THE DEVELOPMENT OF SOLUTION ALGORITHMS FOR COMPRESSIBLE FLOWS

by

David Christopher Slack

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Aerospace Engineering

APPROVED:

[illegible]

R. W. Walters, Chairman

[illegible]

B. Grossman

[illegible]

J. A. Schetz

W. H. Mason

W. Mason

[illegible]

W. L. Neu

January, 1991

Blacksburg, Virginia

c.2

LD
5655
V856
1991
S633
c.2

THE DEVELOPMENT OF SOLUTION ALGORITHMS FOR COMPRESSIBLE FLOWS

by

David Christopher Slack

Committee Chairman: Robert W. Walters

Aerospace Engineering

Abstract

This work investigates three main topics. The first of these is the development and comparison of time integration schemes on two-dimensional unstructured meshes. Both explicit and implicit solution algorithms for the two-dimensional Euler equations on unstructured grids are presented. Cell-centered and cell-vertex finite volume upwind schemes utilizing Roe's approximate Riemann solver are developed. For the cell-vertex scheme, a four stage Runge-Kutta time integration with and without implicit residual averaging, a point Jacobi method, a symmetric point Gauss-Seidel method, and two methods utilizing preconditioned sparse matrix solvers are investigated. For the cell-centered scheme, a Runge-Kutta scheme, an implicit tridiagonal relaxation scheme modeled after line Gauss-Seidel, a fully implicit LU decomposition, and a hybrid scheme utilizing both Runge-Kutta and LU methods are presented. A reverse Cuthill-McKee renumbering scheme is employed for the direct solver in order to decrease CPU time by reducing the fill of the Jacobian matrix. Comparisons are made for both first-order and higher-order accurate solutions using several different time integration algorithms. Higher-order accuracy is achieved by using multi-dimensional monotone linear reconstruction procedures. Results for flow over a transonic circular arc are compared for the various time integration methods. The second topic involves an interactive adaptive remeshing algorithm. The interactive adaptive remeshing algorithm utilizing a frontal grid generator is compared to a single grid calculation. Several device dependent interactive graphics interfaces have been developed along with a device independent DI-3000 interface which can be employed on any computer that has the supporting software including the Cray-2 supercomputers Voyager and Navier. Solutions for two-dimensional, invicid flow over a transonic circular arc and a Mach 3.0 internal flow with an area change are examined. The final topic examined in this work is the capabilities developed for a structured three-dimensional code called GASP. The capabilities include: generalized chemistry and thermodynamic modeling, space marching, memory management through the use of binary C Input/Ouput, and algebraic and two-equation eddy viscosity turbulence modeling. Results are given for a Mach 1.7 three-dimensional analytic forebody, a Mach 1.38 axisymmetric nozzle with hydrogen-air combustion, a Mach 14.1 15^o ramp, and Mach 0.3 viscous flow over a flat plate. The incorporation of these capabilities and the two-dimensional unstructured time integration schemes into a three-dimensional unstructured solver is also discussed.

Acknowledgements

The author would like to thank his advisor Dr. Robert W. Walters for all his help and support without which this work would not have been possible. The author would also like to thank the other members of his graduate committee : Dr. Joseph A. Schetz, Dr. Bernard Grossman, Dr. Wayne Neu, and Dr. William Mason for their assistance both as teachers of their coursework and as committee members. It would be negligent not to mention the contributions of his distinguished colleagues Dr. Pasquale Cinnella, Dr. David Whitaker, and Dr. William D. McGrory. Dr. Cinnella provided much of the coding in GASP related to the chemistry and thermodynamic modeling and laminar viscous contributions. Dr. Whitaker provided the cell-vertex two-dimensional unstructured code, while Dr McGrory aided in the development of the cell-centered unstructured code and provided advice and insight into the discussion of unstructured grid generation. The development of GASP and the cell-centered unstructured code was sponsored by NASA Langley Research Center under NASA Grants NAG-1-776 and NAG-1-992. Finally the author would like to thank his beloved wife Lori for all of her understanding and support which has greatly helped him achieve his goal.

Table of Contents

Acknowlegements iv

Table of Contents v

List of Illustrations x

List of Symbols xv

I. Introduction 1

II. Mesh Generation 8

2.1 Advancing Front Grid Generator 8

2.2 Adaptive Mesh Regeneration 14

2.3 Interactive Graphics 15

III. Governing Equations 17

3.1 Euler Equations 17

3.2 Spatial Discretization 18

3.2.1 Cell-Centered 18

3.2.2 Cell-Vertex 18

3.2.3 Flux Calculation 21

3.3 Roe's Flux Difference Splitting 21

3.4 Steger-Warming Jacobian 25

3.5 Constructing the Left and Right Fluid States 26

3.5.1 Cell-Centered Higher-Order Correction 26

3.5.2 Cell-Vertex Higher-Order Correction 29

IV. Time Integration on Unstructured Meshes 32

4.1 Runge-Kutta Time Integration 34

4.1.1 Runge-Kutta with Residual Smoothing 35

4.2 Point Jacobi Time Integration 36

4.3 Point Gauss-Seidel Time Integration 36

4.4 A Line Gauss-Seidel Type Time Integration 37

4.4.1 Tridiagonal Renumbering Scheme 38

4.5 LU Decomposition Time Integration 41

4.5.1 Reverse Cuthill-McKee 42

4.5.2 Skyline Solver 43

4.6 Hybrid Runge-Kutta / LU Decomposition Time Integration 45

4.7 Sparse Matrix Solvers 46

4.7.1 GMRES Acceleration 46

4.7.2 CHEBYCODE Acceleration 49

V. Results for Two-Dimensional Unstructured Solvers .. 51

5.1 Time Integration Scheme Comparison 51

5.1.1 Coarse 1005 Element Mesh 51

5.1.2 Medium 1999 Element Mesh 57

5.1.3 Fine 4008 Element Mesh 62

5.1.4 Spectral Radius and Time Convergence Rates 67

5.1.5 Summary of Overall Performance 70

5.1.6 Pressure Plot Comparison 71

5.2 Remeshing Strategy Comparison 76

5.2.1 Transonic Circular Arc in a Channel 76

5.2.2 Mach 3.0 Internal Flow with Area Change 83

VI. Governing Equations in Three Dimensions 89

6.1 Navier Stokes Equations 89

6.2 Simplified Governing Equations 93

6.3 Reynolds-Averaged Navier Stokes Equations 96

6.3.1 Algebraic Turbulence Modeling 97

6.3.2 Two-Equation Turbulence Modeling 98

VII. Time Integration on Structured Meshes 101

7.1 Explicit Runge-Kutta Time Integration 101

7.2 Implicit Time Integration 102

7.2.1 LU Decomposition in a Plane 102

7.2.2 Approximate Factorization in a Plane 104

7.2.3 Storage and Vectorization for LU and AF 104

7.3 Space Marching 106

VIII. Results for Three-Dimensional Structured Solver 108

8.1 Analytic Forebody 108

8.2 Supersonic Nozzle with Hydrogen-Air Chemistry 113

8.3 Holden's 15^o Ramp 113

8.4 Viscous Flow over Flat Plate 120

IX. Unstructured Applications in Three Dimensions....123

X. Conclusions 127

References..130

Appendix A..135

Appendix B..142

Appendix C..144

Vita 145

List of Illustrations

Figure 1. Comparison of structured and unstructured mesh for SR71 crossflow plane 3

Figure 2. Background mesh for circular arc in a channel flow 9

Figure 3. Grid input file for circular arc in a channel flow 10

Figure 4. Partial grid for circular arc in a channel flow 12

Figure 5. Ideal triangle shape for advancing front grid generation 13

Figure 6. Control volume for cell-centered finite volume discretization 19

Figure 7. Control volume for cell-vertex finite volume discretization 20

Figure 8. (a) Typical cell interface with left and right states and (b) one dimensional Riemann problem 23

Figure 9. Integration path for cell-centered gradient calculation 28

Figure 10. Phantom cells for cell-vertex gradient calculation 31

Figure 11. Block tridiagonal matrix for line Gauss-Seidel calculation 39

Figure 12. A single subsection of block tridiagonal matrix for line Gauss-Seidel calculation 40

Figure 13. Jacobian matrix after Reverse Cuthill-McKee renumbering 44

Figure 14. Residual versus iteration plots for 1005 element mesh, first-order transonic circular arc in a channel........ 52

Figure 15. Residual versus CPU time plots for 1005 element mesh, first-order transonic circular arc in a channel.................................. 53

Figure 16. Residual versus iteration plots for 1005 element mesh, higher-order transonic circular arc in a channel.................................. 55

Figure 17. Residual versus CPU time plots for 1005 element mesh, higher-order transonic circular arc in a channel.............................. 56

Figure 18. Residual versus iteration plots for 1999 element mesh, first-order transonic circular arc in a channel.................................. 58

Figure 19. Residual versus CPU time plots for 1999 element mesh, first-order transonic circular arc in a channel.................................. 59

Figure 20. Residual versus iteration plots for 1999 element mesh, higher-order transonic circular arc in a channel.................................. 60

Figure 21. Residual versus CPU time plots for 1999 element mesh, higher-order transonic circular arc in a channel.............................. 61

Figure 22. Residual versus iteration plots for 4008 element mesh, first-order transonic circular arc in a channel.................................. 63

Figure 23. Residual versus CPU time plots for 4008 element mesh, first-order transonic circular arc in a channel.................................. 64

Figure 24. Residual versus iteration plots for 4008 element mesh, higher-order transonic circular arc in a channel.................................. 65

Figure 25. Residual versus CPU time plots for 4008 element mesh, higher-order transonic circular arc in a channel.................................. 66

Figure 26. Spectral radius comparison for transonic circular arc in a channel . 68

Figure 27. Time convergence rate plots for transonic circular arc in a channel 69

Figure 28. (a) 1005 element uniform mesh and (b) first-order pressure contours (c) higher-order pressure contours for transonic circular arc in a channel 73

Figure 29. (a) 4008 element uniform mesh and (b) first-order pressure contours (c) higher-order pressure contours for transonic circular arc in a channel 74

Figure 30. (a)-(b) Surface pressure comparison for transonic circular arc in a channel 75

Figure 31. (a)-(b) Residual plots for remeshing strategy of transonic circular arc in a channel 78

Figure 32. (a) uniform 2439 element mesh, (b) higher-order uniform mesh pressure contours, (c) adapted 2389 element mesh, and (d) higher-order adapted mesh pressure contours for transonic circualr arc in a channel 79

Figure 33. (a)-(j) First-order remeshing sequence for transonic circualr arc in a channel 81

Figure 34. (a)-(j) Higher-order remeshing sequence for transonic circualr arc in a channel 82

Figure 35. (a)-(b) Residual plots for remeshing strategy of supersonic nozzle . 85

Figure 36. (a) uniform 6263 element mesh, (b) higher-order uniform mesh pressure contours, (c) adapted 6247 element mesh, and (d) higher-order adapted mesh pressure contours for supersonic nozzle 86

Figure 37. (a)-(j) First-order remeshing sequence for supersonic nozzle 87

Figure 38. (a)-(f) Higher-order remeshing sequence for supersonic nozzle 88

Figure 39. Flat storage of matrix for structured LU and AF solvers 105

Figure 40. Analytic forebody configuration 109

Figure 41. Symmetry plane and crossflow plane grid for analytic forebody .. 110

Figure 42. Pressure contours in symmetry plane for analytic forebody 111

Figure 43. Surface pressure comparison of analytic forebody calculation with experiment 112

Figure 44. Axisymmetric supersonic combustion nozzle configuration 114

Figure 45. Volume averaged temperature plot compared with results from Drummond 115

Figure 46. Sketch of Holden's 15^o Ramp 116

Figure 47. Iteration versus plane number for Holden's 15^o Ramp 118

Figure 48. (a) Skin friction, (b) heat transfer, (c) Pressure distribution comparison with experiment for Holden's 15^o Ramp 119

Figure 49. Similarity profiles for subsonic laminar flat plate compared with Blasius solution 121

Figure 50. (a) u^+ vs. y^+ (b) K^+ vs. y^+ (c) ϵ^+ vs. y^+ for supersonic turbulent flat plate 122

Figure 51. Grid for supersonic nozzle broken into streamwise sections 125

List of Symbols

A	Jacobian matrix
$\tilde{A}$	Roe-averaged matrix
a	speed of sound
$\tilde{a}$	Roe-averaged speed of sound
c	diameter of convex hull for Chebyshev iteration
c_p	specific heat
c_{1-6}	coefficients for two-equation turbulence model
C_i	mesh cell i
d	center of convex hull for Chebyshev iteration
D	binary diffusion coefficient
$\mathcal{D}$	divergence of velocity
D_{12}	binary diffusion coefficient
e_k	eigenvectors
e_n	nonequilibrium portion of internal energy
e_0	total internal energy per unit mass
$\dot{e}_n$	nonequilibrium energy production rate
E_i	local error of cell i
$\vec{F}$	total flux vector
f, g	two-dimensional cartesian components of flux vector
$\hat{F}$, $\hat{G}$, $\hat{H}$	inviscid flux vectors
$\hat{F}_v$, $\hat{G}_v$, $\hat{H}_v$	viscous flux vectors
h_i	local mesh size of cell i
h_0	total enthalpy per unit mass
H	total enthalpy

$\tilde{H}$	Roe-averaged total enthalpy
I	identity matrix
J	Jacobian of coordinate transformation
k	thermal conductivity
k_l, k_t	laminar, turbulent thermal conductivity
K	mass-averaged turbulent kinetic energy
L, U	LU decomposition matrices
$\mathcal{L}$	strain rate invariant
L_2	Euclidean norm
l	mixing length
Le	Lewis number
M	# of inner iterations, # of nonequilibrium energy contributions, or lower triangular matrix
M_∞	freestream Mach number
M_ξ	streamwise Mach number
$\hat{n}$	outward facing unit normal vector
N	# of unknowns, # of species, or upper triangular matrix
p	pressure
p_k	orthonormal search directions for GMRES iteration
p^m	update vector for Chebyshev iteration
Pr_l, Pr_t	laminar, turbulent Prandtl number
q	velocity magnitude
Q	vector of conserved variables
Q_L, Q_R	left, right fluid state at an interface
ΔQ	change in vector Q in time
R	residual for time integration algorithms

R_i	species gas constant
s	strectching factor or Van Albada's limiter
s_i	primitive variables for two-equation turbulence model
Δs	length of cell face
S	vector of primitive variables for two-equation turbulence model
$\hat{S}$	vector of state variables for two-equation turbulence model
t	time
Δt	time step
T	temperature or tridiagonal submatrix
$\tilde{T}$, $\tilde{T}^{-1}$	right, left eigenvectors
u, v, w	cartesian components of velocity
$\tilde{u}$, $\tilde{v}$, $\tilde{w}$	Roe-averaged components of velocity
$\bar{U}$, $\bar{V}$, $\bar{W}$	contravariant components of velocity
$\vec{U}$	velocity vector
V	cell volume
V_i	vertex in center of cell i
$\hat{V}_s$	species diffusion velocity
W	vector of production rates
x, y, z	cartesian space coordinates

Greek symbols

α	weighting coefficients or wave strengths
$\vec{\alpha}$	stretching direction
δ	element size normal to stretching direction
δ_{ij}	Kroneker delta
Δ_+	forward difference

Δ_-	backward difference
ϵ	dissipation rate of turbulent kinetic energy or small parameter
γ	ratio of specific heats
κ	accuracy controller for MUSCL differencing
$\kappa(i)$	list of surrounding cells
λ	eigenvalues or spectral radius
Λ	diagonal matrix of eigenvalues
λ_t	time convergence rate
μ	dynamic viscosity
μ_l, μ_t	laminar, turbulent dynamic viscosity
∇	gradient operator
∇^2	Laplacian operator
Φ	source term for two-equation turbulence model
Φ_A	limiter for higher order cell-centered scheme
ρ	density
$\tilde{\rho}$	Roe-averaged density
ρ_i	species density
$\dot{\rho}_i$	species chemical production rate
Θ	heat flux contribution
$\tau_{x_i x_j}$	shear stress tensor
ω	vorticity or relaxation term for Vigneron technique
Ω	area of domain
ξ, η, ζ	generalized space coordinates
$\|\nabla\hat{\xi}\|$, $\|\nabla\hat{\eta}\|$, $\|\nabla\hat{\zeta}\|$	unit normals to coordinate surfaces

I. Introduction

The design of aircraft in the past has relied heavily on experimental data obtained from wind tunnels. In recent years, Computational Fluid Dynamics (CFD) has taken on a small part of the design burden. Several CFD codes have already been widely used and shown excellent comparison with experiments. However, the majority of these codes were developed and have been used for subsonic, transonic, and relatively low Mach number supersonic problems. There is a need now to develop efficient codes capable of accurately modeling the more complex physics, chemistry, and thermodynamics associated with hypersonic flight. Recently, upwind methods have been extended to include both chemistry and thermodynamic models that will be required for the accurate prediction of hypersonic flowfields[1,2,3,4]. The problem that remains is to make these codes efficient enough both in terms of storage and CPU time to calculate complicated three-dimensional flowfields.

One possible area for improvement and a subject of current research in CFD is the use of an unstructured indexing scheme. Structured solvers do not lend themselves to simple mesh refinement techniques needed to accurately resolve the physics of the flowfield without placing excessive elements in benign regions. By reducing the local mesh spacing in the vicinity of large flow gradients while increasing the mesh spacing in benign regions, a much more accurate solution can be obtained without significantly increasing the computational work. For a complicated three-dimensional flow, this could result in large savings in both storage and CPU time by decreasing the number of cells necessary to obtain accurate results.

Impressive results have been obtained for a wide range of problems utilizing unstructured flow solvers[5,6,7,8,9]. Structured flow solvers have also shown difficulties in predicting satisfactory results around complex geometries due to mesh irregularities. The superior geometric flexibility of unstructured over structured grids is demonstrated in Fig. 1 for a relatively simple body. A typical cross-section of the SR-71 reconnaissance aircraft is shown in which the left half has been discretized by a structured mesh technique and the right half has been discretized by an unstructured grid technique. The cells in the structured mesh are highly skewed while the cell distribution in the unstructured mesh is quite smooth. The highly skewed mesh will lead to solution inaccuracies.

Various techniques for generating unstructured meshes have been proposed in the past. Two of the more popular methods include Delaunay triangulation and the Advancing Front method. For simplicity, these methods will be discussed in two dimensions although both methods have been successfully implemented in three dimensions[10,11]. When using Delaunay triangulation a method must be developed to generate a set of nodes in a plane to be triangulated. From these nodes, Dirichlet regions are constructed. Dirichlet regions are polygons defined by the locus of all points closer to each node than any other. In most cases, the vertices of the polygons represent places where three regions come together. Such vertices are equidistant from three nodes, and all other nodes are farther away. These three nodes can be connected with line segments to form a triangle. When this is done for every polygon vertex, the triangles tessellate the plane. This method has the advantage that it is relatively simple and fast. However, Delaunay triangulation results in triangles that are as close to equilateral as possible which prohibits the possibility of directional refinement.

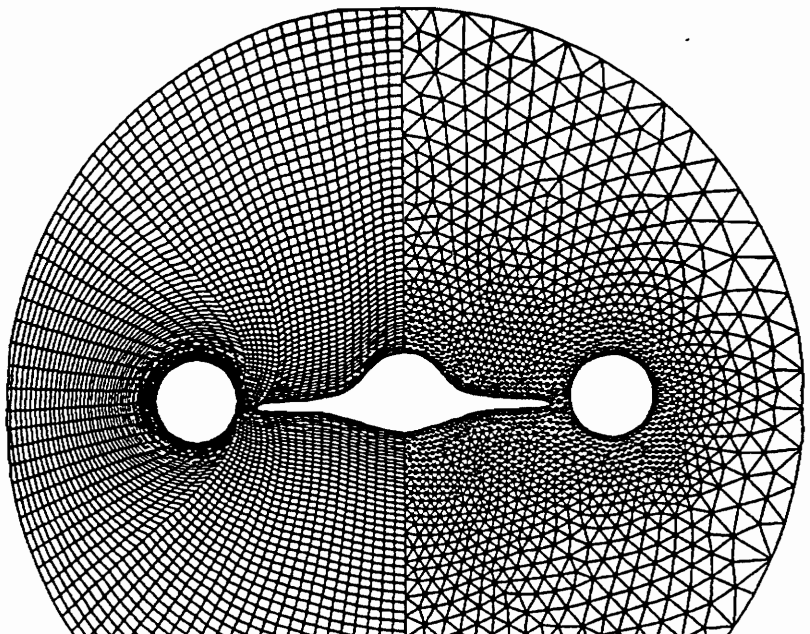


Figure 1) Comparison of structured and unstructured mesh for SR-71 crossflow plane

The advancing front grid generation technique requires input of stretching parameters. These parameters are given in the context of a coarse background grid of triangular elements which covers the solution domain. They are first used to place nodes along the boundaries of the computational region. The sides connecting these nodes form the initial generation front. Elements are added by interpolating the stretching parameters from the background grid, the entire front is updated, and the process repeated. Once the entire domain has been triangulated, a smoothing routine is performed to improve the quality of the generated mesh. The advancing front method has the distinct advantage of generating elements in regions of interest and can be utilized to provide directional refinement to align elongated elements parallel to flow discontinuities.

Since upwind schemes use locally one-dimensional models of the flow, it is important for increased accuracy that the faces of the triangles, across which the fluxes are calculated, align themselves with major flow features such as shocks and expansions. This results in a locally one-dimensional problem which when approximated with a locally one-dimensional flux calculation will be more accurate than approximating a multi-dimensional problem with a locally one-dimensional flux calculation.

It is straightforward with the advancing front grid generation process to utilize mesh regeneration in order to obtain an adapted grid solution. In general, a very simple background mesh with no stretching is input for computing the initial grid. The initial grid will then become the background grid for the first mesh regeneration. This process requires utilizing a measure of the error in the computed solution on the initial grid to obtain the strectching parameters at the nodes. An improved mesh can then be generated with the previous grid as a background grid along with these computed parameters.

One drawback of unstructured flow solvers to date has been slow convergence rates when compared to structured flow solvers. This has been in part due to the difficulty of vectorization caused by the indirect addressing needed in unstructured solvers. This problem can and has been overcome with the use of gather and scatter routines and the coloring of the faces of the triangles in the mesh. Coloring involves ordering the faces such that faces with common cells are placed into separate groups or colors. Consequently, the contribution of each face to the cell flux can be vectorized over each color.

Another convergence problem associated with unstructured solvers has been the lack of implementation of implicit schemes. Implicit schemes have been well established and refined for structured meshes. Schemes such as line Gauss-Seidel, approximate factorization (AF) , and banded LU decomposition have been widely used in both two and three dimensions and have yielded much faster convergence rates than explicit schemes for upwind methods[12]. Approximate factorization, which is a very efficient time integration scheme, utilizes the logical structure of the grid. As a result, it has no application to unstructured meshes. However, other implicit methods have and are being developed. Because this work is still in the developmental stages, most of this work is being done in two dimensions. Among these include variable-banded LU decomposition, a relaxation method similar to line Gauss-Seidel, Point Gauss-Seidel, Point Jacobi, and sparse matrix solvers such as Generalized Minimum Residual (GMRES)[13] and the Chebyshev solver CHEBYCODE.[14]

Obtaining smooth high-order approximations has also been a problem in algorithm development on unstructured meshes. On structured meshes, higher order spatial accuracy can be achieved for upwind methods via MUSCL differencing[15]. Instead of using gradients of the flux to make higher order flux calculations, MUSCL differencing uses gradients of the state variables to construct more accurate values of these variables on either side of an interface. These values are then used to calculate a second order accurate approximation to the flux across that interface. Several authors developed limiting schemes for MUSCL differencing that automatically reduced to locally first-order accurate near discontinuities to avoid oscillations. Stoufflet,et al.[9] developed a method which constructs higher order values of the state variables at an interface in much the same way as it is done on structured meshes. Because of the similarities, the same limiters used on structured meshes apply. Barth[5] proposed a scheme that linearly reconstructs the cell averaged data. This method requires calculating the gradient of the state variables by numerically integrating around a path of surrounding cell data. A monotone preserving limiter is then imposed to achieve smooth results around flow discontinuities.

A problem that has been growing with the complexity of CFD codes is finding personnel qualified enough to understand all of the aspects necessary to obtain accurate solutions to complicated problems. Few codes have an automatic grid generator capable of discretizing a general problem, and even fewer have such a grid generator coupled to the flow solver. This means the engineer is many times left with the task of writing a grid generator for each problem. If the engineer is fortunate enough to have access to a grid generator capable of discretizing a general problem, a variety of obstacles are still faced. Many of the "automatic" grid generators available are as or more complicated than the flow solvers to use. Also, it sometimes takes several attempts at generating a mesh, running the flow solver, and analyzing the solution before an accurate grid converged solution is obtained. However, by developing a graphics interface for input, output, and overall grid generator and flow solver control, capable of running on a range of popular workstations, much time and effort can be saved.

