary
velocity residual
```

```
  end
end
if MBY > 0
  R(BCY(:,1)+NNODES) = zeros(MBY,1);       % Zero out Y boundary
position residual
  if DOFNODE == 4
    R(BCY(:,1)+3*NNODES) = zeros(MBY,1); % Zero out Y boundary
velocity residual
  end
end

%************************************ SCALING ROUTINE ***************
% Scale the equations to make the associated residuals roughly the same
% order of magnitude for equivalent perturbations.  Greatly aids
% orthogonality in LHS modified Gram Schmidt process
%*********************************************************************
global STRUCT_SCALE

temp=length(R)/DOFNODE;
if DOFNODE == 4
  R(2*temp+1:3*temp)=R(2*temp+1:3*temp)/STRUCT_SCALE(3);
  R(3*temp+1:4*temp)=R(3*temp+1:4*temp)/STRUCT_SCALE(4);
end

R(1:temp)=R(1:temp)/STRUCT_SCALE(1);
R(temp+1:2*temp)=R(temp+1:2*temp)/STRUCT_SCALE(2);
```

APPENDIX G

GRID ROUTINES

APPENDIX G

GRID ROUTINES

The grid motion routines implement the continuum solid equation set to determine grid deformation as described in Chapter 9. Much of the coding duplicates that in Appendix F but was maintained as separate subroutines to provide flexibility. The architecture follows the general logic described in Appendix C and the structural solver logic in Appendix F.

Figure G-1 illustrates the interaction between the generic routines of the numerical solver and the specific routines for the grid motion equation set. The shaded boxes are specific to the grid motion calculation while the unshaded boxes are generic. As usual, a top level script instigates the solution process and identifies the subroutines containing the right and left hand side residual calculation logic. Figures G-2 and G-3 expand on the hierarchy in the right and left hand side residual subroutines.

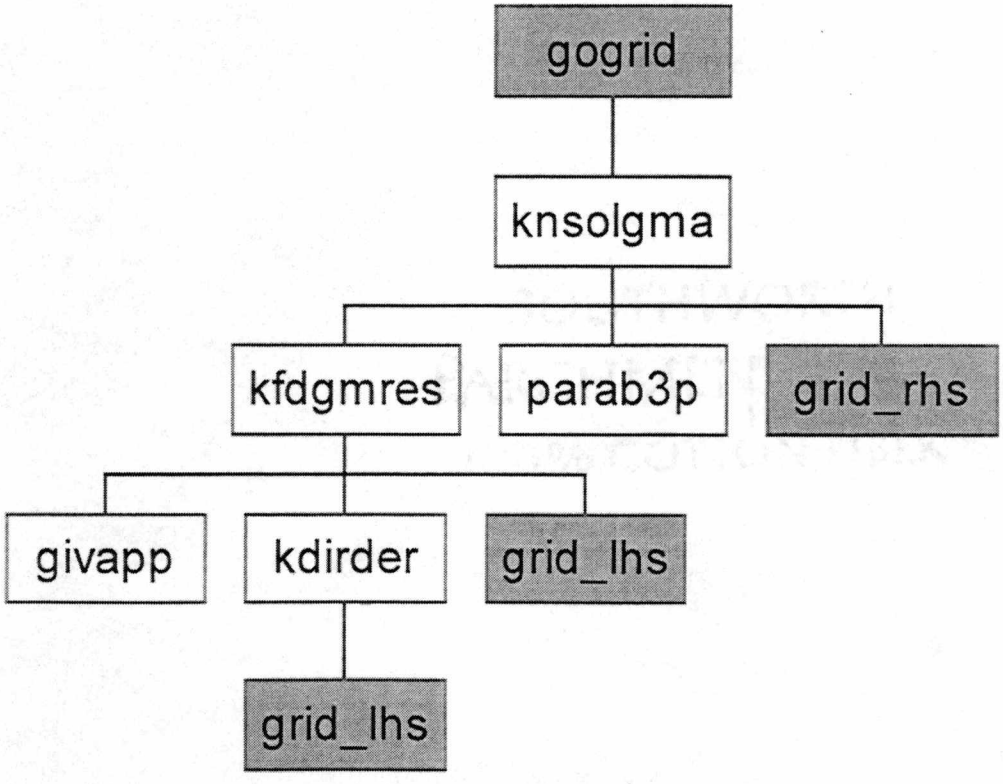


Figure G-1: Fluid Grid Deformation Master Hierarchy

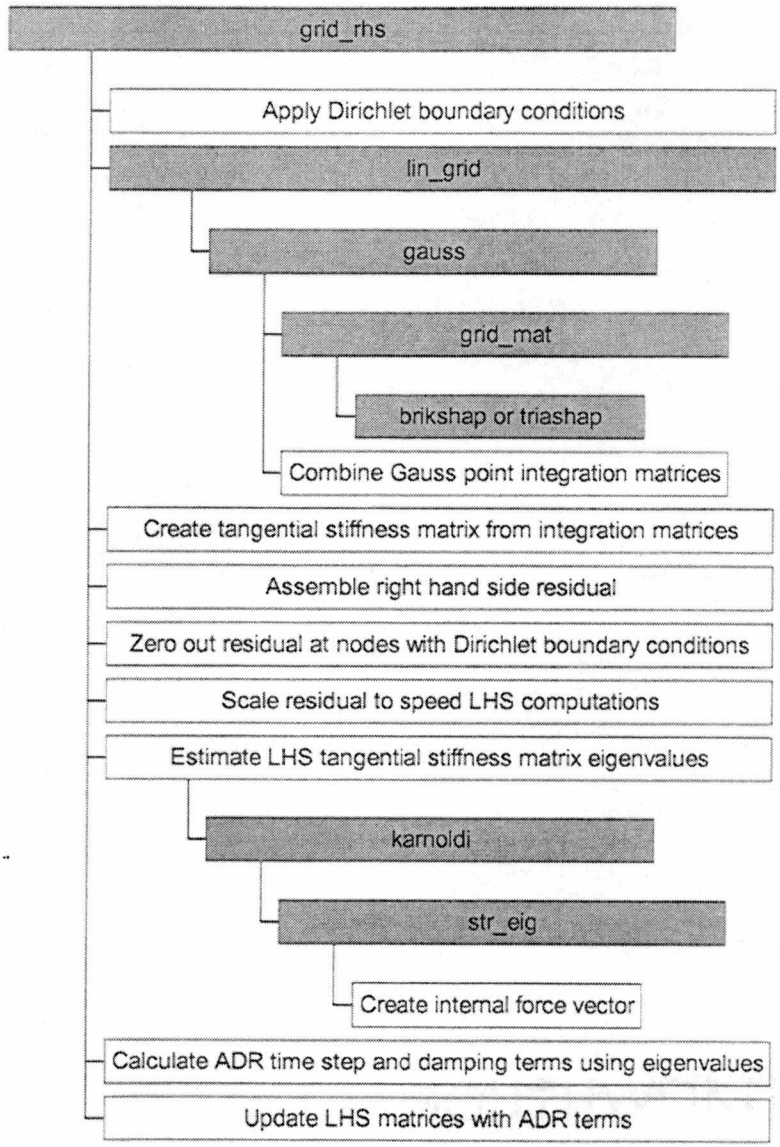


Figure G-2: Right Hand Side Fluid Grid Deformation Residual

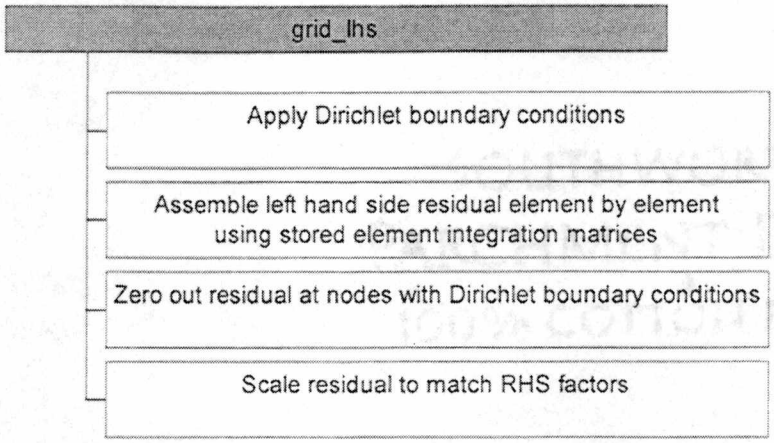


Figure G-3: Left Hand Side Fluid Grid Deformation Residual

A quick comparison of Figure G-2 and Figure F-2 shows the *grid_rhs* subroutine matches the *str_rhs* with only minor exceptions. The *lin_grid* subroutine re-implements the *lforce* subroutine and the eigenvalue estimates using the *karnoldi* subroutine use the exact same *str_eig* module as the structural solver. The left hand side residual logic illustrated in Figure G-3 is the same as that shown in Figure F-3.

script gogrid - Master routine for calculating motion of a single fluid domain given Dirichlet boundary conditions

function grid_rhs
- Combines different routines to form the static or dynamic grid deformation equations based on treating it as a solid continuum
- Treats time and pseudo-time as a system of first order differential equations
- Want R = 0 at equilibrium

function lin_grid - Calculates the internal forces at the nodes based on linear elastic material properties

function grid_mat
- Forms the tensor function Transpose(B)*B for the linear stiffness matrix using linear quadralateral (brick) or triangle shape functions
- Forms the pseudo time mass matrix for LHS operations and breaks pseudo time motion into a first order system of equations in DX, DY, VX, and VY
- Intended for use in gauss quadrature

function grid_lhs
- Combines different routines to form the static or dynamic grid deformation equations based on treating it as a solid continuum
- Treats time and pseudo-time as a system of first order differential equations

```
% Master script program
%
% initialize variables
% XN, YN = original X and Y location
% ELMAT  = nodes assigned to elements
% MP     = material properties
% NN     = Gauss integration order
% DOFNODE= number of degrees of freedom per node
% PTX,PTY= external nodal loads in X and Y direction
% BCX,BCY= displacement boundary conditions in X and Y directions
% DX, DY = initial displacement vectors in X and Y directions
%
% load nozzle2g  % nozzle grid
%
% eval(['load ',name]);

global GSTIFF
eval(['load ',name]);
%DOFNODE = 4;
DOFNODE = 2;    %xxxx

% set some Newton (outer) and GMRES (inner) iteration parameters
TOL=[1E-6,1E-6];        % [absolute error tolerance, relative error
tolerance]
[MX,NX] = size(BCX);
[MY,NY] = size(BCY);
NNODES = length(XN);

% Max number of GMRES iterations min(384, rank)
LIN_ITS = min(96,DOFNODE*NNODES-MX-MY);

% [max Newton steps, max linear iterations, etamax, linear soln method,
max restarts]
% 10 Newton steps
% 40 GMRES iterations per try
% .5 etamax
% 2 = Restarted GMRES(LIN_ITS)
% 4 = max number of GMRES restarts -> linear its max/Newton step
(Armillo reductions)

PARMS=[10,LIN_ITS,.5,2,8];  %xxxx
T_TOL=[1E-6,1E-9,2];    % [absolute time error tolerance, relative
error tolerance,DOF per node]
T_STEP=5;               % max time steps

LIN = 1;                % 1=linear, 2=nonlinear
DEL_T = 0;              % physical time step size
%DEL_TAU = 1;           % pseudo time step size
DEL_TAU = 0;            %xxxx

% decide how many load increments to use
NL = 1;
```

```
DX = zeros(size(XN));
DY = zeros(size(YN));
DX(BCX(:,1))=BCX(:,2);
DY(BCY(:,1))=BCY(:,2);

if DOFNODE == 4
  VX = zeros(size(XN));
  VY = zeros(size(YN));
  VX(BCX(:,1))=BCX(:,3);
  VY(BCY(:,1))=BCY(:,3);
end

flops(0);
tic

%  Initialize matrices for element by element storage
[NEL,LARGEST]=size(ELMAT);

% GSTIFF = [c*M/del_tau+KO+KL+KS , M/del_tau]
%          [I/del_tau            , -I       ]
%GSTIFF=zeros(NEL,(4*LARGEST)^2);
GSTIFF=zeros(NEL,(DOFNODE*LARGEST)^2);

%for i = 1:NL
%SCALE=i/NL;
SCALE=1;

FNRM=1;
STOP_TOL=T_TOL(1) + T_TOL(2)*FNRM;
ITC = 0;

if DOFNODE == 4
  SOLN = [DX;DY;VX;VY];
else
  SOLN = [DX;DY];
end

T_STEP=1;

while(FNRM > STOP_TOL & ITC < T_STEP)
  ITC=ITC+1;
  NODE_VEC = SOLN;

  figure(1)
  clg
  set(1,'Position',[2 30 447 340]);
  figure(2)
  clg
  set(2,'Position',[456 414 371 310]);
  figure(3)
  clg
  set(3,'Position',[2 415 447 308]);
  figure(4)
  clg
  set(4,'Position',[456 30 370 339]);
```

```
  [SOLN, HISTORY, IERROR, STATS] =
knsolgma('grid_lhs','grid_rhs',TOL,PARMS,NODE_VEC,...