The present research involves two separate pieces of technology in CFD and how they can be combined to yield a general three-dimensional solver capable of computing complex flowfields. The first segment of this work brings together, compares, and extends the current two-dimensional unstructured flow solver technology. This is accomplished through the use of a cell-centered code developed by the author and a cell-vertex code developed by Whitaker[8]. A comparison of several time integration schemes from these two codes including both first and higher order calculations on a variety of mesh sizes is given. A mesh adaptation strategy which utilizes an interactive graphics interface is also discussed and compared to a single grid calculation. The adaptive grid generation code was provided by Löhner[6]. It was coupled to the cell-centered code, and the interactive graphics capabilities were added by the author. The second segment of this research focuses on the current capabilities for modeling complex physics and storage management techniques developed for a three dimensional structured flow solver called GASP. Much of the complex physical modeling was developed by Grossman and Cinnella[1,2,3,4], while the time integration, turbulence modeling and storage management techniques were provided by the author. The ultimate purpose of this research is to lay a foundation for the development of a CFD package which includes automatic grid generation, a flow solver, and a portable graphics interface which brings it all together into a powerful, easy to use tool for engineering applications.

II. Mesh Generation

2.1 Advancing Front Grid Generator

The procedure used for generating triangular elements about an arbitrary configuration is an advancing front method[10]. The actual code used was developed by Löhner[6]. This technique requires input of stretching parameters $\hat{\alpha}$, δ, and s. These parameters are given in the context of a coarse background grid of triangular elements which covers the solution domain. The background grid is input in the form of triangles and their surrounding nodes. Generally, the simplest choice is used for the initial background grid. A rectangular region, for example, would simply be broken down into two triangles. It is also possible that the background mesh cover more than the computational domain, so even a complex domain can be covered by a rectangle.

A sample initial background grid for a transonic channel flow is shown in Fig. 2. The background grid is simply two triangles, labeled 1 and 2, and four nodes labeled 1-4. Figure 3 shows the grid input file used to generate a 1000 cell mesh for this problem. The first part of the file describes the boundaries of the computational domain by breaking it into line segments. The second part of this file (after the label background grid information) relates to the background grid. The first information given for the background grid is the number of cells (triangles) and nodes, and then the surrounding nodes for each of the cells. The x and y locations are given for the nodes, and finally the stretching parameters are given for each node of the background grid. The stretching parameters are first used to place nodes along the boundaries of the computational region. The sides connecting these nodes called faces form the initial generation front. Cells are added beginning with the smallest face of the front.

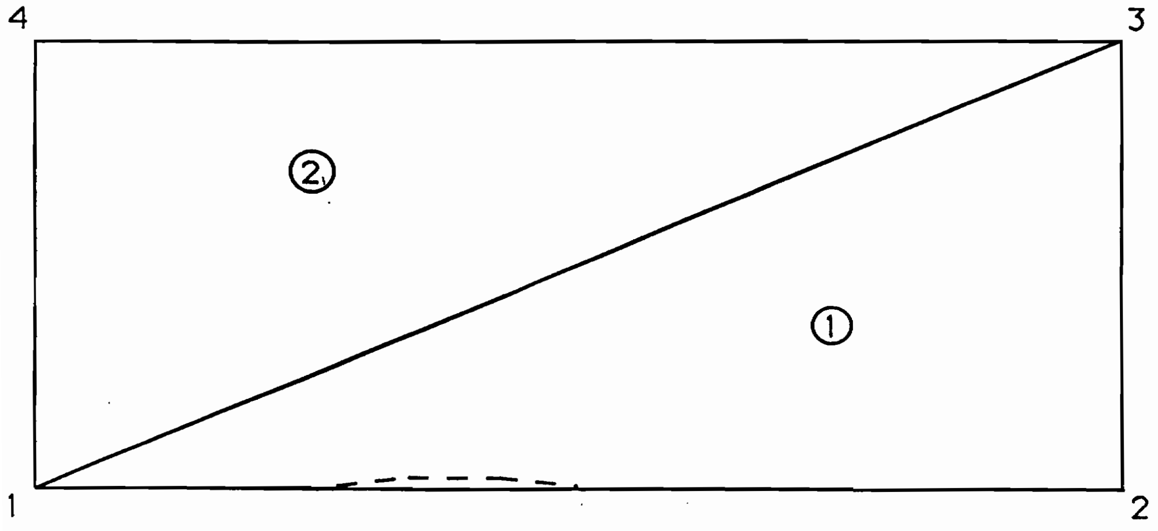


Figure 2) Background mesh for circular arc in a channel flow

```
 ndimn    npoli    nline
   2       15        6
 coords. of points
 1   -0.14166667e+01 0.00000000e+00
 2   -0.83333373e-01 0.00000000e+00
 3    0.33333302e-01 0.54133283e-02
 4    0.14999998e+00 0.21419998e-01
 5    0.26666665e+00 0.32853331e-01
 6    0.38333333e+00 0.39713331e-01
 7    0.50000000e+00 0.41999999e-01
 8    0.61666667e+00 0.39713334e-01
 9    0.73333335e+00 0.32853331e-01
 10    0.85000002e+00 0.21419996e-01
 11    0.96666670e+00 0.54133283e-02
 12    0.10833334e+01 0.00000000e+00
 13    0.35833334e+01 0.00000000e+00
 14    0.35833334e+01 0.20730000e+01
 15   -0.14166667e+01 0.20730000e+01
 nr   type  nr.of pts.      pts.                      bound. cond.
 1     1       2             1 2                        2 1 0
 2     3      11      2 3 4 5 6 7 8 9 10 11 12          2 1 0
 3     1       2            12 13                       2 1 0
 4     1       2            13 14                       3 3 0
 5     1       2            14 15                       2 2 0
 6     1       2            15 1                        1 4 0
background grid
NELEB NPOIB
    2     4
INTMAT
1    1  2  3
2    1  3  4
COORDINATES OF THE NODES
 1   -0.14166667e+01 0.00000000e+00
 2    0.35833334e+01 0.00000000e+00
 3    0.35833334e+01 0.20730000e+01
 4   -0.14166667e+01 0.20730000e+01
UNKNOWN AT THE NODES
1  1.0 0.0 0.028  1.0
2  1.0 0.0 0.028  1.0
3  1.0 0.0 0.028  1.0
4  1.0 0.0 0.028  1.0
```

Figure 3) Grid input file for 1005 element mesh of circular arc in a channel flow

The reason behind this is simple. In a typical problem, there will be regions of larger cells and regions of smaller cells. If the larger cells were generated first, there might not be room left to properly generate the smaller cells, so it makes sense to begin with the smaller faces which will result in smaller cells being generated first.

Figure 4 shows the transonic channel flow problem in the middle of the grid generation process. This diagram is not an actual grid that has been generated, but it is an example of a possible scenario. The outer boundary of the grid was the initial generation front. The set of faces on the interior of the grid that divide the generated mesh from the ungenerated region represent the current generation front. The face shown in bold is the current face that is being eliminated from the front by adding a new cell. Stretching parameters are interpolated from the background grid to the center of this face. From these parameters, an ideal point (labeled A) is generated. Figure 5 shows a diagram of how the ideal point is obtained from the values of the stretching parameters. The vector $\vec{\alpha}$ denotes the direction of stretching, with δ being the element size at right angles to $\vec{\alpha}$ and $s\delta$ being the element size in the direction of stretching.

Once the ideal point has been computed, it is necessary to perform checks to see if this point should be used to from a new cell. The first check is to see whether or not an existing node from the current front is close enough to the ideal point that it could be used in place of the ideal point to form a new cell. Once a node is selected, whether it be the ideal point or an existing node, a test is performed to check if the new cell would cross any of its faces with the current front. If this test is not passed, a new node is selected until one is found that passes the test. When one is found, the new cell is formed and the current face is removed from the front. This process repeated until the entire domain has been triangulated. A smoothing routine is then performed to improve the quality of the generated mesh.

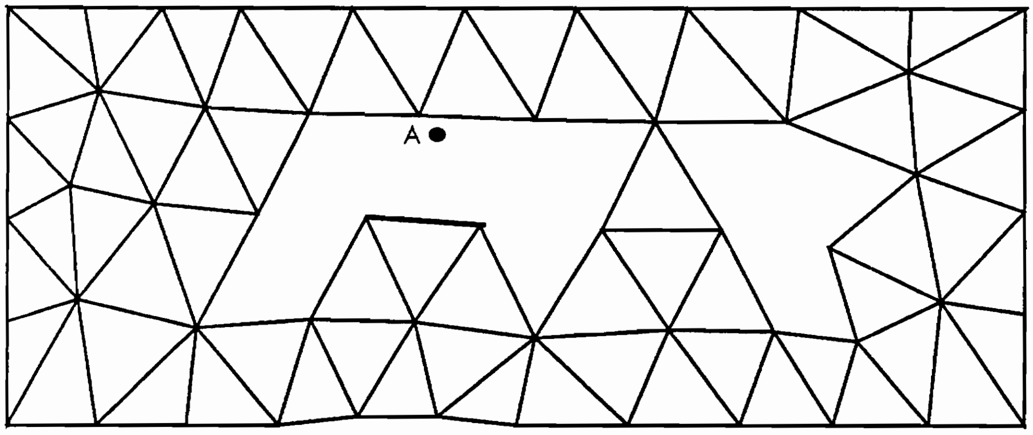


Figure 4) Partial grid for circular arc in a channel flow

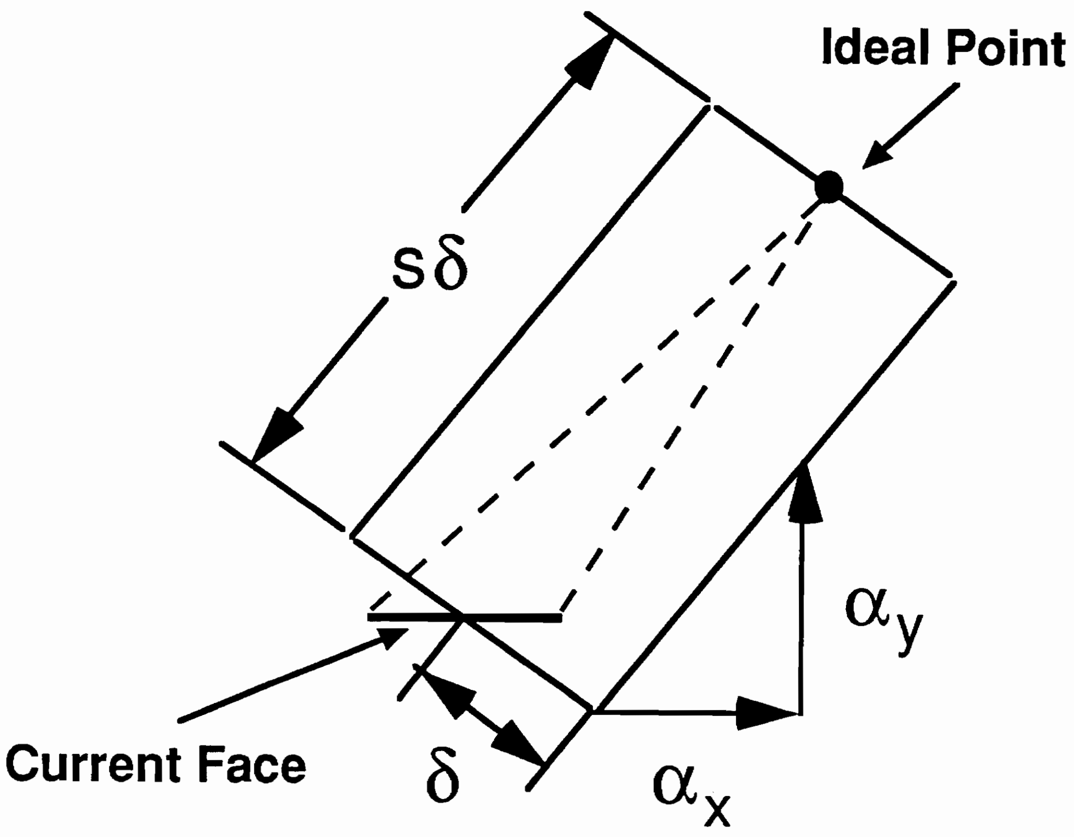


Figure 5) Ideal triangle shape for advancing front grid generation

2.2 Adaptive Mesh Regeneration

The grid generation process described requires input of a background mesh and stretching parameters. In general, a very simple background mesh with no streching is input for computing the initial grid. Once a solution has been computed on the initial grid, this grid will then become the background grid for the first mesh regeneration. This process requires utilizing some measure of the error in the computed solution on the initial grid to obtain the parameters $\hat{\alpha}$, δ, and s at the nodes. An improved mesh can then be generated with the initial grid as a background grid along with these computed parameters.

If the density ρ is chosen as the variable used to determine the error, then an estimate for the root mean square value of the local error is given by Morgan, et al.[10] as

$$E_i^{RMS} = \left(\frac{\int_0^{h_e} E_i^2 dx}{h_i} \right)^{\frac{1}{2}} \propto h_i^2 \|\frac{d^2 \hat{\rho}}{dx^2}\|_i \tag{2.2.1}$$

where h_i is the local element size, and $\hat{\rho}$ denotes the computed solution. A criterion for a uniform value of the error over the entire domain would then be

$$h_i^2 \|\frac{d^2 \hat{\rho}}{dx^2}\|_i = constant \tag{2.2.2}$$

which suggests that the value δ on the new mesh should be computed so that

$$\delta_i^2 \|\frac{d^2 \hat{\rho}}{dx^2}\|_i = constant \tag{2.2.3}$$

For two dimensional problems, equation (2.2.3) is employed in the local principle directions x and y separately

$$\delta_x^2 \|\frac{\partial^2 \hat{\rho}}{\partial x^2}\|_i = \delta_y^2 \|\frac{\partial^2 \hat{\rho}}{\partial y^2}\|_i = constant \tag{2.2.4}$$

The variables δ_x and δ_y represent typical local dimensions of a triangle.

This mesh regeneration procedure is generally performed several times in order to obtain a well refined mesh in as little CPU time as possible. At each remeshing stage the solution on the old grid is interpolated onto the newly generated grid. This is accomplished by simply using the values of the nearest point on the old mesh as the values at each point on the new mesh. This is not conservative, nor is it extremely accurate, but it is very fast, and the error introduced is dominated by high frequencies and hence will be damped out quickly. It should be emphasized that though the intermediate interpolation is not conservative, the solution on the final mesh will be conservative. This process is repeated a fixed number of times before turning the remeshing off and converging the solution to the steady-state on the final mesh.

2.3 Interactive Graphics

It is very difficult to predict when and how many remeshing stages will be required. Therefore, a graphics interface has been developed to allow the user to have a certain degree of control in guiding the flow solver. For the purpose of portability, all of the device dependent graphics subroutines have been separated from the main flow solver. Currently, three separate graphic libraries have been developed: a Sun GKS, an Iris4D, and a DI-3000 library. The DI-3000 graphics package is device independent and will work on any computer with the DI-3000 software installed such as the Cray-2 supercomputers Voyager and Navier.

When the flow solver is implemented, the user has the ability to execute in either automatic or interactive mode. If the automatic mode is chosen, the solver will execute without user interaction. If interactive mode is chosen, the solver will pause for input just before each remeshing step is to take place. The user then has the ability to view the current solution in the form of pressure contours and the current mesh, and then decide whether to remesh or continue computing on the current mesh. In this way, the user is able to guide the solver avoiding unneccessary or premature remeshing.

III. Governing Equations

3.1 Euler Equations

The two-dimensional Euler equations in integral form can be written as:

$$\frac{\partial}{\partial t}\iint_{\Omega} Q \, d\Omega + \oint_{S} \vec{F}\cdot\hat{n}\, ds = 0 \tag{3.1.1}$$

with

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_0 \end{bmatrix} \tag{3.1.2}$$

$$\vec{F} = f\hat{i} + g\hat{j} \tag{3.1.3}$$

and

$$f = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ \rho u h_0 \end{bmatrix} \qquad g = \begin{bmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ \rho v h_0 \end{bmatrix} \tag{3.1.4}$$

where Ω is the area of the domain and S surrounds Ω. The density is ρ, the velocity vector is $\vec{U} = (u\hat{\imath} + v\hat{\jmath})$, the total energy per unit mass is e_0, the total enthalpy per unit mass is h_0, the pressure is p, and the outward facing unit normal is $\hat{n}$. The system of equations is closed by the equation of state for a perfect gas:

$$p = \rho(\gamma - 1)\left[e_0 - \frac{(u^2+v^2)}{2}\right] \tag{3.1.5}$$

where γ is the ratio of specific heats and typically taken as 1.4 for air.

3.2 Spatial Discretization

3.2.1 Cell-Centered Discretization

A cell-centered finite-volume discretization stores the conserved variables at the centroid of each triangle, and the edges of a triangle define the faces of the control volume or cell, as shown in Fig. 6. The value stored at the centroid of the triangle is considered to be an area average value for the entire finite-volume cell. The number of cell faces is the same as the number of neighboring cells which will always be three for this scheme. The boundary values of the conserved variables are stored at the center of the cell faces along the boundary.

3.2.2 Cell-Vertex Discretization

A cell-vertex finite-volume discretization stores the conserved variables at the vertices of the triangles, and the faces of the control volume are formed by connecting the centroids of the triangles surrounding each vertex to the midpoints of the edges emanating from each vertex, as shown in Fig. 7. Each vertex then becomes the approximate cell center of the control volume created around it. The value stored at each vertex in the mesh is then considered to be the area averaged value for that finite-volume cell. In the cell-vertex case the number of cell faces varies. For a particular cell (vertex) it is equal to the number of its neighboring cells (vertices). The outward facing normal of the face in the cell-vertex scheme is found as an area-weighted average of the sides defining the control volume face. The boundary values of the conserved variables are stored at the vertices located on the boundaries.

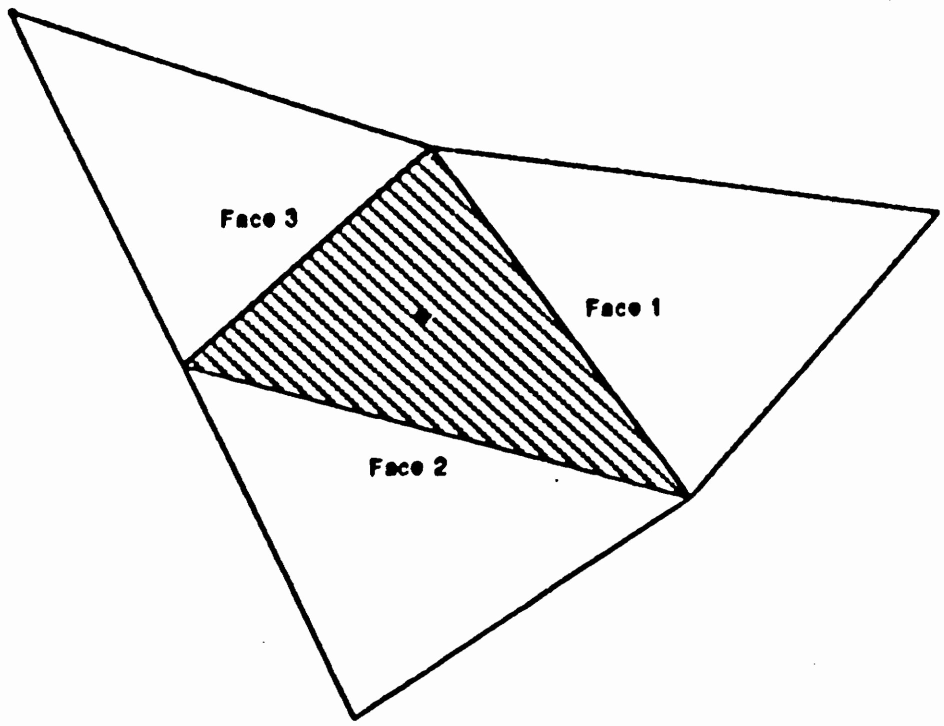


Figure 6) Control volume for cell-centered finite volume discretization

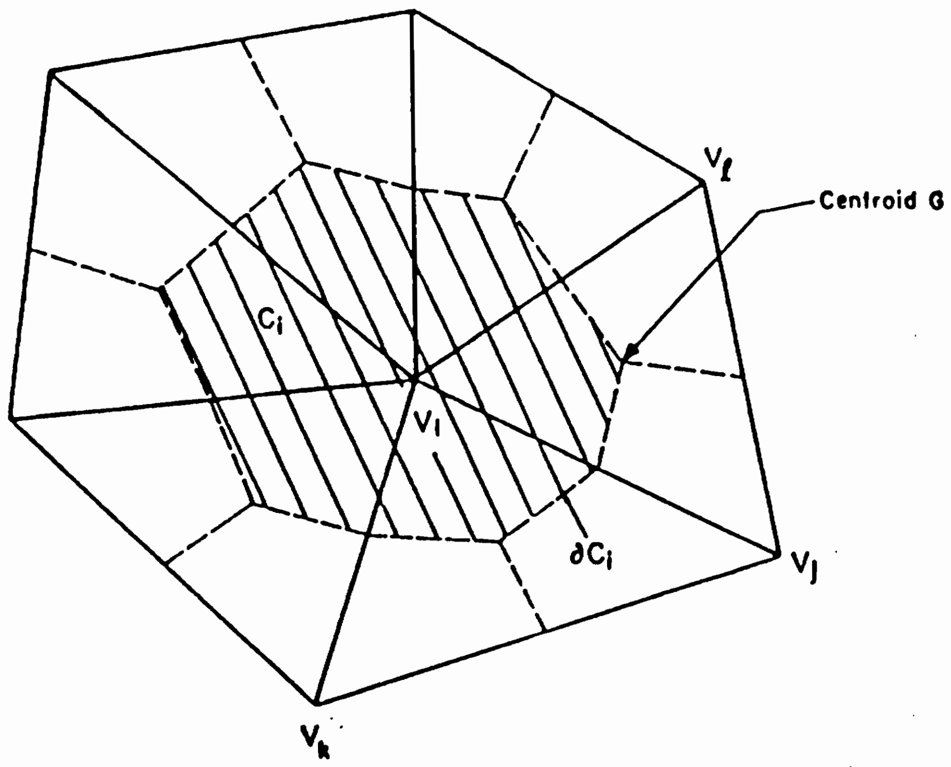


Figure 7) Control volume for cell-vertex finite volume discretization

3.2.3 Flux Calculation

For both the cell-centered and cell-vertex discretizations, equation (3.1.1) can be rewritten in the form:

$$V(C_i)\frac{\partial Q}{\partial t} = -\oint_{\partial C_i} \vec{F} \cdot \hat{n}\, ds \tag{3.2.1}$$

where C_i denotes a cell or control volume and $V(C_i)$ is the area of C_i.

To evaluate the right hand side of (3.2.1), we sum the flux vectors at each face of C_i. The resulting flux integral can be written in discrete form as:

$$\oint_{\partial C_i} \vec{F} \cdot \hat{n}\, ds = \sum_{j=\kappa(i)} F_{i,j} \Delta s \tag{3.2.2}$$

where $F_{i,j}$ is the numerical approximation for the flux at each cell face, Δs is the length of each cell face, and $\kappa(i)$ is a list of neighboring cells.

3.3 Roe's Flux Difference Splitting

Roe's flux difference splitting calculates the flux at an interface by approximating the solution of a one-dimensional Riemann problem[16]. Gudonov[17,18] originally solved the one-dimensional Riemann problem exactly; however, this problem required an iterative solution of the wave interaction at each interface which is computationally inefficient. Figure 8a shows a typical interface with two discrete fluid dynamic states labeled Q_L and Q_R. The associated one-dimensional Riemann problem for the Euler equations is given in Fig. 8b. The outside waves represent either shock waves or expansion fans while the inner wave is a contact discontinuity. F_L and F_R represent the fluxes based on the two states Q_L and Q_R respectively. The flux at the interface can be determined by solving for the difference in the flux across the characteristic waves shown in the diagram.

When the flux F is a linear function of Q, one can write

$$\frac{\partial Q}{\partial t} + A\frac{\partial Q}{\partial x} = 0 \tag{3.3.1}$$

where $A = \partial F/\partial Q$. The exact solution of the Riemann problem can be written in terms of the flux difference as

$$F_R - F_L = \sum_{k=1}^{m} \alpha_k \lambda_k e_k \tag{3.3.2}$$

where m is the vector length of Q, and λ_k, e_k, and α_k are the eigenvalues (wave speeds), eigenvectors, and wave strengths of the k^{th} wave respectively.

In the case of the Euler equations, however, F is a nonlinear function of Q. Roe's scheme solves the approximate linearized problem

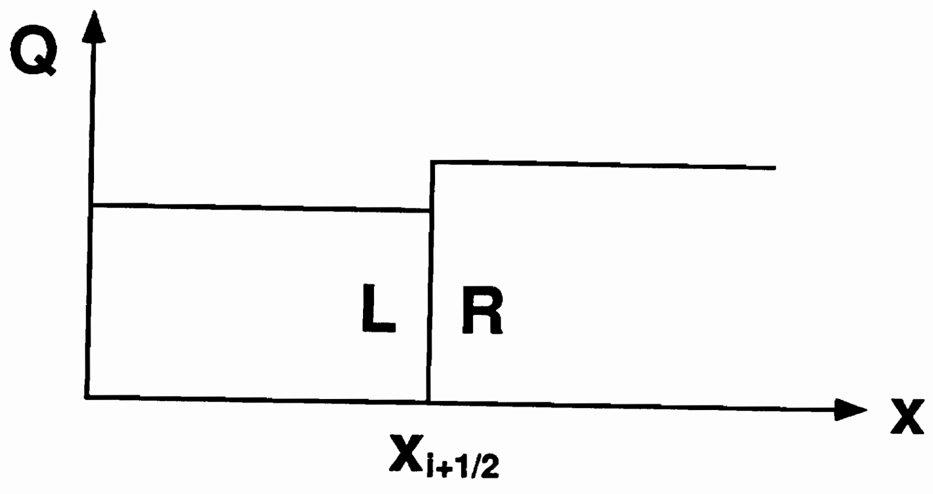
$$\frac{\partial Q}{\partial t} + \tilde{A}\frac{\partial Q}{\partial x} = 0 \tag{3.3.3}$$

where $\tilde{A}$ is called the Roe-averaged matrix. In order to satisfy the Rankine-Hugoniot shock jump condition, the Roe-averaged matrix is constructed so that a flux difference between two fluid states is satisfied identically by

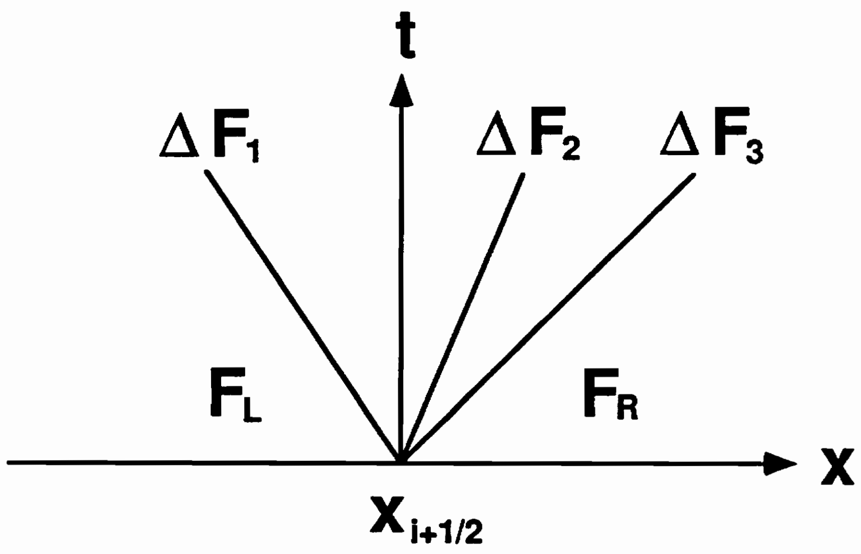
$$\Delta F = F(Q_R) - F(Q_L) = \tilde{A}(Q_R - Q_L) = \tilde{A}\Delta Q \tag{3.3.4}$$

Referring back to Fig. 8, the value of the flux at the interface $F_{i+\frac{1}{2}}$ can be found either by beginning with the value of the flux at the left state F_L and adding the flux differences across waves with negative eigenvalues, or by beginning with the value of the flux at the right state F_R and subtracting the flux differences across waves with positive eigenvalues. By taking the average of these two methods, Roe finds that the flux at the interface is given by

$$F_{i+\frac{1}{2}} = \frac{1}{2}[F(Q_L) + F(Q_R) - |\tilde{A}|\Delta Q] \tag{3.3.5}$$



(a)



(b)

Figure 8) (a) Typical cell interface with left and right fluid states,
(b) one dimensional Riemann problem with characteristic waves

The absolute value symbols placed around the matrix simply indicate that the absolute values of the eigenvalues of $\tilde{A}$ were used to evaluate $\tilde{A}$. The matrix $\tilde{A}$ can be represented by

$$\tilde{A} = \tilde{T}\Lambda\tilde{T}^{-1} \tag{3.3.6}$$

where $\tilde{T}$ are the right eigenvectors, $\tilde{T}^{-1}$ are the left eigenvectors, and Λ is a diagonal matrix of the eigenvalues of $\tilde{A}$. The difference in the flux can then be written as

$$\Delta F = |\tilde{A}|\Delta Q = \tilde{T}|\Lambda|\tilde{T}^{-1}\Delta Q \tag{3.3.7}$$

where

$$\tilde{T} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ \tilde{u} & 0 & \tilde{u}+\tilde{a} & \tilde{u}-\tilde{a} \\ \tilde{v} & 1 & \tilde{v} & \tilde{v} \\ \frac{\tilde{q}^2}{2} & \tilde{v} & \tilde{h}_0+\tilde{a}\tilde{u} & \tilde{h}_0-\tilde{a}\tilde{u} \end{bmatrix} \tag{3.3.8}$$

and

$$\Lambda = diag(\tilde{u}, \tilde{u}, \tilde{u}+\tilde{a}, \tilde{u}-\tilde{a}) = diag(\tilde{\lambda}_1, \tilde{\lambda}_2, \tilde{\lambda}_3, \tilde{\lambda}_4) \tag{3.3.9}$$

and

$$\tilde{T}^{-1}\Delta Q = \frac{1}{2\tilde{a}^2}\begin{bmatrix} 2(\tilde{a}^2\Delta\rho - \Delta p) \\ \Delta p + \tilde{\rho}\tilde{a}\Delta u) \\ \Delta p - \tilde{\rho}\tilde{a}\Delta u) \end{bmatrix} \tag{3.3.10}$$

In this notation, $\tilde{T}^{-1}\Delta Q$ represents the wave strengths. The variables inside the matrices are called Roe-averaged variables. They can be found from

$$\tilde{\rho} = (\rho_L\rho_R)^{\frac{1}{2}} \qquad \tilde{h}_0 = \frac{\rho_L^{\frac{1}{2}}h_{0L} + \rho_R^{\frac{1}{2}}H_{0R}}{\rho_L^{\frac{1}{2}} + \rho_R^{\frac{1}{2}}}$$

$$\tilde{u} = \frac{\rho_L^{\frac{1}{2}}u_L + \rho_R^{\frac{1}{2}}u_R}{\rho_L^{\frac{1}{2}} + \rho_R^{\frac{1}{2}}} \qquad \tilde{a}^2 = (\gamma-1)(\tilde{h}_0 - \frac{\tilde{u}^2}{2})$$

3.4 Steger-Warming Jacobian

When implementing implicit schemes, it is necessary to calculate a Jacobian matrix given by

$$A = \frac{\partial F}{\partial Q} \tag{3.4.1}$$

Because Roe's flux involves complicated expressions for the Roe-averaged variables, it is cumbersome to fully linearize. One approximation to the Roe Jacobian matrix is a Steger-Warming Jacobian matrix[19]. The Steger-Warming Jacobian can be represented by

$$A^{\pm} = T\Lambda^{\pm}T^{-1} \tag{3.4.2}$$

where T are the right eigenvectors of A given by

$$T = \begin{bmatrix} 1 & 0 & 1 & 1 \\ u & 0 & u+a & u-a \\ v & 1 & v & v \\ \frac{q^2}{2} & v & h_0 + au & h_0 - au \end{bmatrix} \tag{3.4.3}$$

and $\Lambda^{\pm}$ is a diagonal matrix of the positive and negative eigenvalues of A

$$\Lambda^{\pm} = diag(\lambda_1^{\pm}, \lambda_2^{\pm}, \lambda_3^{\pm}, \lambda_4^{\pm}) \tag{3.4.4}$$

and T^{-1} are the left eigenvectors of A given by

$$T^{-1} = \begin{bmatrix} 1 - \phi q^2 & 2\phi u & 2\phi v & -2\phi \\ -v & 0 & 1 & 0 \\ \phi(\frac{q^2}{2} - \frac{u}{2a}) & -\phi u + \frac{1}{2a} & -\phi v & \phi \\ \phi(\frac{q^2}{2} + \frac{u}{2a}) & -\phi u - \frac{1}{2a} & -\phi v & \phi \end{bmatrix} \tag{3.4.5}$$

where

$$\phi = \frac{\gamma - 1}{2a^2} \qquad q^2 = u^2 + v^2 \tag{3.4.6}$$

With this formulation the A^+ matrix represents the contributions from the positive wave speeds of the left state Q_L, and the A^- matrix contains the contributions from the negative wave speeds of the right state Q_R.

3.5 Constructing the Left and Right Fluid States

To apply Roe's characteristic-based flux difference splitting scheme[16], it is first necessary to obtain two discrete fluid dynamic states separated by a cell interface. In the approach taken here, the state variables are interpolated from the cell centers to the cell faces creating a left and right state for each interior face. For first-order accuracy, a Taylor's series approximation to the left and right fluid states is simply

$$Q_L = Q(C_i), \qquad Q_R = Q(C_j) \tag{3.5.1}$$

where the R,L subscripts indicate right and left fluid states, and C_i and C_j represent adjacent cells i and j, respectively. This represents a piecewise constant cell distribution of the flow variables. In order to obtain higher-order results, this is replaced by assuming a piecewise linear distribution of cell data. From this reconstruction, higher-order left and right states are found, and their values are limited so that no new extrema are generated.

3.5.1 Cell-Centered Higher-Order Correction

A piecewise linear redistribution of the cell averaged flow variables is given by Barth[5] as

$$Q(x,y) = Q(x_0,y_0) + \nabla Q \cdot \vec{r} \tag{3.5.2}$$

where $\vec{r}$ is the vector from the cell center (x_0, y_0) to any point (x, y) in the cell, and ∇Q represents the solution gradient in the cell. Note that this equation is simply the first-order accurate Taylor approximation plus a higher-order correction. With this approximation, the solution gradient ∇Q is constant in each cell and can be computed from

$$\nabla Q_A = \frac{1}{S_\Omega} \oint_\Omega Q \; \hat{n} \; d\Omega \tag{3.5.3}$$

where S_Ω is the area contained in the path of integration. For the cell-centered case, the path chosen passes through the centroids of the cells B,C, and D which surround cell A, as indicated in Fig. 9.

Consider a limited version of the linear function about the centroid of cell A

$$Q(x,y)_A = Q(x_0,y_0)_A + \Phi_A \nabla Q_A \cdot \vec{r}_A, \qquad \Phi_A \;\epsilon\; [0,1] \tag{3.5.4}$$

In order to find the value of Φ_A, a monotonicity principle is enforced on the unlimited quantities $Q_{A_i} = Q(x_i,y_i)_A$ calculated in (3.5.2) at the faces of cell A. It requires that the values computed at the faces must not exceed the maximum and minimum of neighboring cell values including the value in cell A, i.e. that

$$Q_A^{min} \leq Q_{A_i} \leq Q_A^{max} \tag{3.5.5}$$

where $Q_A^{min} = min(Q_A, Q_{neighbors})$ and $Q_A^{max} = max(Q_A, Q_{neighbors})$. The value Φ_A can now be calculated for each face i of cell A as

$$\Phi_{A_i} = \begin{cases} min\left(1, \dfrac{Q_A^{max} - Q_A}{Q_i - Q_A}\right), & if \;\; Q_i - Q_A > 0 \\ min\left(1, \dfrac{Q_A^{min} - Q_A}{Q_i - Q_A}\right), & if \;\; Q_i - Q_A < 0 \\ 1 & if \;\; Q_i - Q_A = 0 \end{cases} \tag{3.5.6}$$

with $\Phi_A = min(\Phi_{A_i})$, where i is the index of the set of faces surrounding cell A. New limited values for Q_{A_i} at each of the faces of cell A are then calculated from equation (3.5.4) using the the value of Φ_A calculated for the cell.

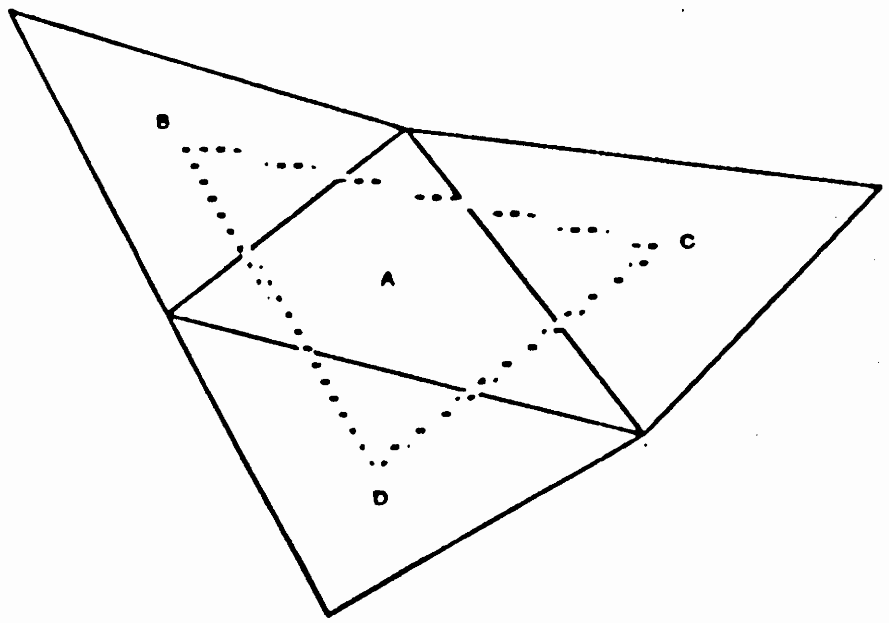


Figure 9) Integration path for cell-centered gradient calculation

3.5.2 Cell-Vertex Higher-Order Correction

Alternatively, an approach shown in Fig. 10 can be used to evaluate ∇Q for the cell vertex scheme[8]. For each edge surrounding vertex V_i in cell C_i, a search can be made of the triangular elements surrounding nodes V_i and V_j which define each edge. Triangular elements which contain the line defined by the vectors, $\overrightarrow{V_{i'}V_i}$, and $\overrightarrow{V_jV_{j'}}$ and surrounding nodes, V_i and V_j, respectively, are used to evaluate the gradient for higher-order MUSCL differencing. Phantom nodes, $V_{i'}$ and $V_{j'}$, define the points where data is necessary to evaluate a MUSCL differencing formula in a structured grid manner. Higher-order data is reconstructed at the midpoint of the edge defined by vertices V_i and V_j. The MUSCL differencing formulas for Fig. 10, can be written as

$$\begin{aligned}(Q_L)_{i,j} &= Q_i + \frac{s}{4}\left\{(1-\kappa s)\Delta_- + (1+\kappa s)\Delta_+\right\} Q_i \\ (Q_R)_{i,j} &= Q_j - \frac{s}{4}\left\{(1-\kappa s)\Delta_+ + (1+\kappa s)\Delta_-\right\} Q_j\end{aligned} \tag{3.5.7}$$

where

$$\begin{aligned}\Delta_-(Q_i) &= Q_i - Q_{i'} \qquad & \Delta_+(Q_i) &= Q_j - Q_i \\ \Delta_-(Q_j) &= Q_j - Q_i \qquad & \Delta_+(Q_j) &= Q_{j'} - Q_j\end{aligned} \tag{3.5.8}$$

where s is Van Albada's limiter[20]

$$s = \frac{2\Delta_+\Delta_- + \delta}{(\Delta_+)^2 + (\Delta_-)^2 + \delta} \tag{3.5.9}$$

and δ is a small number used to prevent division by zero in regions of small gradient.

The limiter, s, serves to limit the accuracy of the MUSCL-differencing formulas in regions of large gradient. This limiting causes a loss of accuracy in the solution but does insure that oscillations in the numerical solution near discontinuities are minimized or prevented. Van Albada's limiter was used because it acted in a continuously differentiable manner which tended to prevent limit cycles in the convergence history.

The parameter κ controls the accuracy of the MUSCL-differencing. With $\kappa = -1$, the solution is second-order accurate and fully upwinded. Using $\kappa = 1/3$, the solution is upwind biased. The backward and forward differences to imaginary points can be approximated by

$$\begin{aligned} \Delta_-(Q_i) &= \vec{\nabla} Q_i \cdot \overrightarrow{V_{i'}V_i} = \vec{\nabla} Q_i \cdot \overrightarrow{V_iV_j} \\ \Delta_+(Q_j) &= \vec{\nabla} Q_j \cdot \overrightarrow{V_jV_{j'}} = \vec{\nabla} Q_j \cdot \overrightarrow{V_iV_j} \end{aligned} \tag{3.5.10}$$

The notation $\vec{\nabla} Q_i$ and $\vec{\nabla} Q_j$ represent the gradient in the triangular element surrounding V_i and containing vector $\overrightarrow{V_{i'}V_i}$ and surrounding V_j and containing vector $\overrightarrow{V_jV_{j'}}$, respectively.

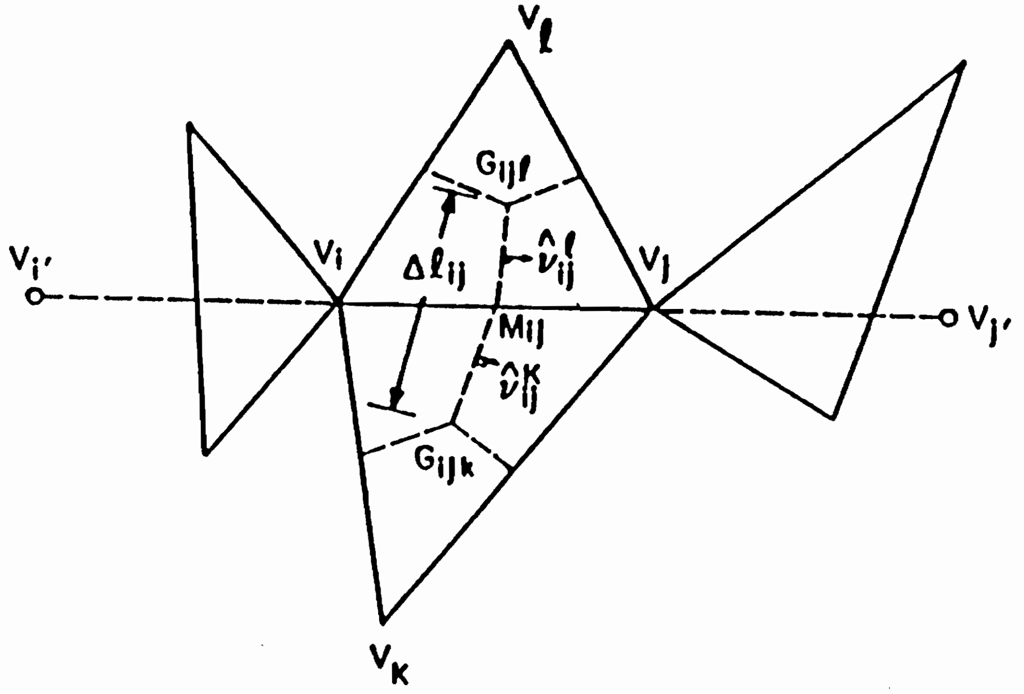


Figure 10) Phantom cells for cell-vertex gradient calculation

line Gauss-Seidel type relaxation, and LU decomposition algorithms all utilize this form of time integration. The LU decomposition calculates an exact solution to the linear system given in equation (4.0.4), while the other implicit schemes calculate approximations to the solution. However, due to the approximate linearization through the use of a Steger-Warming Jacobian and nonlinearity of the residual, the solution to equation (4.0.4) is only an approximation itself of the solution to equation (4.0.2).

For solving linear problems using classic relaxation schemes, it is convenient to decompose A according to

$$A = M + D + N \tag{4.0.6}$$

where M is a lower triangular matrix, D is a diagonal matrix, and N is an upper triangular matrix.

In the last several years, extensive research has been directed towards the use of sparse matrix solvers for the solution of non-symmetric and non-positive definite matrices[21,22,23]. The matrices involved in the present work are non-symmetric and sparse. For a two-dimensional perfect gas problem with 1000 vertices in a cell-vertex code, there are 4000 unknowns and $16 \cdot 10^6$ matrix coefficients. Only about 28,000 to 32,000 coefficients are typically non-zero. Many of the sparse matrix solvers are iterative and only require that the user supply a subroutine to compute a matrix multiply of A and a vector. Generally, the matrix multiply is the most expensive operation required to perform a single iteration of the sparse matrix solver.

IV. Time Integration on Unstructured Meshes

In order to obtain a steady-state solution, the spatially discretized Euler equations must be integrated in time. The time integration scheme is independent of the spatial discretization used. The two basic types of time integration schemes studied here are explicit and implicit algorithms. Both types make use of

$$V\frac{\partial Q}{\partial t} = R(Q) \tag{4.0.1}$$

where $R(Q)$ is the residual and is a discrete representation of the steady-state terms. Explicit time integration evaluates the residual at time level n, while implicit time integration evaluates the residual at time level $n+1$. Therefore, Euler implicit time integration can be represented in delta form as

$$V\frac{\Delta Q}{\Delta t} = R^{n+1} \tag{4.0.2}$$

where $\Delta\{\} = \{\}^{n+1} - \{\}^{n}$ and V is the cell area. In order to solve the equation in this form, it is necessary to linearize the equation in time which can be accomplished by

$$V\frac{\Delta Q}{\Delta t} = R^{n} + \frac{\partial R^{n}}{\partial Q}\Delta Q \tag{4.0.3}$$

Equation 4.0.3 can then be written as the linear system

$$A\Delta Q = R^{n} \tag{4.0.4}$$

where

$$A = \left(\frac{V}{\Delta t}I - \frac{\partial R^{n}}{\partial Q}\right). \tag{4.0.5}$$

For fluid dynamic problems, A is generally a block matrix of dimension N (the number of cells) with a variable bandwidth. The point Jacobi, point Gauss-Seidel,

4.1 Runge-Kutta Time Integration

Runge-Kutta type time integration was developed by Jameson[24] as a variation of the classical Runge-Kutta methods. An m-stage Runge-Kutta scheme is

$$\begin{aligned}
Q^{(0)} &= Q^{(n)} \\
Q^{(1)} &= Q^{(0)} + \alpha_1 \frac{\Delta t}{V} R(Q^{(0)}) \\
&\vdots \\
Q^{(m-1)} &= Q^{(0)} + \alpha_{m-1} \frac{\Delta t}{V} R(Q^{(m-2)}) \\
Q^{(m)} &= Q^{(0)} + \alpha_m \frac{\Delta t}{V} R(Q^{(m-1)}) \\
Q^{(n+1)} &= Q^{(m)}
\end{aligned} \qquad (4.1.1)$$

where $Q^{(n)}$ is the vector of state variables at time n, $R(Q)$ is the right-hand side of (3.2.1), V is the cell area, and the α's are weighting factors given by

$$\alpha_k = \frac{1}{m-k+1} \qquad for \quad k = 1, ..., m \qquad (4.1.2)$$

These values for α_k yield m-order time accuracy for a linear system. However for non-linear problems, the 4-stage scheme which was investigated in both the cell-centered and cell-vertex code is second order accurate in time. The weighting coefficients used in this work were experimentally determined to accelerate convergence to the steady-state in an upwind, structured code[25] and are given by

$$\alpha_1 = 0.15 \quad \alpha_2 = 0.3275 \quad \alpha_3 = 0.57 \quad \alpha_4 = 1 \qquad (4.1.3)$$

Runge-Kutta time integration is easily vectorized and simple to code. It also requires very little storage when compared to implicit schemes. However, it requires a small Courant number for stability and converges slowly as compared to implicit schemes.

4.1.1 Runge-Kutta with Residual Smoothing

The Runge-Kutta scheme could be accelerated by applying implicit residual smoothing at every stage of the time integration[26]. This was performed only for the cell-vertex code, but could be easily incorporated into the cell-centered code. Residual smoothing is essentially a Laplacian filtering of the numerical values of the residuals. The residuals were smoothed for every stage of the Runge-Kutta solver. The following equation was solved for the new value of the residual

$$\bar{R}_i = R_i + \epsilon \nabla^2 \bar{R}_i \tag{4.1.4}$$

where $\bar{R}_i$ is the Laplacian filtered value of R_i. The undivided Laplacian, $\nabla^2 \bar{R}_i$, can be represented on an unstructured mesh as

$$\nabla^2 \bar{R}_i = \sum_{j=\kappa(i)} (\bar{R}_j - \bar{R}_i) \tag{4.1.5}$$

The variable $\kappa(i)$ contains a list of vertices that surround vertex V_i and are adjacent to V_i. The resulting implicit equation for $\bar{R}_i$ is solved by Jacobi iteration. Typically, two Jacobi iterations were performed with $\epsilon = 0.5$. Smoothing the residuals is computationally efficient compared to evaluating the residuals using an upwind scheme. It also allows for a larger maximum time step. As a result, residual smoothing is very effective in improving the overall convergence of the Runge-Kutta time integration scheme.