NODE_VEC,XN,YN,ELMAT,PTX,PTY,SCALE,BCX,BCY,MP,DEL_T,DEL_TAU, ...
                      NN,DOFNODE,LIN);

disp([STATS(:,1),zeros(size(STATS(:,2))),STATS(:,3:5)])
format long
disp(STATS(:,2))
format short

FNRM = norm(SOLN-NODE_VEC)/sqrt(length(SOLN))

end

DX = SOLN(1:NNODES);
DY = SOLN(NNODES+1:2*NNODES);
if DOFNODE == 4
  VX = SOLN(2*NNODES+1:3*NNODES);
  VY = SOLN(3*NNODES+1:4*NNODES);
end

toc
flops
figure(1)
semilogy(HISTORY(:,1))
```

GRID_RHS

```
function R =
grid_rhs(NODE_VEC,dummy,XN,YN,ELMAT,PTX,PTY,SCALE,BCX,BCY,MP,DEL_T,DEL_
TAU,NN, ...
                      DOFNODE,XM1,VM1)

% function R =
grid_rhs(NODE_VEC,dummy,XN,YN,ELMAT,PTX,PTY,SCALE,BCX,BCY,MP,DEL_T,DEL_
TAU,NN, ...
%                     DOFNODE,XM1,VM1)
%
%  4 Aug 98 - created from struct eqn
%  6 Nov 98 - updated to include 1st order system in time
%
%
% Combines different routines to form the static or dynamic grid
deformation equations
% based on treating it as a solid continuum.  Treats time and pseudo-
time as a system
% of first order differential equations.
%
% c*d/dt (dx;dy) + rho*d/dt (u;v) + K*(dx; dy)                = 0
%   d/dt (dx;dy)                                  -  d/dt (u;v) = 0
%                          or
```

```
% [c, rho]*transpose{dx, dy, u, v}+[K, 0]*transpose{dx, dy, u, v}=0
% [I,  0 ]      dt                  [0,-I]
%
% c is not used
% Output is the residual (want R = 0 at equilibrium).
%
% INPUT VARIABLES:
% NODE_VEC = nodal deformation and velocity vector (DX; DY; U; V)
% dummy = place holder for GMRES solver
% XN = X location of nodes
% YN = Y location of nodes
% ELMAT = matrix listing the nodes assigned to each element
% PTX = external point (nodal) loads in the X direction
% PTY = external point (nodal) loads in the Y direction
% SCALE = scaling factor used to gradually apply loads
% BCX = X deformation (and velocity) Dirichlet boundary conditions
[node #, X, VX, solid node #]
% BCY = Y deformation (and velocity) Dirichlet boundary conditions
[node #, Y, VY, solid node #]
% MP = material properties (E...lbf/sqft, Nu, density...lbm/cuft)
% DEL_T = physical time step size
% DEL_TAU = pseudo time step size
% NN = order of gauss integration
% DOFNODE = number of degrees of freedom per node (4 with pseudo time)
% XM1 = [DX;DY] at t-1
% VM1 = [VX;VY] at t-1
%
% OUTPUT VARIABLES:
% R = residual of summation of external and internal forces
%

global GSTIFF

% initialize variables and workspace
NNODES = length(XN);
R = zeros(size(NODE_VEC));
F_APPLIED = zeros(size(NODE_VEC));
[NEL,VERTICES] = size(ELMAT); % VERTICES is the max number of nodes in
any element
NDOF = VERTICES * DOFNODE;

if VERTICES == 3              % triangle
  FLAT    = zeros(4,12^2);
  MASS_MAT = zeros(12);       % physical time consistent mass matrix
  STIFF_MAT = zeros(12);      % element stiffness matrix (KO + KL)
  INIT_MAT  = zeros(12);      % initial stress matrix (KS)

elseif VERTICES == 4          % rectangle
  FLAT    = zeros(4,16^2);
  MASS_MAT = zeros(16);
  STIFF_MAT = zeros(16);
  INIT_MAT  = zeros(16);
end

%************************************ BOUNDARY CONDITION ROUTINE 1 **
% Form deformation vectors and apply dirichlet boundary conditions
```

```
%*********************************************************************
DX = NODE_VEC(1:NNODES);
DY = NODE_VEC(NNODES+1:2*NNODES);
DX(BCX(:,1))=BCX(:,2);
DY(BCY(:,1))=BCY(:,2);

if DOFNODE == 4
  VX = NODE_VEC(2*NNODES+1:3*NNODES);
  VY = NODE_VEC(3*NNODES+1:4*NNODES);
  VX(BCX(:,1))=zeros(size(BCX(:,2)));     % zero out dirichlet boundary
velocity
  VY(BCY(:,1))=zeros(size(BCY(:,2)));     % zero out dirichlet boundary
velocity
end

if DEL_T ~= 0
  DX1 = XM1(1:NNODES);
  DY1 = XM1(NNODES+1:2*NNODES);
  DX1(BCX(:,1))=BCX(:,2);
  DY1(BCY(:,1))=BCY(:,2);
  if DOFNODE == 4
    VX1 = VM1(1:NNODES);
    VY1 = VM1(NNODES+1:2*NNODES);
    VX1(BCX(:,1))=BCX(:,3);
    VY1(BCY(:,1))=BCY(:,3);
  end
end

% scale applied nodal loads and put into a single vector
F_APPLIED(1:2*NNODES) = SCALE * [PTX; PTY];

% plot deformed structure
figure(3)
clg;
hold on
maxx=max([XN;YN]);
minx=min([XN;YN]);
maxdx=max([abs(DX);abs(DY)]);
if maxdx == 0
  maxdx = 1;
end
amp=.1*(maxx-minx)/maxdx;
amp=10;
plot(XN+amp*DX,YN+amp*DY,'y+')
axis('equal')
grid on
drawnow

%***************************** ELEMENT CALCULATIONS
**********************
% Form the RESIDUAL vector using local vector operations and an element
by
% element approach
%*********************************************************************
***
for j = 1:NEL
```

```
% check the shape of each element and set up its needed workspace size
  INDEX=find(ELMAT(j,:));  % INDEX contains the non-zero node numbers
for the element
  VERTICES = length(INDEX);

  NDOF = VERTICES * DOFNODE;                    % this element
  SLIST = ELMAT(j,INDEX);                       % index into variables stored
in data file

% psuedo time matrix terms
  if DEL_TAU ~= 0
    PSEUDO_MP = 1E5*MP(3)*eye(2);               %xxxx
  else
    PSEUDO_MP = zeros(2);
  end

% FLAT: Build flat matrix of element integration arrays for each
element
% Row 1 = sum of correct stiffness matrices for RHS operations
% Row 2 = sum of all pseudo-time mass matrices for LHS operations
% Row 3 = sum of physical time mass matrices
% Row 4 = sum of initial stress matrices

  FLAT = lin_grid(XN(SLIST),YN(SLIST),DX(SLIST),DY(SLIST),...
              MP,PSEUDO_MP,NN,DEL_T,DEL_TAU,DOFNODE);

% RHS and LHS use
% K0 = small displacement stiffness matrix
% KL = initial displacement matrix (a.k.a. large displacement matrix)
% LHS use only
% KT = K0+KL+KS = tangential stiffness matrix (a.k.a. total stiffness
matrix)
% KS = initial stress matrix (a.k.a. geometric matrix)

  STIFF_MAT = reshape(FLAT(1,1:NDOF^2),NDOF,NDOF);    % element
stiffness matrix [(KO + KL), 0; 0, -I] or [KO + KL]
  GSTIFF(j,:) = FLAT(2,:);                            % pseudo time
mass matrix and stiffness for LHS (incomplete)
  MASS_MAT = reshape(FLAT(3,1:NDOF^2),NDOF,NDOF);     % physical time
consistent mass matrix
  INIT_MAT = reshape(FLAT(4,1:NDOF^2),NDOF,NDOF);     % initial stress
matrix (KS)

  SHUFFLE = [];

  for ii = 1:DOFNODE                                  % Index into
residual vector
      SHUFFLE = [SHUFFLE, SLIST+NNODES*(ii-1)];
  end

  if DEL_T == 0                % form RHS residual for STEADY STATE
structural equations

    R(SHUFFLE(1:2*VERTICES))= R(SHUFFLE(1:2*VERTICES)) ...
```

```
                + STIFF_MAT(1:2*VERTICES,1:2*VERTICES) *
[DX(SLIST);DY(SLIST)] ...
                - F_APPLIED(SHUFFLE(1:2*VERTICES));

%    R(SHUFFLE)= R(SHUFFLE) ...
%                + STIFF_MAT *
[DX(SLIST);DY(SLIST);VX(SLIST);VY(SLIST)] ...
%                - F_APPLIED(SHUFFLE);
                                % NOTE: If STIFF_MAT is of the form
[K,0;0,-I], the -I will
                                % cause nonzero terms in the residual if
the velocity is not zero.

  else                          %   1st order time derivative = (Un-Un-
1)/DT
    R(SHUFFLE)= R(SHUFFLE) ...
                + STIFF_MAT * [DX(SLIST);DY(SLIST);VX(SLIST);VY(SLIST)]
...
                - F_APPLIED(SHUFFLE) ...
                + MASS_MAT ...
                * [DX(SLIST)-DX1(SLIST);DY(SLIST)-DY1(SLIST);...
                   VX(SLIST)-VX1(SLIST);VY(SLIST)-VY1(SLIST)];
  end
end


%************************************ BOUNDARY CONDITION ROUTINE 2 **
% Zero out Dirichlet boundary conditions in the residual
%*********************************************************************
[MBX, NBX] = size(BCX);
[MBY, NBY] = size(BCY);
if MBX > 0
  R(BCX(:,1)) = zeros(MBX,1);             % zeros residual for DX
dirichlet BC
  if DOFNODE ~= 2
    R(BCX(:,1)+2*NNODES) = zeros(MBX,1);  % zeros residual for VX
dirichlet BC
  end
end
if MBY > 0
  R(BCY(:,1)+NNODES) = zeros(MBY,1);      % zeros residual for DY
dirichlet BC
  if DOFNODE ~= 2
    R(BCY(:,1)+3*NNODES) = zeros(MBY,1);  % zeros residual for VY
dirichlet BC
  end
end


%************************************ SCALING ROUTINE ***************
% Scale the equations to make the associated residuals roughly the same
% order of magnitude for equivalent perturbations.  Greatly aids
% orthogonality in LHS modified Gram Schmidt process
%*********************************************************************
global GRID_SCALE
```

```
temp=length(R)/DOFNODE;
if DOFNODE == 4
  GRID_SCALE=[1E5;1E5;1E-3;1E-3];
%  GRID_SCALE=[1E5,1E5,1E-5,1E-5];
%  GRID_SCALE=[1E1;1E1;1E-3;1E-3];
%  GRID_SCALE=[1E1;1E1;1E-1;1E-1];
  R(2*temp+1:3*temp)=R(2*temp+1:3*temp)/GRID_SCALE(3);
  R(3*temp+1:4*temp)=R(3*temp+1:4*temp)/GRID_SCALE(4);
else
  GRID_SCALE=[1;1];
end

R(1:temp)=R(1:temp)/GRID_SCALE(1);
R(temp+1:2*temp)=R(temp+1:2*temp)/GRID_SCALE(2);

% plot the scaled residual
%figure(1)
%clg
%plot(RHSR)
%drawnow

global RHSR
RHSR=R;

%************************************ EIGENVALUE ROUTINE ************
% run 10 or 20 steps of an Arnoldi process to estimate the largest
% and smallest eigenvalues of (KO+KL+KS)/M.  Used to set damping
% coefficient and pseudo-time step for LHS operations.
%*********************************************************************
if DEL_TAU ~= 0

  [maxee,minee,minabs,error,total_iters] = karnoldi('str_eig',[1E-
9,20,1],...

ones(2*NNODES,1)/sqrt(2*NNODES),...