4.2 Point Jacobi Time Integration

The point Jacobi method was used in the cell-vertex code to compute an approximate solution to the linear system. The block matrices on the diagonal were inverted and multiplied by the right hand side.

$$\Delta Q = D^{-1} R^n \tag{4.2.1}$$

where D is a diagonal matrix of block matrices from the diagonal of A. As can easily be seen, the method is actually a block Jacobi iteration with the blocks matrices being formed from the linearization of the coupled equations at each grid point. The vector Q is updated after ΔQ is found and a new residual vector and Jacobian is computed. In the point Jacobi method, it is not necessary to compute the off diagonal terms in the A matrix. Since only the diagonal terms are considered, there is no coupling between elements so the scheme is easily vectorized over the elements.

4.3 Point Gauss-Seidel Time Integration

To implement the point Gauss-Seidel method, the off diagonal terms of the Jacobian matrix A must be found. The iteration sequence was identical to point Jacobi, except that the off-diagonal terms of matrix A multiplied the current approximation to ΔQ and were subtracted from the residual. The point Gauss-Seidel method can be written for $i = 1, ..., N$ as

$$\Delta Q_i^{(1)} = D_{i,i}^{-1} \left[R_i^n - \sum_{j=1}^{i-1} M_{i,j} \Delta Q_j^{(1)} - \sum_{j=i}^{N} N_{i,j} \Delta Q_j^{(0)} \right] \tag{4.3.1}$$

The superscripts on ΔQ refer to the inner iteration number of the Gauss-Seidel method on the linear system. Typically, $\Delta Q^{(0)}$ is an initial guess for the Gauss-Seidel solver and $\Delta Q^{(1)}$ is used to update Q^n to Q^{n+1}. Due to the recursive nature of the point Gauss-Seidel algorithm, complete vectorization of this method on a modern vector supercomputer is not possible. Point Gauss-Seidel can be made symmetrical by sweeping through the list of vertices in both directions before updating Q. Utilizing equation (4.3.1) as the first direction, symmetric point Gauss-Seidel can be written for $i = N, ..., 1$ as

$$\Delta Q_i^{(2)} = D_{i,i}^{-1}\left[R_i^N - \sum_{j=1}^{i} M_{i,j}\Delta Q_j^{(1)} - \sum_{j=i+1}^{n} N_{i,j}\Delta Q_j^{(2)}\right] \tag{4.3.2}$$

with $\Delta Q^{(2)}$ being used to update Q. As with the Point Jacobi method, this scheme was only investigated in the cell-vertex code. The implementation of these schemes in a cell-centered code is straightforward.

4.4 A Line Gauss-Seidel Type Time Integration

An approach similar to the line Gauss-Seidel method in use on structured meshes was developed for the cell-centered computer code. In order to perform the line Gauss-Seidel type relaxation, it is first necessary to renumber the cells to get as many elements of A as possible into block tridiagonal form. The resulting matrix A is given in Fig. 11 for a 1005 element mesh. The matrix can be subdivided into several submatrices (denoted by the blocks) of variable length. A close up of the third submatrix, Fig. 12, clearly shows the block tridiagonal form. Each submatrix is then solved by an equation of the form

$$T\Delta Q = R(Q^n, Q^{n+1}) \tag{4.4.1}$$

where T denotes a tridiagonal submatrix, and the residual becomes a function of Q^n and Q^{n+1}. Since the submatrices are independent from each other, the inversion of the submatrices can be vectorized over the number of submatrices. Vectorization can only be done for submatrices with an equal number of elements. When the matrix A is subdivided, a minimum number of elements per submatrix is imposed.

The solution procedure is then vectorized for the elements in every submatrix up to the minimum number of elements allowed, and the remainder of the elements for each submatrix which is larger than the minimum length are computed in scalar mode.

Once the block tridiagonal submatrices have been inverted, the values of ΔQ are solved for sequentially over each submatrix of the matrix A. The elements to the left of the submatrices on a forward sweep (or the elements to the right of the submatrices on a reverse sweep) through the matrix are included in the solution procedure through relaxation. After the values for ΔQ of a submatrix are calculated, the residual for the next submatrix is computed using these new updated values which results in a non-linear update of the residual.

4.4.1 Tridiagonal Renumbering Scheme

The renumbering scheme developed to put as many elements as possible next to the diagonal of the matrix A was very straightforward. Initially a list was compiled of all of the available cells for renumbering which included all of the cells at first. Once a cell was chosen, it was removed from the list and was no longer considered in any part of the selection process. The first cell chosen was simply the first cell given by the grid generator. The next cell would be one of the neighbors of this cell. If the neighboring cell was a boundary cell, it was chosen above all others. Otherwise the cell with the most neighbors was selected. Eventually, a cell will be chosen that has no neighbors still on the list. This always occurs with a boundary cell. When this happens, the only way to continue renumbering is to select a new starting cell from the available list and repeat the process described above.

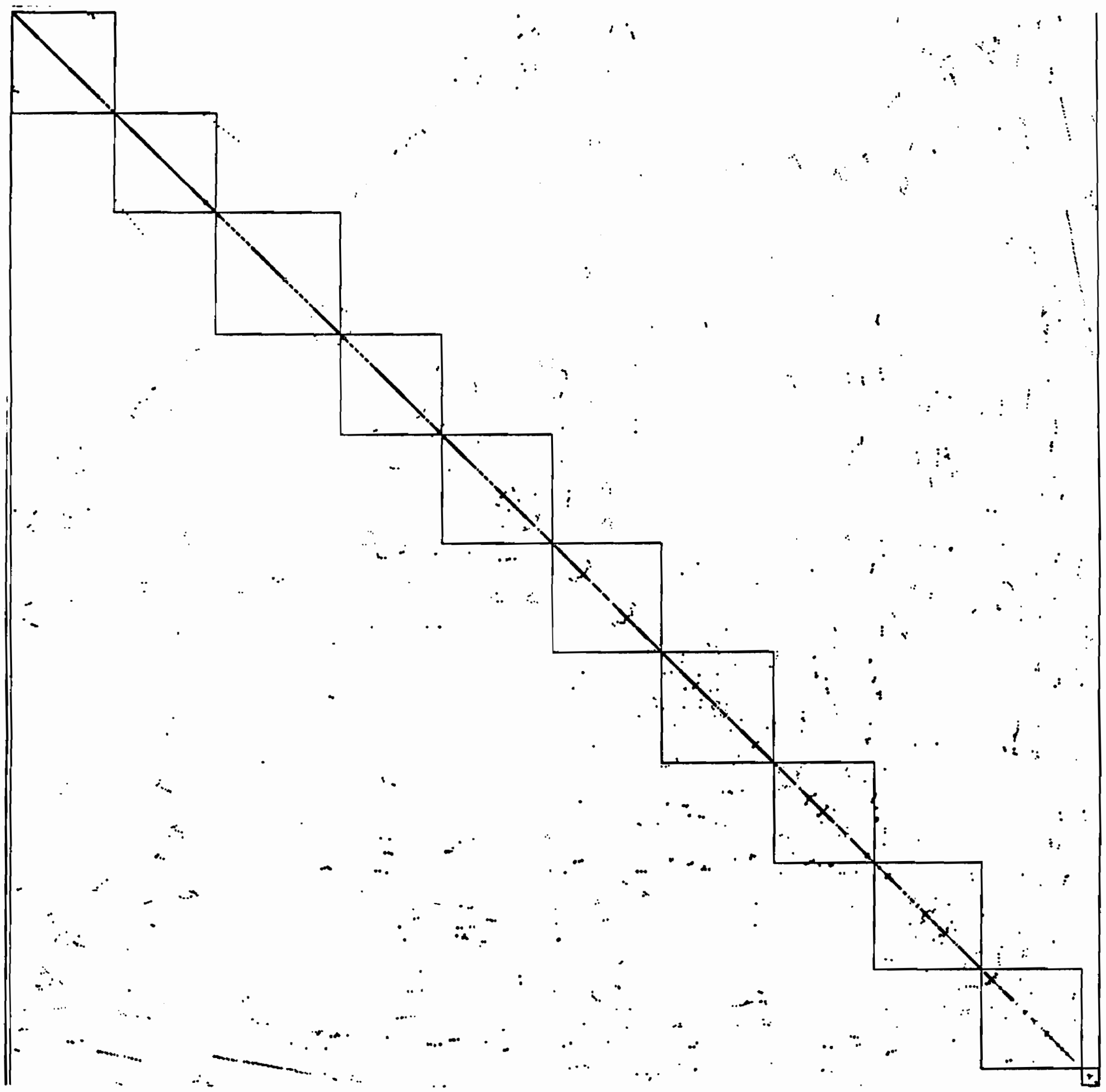


Figure 11) Tri-diagonal form of Jacobian matrix for line Gauss-Seidel relaxation

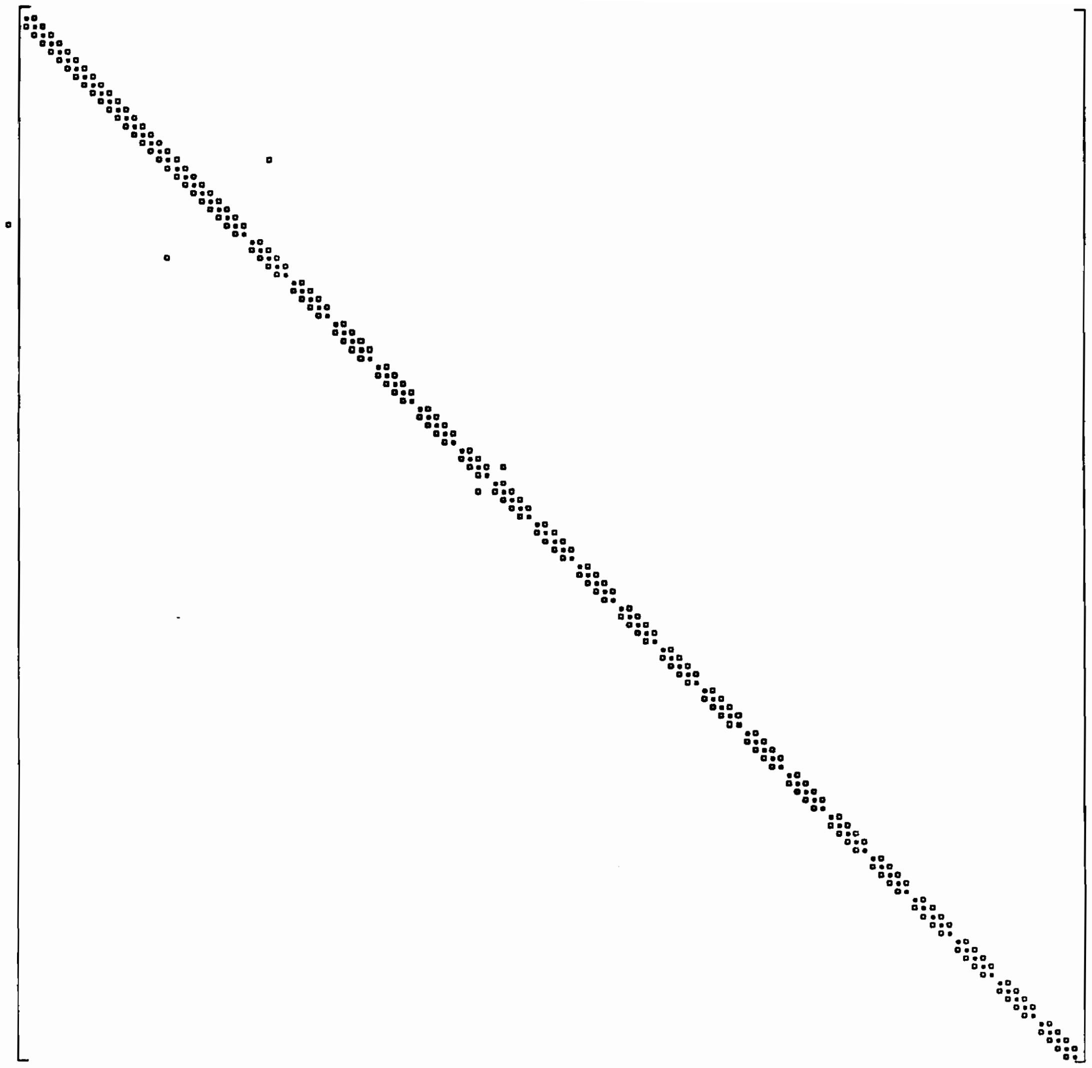


Figure 12) Close-up of third submatrix of Jacobian for line Gauss-Seidel relaxation

This results in a natural sectioning of the block tridiagonal portion of the matrix. However, if each natural break was used to break the matrix into submatrices, there would be some very short submatrices especially toward the end of the matrix when the available list is very short. This would result in large discrepancies in the size of submatrices which would result in too much scalar mode computation. Therefore, a minimum number of cells per submatrix is imposed by the user. If the natural break occurs on or after the minimum selected number of cells per submatrix, then a new block is formed. Otherwise, the break is ignored and treated as any other part of that submatrix.

The reason for selecting a boundary cell first now becomes more apparent. The only path to a boundary cell is through the cell neighboring the boundary cell. If it were passed by, the boundary cell would be selected by itself sometime later in the renumbering scheme. It would have no coupling with interior points and the implicit boundary conditions would be nullified resulting in poor convergence.

This renumbering scheme is not very sophisticated and could be improved to yield better performance. Two areas of potential improvement are: the discrepancies in the size of submatrices needs to be reduced, and the method for selecting a new starting cell is random.

4.5 LU Decomposition Time Integration

The LU (Lower-Upper) decomposition method which was implemented in he cell-centered code attempts to solve the system A exactly in the absence of round-off error. The solution procedure involves factoring the linearized coefficient matrix A into the product of a lower triangular and upper triangular matrix (L and U) such that

$$[LU]\Delta Q = R^n \tag{4.5.1}$$

One then solves the system $L\Delta Q^* = R^n$ by forward substitution and $U\Delta Q = \Delta Q^*$ by backward substitution. This procedure reduces to Newton's method as $\Delta t \rightarrow \infty$, and as a result it exhibits quadratic convergence to the solution of the non-linear system of equations $R(Q^{n+1})$ when the left hand side of the linear system is a Newton linearization of the right hand side.

A reuse capability involves freezing the LU decomposition in time and performing only the relatively inexpensive forward and backward substitutions to advance the solution vector Q. This can be represented by the algorithm

$$\Delta Q^{n+k} = (U^{-1}L^{-1})^n R(Q^{n+k}, Q^{n+k+1}) \tag{4.5.2}$$

where n represents the time at which the LU is frozen, and $n+k$ is the current time step. This approach is very efficient during the latter stages of convergence once the basic flow physics have been established.

4.5.1 Reverse Cuthill-McKee

A Reverse Cuthill-McKee[27] renumbering scheme was investigated as a preprocessor for the LU decomposition in order to reduce the fill of the Jacobian matrix (Fig. 13), thus reducing the storage requirements and CPU time required. The algorithm divides the cells into level sets, S_i, with S_1 consisting of a single cell. The next, S_2, consists of all the neighbors of this cell. The set S_3 consists of all the neighbors of the cells in S_2 that are not in S_1 or S_2. The general set S_i consists of all the neighbors of the cells of S_{i-1} that are not contained in any of the previous sets S_{i-1}, S_{i-2}, ... S_1. The cells in any given set S_i have a specific order based on the number of neighboring cells each cell in S_i has in the set S_{i+1}. The cells in the set are ordered from the greatest number of neighbors to the least number. Globally, the cells are numbered starting with the cell in the first set S_1. Cells from the set

S_2 are then taken in order, followed by those in set S_3, and so on until all the cells are renumbered. The final step is then to reverse the order of all the cells.

With Reverse Cuthill-McKee, the selection of the starting cell can sometimes significantly affect the overall results of the renumbering. In the code, the first cell was chosen arbitrarily to be the first cell generated by the grid generator, which turns out to be a cell on the boundary, usually in a corner. This choice appeared to work well, but more could be done to study various choices or enhance the selection of the starting cell.

A key factor in the choice of the Reverse Cuthill-McKee renumbering scheme is that it is extremely fast. Other more complicated schemes such as nested dissection may sometimes result in faster LU decomposition times, however, the cost of the renumbering can be much greater. This can pay off when the renumbering needs to be done only once as is the case when a single unchanging grid structure is used. In this case, however, with grid adaptation and regeneration, the renumbering step must be performed on each new grid and therefore it needs to be fast.

4.5.2 Skyline Solver

After the Reverse Cuthill-McKee renumbering, the sparsity of the variable-banded matrix A can be exploited. Because LU decomposition without pivoting preserves the band structure, it is only necessary to store and compute those coefficients inside the variable bandwidth. It is difficult to specify a given amount of storage for a given number of cells because the storage is affected by other factors such as the shape of the domain and the cell distribution inside the domain. The variable bandwidth solver vectorizes over the local bandwidth, just as a banded solver does for a structured grid.

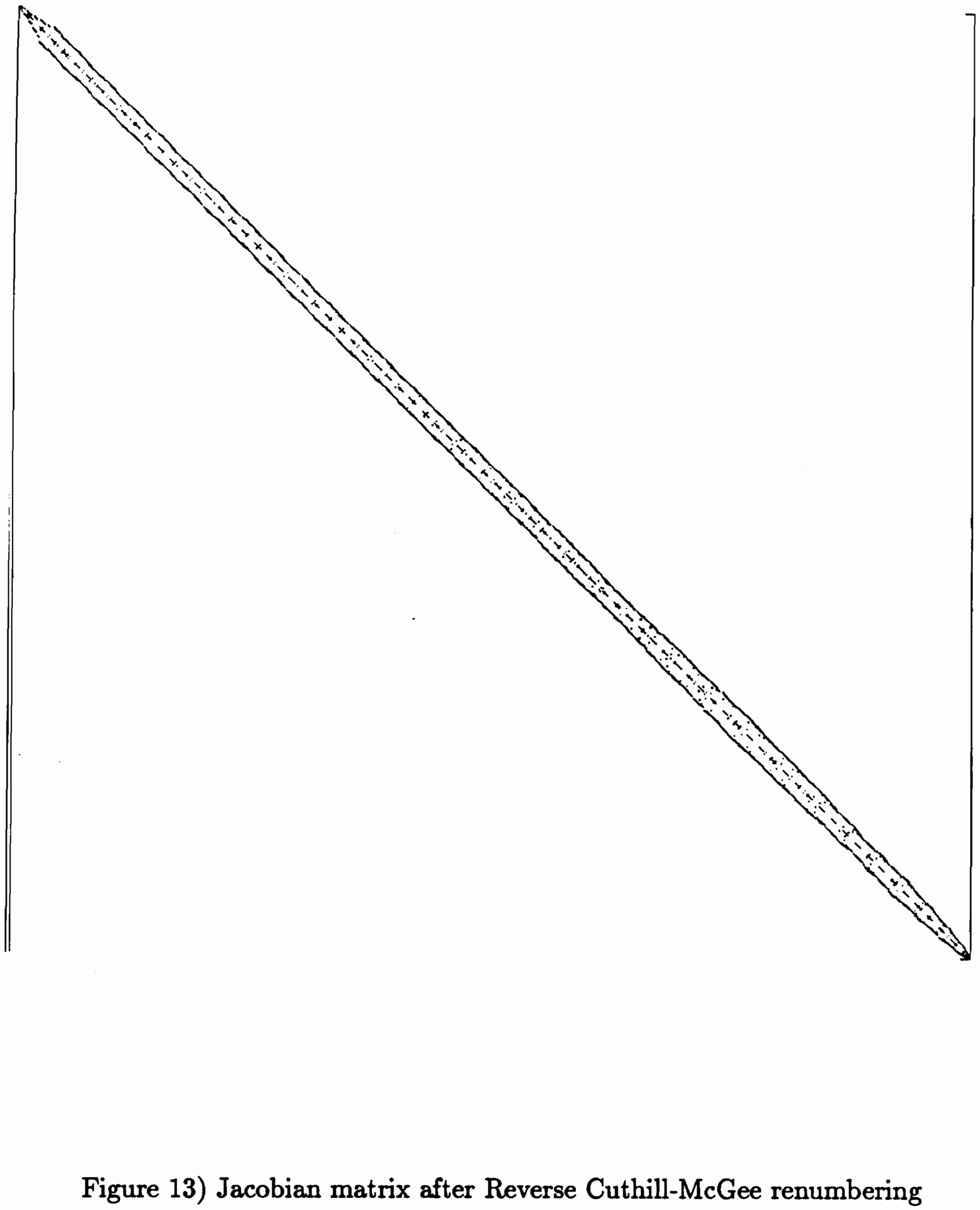


Figure 13) Jacobian matrix after Reverse Cuthill-McGee renumbering scheme for LU decomposition algorithm

Because the matrix A has a structure that is physically symmetric in that it has the same number of columns above the diagonal as it has rows to the left of the diagonal, only one parameter needs to be calculated from the cell connectivity information. This parameter contains the location of the diagonal term for each row of the matrix. From this parameter, the structure of the matrix is implied, and the LU decomposition and backward and forward substitutions can be performed. The calculation of this parameter is cheap and needs to be done only once for each grid since the structure of the matrix remains unchanged from iteration to iteration. Other terms used in the decomposition and solution process related to the structure of the matrix which are calculated from this parameter need to be calculated every iteration and result in a small penalty when compared to a structured banded solver. These terms could be calculated only once, however, the resulting storage penalty would be on the order of five times the storage of the LU matrix which is far too great to consider.

4.6 Hybrid Runge-Kutta / LU Decomposition Time Integration

The hybrid scheme is simply a combination of the Runge-Kutta and LU decomposition with a reuse of the Jacobian. The scheme starts with a Runge-Kutta iteration which is performed until a specified tolerance of the residual is met. This tolerance is generally chosen such that the solution has progressed through much of the transient so that the Jacobian used by the LU decomposition method will be relatively invariant from iteration to iteration. The LU decomposition is then performed only once, and the decomposition elements are reused for the remaining iterations.

This algorithm was the one chosen to compare convergence times of the remeshing strategy versus a solution on a single mesh. It was chosen because it was shown to be one of the better strategies overall, and because it is particularly useful for the remeshing strategy. The Runge-Kutta method is very efficient in damping out the high frequency errors which are introduced by the interpolation of the flow variables onto a newly generated mesh. After the errors are damped out, the solution is sufficiently close to the converged solution on that mesh such that only one LU decomposition needs to be performed. The solution can then be converged using only the backward and forward substitution step which is the inexpensive part of the exact solution procedure.

4.7 Sparse Matrix Solvers

4.7.1 GMRES Acceleration

The Generalized Minimum Residual (GMRES) algorithm was developed by Saad and Schultz[13] as a generalization of the preconditioned conjugate gradient algorithm for non-symmetric matrices. Both GMRES and the conjugate gradient method can be classified as Krylov subspace methods. GMRES converges the linear problem by eliminating the error associated with all non-unity eigenvalues present in matrix A given in the equation

$$A\Delta Q - R^n = 0 \tag{4.7.1}$$

In the worst case, none of the eigenvalues of the linear system have a value of one, and the error associated with all eigenvalues must be eliminated. Eigenvalues that have a value of one do not have any error associated with the solution of the problem and do not have to be eliminated by the sparse matrix solver. The eigenvalues of the linear system can be clustered about one by premultiplying the linear system by a matrix preconditioner. Preconditioning was performed in this work by using the Point-Jacobi algorithm in the form

$$D^{-1}(A\Delta Q - R^n) = F(\Delta Q) = 0 \tag{4.7.2}$$

The purpose of the GMRES algorithm can now be thought of as an iteration process to obtain a better guess to the update vector ΔQ. In this sense, the GMRES algorithm accelerates the convergence of the point-Jacobi algorithm. For each point-Jacobi time step, the GMRES solver is performed M times where m denotes the inner iteration number such that $m = 0, 1, ..., M$. The initial guess for the GMRES algorithm is

$$\Delta Q^0 = \Delta Q^n \tag{4.7.3}$$

The directional derivative of a function F at ΔQ in the direction p can be represented by

$$\frac{\partial F(\Delta Q)}{\partial p} = \lim_{\epsilon \to 0} \frac{F(\Delta Q + \epsilon p) - F(\Delta Q)}{\epsilon} \tag{4.7.4}$$

where ϵ is chosen to be a sufficiently small number. In order to advance the solution using GMRES on a given inner iteration m, it is necessary to construct k orthonormal search directions $p_1, p_2, ..., p_k$ where k is the dimension of the Krylov subspace. The method for finding the search directions was given by Wigton, et al[28]. The first search direction can be found by

$$p_1 = F(\Delta Q^m) \tag{4.7.5}$$

Normalize p_1

$$p_1 = \frac{p_1}{||p_1||} \tag{4.7.6}$$

The remaining $k - 1$ search directions can be obtained from

$$p_{j+1} = \frac{\partial F(\Delta Q^m)}{\partial p_j} - \sum_{i=1}^{j} b_{ij} p_i \tag{4.7.7}$$

where

$$b_{ij} = (\frac{\partial F(\Delta Q^m)}{\partial p_j}, p_i) \tag{4.7.8}$$

so that

$$(p_{j+1}, p_i) = 0 \quad for \;\; i = 1, 2, ..., j \tag{4.7.9}$$

Normalize p_{j+1}

$$p_{j+1} = \frac{p_{j+1}}{||p_{j+1}||} \tag{4.7.10}$$

A new approximation for the ΔQ vector can be found from

$$\Delta Q^{m+1} = \Delta Q^m + \sum_{j=1}^{k} a_j p_j \tag{4.7.11}$$

where the unknown coefficients a_j are chosen to minimize the function

$$||F(\Delta Q^{m+1})||^2 = ||F(\Delta Q^m + \sum_{j=1}^{k} a_j p_j)||^2 \tag{4.7.12}$$

$$\simeq ||F(\Delta Q^m) + \sum_{j=1}^{k} a_j \frac{\partial F(\Delta Q^m)}{\partial p_j}||^2 \tag{4.7.13}$$

Now solve this least squares problem for a_j by the QR algorithm available in LINPACK[29]. Once the values for a_j have been obtained, then the values for ΔQ^{m+1} can be calculated from equation 4.7.11. The GMRES solver can then be restarted using the new values for ΔQ.