[DX;DY],[],XN,YN,ELMAT,BCX,BCY,2,4);

% determine c and del_tau
  ALPHA=4;
  C=ALPHA*sqrt(min(minee))
  DEL_TAU=2/sqrt(max(maxee))
%C=1
%DEL_TAU=1

% divide top right and lower left blocks by del_tau
% add c*upper right block/del_tau to KO+KL+KS

  [NEL,VERTICES] = size(ELMAT);     % VERTICES is the max number of
nodes in any element
  NDOF = VERTICES * DOFNODE;
  TEMP = zeros(NDOF);          % max space needed

  for j = 1:NEL

% check the shape of each element and set up its needed workspace size
```

```
    INDEX=find(ELMAT(j,:));  % INDEX contains the non-zero node numbers
for the element
    VERTICES = length(INDEX);
    V2 = 2*VERTICES;
    V4 = 4*VERTICES;
    NDOF = VERTICES * DOFNODE;                    % this element

    TEMP = zeros(NDOF);
    TEMP = reshape(GSTIFF(j,1:NDOF^2),NDOF,NDOF);     % unwrap element
GSTIFF matrix
    TEMP(1:V2,1:V2) = TEMP(1:V2,1:V2) + TEMP(1:V2,V2+1:V4)*C/DEL_TAU;
      % KT block
    TEMP(1:V2,V2+1:V4) = TEMP(1:V2,V2+1:V4)/DEL_TAU;
      % Mass block
    TEMP(V2+1:V4,1:V2) = TEMP(V2+1:V4,1:V2)/DEL_TAU;
      % Identity block

    GSTIFF(j,1:NDOF^2) = reshape(TEMP,1,NDOF^2);      % rewrap element
GSTIFF matrix
  end

% plot eigenvalues from each Arnoldi step to visualize convergence
  figure(4)
  plot(maxee)
  hold on
  plot(minee,'r')
end

figure(1)
clg
```

LIN_GRID

```
function FLAT =
lin_grid(XN,YN,DX,DY,MP,PSEUDO_MP,NN,DEL_T,DEL_TAU,DOFNODE);
%function FLAT =
lin_grid(XN,YN,DX,DY,MP,PSEUDO_MP,NN,DEL_T,DEL_TAU,DOFNODE);
%
% 4 Aug 98 - created from struct eqn
% 6 Nov 98 - updated to include system 1st order in time
%
% Calculates the internal forces at the nodes based on linear
% elastic material properties
%
% XN    = column vector of global X POSITION of element nodes
% YN    = column vector of global Y POSITION of element nodes
% DX    = column vector of global X DEFORMATION of element nodes (not
used)
% DY    = column vector of global Y DEFORMATION of element nodes (not
used)
% MP    = vector of material properties {E, Nu}.  Assumes constant
%       material properties and plane stress case (easily modified)
```

```
% PSEUDO_MP = artificial material properties for pseudo time mass
matrix
% NN     = integer describing the order of integration to use in
%        gauss quadrature (1,2,3 for linear, quadratic, cubic)
%        NN = 2 is the normal value
% DEL_T = physical time step size
% DEL_TAU = pseudo time step size
% DOFNODE number of degrees of freedom per node
%
% FLAT: Build flat matrix of element integration arrays for each
element
% Row 1 = sum of correct stiffness matrices for RHS operations
% Row 2 = sum of all PDE matrices for LHS operations
% Row 3 = sum of physical time mass matrices
% Row 4 = sum of initial stress matrices

% initialize
NNODES = length(XN);
NDOF = DOFNODE*NNODES;

% plane stress D11 = E/(1-Nu^2)  D12 = Nu*D11  D33 = (D11-D12)/2
% This could be moved inside the element loop if variable material
properties
% were needed
D11 = MP(1)/(1-MP(2)^2);
D12 = D11 * MP(2);
D33 = (D11 - D12)/2;

if NNODES == 3              % triangle
  REGION = 2;
elseif NNODES == 4          % rectangle
  REGION = 1;
end

FLAT = zeros(4,NDOF^2);
W    = zeros(NDOF);
WT   = zeros(NDOF);
STIFFNESS = zeros(NDOF);
KK = zeros(2*NNODES);
TR   = zeros(NDOF);
MASS = zeros(NDOF);
P_MASS = zeros(NDOF);

% FLAT memory area used here for convenience.  Definitions below will
% be updated before completion of routine to be consistent with calling
% routine.
%
% Build PDE integration for each element
% FLAT: Build flat matrix of element integration arrays for each
element
% Row 1 = sum of stiffness matrices for [DX;DY] operations
% Row 2 = sum of physical time mass matrices for [DX;DY] operations
% Row 3 = sum of pseudo-time mass matrices
% Row 4 = sum of initial stress matrices
```

```
FLAT =
gauss('grid_mat',NN,REGION,XN,YN,MP,PSEUDO_MP,DEL_T,DEL_TAU,DOFNODE);

STIFFNESS = reshape(FLAT(1,1:NDOF^2),NDOF,NDOF);      % [W,0;0,-I] or
[W]
MASS = reshape(FLAT(2,1:NDOF^2),NDOF,NDOF);
P_MASS = reshape(FLAT(3,1:NDOF^2),NDOF,NDOF);         % [0,P_M;I,0] or
[0]

% form correct stiffness matrix for element
W = STIFFNESS(1:2*NNODES,1:2*NNODES);

% block pseudo-transpose - reverses indices on indicial notation (Wac
vs. Wca)
WT = [W(1:NNODES,1:NNODES),      W(NNODES+1:2*NNODES, 1:NNODES);
      W(1:NNODES, NNODES+1:2*NNODES),
W(NNODES+1:2*NNODES,NNODES+1:2*NNODES)];

% block matrix trace
TR = [W(1:NNODES,1:NNODES)+W(NNODES+1:2*NNODES,NNODES+1:2*NNODES),
zeros(NNODES,NNODES);
      zeros(NNODES,NNODES),
W(1:NNODES,1:NNODES)+W(NNODES+1:2*NNODES,NNODES+1:2*NNODES)];

% linear stiffness matrix - technically not the same form as in
Zienkiewicz & Taylor
% but provides the same answer
KK = D12 * W + D33*(WT + TR);
STIFFNESS(1:2*NNODES,1:2*NNODES) = KK;                % [KK,0;0,-I] or
[W]

% FLAT: Build flat matrix of element integration arrays for each
element
% Row 1 = sum of correct stiffness matrices for RHS operations
% Row 2 = sum of all PDE matrices for LHS operations
% Row 3 = sum of physical time mass matrices
% Row 4 = sum of initial stress matrices

FLAT = zeros(4,NDOF^2);
%FLAT = [reshape(STIFFNESS,1,NDOF^2); ...
%        reshape(P_MASS + MASS + STIFFNESS,1,NDOF^2); ...
%        reshape(MASS,1,NDOF^2); ...
%        zeros(1,NDOF^2)];
FLAT = [reshape(STIFFNESS,1,NDOF^2); ...
        reshape(P_MASS + STIFFNESS,1,NDOF^2); ...
        reshape(MASS,1,NDOF^2); ...
        zeros(1,NDOF^2)];
```

GRID_MAT

```
function MFLAT =
grid_mat(ZETA,XN,YN,MP,PSEUDO_MP,DEL_T,DEL_TAU,DOFNODE)
%function MFLAT =
grid_mat(ZETA,XN,YN,MP,PSEUDO_MP,DEL_T,DEL_TAU,DOFNODE)
%
%  4 Aug 98 - created from struct eqn
%  6 Nov 98 - updated to include 1st order system in time
%
% Forms the tensor function Transpose(B)*B for the linear stiffness
% matrix using linear quadralateral (brick) or triangle shape
functions.
% Similar to the function (dNi/dXj) * transpose(dNi/dXj) * Area but
also
% forms the cross elements needed in the stiffness calculations.
%
% Also forms the pseudo time mass matrix for LHS operations and breaks
% pseudo time motion into a first order system of equations in DX, DY,
VX,
% and VY.
%
% The module checks ZETA to figure out which shape to use.
% Routine assumes a vector parameter (2-D) is to be distributed over
% the element.  Intended for use in gauss quadrature.
%
% ZETA = row vector of current position in local coordinates
% XN   = column vector of global X position of the element nodes
% YN   = column vector of global Y position of the element nodes
% MP    = vector of material properties {E, Nu, Rho}.  Assumes constant
%       material properties and plane stress case (easily modified)
% PSEUDO_MP = properties in pseudo time mass matrix [diag X, off XY;
off YX, diag Y]
% DEL_T = physical time step size
% DEL_TAU = pseudo time step size
% DOFNODE = number of degrees of freedom per node
%
%  MFLAT =  flat matrix of element integration matrices at the current
location
%  MFLAT = [reshape(B211,1,NDOF^2); ...
%           reshape(MASS,1,NDOF^2); ...
%           reshape(P_MASS,1,NDOF^2); ...
%           zeros(1,NDOF^2)] * AE;

NNODES = length(XN);
NDOF = NNODES*DOFNODE;
MFLAT = zeros(4,NDOF^2);
SHAPE=length(ZETA);

if SHAPE == 3                    % Triangle

  DZJDXK = zeros(3,2);
  DNIDZJ = zeros(3,3);
  NI     = zeros(3,1);
```

```
  DXKDZJ = zeros(3,2);
  DNIDXK = zeros(3,2);
  [DZJDXK, DNIDZJ, NI, DXKDZJ, AE] = triashap(ZETA, XN, YN);

elseif SHAPE == 2                % Brick

  DZJDXK = zeros(2,2);
  DNIDZJ = zeros(4,2);
  NI     = zeros(4,1);
  DXKDZJ = zeros(2,2);
  DNIDXK = zeros(4,2);
  [DZJDXK, DNIDZJ, NI, DXKDZJ, AE] = brikshap(ZETA, XN, YN);

end
DNIDXK = DNIDZJ*DZJDXK;

% physical time mass matrix
if DEL_T == 0
  MASS = zeros(NDOF);
else
  MASS = kron(eye(DOFNODE),NI*NI') / DEL_T;
end

% pseudo time mass matrix
P_MASS = zeros(NDOF);
if DEL_TAU == 0
                                    % do nothing
else
  % term needing c*M/del_tau will be created in STR_RHS using
eigenvalues
  P_MASS(1:2*NNODES,2*NNODES+1:4*NNODES) = kron(PSEUDO_MP,NI*NI');
  P_MASS(2*NNODES+1:4*NNODES,1:2*NNODES) = kron(eye(2),NI*NI');
end

% basis for stiffness matrix
B211 = [DNIDXK(:,1);DNIDXK(:,2)] * [DNIDXK(:,1);DNIDXK(:,2)]';
if DOFNODE == 2
  STIFFNESS = B211;
else
  STIFFNESS = [B211,zeros(size(B211));zeros(size(B211)),-
kron(eye(2),NI*NI')];
end

% MFLAT: Build flat matrix of element integration arrays for each
element
% Row 1 = sum of stiffness matrices for [DX;DY;VX;VY] operations
% Row 2 = sum of physical time mass matrices for [DX;DY;VX;VY]
operations
% Row 3 = sum of pseudo-time mass matrices [DX;DY;VX;VY]
% Row 4 = sum of initial stress matrices

MFLAT = [reshape(STIFFNESS,1,NDOF^2); ...
         reshape(MASS,1,NDOF^2); ...
         reshape(P_MASS,1,NDOF^2); ...
         zeros(1,NDOF^2)] * AE;
```

GRID_LHS

```
function R =
grid_lhs(NODE_VEC,dummy,XN,YN,ELMAT,PTX,PTY,SCALE,BCX,BCY,MP,DEL_T,DEL_
TAU,NN, ...
                    DOFNODE,XM1,VM1)

% function R =
grid_lhs(NODE_VEC,dummy,XN,YN,ELMAT,PTX,PTY,SCALE,BCX,BCY,MP,DEL_T,DEL_
TAU,NN, ...
%                   DOFNODE,XM1,VM1)
%
% 29 Aug 98 - created from structural left hand side
%
% Combines different routines to form the static or dynamic grid
deformation equations
% based on treating it as a solid continuum.  Treats time and pseudo-
time as a system
% of first order differential equations
%
% c*d/dt (dx;dy) + rho*d/dt (u;v) + K*(dx; dy)                = 0
%   d/dt (dx;dy)                                 -  d/dt (u;v) = 0
%
% [c, rho]*transpose{dx, dy, u, v}+[K, 0]*transpose{dx, dy, u, v}=0
% [I,  0 ]      dt                  [0,-I]
%
% Output is the residual (want R = 0 at equilibrium).
%
% INPUT VARIABLES:
% NODE_VEC = nodal deformation vector (DX; DY; U; V)
% dummy = place holder for GMRES solver
% XN = X location of nodes
% YN = Y location of nodes
% ELMAT = matrix listing the nodes assigned to each element
% PTX = external point (nodal) loads in the X direction
% PTY = external point (nodal) loads in the Y direction
% SCALE = scaling factor used to gradually apply loads
% BCX = X deformation (and velocity) Dirichlet boundary conditions
[node #, X, VX, solid node #]
% BCY = Y deformation (and velocity) Dirichlet boundary conditions
[node #, Y, VY, solid node #]
% MP = material properties (E, Nu, density)                    (not
used on LHS)
% DEL_T = physical time step size                              (not
used on LHS)
% DEL_TAU = pseudo time step size                              (not
used on LHS)
% NN = order of gauss integration                              (not
used on LHS)
% DOFNODE = number of degrees of freedom per node (4 with pseudo time)
% XM1 = [DX;DY] at t-1                                         (not
used on LHS)
% VM1 = [VX;VY] at t-1                                         (not
used on LHS)
```

```
%
% OUTPUT VARIABLES:
% R = residual of summation of external and internal forces
%

global GSTIFF

% initialize variables and workspace
NNODES = length(XN);
R = zeros(size(NODE_VEC));          % Space for RESIDUAL vector
F_APPLIED = zeros(size(NODE_VEC));  % Space for APPLIED LOAD vector
[NEL,VERTICES] = size(ELMAT);       % VERTICES is the max number of
nodes in any element

LHS_MAT = zeros(VERTICES*DOFNODE);  % Set up max space for LHS jacobian
matrix
SHUFFLE = zeros(1,VERTICES*DOFNODE);% Index

%************************************ BOUNDARY CONDITION ROUTINE 1 **
% Form deformation vectors and apply dirichlet boundary conditions
%*************************************************************************
DX = NODE_VEC(1:NNODES);
DY = NODE_VEC(NNODES+1:2*NNODES);
DX(BCX(:,1))=BCX(:,2);              % set dirichlet X boundary
deformation
DY(BCY(:,1))=BCY(:,2);              % set dirichlet Y boundary
deformation
if DOFNODE == 4
  VX = NODE_VEC(2*NNODES+1:3*NNODES);
  VY = NODE_VEC(3*NNODES+1:4*NNODES);
  VX(BCX(:,1))=BCX(:,3);                % set dirichlet X boundary
velocity
  VY(BCY(:,1))=BCY(:,3);                % set dirichlet Y boundary
velocity
end

if DEL_T ~= 0
  DX1 = XM1(1:NNODES);
  DY1 = XM1(NNODES+1:2*NNODES);
  DX1(BCX(:,1))=BCX(:,2);
  DY1(BCY(:,1))=BCY(:,2);
  VX1 = VM1(1:NNODES);
  VY1 = VM1(NNODES+1:2*NNODES);
  VX1(BCX(:,1))=BCX(:,3);
  VY1(BCY(:,1))=BCY(:,3);
end

% scale applied nodal loads and put into a single vector
F_APPLIED(1:2*NNODES) = SCALE * [PTX; PTY];

%***************************** ELEMENT CALCULATIONS
**********************
% Form the RESIDUAL vector using local vector operations and an element
by
% element approach
```

```
%*********************************************************************
***
for j = 1:NEL

% check the shape of each element and set up its needed workspace size

  INDEX=find(ELMAT(j,:));     % INDEX contains the non-zero node
numbers for the element
  VERTICES = length(INDEX);
  NDOF = VERTICES * DOFNODE;  % NDOF for the element
  SLIST = ELMAT(j,INDEX);     % index into variables stored in data
file

  LHS_MAT = reshape(GSTIFF(j,1:NDOF^2),NDOF,NDOF);    % recall stored
LHS integration matrix

  SHUFFLE = [];
  for ii = 1:DOFNODE                                   % Index into
residual vector
    SHUFFLE = [SHUFFLE, SLIST+NNODES*(ii-1)];
  end

% form LHS residual for structural equations
  if DOFNODE == 4
    R(SHUFFLE)= R(SHUFFLE) ...
                + LHS_MAT * [DX(SLIST);DY(SLIST);VX(SLIST);VY(SLIST)]
...
                - F_APPLIED(SHUFFLE);
  else
    R(SHUFFLE)= R(SHUFFLE) ...
                + LHS_MAT * [DX(SLIST);DY(SLIST)] ...
                - F_APPLIED(SHUFFLE);
  end
end


%********************************** BOUNDARY CONDITION ROUTINE 2 **
% Zero out Dirichlet boundary conditions in the residual
%*********************************************************************
[MBX, NBX] = size(BCX);
[MBY, NBY] = size(BCY);
if MBX > 0
  R(BCX(:,1)) = zeros(MBX,1);              % Zero out X boundary
position residual
  if DOFNODE == 4
    R(BCX(:,1)+2*NNODES) = zeros(MBX,1);  % Zero out X boundary
velocity residual
  end
end
if MBY > 0
  R(BCY(:,1)+NNODES) = zeros(MBY,1);       % Zero out Y boundary
position residual
  if DOFNODE == 4
    R(BCY(:,1)+3*NNODES) = zeros(MBY,1);  % Zero out Y boundary
velocity residual
  end
```

```
end

%************************************ SCALING ROUTINE ***************
% Scale the equations to make the associated residuals roughly the same
% order of magnitude for equivalent perturbations.  Greatly aids
% orthogonality in LHS modified Gram Schmidt process
%*********************************************************************
global GRID_SCALE

temp=length(R)/DOFNODE;
if DOFNODE == 4
  R(2*temp+1:3*temp)=R(2*temp+1:3*temp)/GRID_SCALE(3);
  R(3*temp+1:4*temp)=R(3*temp+1:4*temp)/GRID_SCALE(4);
end

R(1:temp)=R(1:temp)/GRID_SCALE(1);
R(temp+1:2*temp)=R(temp+1:2*temp)/GRID_SCALE(2);
```

APPENDIX H

COUPLED MASTER ROUTINES

APPENDIX H

COUPLED MASTER ROUTINES

The sequence of operations within a fluid/solid/grid domain is the same whether performing a loosely coupled fluid-structure calculation or one that is tightly coupled. The difference is the order in which the various right hand side (RHS) and left hand side (LHS) domains are addressed and how frequently the boundary interactions between the domains are updated. This is completely determined by high-level routines whose purpose is to form right and/or left hand side residuals.

For example, in the loosely coupled approach the fluid conservation laws are solved to convergence using both linear (inner/GMRES) and nonlinear (outer/Newton) with the fluid grid motion and solid deformation frozen. Only fluid flow residuals are passed to the partial differential equation solver. This fluid solution is then used to update loads on the solid. Next, the solid deformation is solved to convergence with fluid loads frozen. The solid solution (deformation) is used to update Dirichlet boundary conditions for the fluid grid location. Finally, the fluid grid deformation is solved to convergence with physical solid deformations frozen. The process is repeated until the individual domain solutions stop changing and the domain residuals are sufficiently small. This is shown graphically in Figure H-1. Outer/Newton iterations are performed in each domain to reach convergence so the level (m) represents an overall convergence level.

In the tightly coupled solution process, all domains are updated with the latest information from each of the other domains each time a residual is formed. The residual passed to the partial differential equation solver is a single vector made up of the residual vectors from each of the individual domains. This composite residual vector is then treated as any other residual by the numerical solver. The high level routines responsible for forming the composite residual simply call the same subroutines as above to form residuals for each of the domains and then stack them together into a single vector. This is true for both the left and right hand side residuals. The impact is most prominent on the left hand side. Each inner/GMRES iteration (guess) has updated information from all domains allowing the domains to fully interact. This is shown graphically in Figure H-2 where the levels (n) are now simple outer/Newton iterations and the

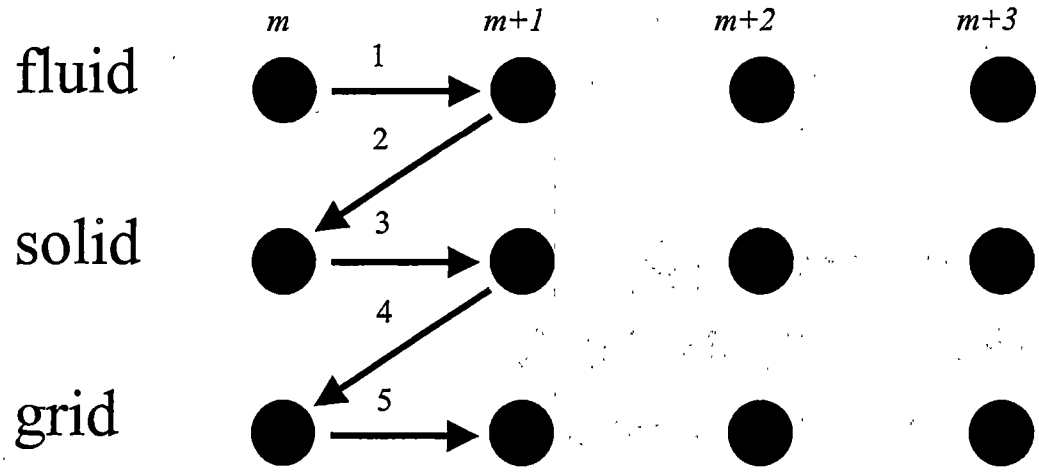


Figure H-1: Loosely Coupled Solution Process

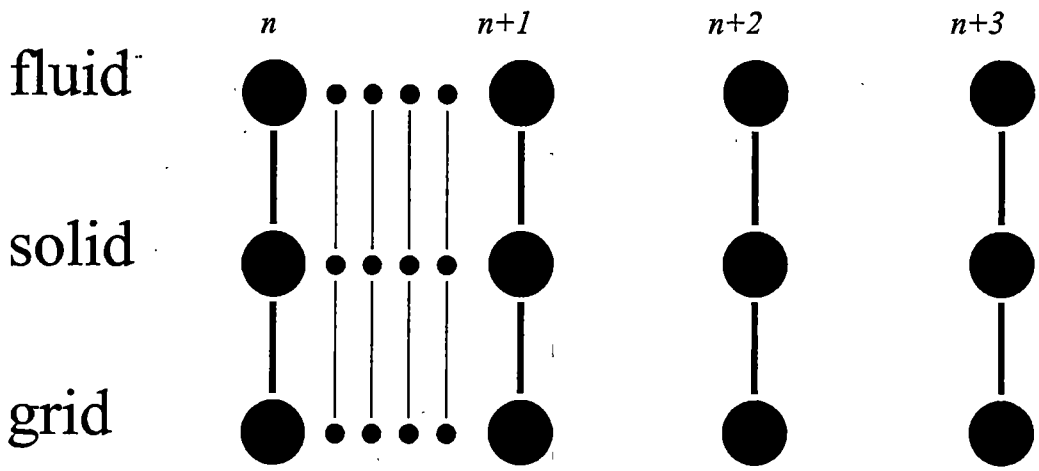


Figure H-2: Tightly Coupled Solution Process

overall system converges as the Newton iteration progresses.

Figure H-3 illustrates how the top level routines interact. ***Master*** is an overarching routine that establishes workspace, initializes vectors and matrices, and sets numerical solver parameters such as number of Krylov vectors, number of GMRES restarts allowed, termination criteria, etc. This is also the level where domain iterations (outside the nonlinear Newton iteration) are performed, if needed. The ***f_rhs*** and ***f_lhs*** routines create the respective right and left hand side residuals and may call conservation laws for a single domain or create a composite residual vector by calling conservation law routines from multiple domains.

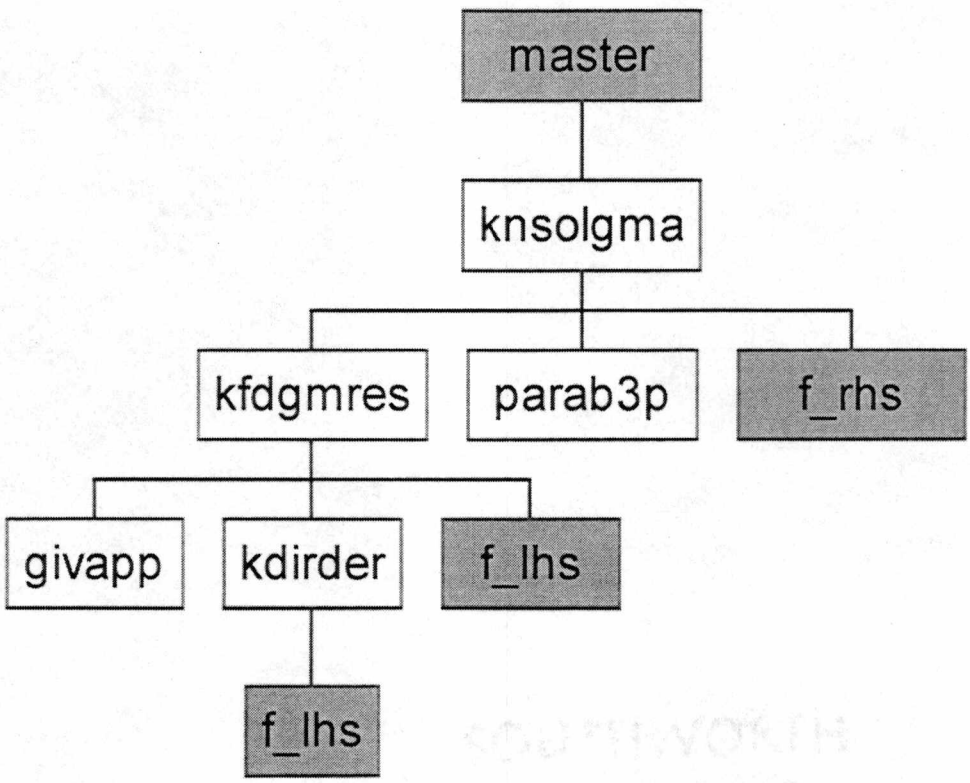


Figure H-3: Top Level Module Interactions

Master Routine Operations

Single domain

1. Set iteration parameters for domain
2. Open domain initialization file
3. Create workspace
4. Initialize vectors and matrices
5. Launch partial differential equation solver using right and left hand side files for single domain

Loosely coupled multiple domain (mas_123)

1. Open domain initialization file
2. Set iteration parameters
3. Create workspace
4. Initialize vectors and matrices
5. Launch partial differential equation solver using right and left hand side files for single domain
6. Update boundary conditions
7. Return to 1. and repeat for next domain
8. Continue until reaching termination criteria

Tightly coupled multiple domain (fsg_mas)

1. Set best iteration parameters for multiple domain residual vector
2. Open domain initialization file
3. Create workspace
4. Initialize portion of vectors and matrices for domain
5. Return to 2. and repeat for next domain
6. Launch partial differential equation solver using right (*fsg_rhs*) and left (*fsg_lhs*) hand side files for multiple domains

The loosely coupled domain approach calls the same left and right hand side files described in Appendices E through G. The tightly coupled domain approach ultimately uses these files as well but adds a left and right hand "organizer" file to call each of the domain files, update boundary conditions, and consolidate the residuals into a single vector. These functions are performed in the overall master routine in the loosely coupled approach.

Right Hand Side (RHS) Operations

fluid

1. Input current pressure (P), X velocity (U), Y velocity (V) and temperature (T) for fluid at grid nodes
2. Input original X and Y location of fluid grid nodes
3. Input current displacements (DX and DY) and velocities (VX and VY) for fluid grid nodes
4. Form $X_{current}$=X+DX and $Y_{current}$ =Y+DY to get updated grid shape
5. Input boundary conditions for P, U, V, T
6. Form element matrices for LHS fluid delta conservation laws
7. Form and store element matrices for LHS fluid grid motion delta conservation law terms
8. Assemble RHS residual for nonlinear fluid conservation laws

solid

1. Input original X and Y locations for solid grid nodes
2. Input current displacements (DX and DY) and velocities (VX and VY) for solid grid nodes
3. Input pressure (P) loads for solid boundary follower forces from fluid flow calculations
4. Input Dirichlet boundary conditions for X, Y and VX, VY
5. Form and store element matrices for LHS solid delta conservation laws
6. Assemble RHS residual for solid (possibly nonlinear) conservation laws on solid grid

grid

1. Input original X and Y locations for fluid grid nodes
2. Input current displacements (DX and DY) and velocities (VX and VY) for fluid grid nodes
3. Input Dirichlet boundary conditions for X, Y and VX, VY from solid grid calculations
4. Form and store element matrices for LHS solid delta conservation laws
5. Assemble RHS residual for solid (possibly nonlinear) conservation laws on fluid grid

Left Hand Side (LHS) Operations

fluid

1. Input current guess for P, U, V, and T at fluid grid nodes
2. Input boundary conditions for P, U, V, and T
3. Unfold element matrices for LHS fluid delta conservation laws
4. Perform initial assembly of LHS residual for linearized fluid delta conservation laws

fluid adjustments for grid motion

1. Input current guess for DX, DY, VX, and VY for fluid grid nodes
2. Input boundary conditions for DX, DY, VX, and VY
3. Unfold element matrices for LHS fluid grid motion delta conservation law term
4. Assemble update to LHS residual for fluid grid motion delta conservation law term
5. Add residual update to initial LHS residual

solid

1. Input current guess for DX, DY, VX, and VY for solid grid nodes
2. Input boundary conditions for DX, DY, VX, and VY
3. Unfold element matrices for LHS solid delta conservation laws
4. Assemble LHS residual for solid delta conservation laws on solid grid

grid

1. Input current guess for DX, DY, VX, and VY for fluid grid nodes
2. Input boundary conditions for DX, DY, VX, and VY
3. Unfold element matrices for LHS solid delta conservation laws
4. Assemble LHS residual for solid delta conservation laws on fluid grid

MAS_123