For a problem with N equations and a Krylov subspace dimension of k, the required storage for the GMRES solver is (k+4)*N[28]. The work of the GMRES solver increases as the square of the subspace dimension, but a larger Krylov subspace dimension will tend to give faster convergence when it is necessary to converge the linear system several orders of magnitude.

In this work, a value of five was used as the Krylov subspace dimension. Twelve restart cycles were used for the GMRES solver before an update of vector Q. The linear system was considered converged when GMRES reduced the residual of the linear system by a factor of 0.05. The parameters used in the present work may not be optimal for all types of problems.

4.7.2 CHEBYCODE Acceleration

CHEBYCODE[14,21,30] was the second sparse matrix solver that was examined in the cell-vertex code. CHEBYCODE is an implementation of the classical Chebyshev algorithm for a system of equations. CHEBYCODE is limited to solving linear systems whose eigenvalues have only positive real parts. Since the linear system is the Jacobian of an upwind scheme, the matrix is positive definite. Hence, the matrix has only eigenvalues with positive real parts. As with GMRES, preconditioning was performed using the Point-Jacobi algorithm in the form

$$D^{-1}(A\Delta Q - R^n) = F(\Delta Q) = 0 \tag{4.7.14}$$

The initial guess for the Chebyshev iteration is

$$\Delta Q^0 = \Delta Q^n \tag{4.7.15}$$

while the initial value used for the update of the ΔQ vector is

$$p^0 = \frac{1}{d} F(\Delta Q^0) \tag{4.7.16}$$

A given number, M, of Chebyshev iterations are performed to find a better approximation to the vector ΔQ. For $m = 0, 1, ..., M$ the Chebyshev iteration given by Elman,et al[31] is

$$\begin{aligned} \Delta Q^{m+1} &= \Delta Q^m + p^m \\ \alpha^{m+1} &= \begin{cases} \frac{2d}{(2d^2-c^2)}, & m = 0 \\ \frac{1}{d-(c/2)^2\alpha^m} & m \geq 1 \end{cases} \\ \beta^{m+1} &= d\alpha^{m+1} - 1 \\ p^{m+1} &= \alpha^{m+1} F(\Delta Q^{m+1}) + \beta^{m+1} p^m \end{aligned} \tag{4.7.17}$$

where d and c are the Chebyshev iteration parameters that describe the center of the convex hull (which encloses the eigenvalue spectrum) and a convex hull diameter, respectively. The values of these two parameters are input initially, but in order to accelerate convergence they are updated during the iteration process. In order to calculate new values for d and c, the maximum and minimum absolute values of the eigenvalues must be known. CHEBYCODE approximates the maximum and minimum absolute values of the eigenvalues by a modified power method[32].

In the present work, the maximum number of iterations, M, used by CHEBYCODE to solve the linear problem was set to 60. The linear problem was considered converged when the residual of the linear problem was reduced by a factor of 0.05. At intervals of 20 CHEBYCODE iterations, the matrix solver would attempt to find new approximations to the maximum and minimum absolute values of the eigenvalues, and hence the values of the iteration parameters d and c. The user manual[14] recommended an average of the eigenvalues as the initial location of the center of the convex hull (which encloses the eigenvalue spectrum) and a convex hull diameter of zero. Therefore, the initial convex hull was set to a value of one (real value) with a diameter of zero.

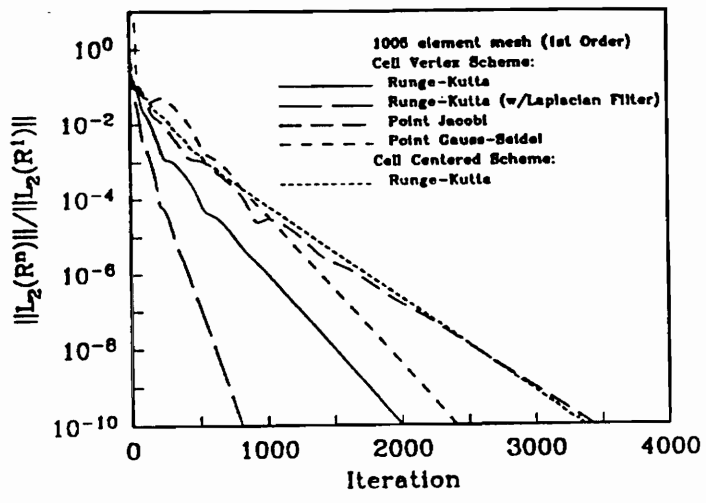
V. Results for Two-Dimensional Unstructured Solvers

5.1 Time Integration Scheme Comparison

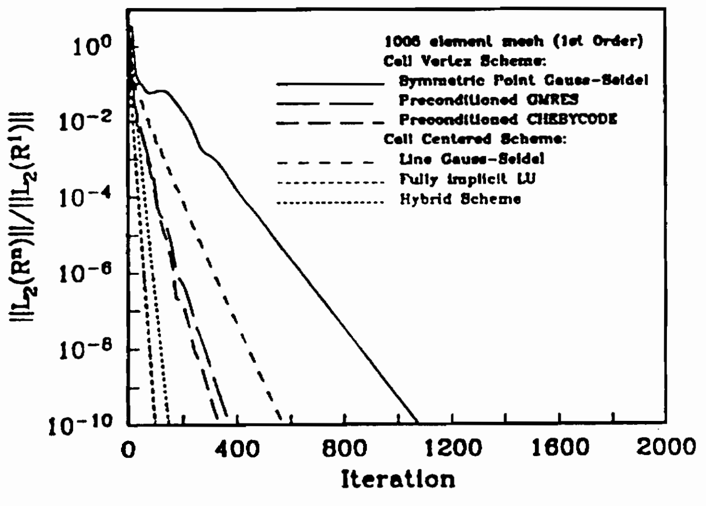
Convergence rates for Euler flow over a transonic ($M_\infty = 0.85$) circular arc in a channel are presented for each of the time integration methods given in chapter IV. The rates are compared on 1005, 1999, and 4008 element meshs for both first-order and higher-order accurate cell-centered and cell-vertex schemes. Convergence rates are plotted versus iteration and computational time. The quantity that is plotted is the L_2 norm of the residual vector. Typically, both schemes (cell-centered and cell-vertex) reused the Jacobian after the solution was converged one order of magnitude. This reuse of the Jacobian was done to minimize computational work. Several unshown numerical experiments verified that there is little or no difference in the convergence rate when the Jacobian is reused after the basic flow physics are established. The higher-order solutions were generated using the MUSCL limiter for the cell-vertex scheme and Barth's limiter for the cell-centered scheme.

5.1.1 Coarse 1005 Element Mesh

Figures 14 and 15 are the convergence histories with respect to iteration and time for the first-order calculation of a circular arc in a channel on a 1005 element mesh. All schemes converged the first-order, 1005 element test case. The best schemes in terms of iteration count and CPU time were the fully implicit methods. The fully implicit LU method (cell-centered) had the lowest iteration count. The hybrid (cell-centered) scheme was the fastest in terms of CPU time, closely followed by CHEBYCODE and GMRES (cell-vertex) methods.

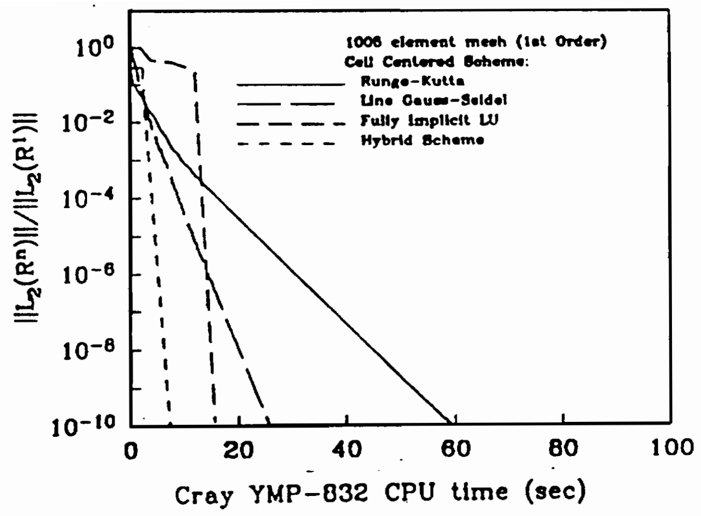


(a)

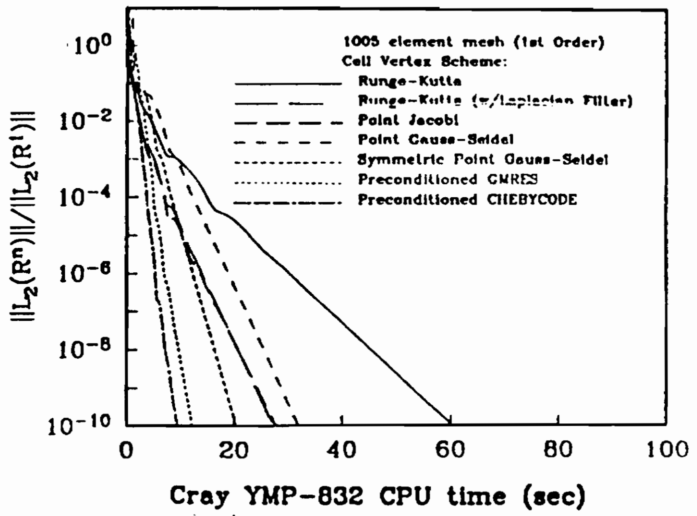


(b)

Figure 14) Residual vs. iteration # for 1005 element mesh first-order transonic circular arc in a channel



(a)

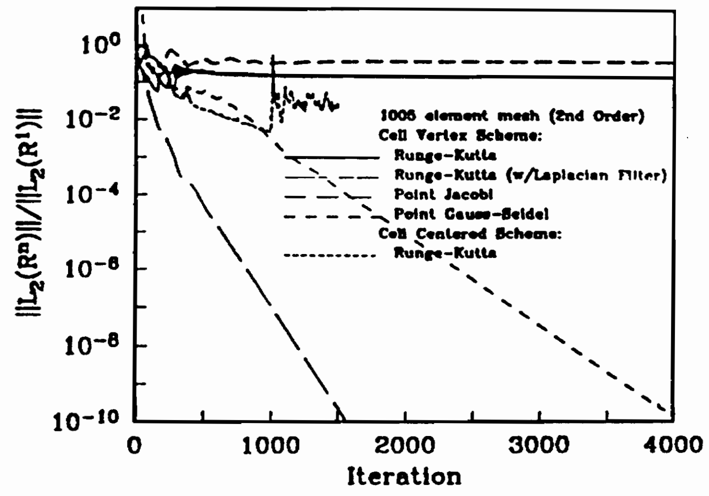


(b)

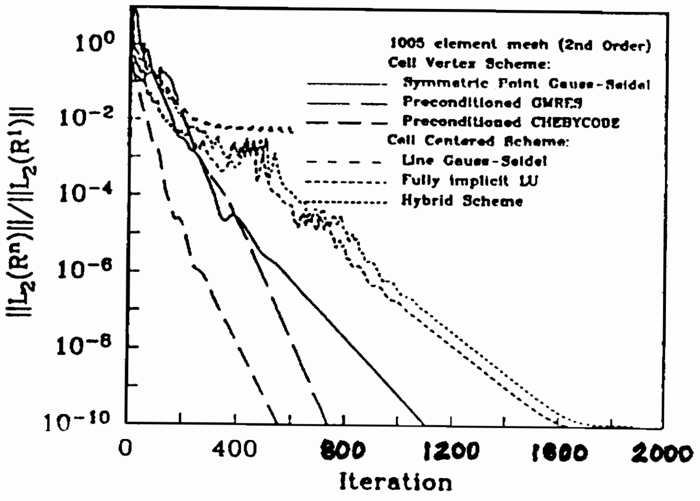
Figure 15) (a) residual vs. Cray-YMP CPU time for cell-centered,
(b) residual vs. Cray-YMP CPU time for cell-vertex,
for 1005 element mesh first-order transonic circular arc in a channel

The fully implicit LU method converged in the fewest number of iterations due to its implicitness and exact solution of the linear system. However, finding the LU of a large matrix requires substantial CPU time. After the Jacobian was frozen (reused), the only operations necessary to solve the system were forward and backward substitutions. The hybrid scheme avoids most of the factorization CPU time by using Runge-Kutta iteration to converge the residual one order of magnitude, then performing only one LU factorization. The rest of the iterations are performed using a frozen Jacobian with backward-forward substitution. These characteristics are clearly evident in the CPU time comparison. Interestingly enough, the iteration count for Runge-Kutta with a cell-centered scheme was twice that of Runge-Kutta for the cell-vertex scheme, yet the CPU times were almost identical. Apparently, the loss in convergence rate is balanced by the cost per iteration. The Gauss-Seidel (cell-vertex) method and the line Gauss-Seidel type relaxation (cell-centered) converged in comparable amounts of CPU time. The symmetric Gauss-Seidel (cell-vertex) was superior to both and was the only cell-vertex method to approach the sparse matrix solvers in speed. The point-Jacobi (cell-vertex) and Runge-Kutta with Laplacian smoothing (cell-vertex) methods converged in comparable amounts of time.

The next series of plots, Figs. 16 and 17, are for the higher-order accurate, 1005 element circular arc in a channel. All the higher-order cases were run with limiting turned on. The fully explicit schemes were not successful in converging the higher-order problem. The only explicit scheme which converged the higher-order problem was Runge-Kutta with Laplacian filtering. The Laplacian filtering of the residual permitted the explicit method to converge this problem.

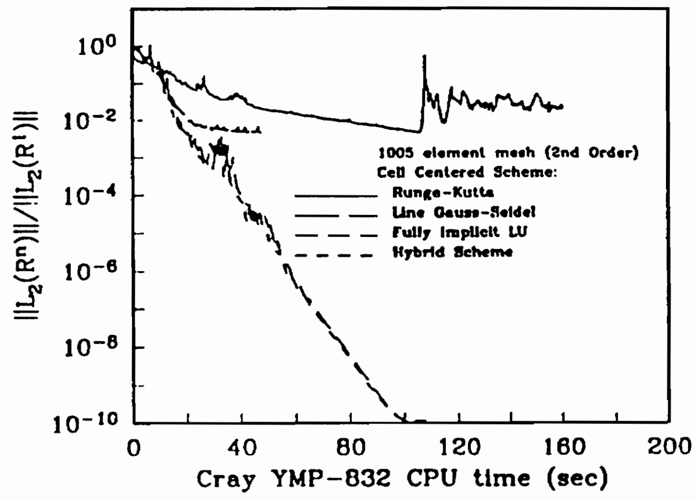


(a)

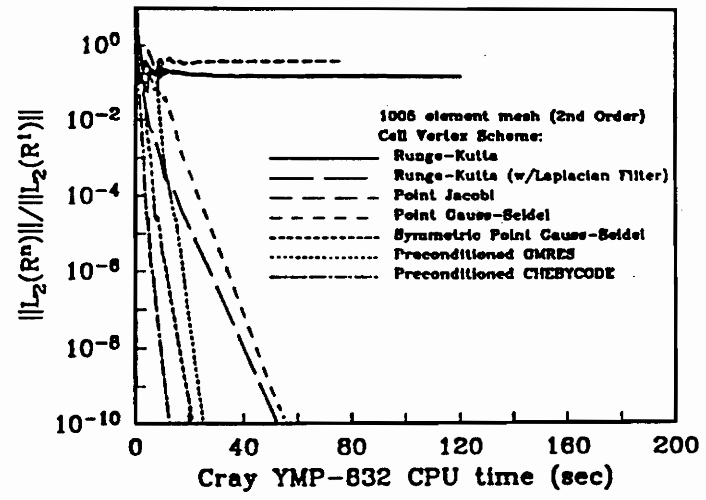


(b)

Figure 16) Residual vs. iteration # for 1005 element mesh higher-order transonic circular arc in a channel



(a)



(b)

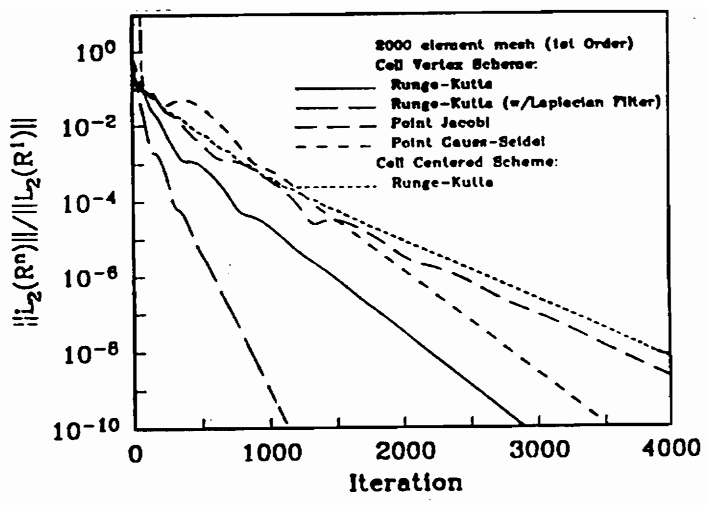
Figure 17) (a) residual vs. Cray-YMP CPU time for cell-centered,
(b) residual vs. Cray-YMP CPU time for cell-vertex,
for 1005 element mesh higher-order transonic circular arc in a channel

The point-Jacobi iteration and line Gauss-Seidel type relaxation were both implicit schemes which were not successful in converging the higher-order problem. CHEBYCODE was clearly the best in terms of both iteration count and CPU time. GMRES was second but convergence relative to CHEBYCODE was degraded as compared to the first-order solution. Symmetric point Gauss-Seidel was comparable in CPU time to GMRES. Both the hybrid and fully implicit LU method were comparable to each other in both CPU time and iteration count. Some test runs were made for the cell-vertex method with the limiter turned off. With the limiter off, all time integration algorithms were able to converge the higher-order problem. Thus, the limiter introduces a non-linearity into the iteration procedure that slows or stops convergence.

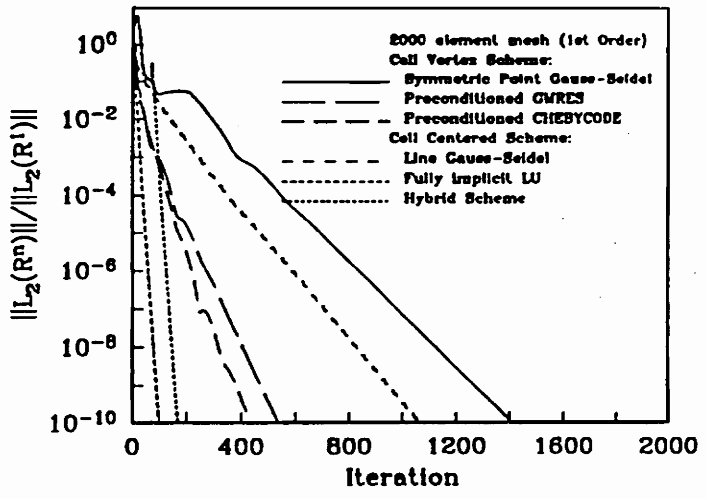
5.1.2 Medium 1999 Element Mesh

In Figs. 18 and 19 the convergence rates of the time integration algorithms on a 1999 element mesh are compared. The iteration plots are almost identical in form to the first-order, 1005 element case with some increase in iteration count for all the methods. The number of iterations for the fully implicit LU method remains almost constant. The more strongly implicit a method was the smaller the percentage increase in computational time.

The higher-order, 1999 element mesh convergence plots are detailed in Figs. 19 and 20. Only three schemes, symmetric Gauss-Seidel, GMRES, and CHEBYCODE, would converge the higher-order solution on the 1999 element mesh. GMRES converged the solution in fewer iterations than CHEBYCODE, but CHEBYCODE still used less CPU time than GMRES.

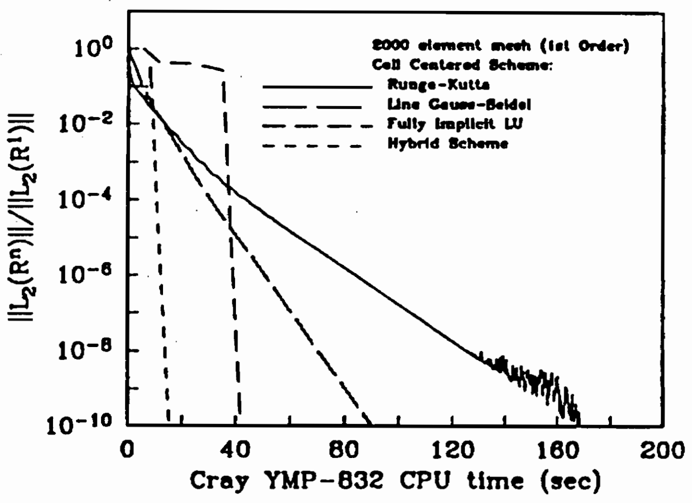


(a)

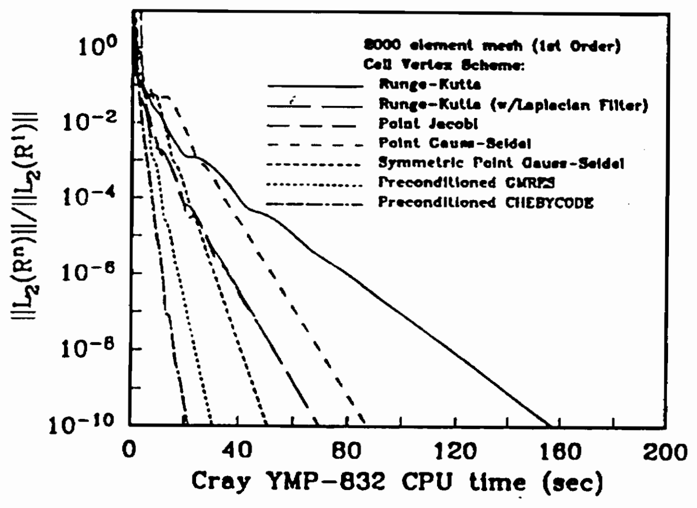


(b)

Figure 18) Residual vs. iteration # for 1999 element mesh first-order transonic circular arc in a channel

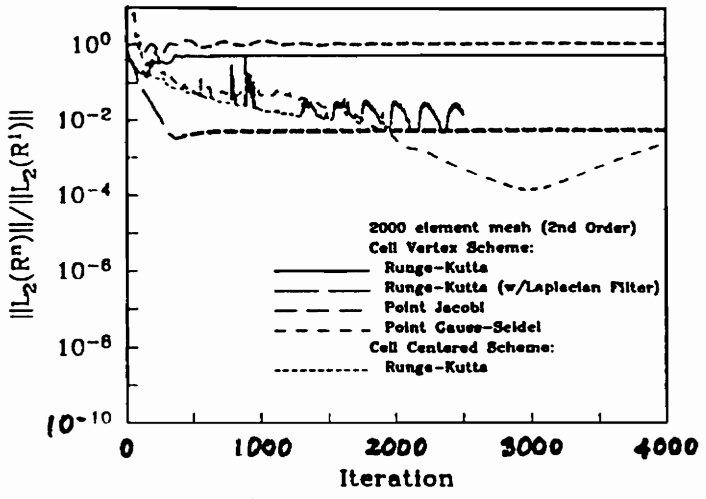


(a)