```
% mas_123
% Master script program for fluid, structure, grid motion
%
% common variables
% XN, YN = vector of X and Y locations of nodes
```

```
%          fluid: current location
%          structure and grid: original location
% ELMAT   = nodes assigned to elements
% MP      = material properties
%          fluid MP = [R, CP, GAMMA]
%                     (R...ft^2/sec^2,CP...ft^2/sec^2, gamma)
%          structure and grid MP = [E, NU, RHP]
%                                  (E...lbf/sqft, Nu,
density...lbm/cuft)
% DOFNODE = degrees of freedom per node
%          fluid = 4
%          structure and grid = 2
% NN =       order of gauss integration (1,2, or 3), 2 for all
% UN =       vector of X velocity of nodes
% VN =       vector of Y velocity of nodes
% DEL_T =    physical time step size
% DEL_TAU = pseudo time step size
% A =       connectivity matrix for viewing grids
% numbers = node numbers to use with connectivity matrix
%
% fluid only
% QINITIAL = matrix of initial conditions for primitive nodal variables
[P, U, V, T]
% BCTYPE =  vector of nodal definition of boundary condition type
%           not necessarily the most efficient implementation but it
forces
%           a single definition of boundary state for nodes
%           type 0 = not on a boundary
%           type 1 = inflow/outflow
%           type 5 = impermeable wall
%           type 6 = combination wall and inflow/outflow (i.e. a corner
node)
%           type 21= definitely inflow
%           type 22= definitely outflow
%           type 26= combination wall and definitely inflow
%           type 27= combination wall and definitely outflow
% BC1     =  matrix of type 1, 6, 21, 22, 26 & 27 boundary conditions
%           [node number, P, U, V, T]
% MOUT_PTR =nodal vector giving index for boundary nodes in MOUT matrix
%
% structure only
% P =      vector of pressure at each node and a cross reference to a
fluid
%          flow node producing the pressure [P, cross reference]
% P_REF = reference pressure for structure surface away from flow
%
% structure and grid only
% NODE_VEC = nodal deformation and velocity vector [DX; DY; (U; V)]
% dummy = place holder
% PTX = point (nodal) loads in the X direction
% PTY = point (nodal) loads in the Y direction
% SCALE = scaling factor used to gradually apply loads
% BCX = X deformation (and velocity) Dirichlet boundary conditions
%       [node #, X, VX, solid node #]
% BCY = Y deformation (and velocity) Dirichlet boundary conditions
%       [node #, Y, VY, solid node #]
```

```
% XM1 = [DX;DY] at t-1
% VM1 = [VX;VY] at t-1
%
% global variables
%
% fluid
% MFLOW =   flat file containing summation of element integration
matrices for LHS
% MOUT  =   matrix of nodal RH rule tangent vectors and boundary type
information
%           [node number, dx, dy, type (1 - 54), index into BC1, DX,
DY]
%           BC conditions 1=supersonic inflow,  2=subsonic inflow
%                         3=supersonic outflow, 4=subsonic outflow
%                         5=impermeable wall,
%                         51=wall+super in,     52=wall+sub in
%                         53=wall+super out,    54=wall+sub out
%           [dx,dy] = wall tangent and in/out tangent
%           [DX,DY] = in/out tangent if [dx,dy] has wall tangent for a
corner,
%                     otherwise original direction of velocity vector
% GFLOW =   flat file containing summation of element integration
matrices for LHS
%           grid motion in pseudo time
%
% structure
% STIFF =   flat file containing summation of element integration
matrices for LHS
%           structural internal force vector (stiffness), physical time
(if used)
%           and pseudo time (if used) calculations
%
% grid
% GSTIFF =  flat file containing summation of element integration
matrices for LHS
%           grid internal force vector (stiffness), physical time (if
used) and
%           pseudo time (if used) calculations

global MFLOW MOUT GFLOW STIFF GSTIFF

T_STEP=1;               % max time steps
DEL_T = 0;              % physical time step size, 0 = steady state

flops(0);
tic

%************************************************************************
**************
% Set up space for global matrices
%************************************************************************
**************

FLUID_NAME = 'nozzle2c'
FLUID_NNODES = 192
FLUID_DOFNODE = 4
```

```
FLUID_DOF = FLUID_DOFNODE * FLUID_NNODES
SOLID_NAME = 'wall2b'
SOLID_NNODES = 48
SOLID_DOFNODE = 2
SOLID_DOF = SOLID_DOFNODE * SOLID_NNODES
GRID_NAME = 'nozzle2g'
GRID_NNODES = FLUID_NNODES
GRID_DOFNODE = 2
GRID_DOF = GRID_DOFNODE * GRID_NNODES

SOLN = zeros(FLUID_DOF+SOLID_DOF+GRID_DOF,1);

%********************************** FLUID
*****************************************

% Initialize fluid global matrices for element by element storage
eval(['load ',FLUID_NAME]);
[NEL,LARGEST]=size(ELMAT);
MFLOW=zeros(NEL,(FLUID_DOFNODE*LARGEST)^2);      % LHS element
integration matrix

LBC = length(find(BCTYPE));          % number of boundary nodes
MOUT = zeros(LBC,7);                 % [node number,dx,dy,type,index
into BC1,DX,DY]
MOUT(:,1)=find(BCTYPE);              % load boundary node numbers

GFLOW = zeros(NEL,2*FLUID_DOFNODE*LARGEST^2);    % Fluid flow correction
for grid motion

% load fluid initial conditions
FLUID_INDEX = [1:FLUID_DOF];
SOLN(FLUID_INDEX) = reshape(QINITIAL,FLUID_NNODES*FLUID_DOFNODE,1);

%********************************** SOLID
*****************************************

% Initialize solid global matrix for element by element storage
eval(['load ',SOLID_NAME]);
[NEL,LARGEST]=size(ELMAT);
STIFF=zeros(NEL,(SOLID_DOFNODE*LARGEST)^2);      % LHS element
integration matrix

SNODES = length(XN);

% set initial solid position and velocity
DXS = zeros(SNODES,1);
DYS = zeros(SNODES,1);
if SOLID_DOFNODE == 4
  VXS = zeros(SNODES,1);
  VYS = zeros(SNODES,1);
else
  VXS = [];
  VYS = [];
end
```

```
%********************************** GRID
*****************************************

% Initialize grid global matrix for element by element storage
eval(['load ',GRID_NAME]);
[NEL,LARGEST]=size(ELMAT);
GSTIFF=zeros(NEL,(GRID_DOFNODE*LARGEST)^2);     % LHS element
integration matrix

GNODES = length(XN);

% set grid position and velocity
DXG = zeros(GNODES,1);
DYG = zeros(GNODES,1);
if GRID_DOFNODE == 4
  VXG = zeros(GNODES,1);
  VYG = zeros(GNODES,1);
else
  VXG = [];
  VYG = [];
end

%************************************************************************
**************
% Set up for physical time or load increment stepping
%************************************************************************
**************

SOLN_TM1 = [];
SOLN_TM2 = [];

FNRM=1;
DELTA_SOLN = 1;
ITC = 0;

% start physical time or load increment stepping iterations
while(DELTA_SOLN > 1E-9 & ITC < 5)
  ITC=ITC+1;

% update for actual time stepping
  SOLN_TM2 = SOLN_TM1;
  SOLN_TM1 = SOLN;

% Set up figure windows
  figure(1)
  clg
  set(1,'Position',[2 30 447 340]);
  figure(2)
  clg
  set(2,'Position',[456 414 371 310]);
  figure(3)
  clg
  set(3,'Position',[2 415 447 308]);
  figure(4)
  clg
  set(4,'Position',[456 30 370 339]);
```

```
%***************************** START FLUID
**************************************
%
%*************************************************************************
*************

eval(['load ',FLUID_NAME]);          % open fluid file
FNODES = length(XN);
INDEX = [1 : FLUID_DOF]';

QP = SOLN(INDEX);                    % initially set to QINITIAL
QP1=QP;
QP2=QP;
DEL_TAU = 1;

if GRID_DOFNODE == 4                 % use grid position and velocity
from above
  % do nothing
elseif DEL_TAU ~= 0                  % calculate grid velocity if not
available
  VXG = DXG / DEL_TAU;
  VYG = DYG / DEL_TAU;
else
  VXG = zeros(size(DXG));
  VYG = zeros(size(DYG));
end

  TOL=[1E-6,1E-6,4];
  LIN_ITS = 88;
  PARMS=[20,88,.5,2,5];

figure(1)
clg
figure(2)
clg

%if ITC ~= 1
  [SOLN(INDEX), F_HISTORY, F_IERROR, F_STATS] = knsolgma...
             ('FLOW_LHS','BLOK_RHS',TOL,PARMS,QP,QP,...

XN+DXG,YN+DYG,VXG,VYG,QP1,QP2,DEL_T,DEL_TAU,MP,ELMAT,BCTYPE,...
             BC1,MOUT_PTR,FLUID_DOFNODE,NN);
ITC
disp(['Fluid results'])
disp([F_STATS(:,1),zeros(size(F_STATS(:,2))),F_STATS(:,3:5)])
format long
disp(F_STATS(:,2))
format short

% Update pressure, velocity, temperature
QP = SOLN(INDEX);

% Enforce fluid boundary conditions

QP = reshape(QP,FNODES,FLUID_DOFNODE);
```

```
QP = bc1_lhs(QP,BC1);

SOLN(INDEX) = reshape(QP,FNODES*FLUID_DOFNODE,1);
toc

%end

%****************************** END FLUID
************************************
% END FLUID END FLUID END FLUID END FLUID END FLUID END FLUID END FLUID
END FLUID


%****************************** START STRUCTURE
************************************
%
%************************************************************************
*************

eval(['load ',SOLID_NAME]);                          % open solid file
SNODES = length(XN);
INDEX = [FLUID_DOF+1 : FLUID_DOF+SOLID_DOF]';

NODE_VEC = [DXS;DYS;VXS;VYS];

% need actual node numbers to tie boundary conditions together in fluid
and solid
% frames of reference

P_CROSS_REF = find(P(:,2)~=0);
P(P_CROSS_REF,1) = QP(P(P_CROSS_REF,2),1);           % get pressure
distribution

% calculate updated nodal forces from pressure loads
[PTXS,PTYS] = press(XN+DXS,YN+DYS,ELMAT,P(:,1),NN,SOLID_DOFNODE);

% remove load at dirichlet boundary nodes
PTXS(BCX(:,1)) = PTXS(BCX(:,1))*0;
PTYS(BCY(:,1)) = PTYS(BCY(:,1))*0;

SCALE = 1;
XT_1 = [];
VT_1 = [];
DEL_TAU = 0;

figure(1)
clg
figure(2)
clg

% set some Newton (outer) and GMRES (inner) iteration parameters
  TOL=[1E-6,1E-6,2];
  LIN_ITS = 88;
  PARMS=[10,LIN_ITS,.5,2,8];

  [SOLN(INDEX), S_HISTORY, S_IERROR, S_STATS] = knsolgma...
```

```
                    ('STR_LHS','STR_RHS',TOL,PARMS,NODE_VEC,...
NODE_VEC,XN,YN,ELMAT,PTXS,PTYS,SCALE,BCX,BCY,MP,...
                     DEL_T,DEL_TAU,NN,SOLID_DOFNODE,[1,0]);

ITC
disp(['Solid results'])
disp([S_STATS(:,1),zeros(size(S_STATS(:,2))),S_STATS(:,3:5)])
format long
disp(S_STATS(:,2))
format short

% update solid position and velocity
DXS = SOLN(INDEX(1):INDEX(SNODES));
DYS = SOLN(INDEX(SNODES+1):INDEX(2*SNODES));
if SOLID_DOFNODE == 4
  VXS = SOLN(INDEX(2*SNODES+1):INDEX(3*SNODES));
  VYS = SOLN(INDEX(3*SNODES+1):INDEX(4*SNODES));
else
  VXS = [];
  VYS = [];
end

% Enforce structure boundary conditions
DXS(BCX(:,1))=BCX(:,2);              % set dirichlet X boundary
deformation
DYS(BCY(:,1))=BCY(:,2);              % set dirichlet Y boundary
deformation
if SOLID_DOFNODE == 4
  VXS(BCX(:,1))=BCX(:,3);            % set dirichlet X boundary velocity
  VYS(BCY(:,1))=BCY(:,3);            % set dirichlet Y boundary velocity
end

SOLN(INDEX)=[DXS;DYS;VXS;VYS];
toc

%*************************** END STRUCTURE
****************************************
% END STRUCTURE END STRUCTURE END STRUCTURE END STRUCTURE END STRUCTURE
END STRUCTURE


%*************************** START GRID
****************************************
%
%*************************************************************************
*************

eval(['load ',GRID_NAME]);           % open grid file
GNODES = length(XN);
INDEX = [FLUID_DOF+SOLID_DOF+1 : FLUID_DOF+SOLID_DOF+GRID_DOF]';

% Grid position and velocity are already available.  Update velocity as
it may
% have been changed for fluid calculation
if GRID_DOFNODE ~= 4
```

```
  VXG = [];
  VYG = [];
end

% Need actual node numbers to tie boundary conditions together in grid
and solid
% frames of reference.  Cross reference in BCX and BCY provides
connections.  Must
% be repeated here for use in grid residual calculation.

X_CROSS_REF = find(BCX(:,4)~=0);
Y_CROSS_REF = find(BCY(:,4)~=0);

BCX(X_CROSS_REF,2) = DXS(BCX(X_CROSS_REF,4));
BCY(Y_CROSS_REF,2) = DYS(BCY(Y_CROSS_REF,4));

if GRID_DOFNODE == 4 & SOLID_DOFNODE == 4
  BCX(X_CROSS_REF,3) = VXS(BCX(X_CROSS_REF,4));
  BCY(Y_CROSS_REF,3) = VYS(BCY(Y_CROSS_REF,4));
end

SCALE = 1;
XT_1 = [];
VT_1 = [];
DEL_TAU = 0;

NODE_VEC = [DXG;DYG;VXG;VYG];

figure(1)
clg
figure(2)
clg

% set some Newton (outer) and GMRES (inner) iteration parameters
  TOL=[1E-6,1E-6,2];
  LIN_ITS = 88;
  PARMS=[10,LIN_ITS,.5,2,8];

  [SOLN(INDEX), G_HISTORY, G_IERROR, G_STATS] = knsolgma...
                    ('grid_lhs','grid_rhs',TOL,PARMS,NODE_VEC,...

NODE_VEC,XN,YN,ELMAT,PTX,PTY,SCALE,BCX,BCY,MP,DEL_T,DEL_TAU, ...
                     NN,GRID_DOFNODE,[1,0]);

ITC
disp(['Grid results'])
disp([G_STATS(:,1),zeros(size(G_STATS(:,2))),G_STATS(:,3:5)])
format long
disp(G_STATS(:,2))
format short

% update grid position and velocity
DXG = SOLN(INDEX(1):INDEX(GNODES));
DYG = SOLN(INDEX(GNODES+1):INDEX(GRID_DOF));
if GRID_DOFNODE == 4
  VXG = SOLN(INDEX(2*GNODES+1):INDEX(3*GNODES));
```

```
  VYG = SOLN(INDEX(3*GNODES+1):INDEX(4*GNODES));
else
  VXG = [];
  VYG = [];
end

% Enforce grid boundary conditions
DXG(BCX(:,1))=BCX(:,2);              % set dirichlet X boundary
deformation
DYG(BCY(:,1))=BCY(:,2);              % set dirichlet Y boundary
deformation
if SOLID_DOFNODE == 4
  VXG(BCX(:,1))=BCX(:,3);            % set dirichlet X boundary velocity
  VYG(BCY(:,1))=BCY(:,3);            % set dirichlet Y boundary velocity
end

SOLN(INDEX) = [DXG;DYG;VXG;VYG];
toc
%************************** END GRID
*****************************************
% END GRID END GRID END GRID END GRID END GRID END GRID END GRID END
GRID END GRID


  DELTA_SOLN = max(abs(SOLN-SOLN_TM1))
  FNRM = norm(SOLN-SOLN_TM1)/sqrt(length(SOLN))

end   % physical time or load increment stepping


toc
flops
```

FSG_MAS

```
% Master script program for fluid, structure, grid motion
%
% common variables
% XN, YN = vector of X and Y locations of nodes
%          fluid: current location
%          structure and grid: original location
% ELMAT  = nodes assigned to elements
% MP     = material properties
%          fluid MP = [R, CP, GAMMA]
%                     (R...ft^2/sec^2,CP...ft^2/sec^2, gamma)
%          structure and grid MP = [E, NU, RHP]
%                                  (E...lbf/sqft, Nu,
density...lbm/cuft)
% DOFNODE = degrees of freedom per node
%          fluid = 4
%          structure and grid = 2
% NN =      order of gauss integration (1,2, or 3), 2 for all
```

```
% UN =       vector of X velocity of nodes
% VN =       vector of Y velocity of nodes
% DEL_T =    physical time step size
% DEL_TAU =  pseudo time step size
% A =        connectivity matrix for viewing grids
% numbers =  node numbers to use with connectivity matrix
%
% fluid only
% QINITIAL = matrix of initial conditions for primitive nodal variables
[P, U, V, T]
% BCTYPE =   vector of nodal definition of boundary condition type
%            not necessarily the most efficient implementation but it
forces
%            a single definition of boundary state for nodes
%            type 0 = not on a boundary
%            type 1 = inflow/outflow
%            type 5 = impermeable wall
%            type 6 = combination wall and inflow/outflow (i.e. a corner
node)
%            type 21= definitely inflow
%            type 22= definitely outflow
%            type 26= combination wall and definitely inflow
%            type 27= combination wall and definitely outflow
% BC1     =  matrix of type 1, 6, 21, 22, 26 & 27 boundary conditions
%            [node number, P, U, V, T]
% MOUT_PTR =nodal vector giving index for boundary nodes in MOUT matrix
%
% structure only
% P =        vector of pressure at each node and a cross reference to a
fluid
%           flow node producing the pressure [P, cross reference]
% P_REF =   reference pressure for structure surface away from flow
%
% structure and grid only
% NODE_VEC = nodal deformation and velocity vector [DX; DY; (U; V)]
% dummy =   place holder
% PTX =     point (nodal) loads in the X direction
% PTY =     point (nodal) loads in the Y direction
% SCALE =   scaling factor used to gradually apply loads
% BCX =     X deformation (and velocity) Dirichlet boundary conditions
%          [node #, X, VX, solid node #]
% BCY =     Y deformation (and velocity) Dirichlet boundary conditions
%          [node #, Y, VY, solid node #]
% XM1 =     [DX;DY] at t-1
% VM1 =     [VX;VY] at t-1
%
% global variables
%
% fluid
% MFLOW =    flat file containing summation of element integration
matrices for LHS
% MOUT  =    matrix of nodal RH rule tangent vectors and boundary type
information
%            [node number, dx, dy, type (1 - 54), index into BC1, DX,
DY]
%            BC conditions 1=supersonic inflow,  2=subsonic inflow
```