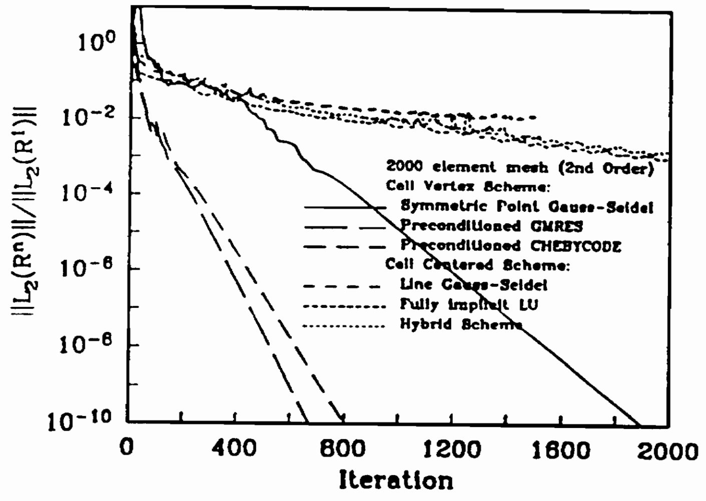


(b)

Figure 19) (a) residual vs. Cray-YMP CPU time for cell-centered,
(b) residual vs. Cray-YMP CPU time for cell-vertex,
for 1999 element mesh first-order transonic circular arc in a channel

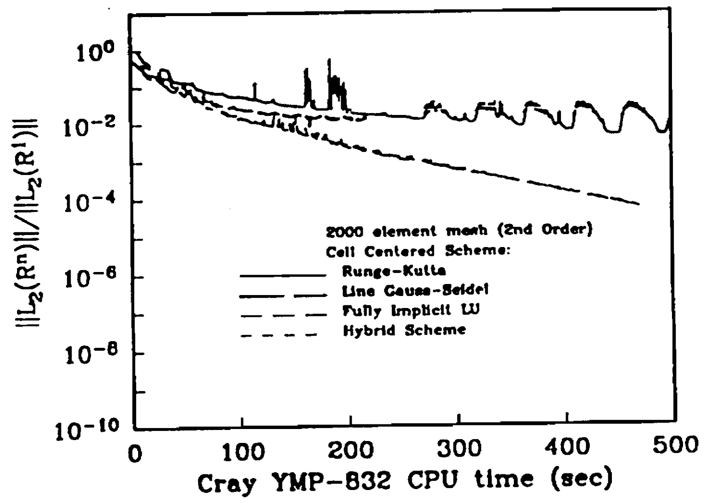


(a)

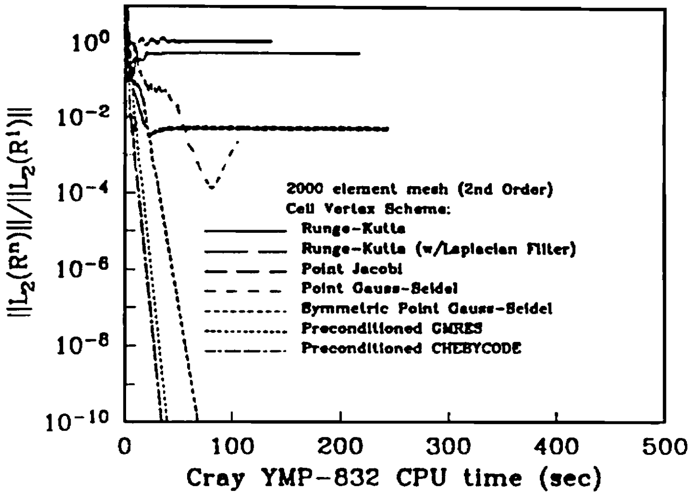


(b)

Figure 20) Residual vs. iteration # for 1999 element mesh higher-order transonic circular arc in a channel



(a)



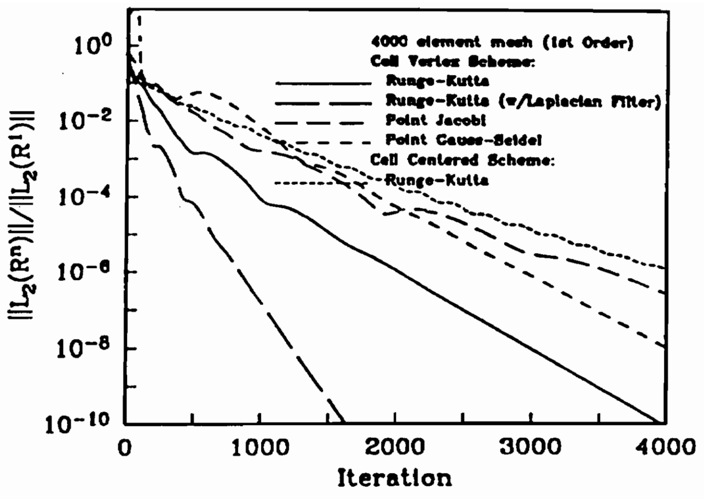
(b)

Figure 21) (a) residual vs. Cray-YMP CPU time for cell-centered,
(b) residual vs. Cray-YMP CPU time for cell-vertex,
for 1999 element mesh higher-order transonic circular arc in a channel

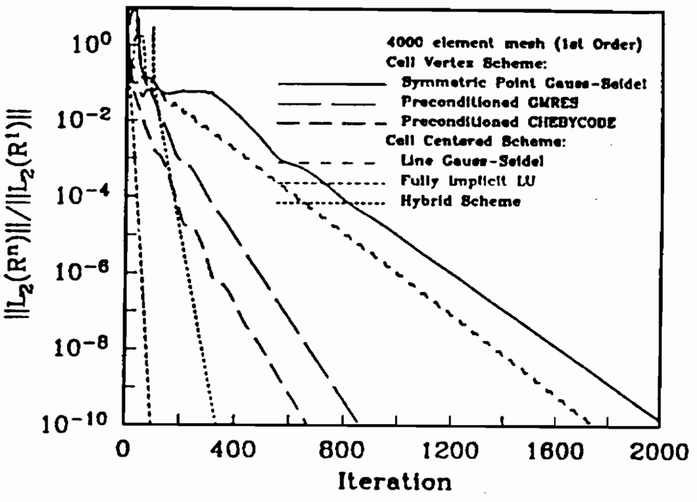
5.1.3 Fine 4008 Element Mesh

Convergence histories are presented in Figs. 22 and 23 for the first-order circular arc in a channel solution on a fine 4008 element mesh. Iteration histories are of the same form as previous first-order solutions. Total iteration count has increased for all methods except the fully implicit LU method. CPU times all increased. The sparse matrix solvers CHEBYCODE and GMRES increase in CPU time was less than the increases seen by the hybrid and fully implicit LU methods. This behavior will become more evident in the spectral radius and time convergence rate plots discussed in the next section.

For the higher-order, 4008 element test case, appearing in Figs. 24 and 25, the Runge-Kutta with Laplacian filtering, symmetric Gauss-Seidel, GMRES, and CHEBYCODE methods were the only ones to converge. The CPU times went up for all methods but again the percentage increase was smallest for GMRES and CHEBYCODE. CHEBYCODE used the least CPU time to converge.

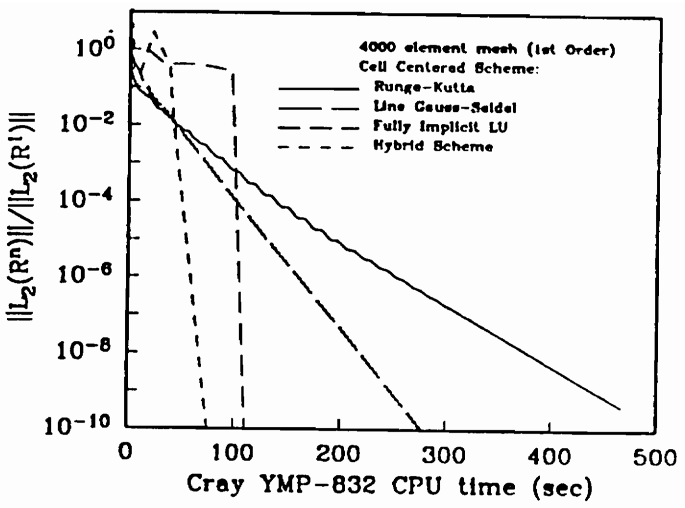


(a)

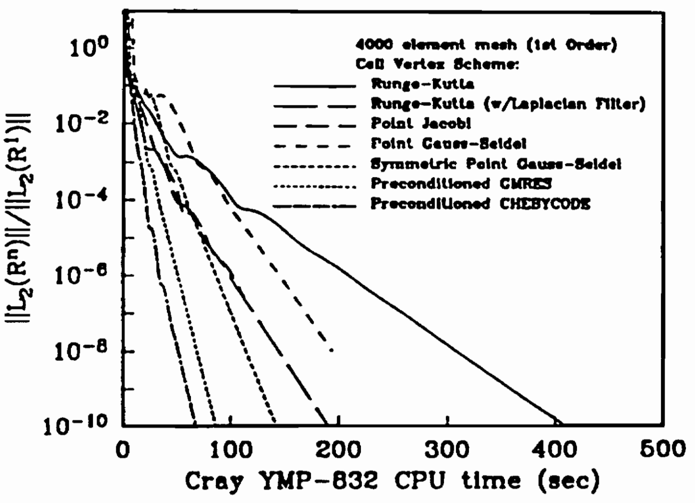


(b)

Figure 22) Residual vs. iteration # for 4008 element mesh first-order transonic circular arc in a channel

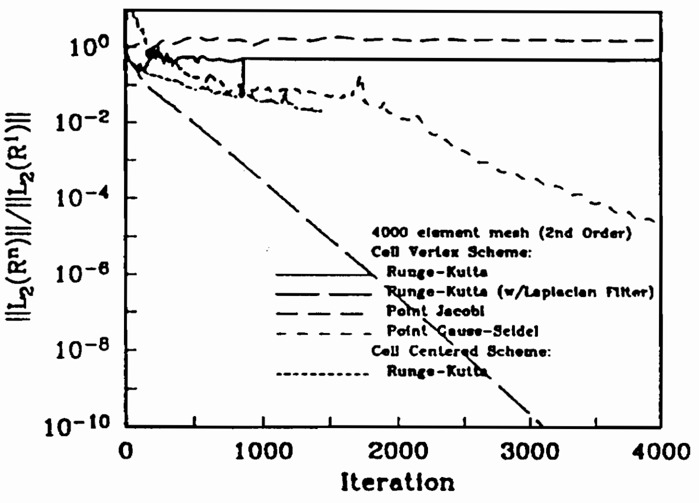


(a)

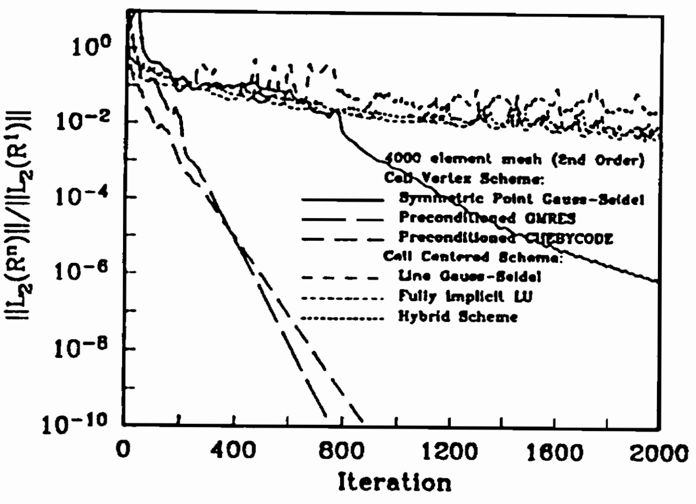


(b)

Figure 23) (a) residual vs. Cray-YMP CPU time for cell-centered,
(b) residual vs. Cray-YMP CPU time for cell-vertex,
for 4008 element mesh first-order transonic circular arc in a channel

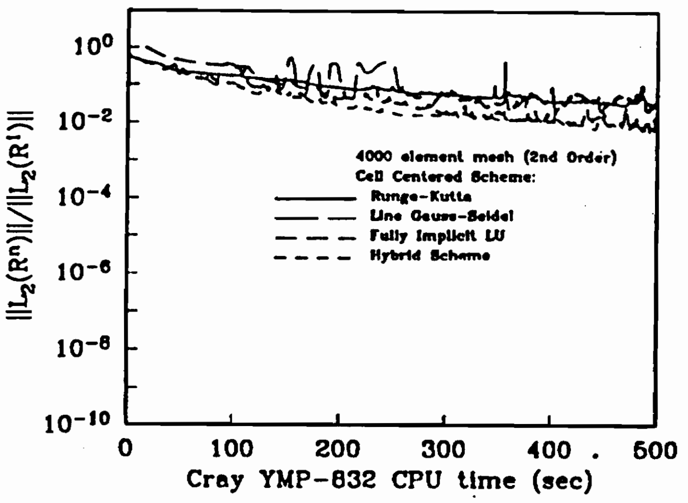


(a)

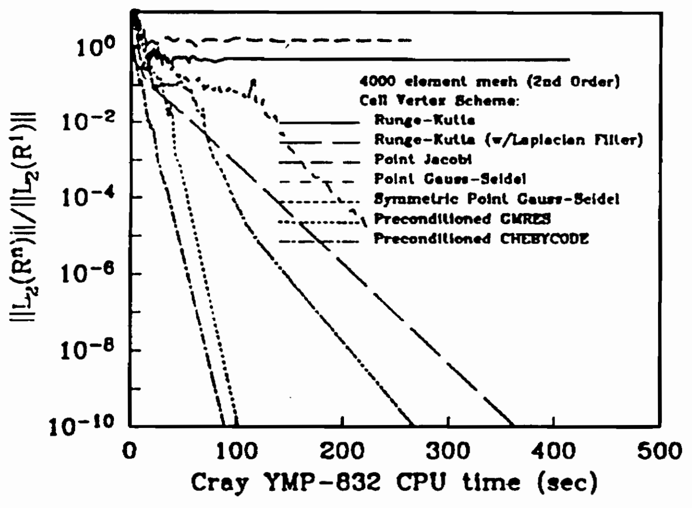


(b)

Figure 24) Residual vs. iteration # for 4008 element mesh higher-order transonic circular arc in a channel



(a)



(b)

Figure 25) (a) residual vs. Cray-YMP CPU time for cell-centered,
(b) residual vs. Cray-YMP CPU time for cell-vertex,
for 4008 element mesh higher-order transonic circular arc in a channel

5.1.4 Spectral Radius and Time Convergence Rate

Figures 26 and 27 detail the spectral radius and time convergence rate of each method with respect to the grid spacing, Δ. The spectral radius, λ, was computed as

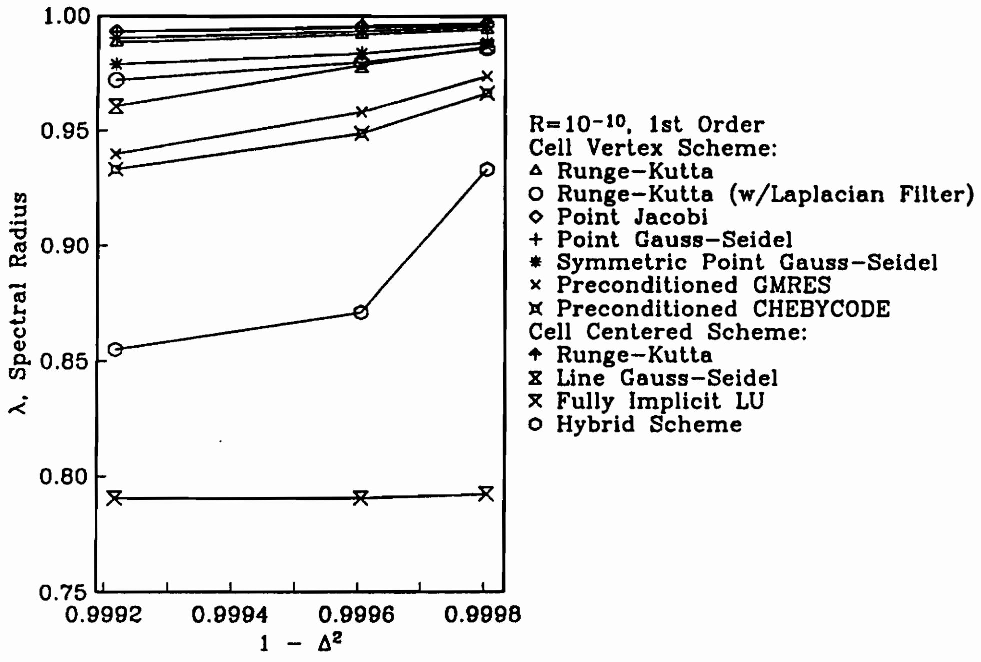
$$\lambda = \left[\frac{\|L_2(R^n)\|}{\|L_2(R^1)\|}\right]^{1/N} \tag{5.1.1}$$

where $L_2(R)$ is the Euclidean norm of the residual and N is the number of unknowns. The time convergence rate, λ_t, was defined as

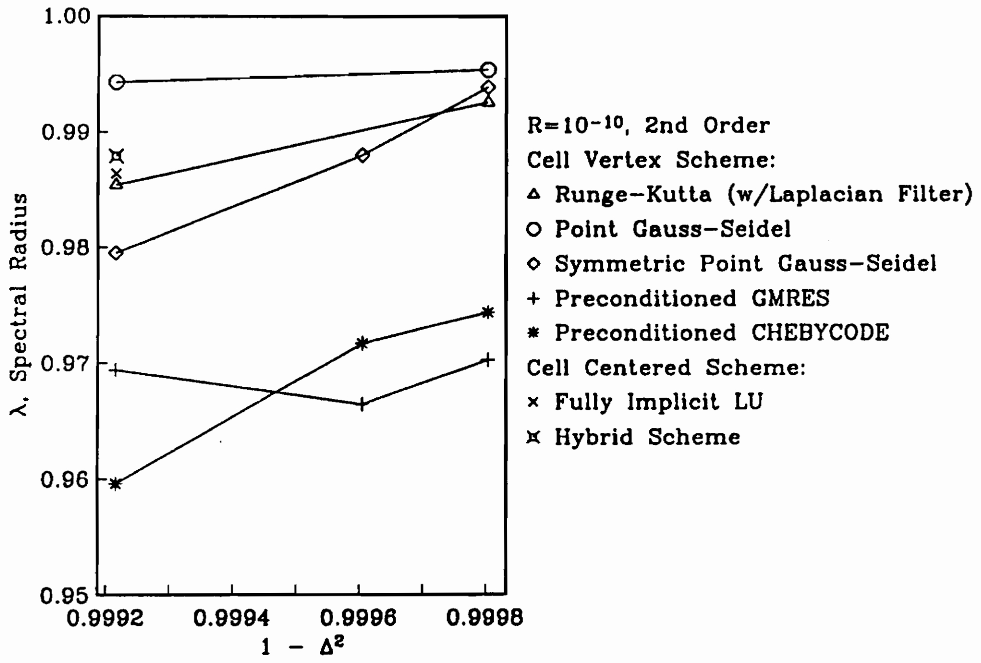
$$\lambda_t = \left[\frac{\|L_2(R^n)\|}{\|L_2(R^1)\|}\right]^{1/t} \tag{5.1.2}$$

where t is the CPU time when iteration n occurred. The time convergence rate could be viewed as the factor by which the L_2 norm of the residual is reduced per one second of CPU time. These plots are extremely important when considering the application of these time integration schemes to three-dimensional problems. The spectral radius and time convergence rate plots give an indication of the possible behavior of these algorithms for larger problems.

Looking at the first-order results, the hybrid scheme degrades rapidly as the mesh increases from 1999 to 4008 elements. However, the fully implicit LU and explicit Runge-Kutta remain along the same basic paths. Therefore, the poor performance of the hybrid method for larger grid sizes can be attributed to either a decrease in the compatibility of the two schemes or to an increase in the percentage of time spent performing Runge-Kutta iterations as the number of elements in the mesh increases. The remaining algorithms follow nearly linear paths with GMRES and CHEBYCODE having the most attractive behavior.

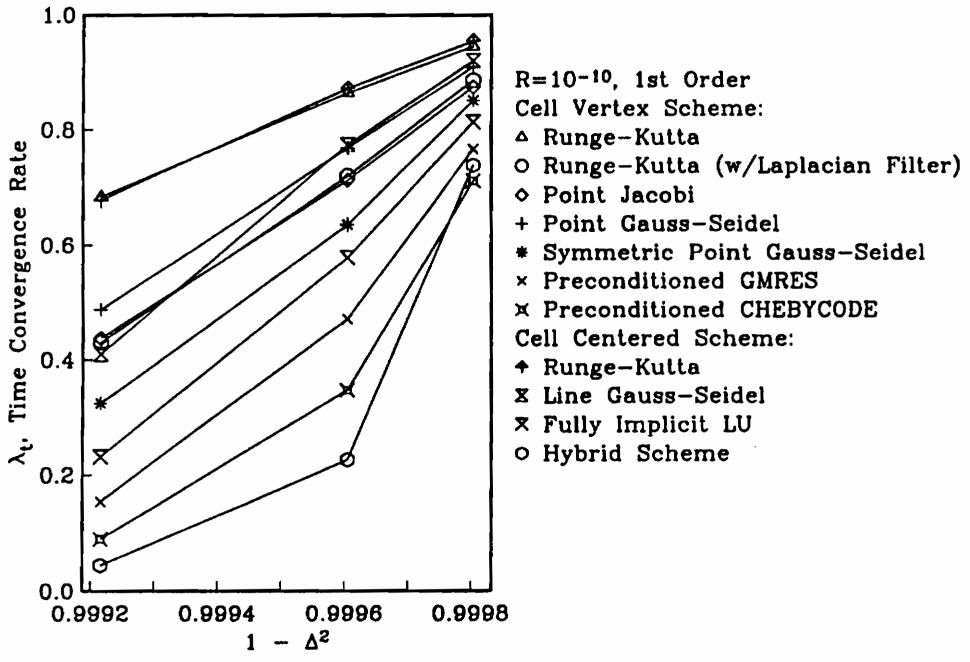


(a)

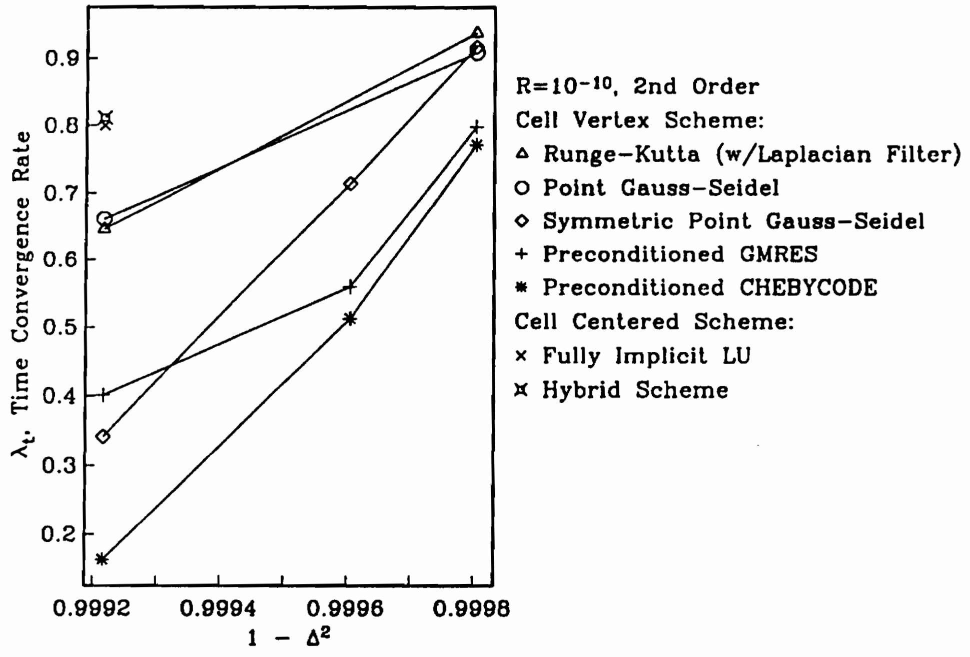


(b)

Figure 26) Spectral radius comparison for transonic circular arc in a channel, $M_\infty = 0.85$



(a)



(b)

Figure 27) (a) Time convergence rate comparison for transonic circular arc in a channel, $M_\infty = 0.85$

The higher-order results contain mainly those algorithms from the cell-vertex code because the limiter in the cell-centered code prevented convergence. The symmetric Gauss-Seidel method degrades quickly with steadily increasing mesh fineness for a higher-order solution. The sparse matrix solvers also degrade with increasing mesh sizes but the asymptotic limit may still be better than the explicit schemes.