```
%                          3=supersonic outflow, 4=subsonic outflow
%                          5=impermeable wall,
%                          51=wall+super in,     52=wall+sub in
%                          53=wall+super out,    54=wall+sub out
%           [dx,dy] = wall tangent and in/out tangent
%           [DX,DY] = in/out tangent if [dx,dy] has wall tangent for a
corner,
%                     otherwise original direction of velocity vector
% GFLOW =   flat file containing summation of element integration
matrices for LHS
%           grid motion in pseudo time
%
% structure
% STIFF =   flat file containing summation of element integration
matrices for LHS
%           structural internal force vector (stiffness), physical time
(if used)
%           and pseudo time (if used) calculations
%
% grid
% GSTIFF =  flat file containing summation of element integration
matrices for LHS
%           grid internal force vector (stiffness), physical time (if
used) and
%           pseudo time (if used) calculations

global MFLOW MOUT GFLOW STIFF GSTIFF

% set some Newton (outer) and GMRES (inner) iteration parameters
% [absolute error tolerance, relative error tolerance, degrees of
freedom/node]
%TOL=[1E-6,1E-9,4];
TOL=[1E-6,1E-6,4];

% [max Newton steps, max linear iterations, etamax, linear soln method,
max restarts]
% 20 Newton steps
% 40 GMRES iterations per start or restart
% .5 etamax (or .9)
% 2 = Restarted GMRES(40)
% 5 = max number of GMRES restarts -> linear its max/Newton step.
%     i.e. GMRES will run 6 times if 5 restarts.  Initial + 5 restarts
PARMS=[20,88,.5,2,5];
%PARMS=[2,2,.5,2,1];

T_STEP=1;                % max time steps
DEL_T = 0;               % physical time step size, 0 = steady state

flops(0);
tic

%**************************************************************************
**************
% Set up space for global matrices
%**************************************************************************
**************
```

```
FLUID_NAME = 'nozzle2c'
FLUID_NNODES = 192
FLUID_DOFNODE = 4
FLUID_DOF = FLUID_DOFNODE * FLUID_NNODES
SOLID_NAME = 'wall2b'
SOLID_NNODES = 48
SOLID_DOFNODE = 2
SOLID_DOF = SOLID_DOFNODE * SOLID_NNODES
GRID_NAME = 'nozzle2g'
GRID_NNODES = FLUID_NNODES
GRID_DOFNODE = 2
GRID_DOF = GRID_DOFNODE * GRID_NNODES

SOLN = zeros(FLUID_DOF+SOLID_DOF+GRID_DOF,1);

%********************************** FLUID
*****************************************

% Initialize fluid global matrices for element by element storage
eval(['load ',FLUID_NAME]);
[NEL,LARGEST]=size(ELMAT);
MFLOW=zeros(NEL,(FLUID_DOFNODE*LARGEST)^2);     % LHS element
integration matrix

LBC = length(find(BCTYPE));          % number of boundary nodes
MOUT = zeros(LBC,7);                 % [node number,dx,dy,type,index
into BC1,DX,DY]
MOUT(:,1)=find(BCTYPE);              % load boundary node numbers

GFLOW = zeros(NEL,2*FLUID_DOFNODE*LARGEST^2);   % Fluid flow correction
for grid motion

% load fluid initial conditions
FLUID_INDEX = [1:FLUID_DOF];
SOLN(FLUID_INDEX) = reshape(QINITIAL,FLUID_NNODES*FLUID_DOFNODE,1);

%********************************** SOLID
*****************************************

% Initialize solid global matrix for element by element storage
eval(['load ',SOLID_NAME]);
[NEL,LARGEST]=size(ELMAT);
STIFF=zeros(NEL,(SOLID_DOFNODE*LARGEST)^2);     % LHS element
integration matrix

%********************************** GRID
*****************************************

% Initialize grid global matrix for element by element storage
eval(['load ',GRID_NAME]);
[NEL,LARGEST]=size(ELMAT);
GSTIFF=zeros(NEL,(GRID_DOFNODE*LARGEST)^2);     % LHS element
integration matrix
```

```
%*************************************************************************
**************
% Set up for physical time or load increment stepping
%*************************************************************************
**************

SOLN_TM1 = [];
SOLN_TM2 = [];

FNRM=1;
STOP_TOL=TOL(1) + TOL(2)*FNRM;
ITC = 0;

% start physical time or load increment stepping iterations
while(FNRM > STOP_TOL & ITC < T_STEP)
  ITC=ITC+1;

% update for actual time stepping
  SOLN_TM2 = SOLN_TM1;
  SOLN_TM1 = [];

% Set up figure windows
  figure(1)
  clg
  set(1,'Position',[2 30 447 340]);
  figure(2)
  clg
  set(2,'Position',[456 414 371 310]);
  figure(3)
  clg
  set(3,'Position',[2 415 447 308]);
  figure(4)
  clg
  set(4,'Position',[456 30 370 339]);

  [SOLN, HISTORY, IERROR, STATS] =
knsolgma('FSG_LHS','fsg_rhs',TOL,PARMS,...
                                          SOLN,SOLN,...

FLUID_NAME,FLUID_DOF,FLUID_DOFNODE,...

SOLID_NAME,SOLID_DOF,SOLID_DOFNODE,...
                                          GRID_NAME ,GRID_DOF
,GRID_DOFNODE,...
                                          SOLN_TM1,SOLN_TM2);

% update SOLN for all necessary boundary conditions

% fluid updates inflow/outflow boundary conditions
  eval(['load ',FLUID_NAME]);
  DUMMY = reshape(SOLN(1:FLUID_DOF),FLUID_NNODES,FLUID_DOFNODE);
  DUMMY = bc1_lhs(DUMMY,BC1);
  SOLN(1:FLUID_DOF) = reshape(DUMMY,FLUID_DOF,1);

% solid updates restricted deformation conditions
  eval(['load ',SOLID_NAME]);
```

```
  SOLN(FLUID_DOF+BCX(:,1))=BCX(:,2);
  SOLN(FLUID_DOF+SOLID_NNODES+BCY(:,1))=BCY(:,2);

% grid updates along solid boundary and fixed deformation conditions
  eval(['load ',GRID_NAME]);
  X_CROSS_REF = find(BCX(:,4)~=0);
  Y_CROSS_REF = find(BCY(:,4)~=0);
  BCX(X_CROSS_REF,2)=SOLN(FLUID_DOF+BCX(X_CROSS_REF,4));
  BCY(Y_CROSS_REF,2)=SOLN(FLUID_DOF+SOLID_NNODES+BCY(Y_CROSS_REF,4));
  SOLN(FLUID_DOF+SOLID_DOF+BCX(:,1))=BCX(:,2);
  SOLN(FLUID_DOF+SOLID_DOF+GRID_NNODES+BCY(:,1))=BCY(:,2);

  disp([STATS(:,1),zeros(size(STATS(:,2))),STATS(:,3:5)])
  format long
  disp(STATS(:,2))
  format short

  FNRM = norm(SOLN)/sqrt(length(SOLN));

end   % physical time or load increment stepping

toc
flops

figure(1)
clg
semilogy(HISTORY(:,1))
```

FSG_RHS

```
function R = fsg_rhs(SOLN,DUMMY,...
                     FLUID_NAME,FLUID_DOF,FLUID_DOFNODE,...
                     SOLID_NAME,SOLID_DOF,SOLID_DOFNODE,...
                     GRID_NAME, GRID_DOF, GRID_DOFNODE,...
                     SOLN_TM1,SOLN_TM2);

% DXS = structure deformation
% DYS = structure deformation
% VXS = structure velocity
% VYS = structure velocity
% DXG = grid deformation
% DYG = grid deformation
% VXG = grid velocity
% VYG = grid velocity

disp('In FSG_RHS now')

R = zeros(FLUID_DOF+SOLID_DOF+GRID_DOF,1);

%***************************** START SETUP
*****************************************
```

```
% Get the grid position, velocity, and boundary conditions set up
properly
%*************************************************************************
*************

eval(['load ',SOLID_NAME]);                    % open solid file
SNODES = length(XN);
INDEX = [FLUID_DOF+1 : FLUID_DOF+SOLID_DOF]';

DXS = SOLN(INDEX(1):INDEX(SNODES));            % get solid position
and velocity
DYS = SOLN(INDEX(SNODES+1):INDEX(2*SNODES));
if SOLID_DOFNODE == 4
  VXS = SOLN(INDEX(2*SNODES+1):INDEX(3*SNODES));
  VYS = SOLN(INDEX(3*SNODES+1):INDEX(4*SNODES));
else
  VXS = [];
  VYS = [];
end

% Enforce structure boundary conditions
DXS(BCX(:,1))=BCX(:,2);                % set dirichlet X boundary
deformation
DYS(BCY(:,1))=BCY(:,2);                % set dirichlet Y boundary
deformation
if SOLID_DOFNODE == 4
  VXS(BCX(:,1))=BCX(:,3);              % set dirichlet X boundary velocity
  VYS(BCY(:,1))=BCY(:,3);              % set dirichlet Y boundary velocity
end

%****************************** END SETUP
**************************************
% END SETUP END SETUP END SETUP END SETUP END SETUP END SETUP END SETUP
END SETUP


%****************************** START GRID
****************************************
%
%*************************************************************************
*************
eval(['load ',GRID_NAME]);                     % open grid file
GNODES = length(XN);
INDEX = [FLUID_DOF+SOLID_DOF+1 : FLUID_DOF+SOLID_DOF+GRID_DOF]';

DXG = SOLN(INDEX(1):INDEX(GNODES));            % get grid position and
velocity
DYG = SOLN(INDEX(GNODES+1):INDEX(GRID_DOF));
if GRID_DOFNODE == 4
  VXG = SOLN(INDEX(2*GNODES+1):INDEX(3*GNODES));
  VYG = SOLN(INDEX(3*GNODES+1):INDEX(4*GNODES));
else
  VXG = [];
  VYG = [];
end
```

```
% need actual node numbers to tie boundary conditions together in grid
and solid
% frames of reference.  Cross reference in BCX and BCY provides
connections.

X_CROSS_REF = find(BCX(:,4)~=0);                % get grid/solid cross
reference
Y_CROSS_REF = find(BCY(:,4)~=0);

% update grid dirichlet boundary condition file to match current solid
shape
BCX(X_CROSS_REF,2) = DXS(BCX(X_CROSS_REF,4));
BCY(Y_CROSS_REF,2) = DYS(BCY(Y_CROSS_REF,4));

if GRID_DOFNODE == 4 & SOLID_DOFNODE == 4
  BCX(X_CROSS_REF,3) = VXS(BCX(X_CROSS_REF,4));
  BCY(Y_CROSS_REF,3) = VYS(BCY(Y_CROSS_REF,4));
end

SCALE = 1;
XT_1 = [];
VT_1 = [];
DEL_T = 0;
DEL_TAU = 0;

disp('starting grid')
R(INDEX) =
grid_rhs([DXG;DYG;VXG;VYG],[],XN,YN,ELMAT,PTX,PTY,SCALE,BCX,BCY,MP,...
                    DEL_T,DEL_TAU,NN,GRID_DOFNODE,XT_1,VT_1);

% Enforce grid boundary conditions for use in fluid calculations
DXG(BCX(:,1))=BCX(:,2);             % set dirichlet X boundary
deformation
DYG(BCY(:,1))=BCY(:,2);             % set dirichlet Y boundary
deformation
if SOLID_DOFNODE == 4
  VXG(BCX(:,1))=BCX(:,3);           % set dirichlet X boundary velocity
  VYG(BCY(:,1))=BCY(:,3);           % set dirichlet Y boundary velocity
end
disp('ending grid')

%************************** END GRID
*****************************************
% END GRID END GRID END GRID END GRID END GRID END GRID END GRID END
GRID END GRID


%************************** START FLUID
*****************************************
%
%*************************************************************************
*************
eval(['load ',FLUID_NAME]);         % open fluid file
FNODES = length(XN);
INDEX = [1 : FLUID_DOF]';
```