5.1.5 Summary of Overall Performance

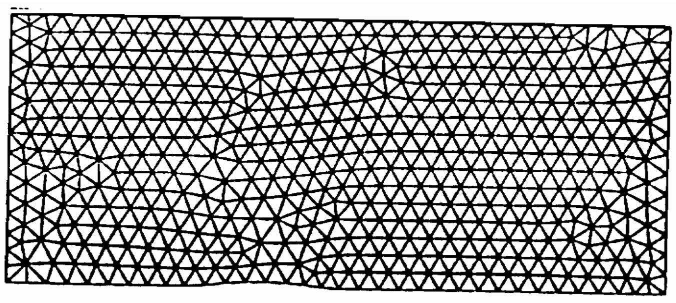
In general, for first-order problems, the fully implicit LU method was fastest in terms of iteration count while the hybrid scheme was best in terms of CPU usage. However, as the mesh size increased, the sparse matrix solvers GMRES and CHEBYCODE appeared more attractive than the hybrid scheme. Also for larger problems such as three-dimensional applications, GMRES and CHEBYCODE require far less storage than the LU and hybid methods. Another attractive scheme for three-dimensional application is the Runge-Kutta with residual smoothing scheme. This scheme requires very little storage, is simple to code, and has reasonable convergence behavior. For higher-order problems, GMRES used the fewest iterations while CHEBYCODE was the fastest. The significant difference between the first- and higher-order results is believed to be mainly due to the different methods used for the higher-order calculations. The line Gauss-Seidel type relaxation, LU and hybrid schemes were computed using Barth's method while Gauss-Seidel, GMRES and CHEBYCODE were converged utilizing MUSCL differencing. The difficulty in converging the higher-order problems was thought to be the highly nonlinear behavior of Barth's limiter and the inconsistency of a first-order left hand side and a higher-order right hand side.

Another concern arose when comparing the line Gauss-Seidel type scheme to the Gauss-Seidel scheme. One would expect the line Gauss-Seidel type method to converge faster than the Gauss-Seidel method as it does for structured mesh calculations because it computes more terms implicitly and has less terms to relax. However, several problems exist with the currently developed line Gauss-Seidel type scheme that could be fixed to make it better. First, the terms inside each submatrix which are not part of the block tridiagonal form are completely neglected. Even if only a few terms per submatrix are neglected, the effect on convergence could be very significant because the solution to each submatrix affects the solution of the remaining submatrices. Second, because the line Gauss-Seidel type scheme utilizes a non-linear update of the residual to relax the terms outside the submatrices, it was not possible to make it symmetrical as was done for the Gauss-Seidel scheme. A possible solution to both of these problems would be to treat the terms that are not in block tridiagonal form exactly as the off-diagonal terms of the Gauss-Seidel method.

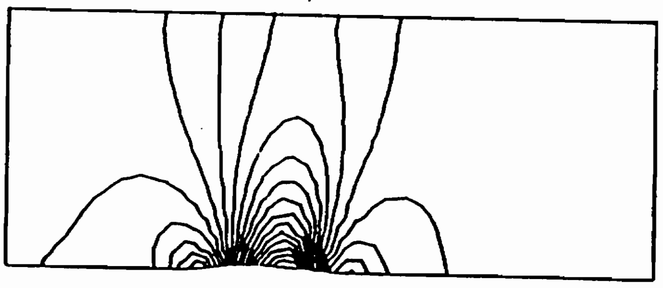
5.1.6 Pressure Plot Comparisons

Figure 28 presents the mesh and pressure contour solutions for the 1005 element test case. Figure 29 contains the corresponding plots for the 4008 element test case. First and higher-order pressure contours are shown for the cell-centered discretization. In this problem, the flow first compresses near the leading edge of the circular arc. It then expands over top until it becomes supersonic near the leading edge causing a normal shock to form. Finally, the flow expands over the trailing edge of the circular arc. The contours in the solutions appear to be smooth except near the normal shock. As expected, the higher-order solutions capture the shock better than the first order solutions. In Fig. 30, a plot of the pressure coefficient c_p on the lower surface is compared between all the first and higher-order solutions for the 1005 and 4008 element test cases and a shock-fitted solution from a GAMM workshop[33].

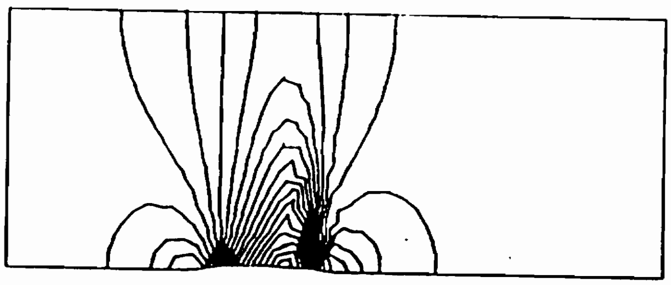
The present results compare quite favorably to the shock-fitted solution from the GAMM workshop. It is easy to see that the finest mesh (4008 elements) higher-order solution has the best comparison.



(a)

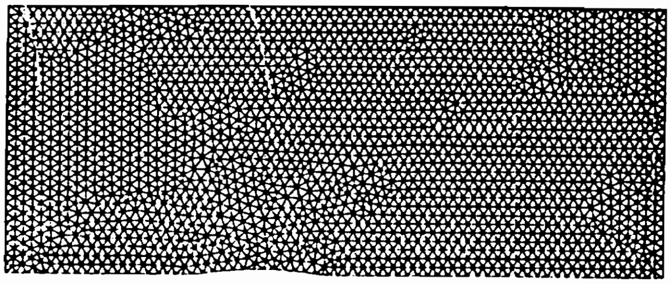


(b)

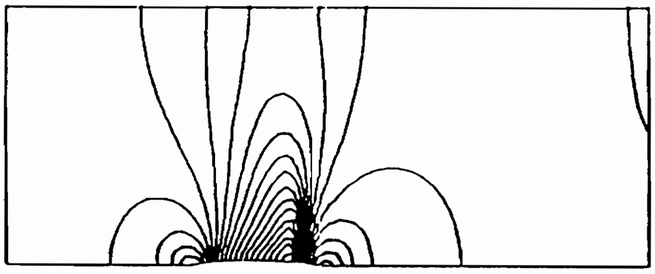


(c)

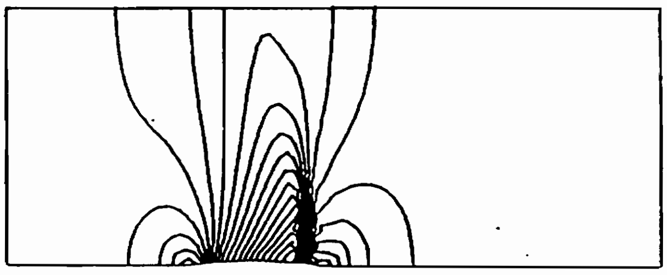
Figure 28) (a) Mesh: 1005 elements, 551 vertices,
(b) first-order cell-centered pressure contours,
(c) higher-order cell-centered pressure contours
for transonic circular arc in a channel, $M_\infty = 0.85$



(a)



(b)



(c)

Figure 29) (a) Mesh: 4008 elements, 2098 vertices,
(b) first-order cell-centered pressure contours,
(c) higher-order cell-centered pressure contours
for transonic circular arc in a channel, $M_\infty = 0.85$

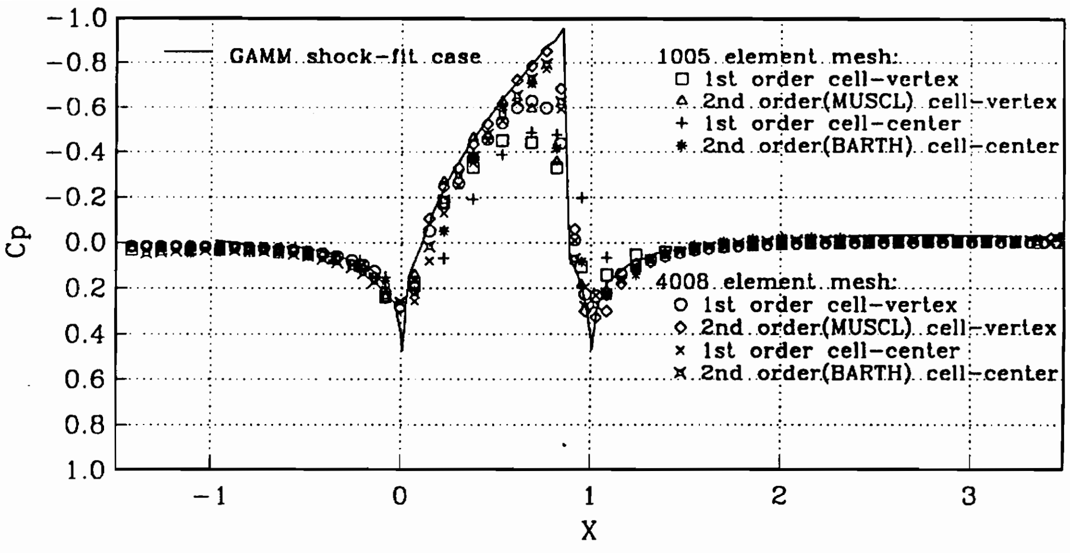


Fig. 30) Comparison of unstructured results with GAMM workshop[33] results on structured mesh, 72x21 grid for coefficient of pressure on lower surface of transonic circular arc in a channel, $M_\infty = 0.85$

5.2 Remeshing Strategy Comparison

Two test cases are presented in this section. The first is the transonic, $M_\infty = 0.85$, circular arc airfoil in a channel which was used for the time integration scheme comparison. The second is a $M_\infty = 3.0$ internal flow with area change. For both cases the solutions and convergence rates of a remeshing strategy as described in section 2.2 are compared with those of an unstructured uniform mesh. The uniform mesh was chosen to have approximately the same number of elements as the final adapted mesh.

The purpose of this comparison is to illustrate that the solution from the remeshing sequence (which includes the time to generate each new mesh and interpolate the variables) takes little or no more time to compute than the solution on the uniform mesh. However, the solution obtained from the remeshing strategy is much more accurate.

All of the solutions shown were computed using the hybrid time integration scheme which performs Runge-Kutta time integration until the normalized residual reaches a value of 0.3. The scheme then computes one LU decomposition, freezes the Jacobian matrix and converges the solution with forward/backward substitutions only. This method was shown previously to be one of the more efficient time integration schemes, and it is particularly useful for the remeshing strategy because the Runge-Kutta method is very efficient in damping out the high frequency errors which are introduced by the interpolation of the flow variables onto a newly generated mesh.

5.2.1 Transonic Circular Arc in a Channel

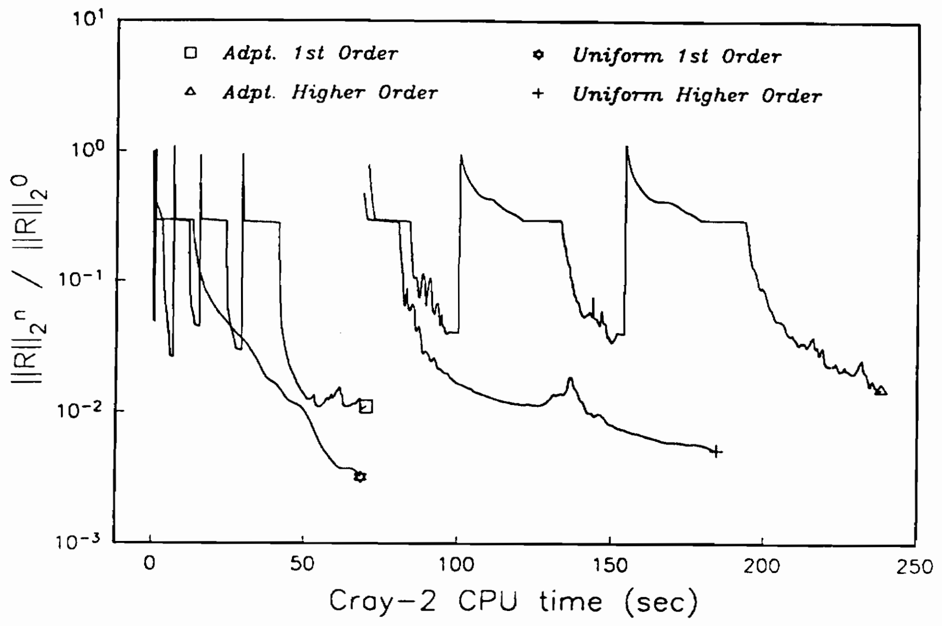
Figure 31(a) shows a plot of the normalized residual (the L_2 norm of the residual divided by the L_2 norm of the free stream residual on the given mesh) versus Cray-2

CPU time for the transonic circular arc. Compared on this plot are the convergence histories of a remeshing strategy and a solution calculated on a uniform 2439 element mesh. The time to generate each new grid for the remeshing strategy was included in the CPU time and is represented in the plot as the time between the bottom of a spike and the peak of a spike. Although the CPU times do increase as the grids become progressively finer, the CPU time to generate a given mesh is small when compared to the time it takes to converge the solution on that mesh.

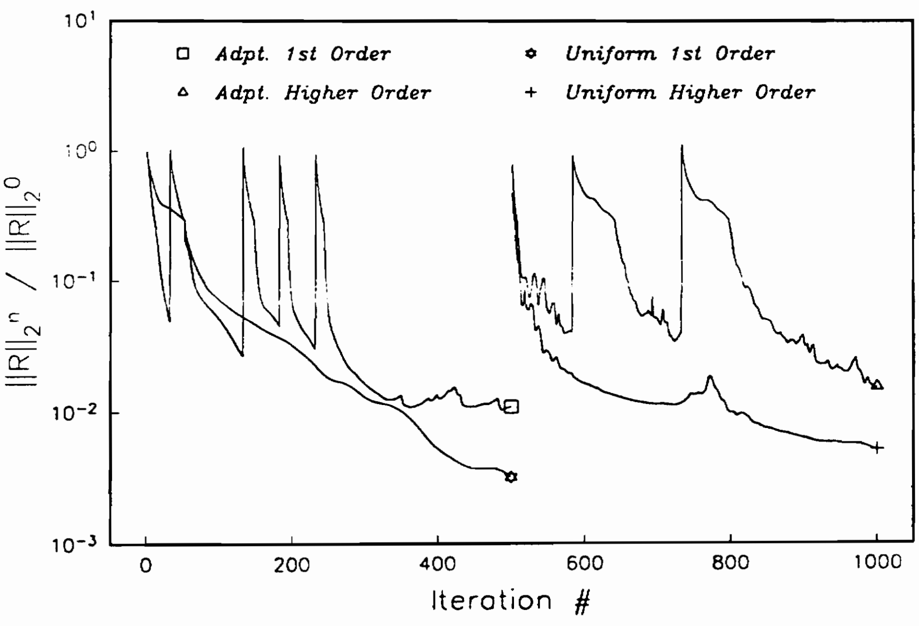
The remeshing sequence started on a very coarse mesh of 79 elements, remeshed the first time at a residual value of 0.05, and then remeshed every 50 time steps thereafter up to a maximum of four remeshes (i.e. five grids total). After four remeshing cycles, the first-order solution was converged on the resulting 2039 element mesh. A higher-order remeshing sequence was then performed beginning with the first-order solution. In this case, only two remeshes were performed before the solution was converged on a 2389 element mesh.

The solution on the uniform mesh was computed using the same number of time steps (iterations) as the remeshing sequence with approximately the same number of elements as the final mesh. This is more evident in Fig. 31(b) which shows the corresponding normalized residual versus iteration plot.

The remeshing strategy takes slightly more CPU time than the solution computed on the uniform mesh. However, because the adaptive remeshing clusters elements in large gradient regions in the flow as can be seen by the 2389 element adapted grid (Fig. 32(a)), the pressure contours for this remeshing strategy, Fig. 32(b), display a much finer shock and better overall solution quality than does the 2439 element uniform mesh (Figs. 32(c) and 32(d)).

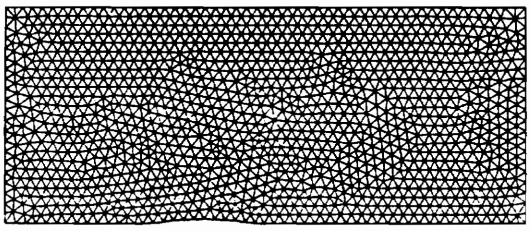


(a)

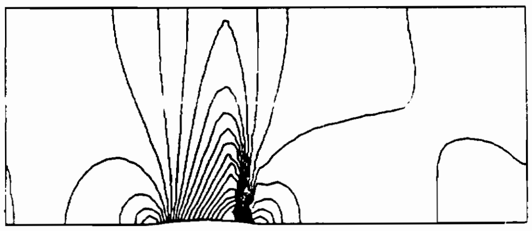


(b)

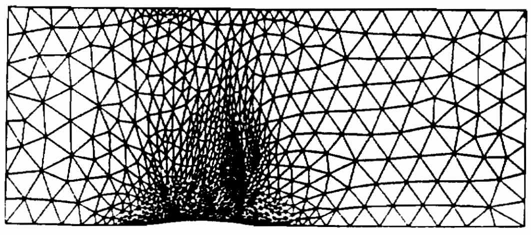
Figure 31) (a) Residual vs. Cray-YMP CPU time, (b) residual vs. iteration for remeshing strategy and single grid calculation of transonic circular arc in a channel, $M_\infty = 0.85$



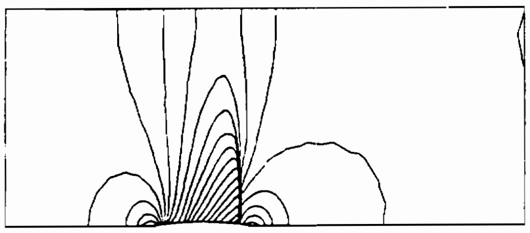
(a)



(b)



(c)



(d)

Figure 32) (a) Uniform 2439 element mesh, (b) uniform mesh pressure contours, (c) adapted 2389 element mesh, (d) adapted mesh pressure contours for higher order transonic circular arc in a channel, $M_\infty = 0.85$

Figure 33 shows the progression of the first-order adaptive remeshing solution starting from the initial 79 element uniform mesh (Fig. 33(a)) and ending with the first-order solution on the 2039 element mesh (Figs. 33(i) & 33(j)). The initial few grids in the sequence rapidly obtain a good long wave pattern, Figs. 33(a)-(d), which is passed through the interpolation procedure to the finer meshes.

Figure 34 shows the higher-order remeshing solution which starts with the final first-order solution. Since the solution is already well defined, the higher-order remeshes just serve to refine the shock further. It is not clear from this calculation whether the higher-order remeshing solution is better than the first-order remeshing solution because of the increased accuracy of the interpolation or because the higher-order calculation involves more mesh elements. From the uniform mesh results given before for the 4008 element mesh (Fig. 29), it was obvious that the higher-order interpolation increased the accuracy of the solution. However, in the adapted case where elements are aligned with flow discontinuities, it is possible for an Euler calculation (which neglects viscous dissipation) that the higher-order interpolation has little effect on the solution.

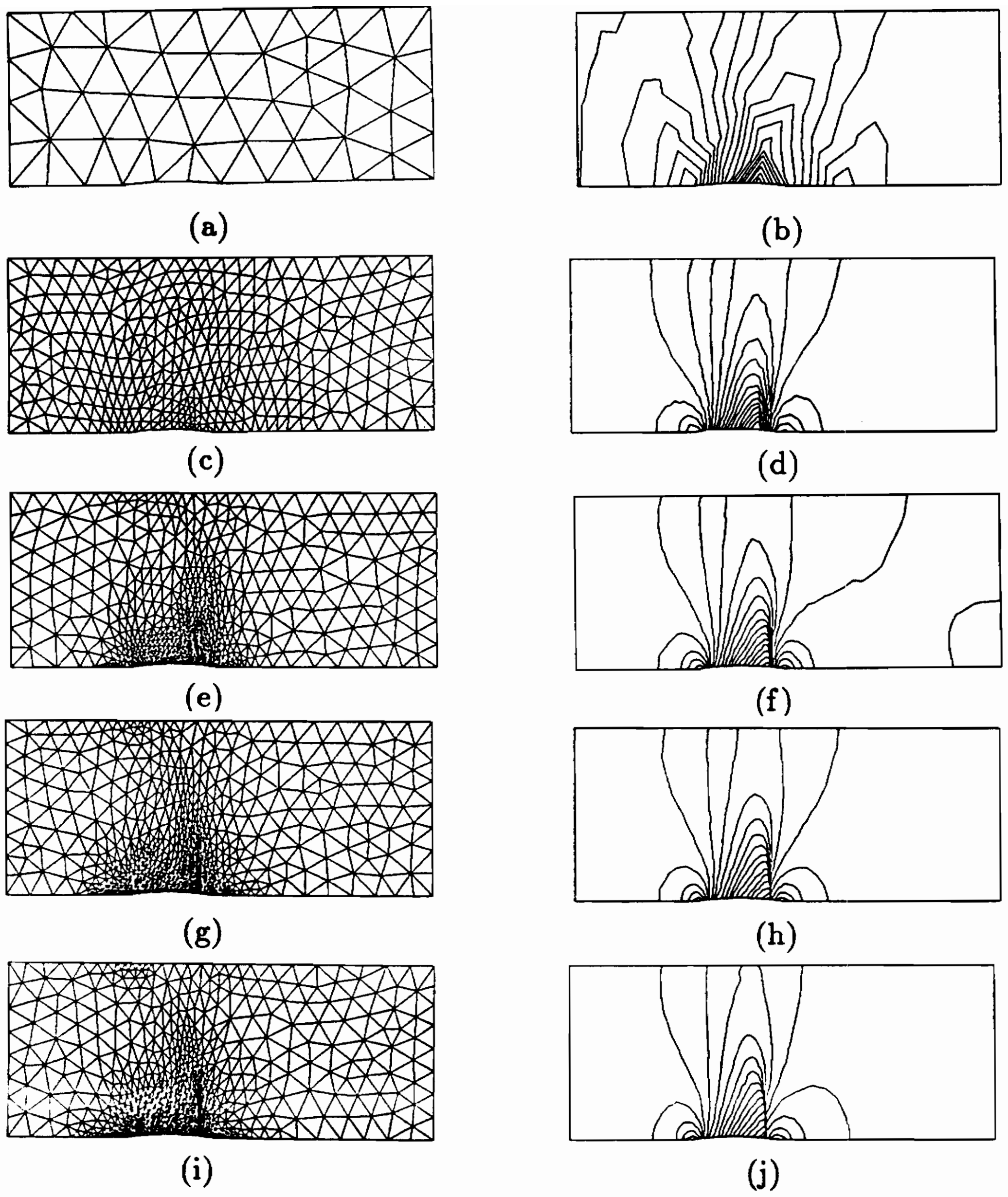


Figure 33) (a) Mesh: 79 elements, 55 vertices, (b) pressure contours
(c) Mesh: 635 elements, 356 vertices, (d) pressure contours
(e) Mesh: 1117 elements, 614 vertices, (f) pressure contours
(g) Mesh: 1695 elements, 916 vertices, (h) pressure contours
(i) Mesh: 2039 elements, 1098 vertices, (j) pressure contours
for first-order adaptive remeshing sequence:
transonic circular arc, $M_\infty = 0.85$

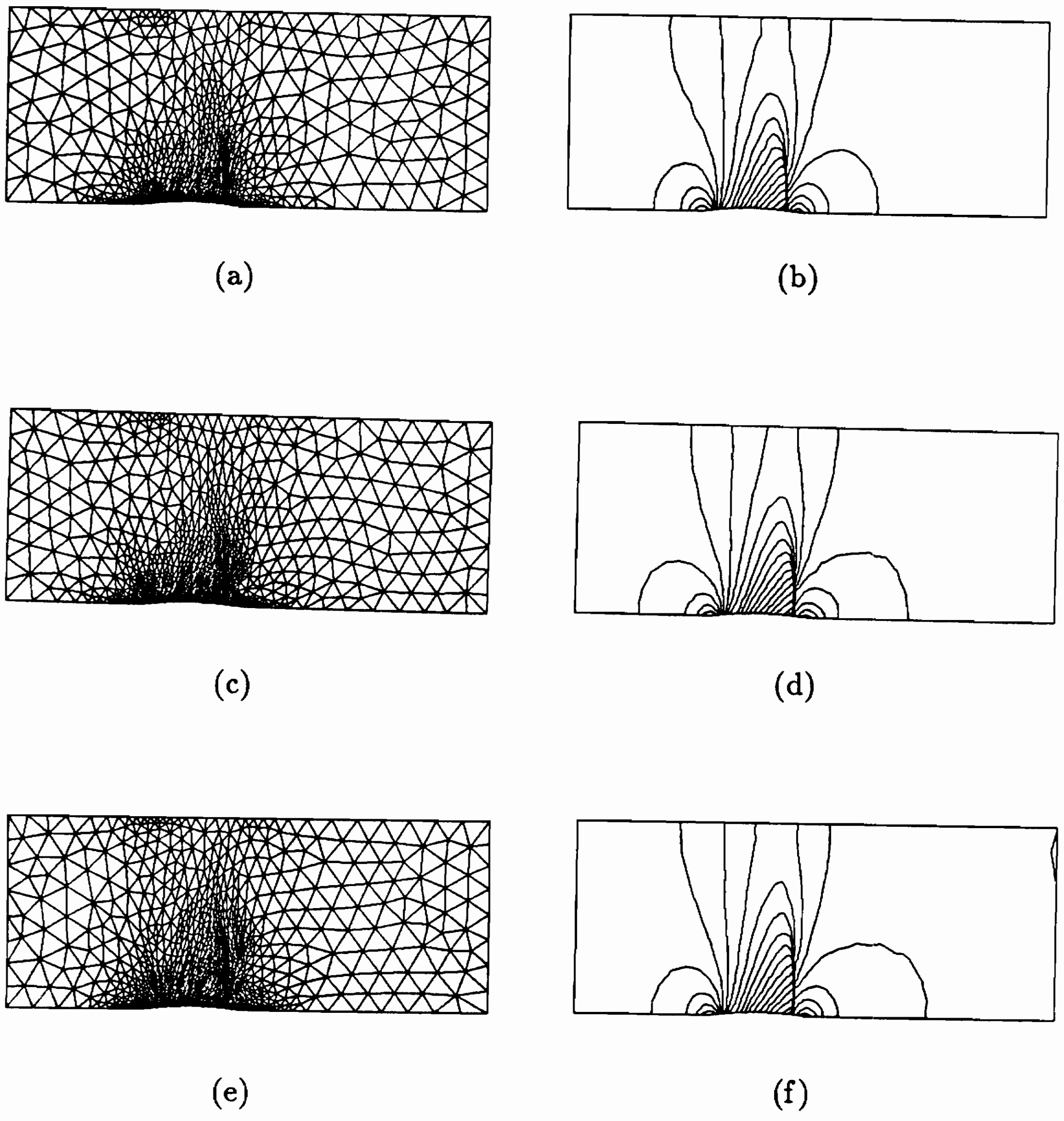


Figure 34) (a) Mesh: 2039 elements, 1098 vertices, (b) pressure contours, (c) Mesh: 2210 elements, 1182 vertices, (d) pressure contours, (e) Mesh: 2389 elements, 1274 (f) pressure contours for higher order adaptive remeshing sequence: transonic circular arc, $M_\infty = 0.85$