```
QP = SOLN(INDEX);
QP1=QP;
QP2=QP;

DEL_T = 0;
DEL_TAU = 1;

if GRID_DOFNODE == 4                    % use grid position and velocity
from above
  % do nothing
elseif DEL_TAU ~= 0                     % calculate grid velocity if not
available
  VXG = DXG / DEL_TAU;
  VYG = DYG / DEL_TAU;
else
  VXG = zeros(size(DXG));
  VYG = zeros(size(DYG));
end

disp('starting fluid residual calculation')
R(INDEX) =
blok_rhs(QP,QP,XN+DXG,YN+DYG,VXG,VYG,QP,QP,DEL_T,DEL_TAU,MP,ELMAT,...
                    BCTYPE,BC1,MOUT_PTR,FLUID_DOFNODE,NN);

% Enforce fluid boundary conditions
QP = reshape(QP,FNODES,FLUID_DOFNODE);
QP = bc1_lhs(QP,BC1);

disp('ending fluid residual calculation')

%****************************** END FLUID
**************************************
% END FLUID END FLUID END FLUID END FLUID END FLUID END FLUID END FLUID
END FLUID


%****************************** START STRUCTURE
**************************************
%
%*************************************************************************
*************

eval(['load ',SOLID_NAME]);                          % open solid file
SNODES = length(XN);
INDEX = [FLUID_DOF+1 : FLUID_DOF+SOLID_DOF]';

% solid position and velocity calculated in setup section above

% need actual node numbers to tie boundary conditions together in fluid
and solid
% frames of reference

P_CROSS_REF = find(P(:,2)~=0);
P(P_CROSS_REF,1) = QP(P(P_CROSS_REF,2),1);

% calculate nodal forces from pressure loads
```

```
[PTX,PTY] = press(XN+DXS,YN+DYS,ELMAT,P(:,1),NN,SOLID_DOFNODE);

% remove load at dirichlet boundary nodes
PTX(BCX(:,1)) = PTX(BCX(:,1))*0;
PTY(BCY(:,1)) = PTY(BCY(:,1))*0;

SCALE = 1;
XT_1 = [];
VT_1 = [];
DEL_T = 0;
DEL_TAU = 0;

disp('starting structure')
R(INDEX) =
str_rhs([DXS;DYS;VXS;VYS],[DXS;DYS;VXS;VYS],XN,YN,ELMAT,PTX,PTY,SCALE,.
..

BCX,BCY,MP,DEL_T,DEL_TAU,NN,SOLID_DOFNODE,[1,0],XT_1,VT_1);

disp('ending structure')

%*************************** END STRUCTURE
****************************************
% END STRUCTURE END STRUCTURE END STRUCTURE END STRUCTURE END STRUCTURE
END STRUCTURE

global RHSR
RHSR = R;


disp('Leaving FSG_RHS')
```

FSG_LHS

```
function R = fsg_lhs(SOLN,DUMMY,...
                     FLUID_NAME,FLUID_DOF,FLUID_DOFNODE,...
                     SOLID_NAME,SOLID_DOF,SOLID_DOFNODE,...
                     GRID_NAME, GRID_DOF, GRID_DOFNODE,...
                     SOLN_TM1,SOLN_TM2);

% DXS = structure deformation
% DYS = structure deformation
% VXS = structure velocity
% VYS = structure velocity
% DXG = grid deformation
% DYG = grid deformation
% VXG = grid velocity
% VYG = grid velocity

R = zeros(FLUID_DOF+SOLID_DOF+GRID_DOF,1);
```

```
%****************************** START SETUP
****************************************
% Get the grid position, velocity, and boundary conditions set up
properly
%*************************************************************************
*************

eval(['load ',SOLID_NAME]);                          % open solid file
SNODES = length(XN);
INDEX = [FLUID_DOF+1 : FLUID_DOF+SOLID_DOF]';

DXS = SOLN(INDEX(1):INDEX(SNODES));                  % get solid position
and velocity
DYS = SOLN(INDEX(SNODES+1):INDEX(2*SNODES));
if SOLID_DOFNODE == 4
  VXS = SOLN(INDEX(2*SNODES+1):INDEX(3*SNODES));
  VYS = SOLN(INDEX(3*SNODES+1):INDEX(4*SNODES));
else
  VXS = [];
  VYS = [];
end

% Enforce structure boundary conditions
DXS(BCX(:,1))=BCX(:,2);                  % set dirichlet X boundary
deformation
DYS(BCY(:,1))=BCY(:,2);                  % set dirichlet Y boundary
deformation
if SOLID_DOFNODE == 4
  VXS(BCX(:,1))=BCX(:,3);                % set dirichlet X boundary velocity
  VYS(BCY(:,1))=BCY(:,3);                % set dirichlet Y boundary velocity
end

%****************************** END SETUP
****************************************
% END SETUP END SETUP END SETUP END SETUP END SETUP END SETUP END SETUP
END SETUP


%****************************** START GRID
****************************************
%
%*************************************************************************
*************
eval(['load ',GRID_NAME]);                           % open grid file
GNODES = length(XN);
INDEX = [FLUID_DOF+SOLID_DOF+1 : FLUID_DOF+SOLID_DOF+GRID_DOF]';

DXG = SOLN(INDEX(1):INDEX(GNODES));                  % get grid position and
velocity
DYG = SOLN(INDEX(GNODES+1):INDEX(GRID_DOF));
if GRID_DOFNODE == 4
  VXG = SOLN(INDEX(2*GNODES+1):INDEX(3*GNODES));
  VYG = SOLN(INDEX(3*GNODES+1):INDEX(4*GNODES));
else
  VXG = [];
  VYG = [];
```

```
end

% need actual node numbers to tie boundary conditions together in grid
and solid
% frames of reference.  Cross reference in BCX and BCY provides
connections.

X_CROSS_REF = find(BCX(:,4)~=0);                % get grid/solid cross
reference
Y_CROSS_REF = find(BCY(:,4)~=0);

% update grid dirichlet boundary condition file to match current solid
shape
BCX(X_CROSS_REF,2) = DXS(BCX(X_CROSS_REF,4));
BCY(Y_CROSS_REF,2) = DYS(BCY(Y_CROSS_REF,4));

if GRID_DOFNODE == 4 & SOLID_DOFNODE == 4
  BCX(X_CROSS_REF,3) = VXS(BCX(X_CROSS_REF,4));
  BCY(Y_CROSS_REF,3) = VYS(BCY(Y_CROSS_REF,4));
end

SCALE = 1;
XT_1 = [];
VT_1 = [];
DEL_T = 0;
DEL_TAU = 0;

R(INDEX) =
grid_lhs([DXG;DYG;VXG;VYG],[],XN,YN,ELMAT,PTX,PTY,SCALE,BCX,BCY,MP,...
                    DEL_T,DEL_TAU,NN,GRID_DOFNODE,XT_1,VT_1);

% Enforce grid boundary conditions
DXG(BCX(:,1))=BCX(:,2);              % set dirichlet X boundary
deformation
DYG(BCY(:,1))=BCY(:,2);              % set dirichlet Y boundary
deformation
if SOLID_DOFNODE == 4
  VXG(BCX(:,1))=BCX(:,3);            % set dirichlet X boundary velocity
  VYG(BCY(:,1))=BCY(:,3);            % set dirichlet Y boundary velocity
end

%************************** END GRID
*****************************************
% END GRID END GRID END GRID END GRID END GRID END GRID END GRID END
GRID END GRID


%***************************** START FLUID
*************************************
%
%*****************************************************************************
*************
eval(['load ',FLUID_NAME]);          % open fluid file
FNODES = length(XN);
INDEX = [1 : FLUID_DOF]';
```

```
QP = SOLN(INDEX);
QP1=QP;
QP2=QP;

DEL_T = 0;
DEL_TAU = 1;

if GRID_DOFNODE == 4                    % use grid position and velocity
from above
  % do nothing
elseif DEL_TAU ~= 0                     % calculate grid velocity if not
available
  VXG = DXG / DEL_TAU;
  VYG = DYG / DEL_TAU;
else
  VXG = zeros(size(DXG));
  VYG = zeros(size(DYG));
end

R(INDEX) =
flow_lhs(QP,QP,XN+DXG,YN+DYG,VXG,VYG,QP,QP,DEL_T,DEL_TAU,MP,ELMAT,...
                    BCTYPE,BC1,MOUT_PTR,FLUID_DOFNODE,NN);

% Enforce fluid boundary conditions
QP = reshape(QP,FNODES,FLUID_DOFNODE);
QP = bc1_lhs(QP,BC1);

%****************************** END FLUID
**************************************
% END FLUID END FLUID END FLUID END FLUID END FLUID END FLUID END FLUID
END FLUID


%****************************** START STRUCTURE
**************************************
%
%*************************************************************************
*************

eval(['load ',SOLID_NAME]);                          % open solid file
SNODES = length(XN);
INDEX = [FLUID_DOF+1 : FLUID_DOF+SOLID_DOF]';

% solid position and velocity calculated above

% need actual node numbers to tie boundary conditions together in fluid
and solid
% frames of reference

P_CROSS_REF = find(P(:,2)~=0);
P(P_CROSS_REF,1) = QP(P(P_CROSS_REF,2),1);

% calculate nodal forces from pressure loads
[PTX,PTY] = press(XN+DXS,YN+DYS,ELMAT,P(:,1),NN,SOLID_DOFNODE);

% remove load at dirichlet boundary nodes
```

```
PTX(BCX(:,1)) = PTX(BCX(:,1))*0;
PTY(BCY(:,1)) = PTY(BCY(:,1))*0;

SCALE = 1;
XT_1 = [];
VT_1 = [];
DEL_T = 0;
DEL_TAU = 0;

R(INDEX) =
str_lhs([DXS;DYS;VXS;VYS],[DXS;DYS;VXS;VYS],XN,YN,ELMAT,PTX,PTY,SCALE,.
..

BCX,BCY,MP,DEL_T,DEL_TAU,NN,SOLID_DOFNODE,[1,0],XT_1,VT_1);

%************************** END STRUCTURE
****************************************
% END STRUCTURE END STRUCTURE END STRUCTURE END STRUCTURE END STRUCTURE
END STRUCTURE
```

VITA

Kevin Kilpatrick was born in Wichita Falls, Texas on 5 May 1962. He moved to Monroe, Louisiana at a young age and attended public schools in the Ouachita Parish Public School system. He graduated Ouachita Parish High School as valedictorian in 1980 and soon entered Louisiana Technical University on an Air Force Reserve Officer Training Corp scholarship in Mechanical Engineering. In 1981, he entered the US Air Force Academy in Colorado Springs, Colorado. Kevin graduated from the Air Force Academy in 1985 with a triple major in Aeronautical Engineering, Astronautical Engineering, and Engineering Sciences. Upon graduation, he entered the active duty Air Force as a Second Lieutenant and was assigned to the Aero Propulsion and Power Laboratory at Wright Patterson Air Force Base in Dayton, Ohio where he performed research and testing of advanced jet engine compressor designs. He also enrolled in the University of Dayton in 1985, completing a Master's Degree in Mechanical Engineering in 1987. In 1991, now Captain Kilpatrick was assigned to the University of Tennessee in Knoxville, Tennessee as an Air Force Reserve Officer Training Corp instructor and Commandant of Cadets. While assigned in Knoxville, Kevin began an advanced program of study in Mechanical Engineering. When he was reassigned to Arnold Air Force Base in Tullahoma, Tennessee in 1994, he continued his studies at the University of Tennessee Space Institute while serving as the lead engineer for wind tunnel test technologies and computational modeling and simulation. Major Kilpatrick was reassigned in 1997 to the Oklahoma Air Logistics Center in Oklahoma City, Oklahoma where he was first appointed Chief of Propulsion Systems Engineering, then Chief of Propulsion Item and Production Management before becoming the acting Chief of the Propulsion Management Division. He continued his doctoral research throughout. In 2000, Major Kilpatrick was again reassigned to Shaw Air Force Base in Sumter, South Carolina as the Maintenance Supervisor in the Component Repair Squadron. He received his doctoral degree in Mechanical Engineering in August, 2001.

Kevin Kilpatrick was selected for promotion to Lieutenant Colonel and is currently serving as the Squadron Maintenance Officer for the 77th Fighter Squadron, the "Gamblers." He is married to the former Carolyn Cummings of Dayton, Ohio and they have two children, Lucas and Samuel.