5.2.2 Mach 3.0 Internal Flow with Area Change

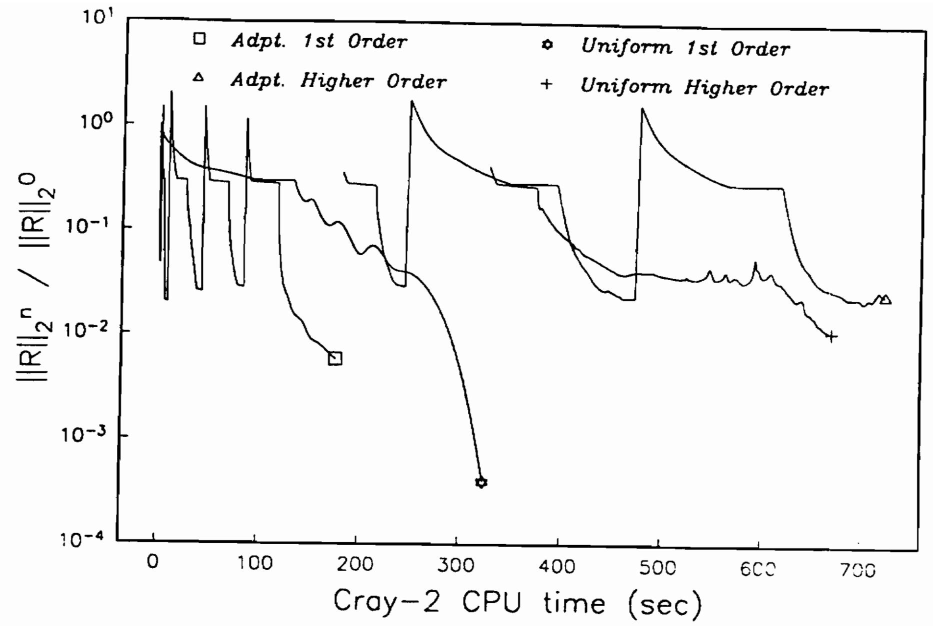
The Mach 3.0 internal flow test case was run utilizing the symmetry in the direction normal to the flow. Therefore all the figures of meshes and contours display the entire domain, while the computational domain consists of the bottom half. The number of elements of a mesh given in this paper reflects the number of elements used for computing the bottom half.

Figure 35(a) shows a plot of the normalized residual versus Cray-2 CPU time for this case. Compared on this plot are the convergence histories of a remeshing strategy similar to that used in the transonic arc case (only the number of elements in each mesh differed) and a solution calculated on a uniform 6263 element mesh. The remeshing sequence started on a very coarse mesh of 85 elements. After four remeshing cycles, the first-order solution was converged on a 4211 element mesh. The higher-order remeshing sequence resulted in a converged solution on a 6247 element mesh. Figure 35(b) shows the corresponding normalized residual versus iteration plot.

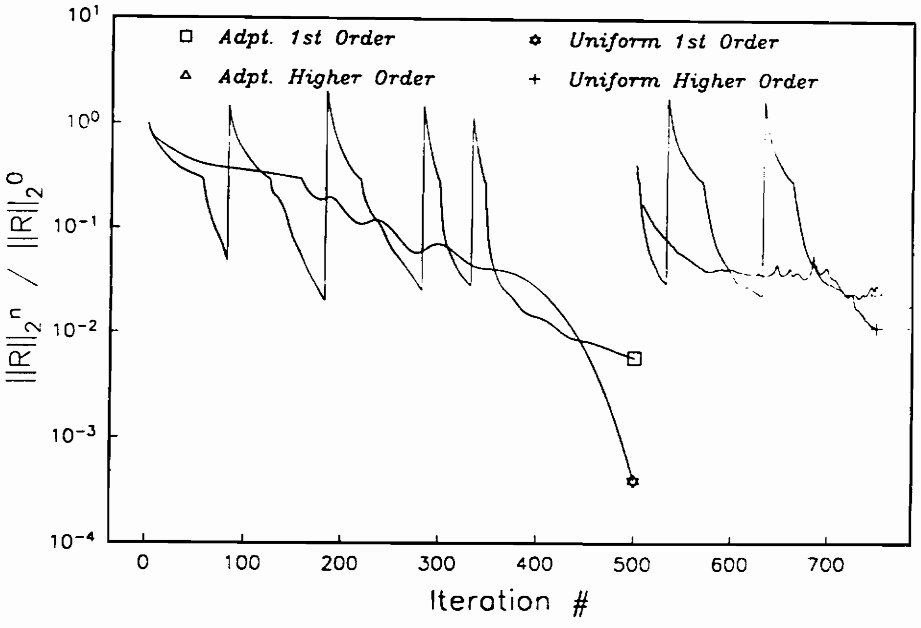
The remeshing strategy takes approximately the same CPU time as the solution computed on the uniform mesh. Since the 6247 element adapted grid in Fig. 36(a) has more elements near the shocks, and since these elements are stretched in the direction of the shocks, the pressure contours for this case (Fig. 36(b)) display much sharper shocks and far less dissipation than the 6263 element uniform mesh (Figs. 36(c) and 36(d)).

Figure 37 shows the first-order adaptive remeshing sequence beginning on the initial 85 element uniform mesh (Fig. 37(a)) and finishing on the 4211 element mesh (Fig. 37(i)). The first few grids in the sequence develop the initial shock, expansion and reflection, Figs. 37(a)-(f), and the latter grids better define these phenomenon.

Figure 38 shows the higher order remeshing sequence which starts with the final first-order solution. The higher-order remeshes make a significant contribution in refining all of the shocks, especially those toward the exit. Once again, as was discussed for the transonic circular arc remeshing problem, it is not clear whether the increase in solution quality was due to the higher-order interpolation or the significant increase in the number of elements in the mesh.



(a)



(b)

Figure 35) (a) Residual vs. Cray-YMP CPU time, (b) residual vs. iteration for remeshing strategy and single grid calculation of internal flow with area change, $M_\infty = 3.0$

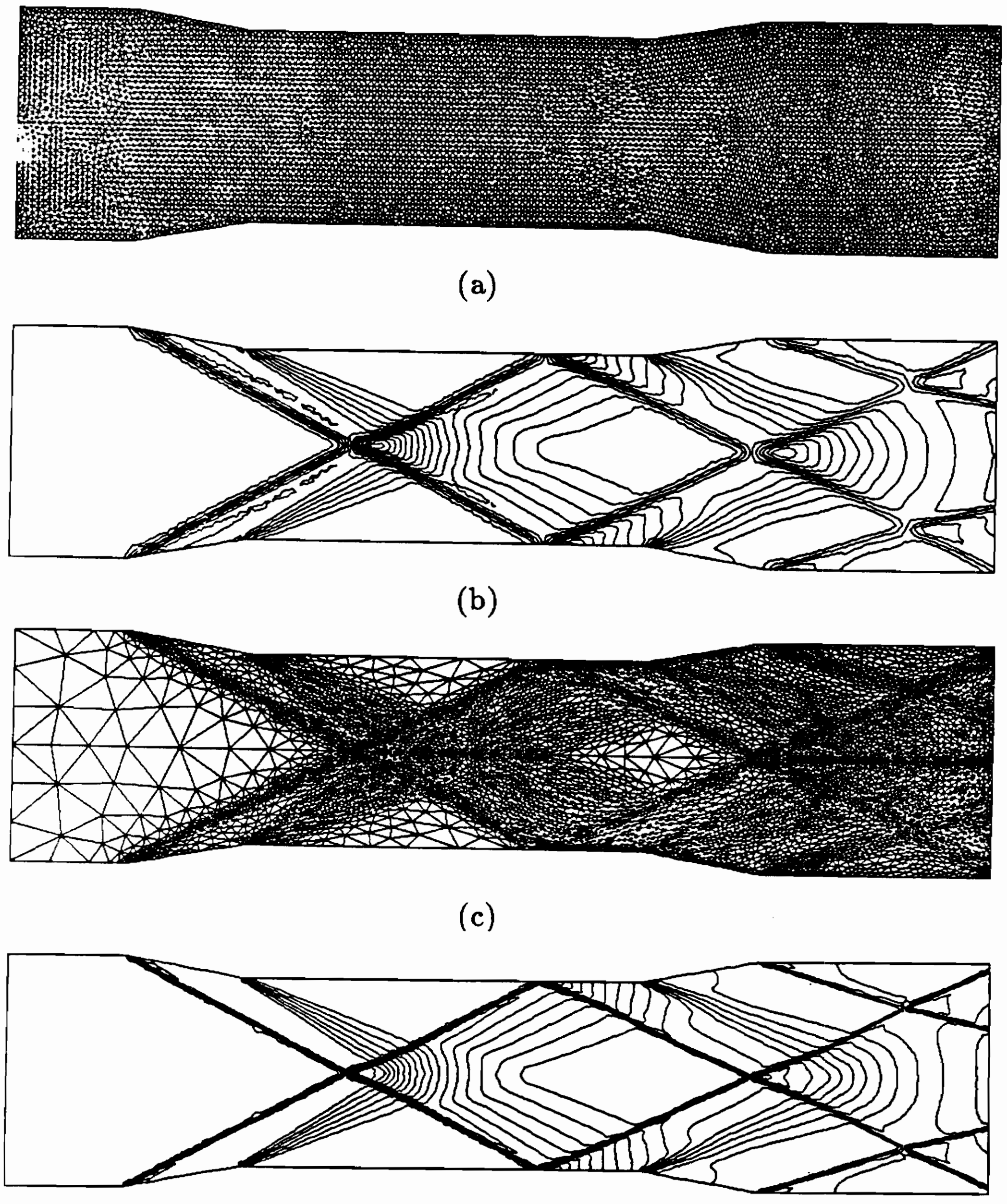


Figure 36) (a) Uniform 6263 element mesh, (b) uniform mesh pressure contours, (c) adapted 6247 element mesh, (d) adapted mesh pressure contours for higher order internal flow with area change, $M_\infty = 3.0$

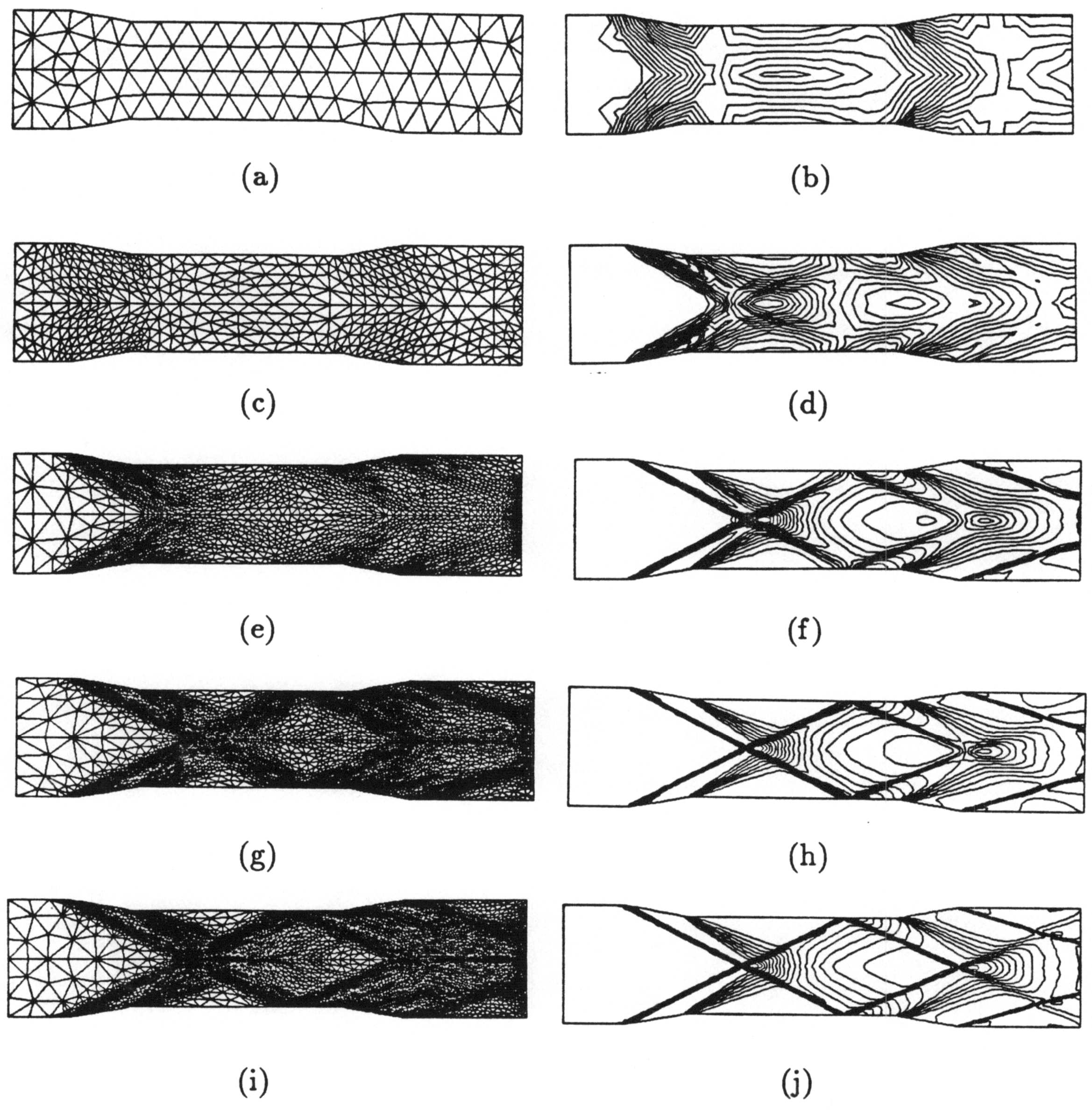


Figure 37) (a) Mesh: 85 elements, 67 vertices, (b) pressure contours
(c) Mesh: 453 elements, 268 vertices, (j) pressure contours
(e) Mesh: 2119 elements, 1140 vertices, (f) pressure contours
(g) Mesh: 3754 elements, 1980 vertices, (h) pressure contours
(i) Mesh: 4211 elements, 2219 vertices, (j) pressure contours
for first-order adaptive remeshing sequence:
internal flow with area change, $M_\infty = 3.0$

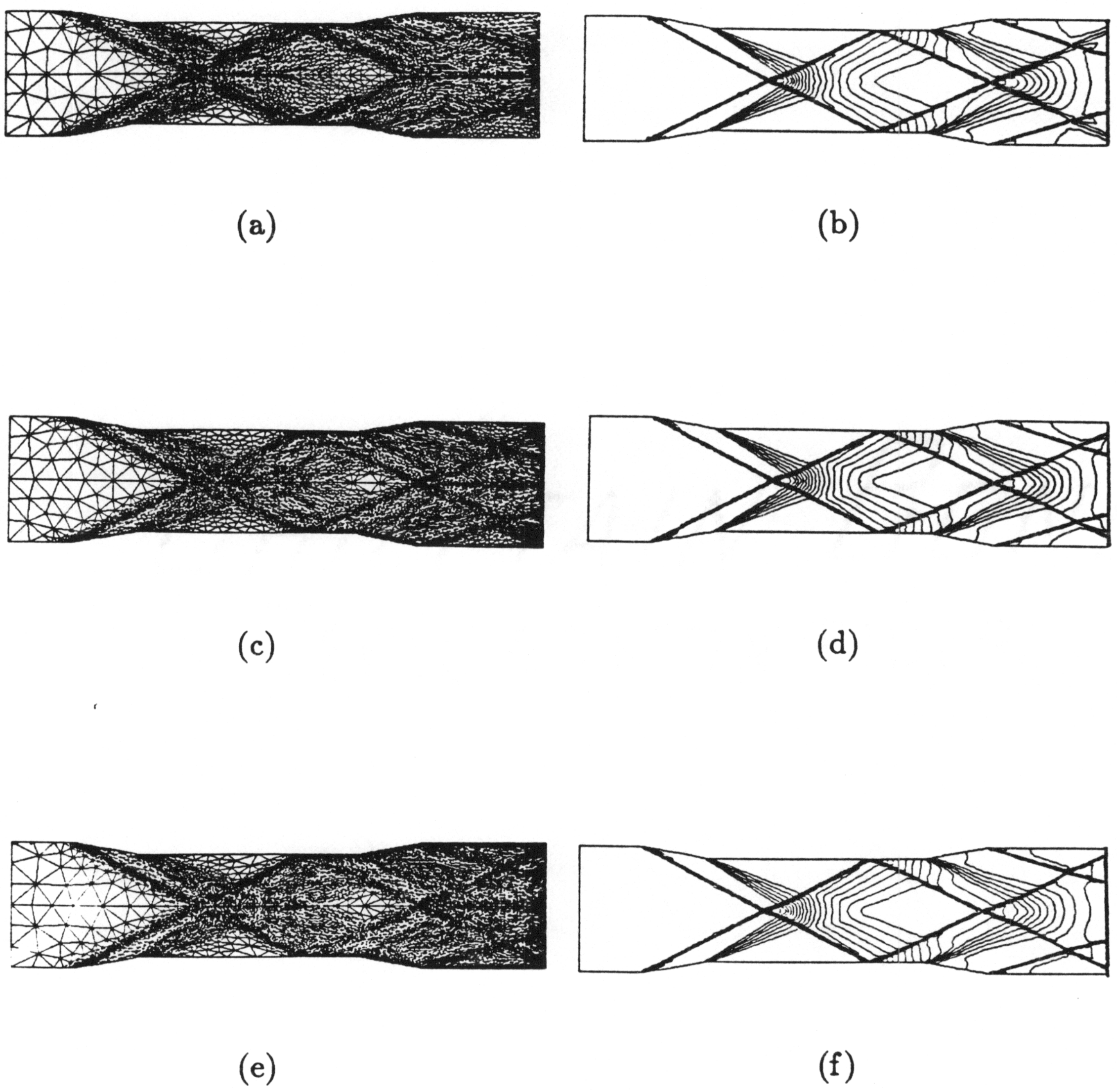


Figure 38) (a) Mesh: 4211 elements, 2219 vertices, (b) pressure contours, (c) Mesh: 5714 elements, 3005 vertices, (d) pressure contours, (e) Mesh: 6247 elements, 3275 vertices, (f) pressure contours for higher-order adaptive remeshing sequence: internal flow with area change, $M_\infty = 3.0$

VI. Governing Equations for Three Dimensions

6.1 Navier-Stokes Equations

The formulation presented here follows the theoretical framework of Grossman and Cinnella[1,2,3,4]. Because of the need to develop efficient computer codes capable of accurately modeling the more complex physics, chemistry, and thermodynamics associated with hypersonic flight, upwind methods have been extended to include such capabilities. This section gives a brief overview of the current modeling developed for the three-dimensional structured computer code GASP. The majority of the coding of the chemistry and thermodynamic models as well as the laminar viscous terms was done by Cinnella[1]. The extension of this work to a three-dimensional unstructured grid will be discussed in chapter IX.

The governing equations for a three-dimensional flow with non-equilibrium chemistry and non-equilibrium internal energy may be written in vector conservation form using generalized coordinates as

$$\frac{\partial}{\partial t}\left(\frac{Q}{J}\right) + \frac{\partial(\hat{F}-\hat{F}_v)}{\partial \xi} + \frac{\partial(\hat{G}-\hat{G}_v)}{\partial \eta} + \frac{\partial(\hat{H}-\hat{H}_v)}{\partial \zeta} = \frac{W}{J}, \tag{6.1.1}$$

where J is the Jacobian of the coordinate transformation between the Cartesian frame $(x,\ y,\ z)$ and the generalized curvilinear frame $(\xi,\ \eta,\ \zeta)$ which is given by

$$J = \frac{\partial(\xi,\ \eta,\ \zeta)}{\partial(x,\ y,\ z)} = det \begin{pmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{pmatrix}, \tag{6.1.2}$$

and Q is the vector of conserved variables of length $N+M+4$ given by

$$Q = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \vdots \\ \rho_N \\ \rho u \\ \rho v \\ \rho w \\ \rho_1 e_{n_1} \\ \rho_2 e_{n_2} \\ \vdots \\ \rho_M e_{n_M} \\ \rho e_0 \end{pmatrix} \tag{6.1.3}$$

where the species density is ρ_i for $i = 1, \ldots N$, the total density is ρ, the mass-averaged velocity vector is $\vec{U} = (u\hat{\imath} + v\hat{\jmath} + w\hat{k})$, the species contribution to the non-equilibrium energy per unit mass is e_{n_i} for $i = 1, \ldots M$, and the total energy per unit mass is e_0.

The vector of source terms W models the effects of finite-rate chemistry and vibrational non-equilibrium by

$$W = \begin{pmatrix} \dot{\rho}_1 \\ \dot{\rho}_2 \\ \vdots \\ \vdots \\ \dot{\rho}_N \\ 0 \\ 0 \\ 0 \\ (\dot{Q}_1)_{n,E} + (\dot{Q}_1)_{n,I} \\ (\dot{Q}_2)_{n,E} + (\dot{Q}_2)_{n,I} \\ \vdots \\ (\dot{Q}_M)_{n,E} + (\dot{Q}_M)_{n,I} \\ 0 \end{pmatrix}, \tag{6.1.4}$$

where $\dot{\rho}_i$ for $i = 1, \ldots N$ is the rate of production of species i, and $(\dot{Q}_i)_{n,E}$ and $(\dot{Q}_i)_{n,I}$ for $i = 1, \ldots M$ are the elastic and inelastic contributions to the source terms of the non-equilibrium energy equation.

The inviscid flux vectors can be written

$$\hat{F} = \frac{|\nabla\xi|}{J}\begin{pmatrix} \rho_1\bar{U} \\ \rho_2\bar{U} \\ \vdots \\ \vdots \\ \rho_N\bar{U} \\ \rho\bar{U}u+\hat{\xi}_x p \\ \rho\bar{U}v+\hat{\xi}_y p \\ \rho\bar{U}w+\hat{\xi}_z p \\ \rho_1 e_{n_1}\bar{U} \\ \vdots \\ \rho_M e_{n_M}\bar{U} \\ \rho h_0\bar{U} \end{pmatrix}, \hat{G} = \frac{|\nabla\hat{\eta}|}{J}\begin{pmatrix} \rho_1\bar{V} \\ \rho_2\bar{V} \\ \vdots \\ \vdots \\ \rho_N\bar{V} \\ \rho\bar{V}u+\eta_x p \\ \rho\bar{V}v+\eta_y p \\ \rho\bar{V}w+\eta_z p \\ \rho_1 e_{n_1}\bar{V} \\ \vdots \\ \rho_M e_{n_M}\bar{V} \\ \rho h_0\bar{V} \end{pmatrix}, \hat{H} = \frac{|\nabla\zeta|}{J}\begin{pmatrix} \rho_1\bar{W} \\ \rho_2\bar{W} \\ \vdots \\ \vdots \\ \rho_N\bar{W} \\ \rho\bar{W}u+\hat{\zeta}_x p \\ \rho\bar{W}v+\hat{\zeta}_y p \\ \rho\bar{W}w+\hat{\zeta}_z p \\ \rho_1 e_{n_1}\bar{W} \\ \vdots \\ \rho_M e_{n_M}\bar{W} \\ \rho h_0\bar{W} \end{pmatrix}, \quad (6.1.5)$$

where $|\nabla\hat{\xi}|$, $|\nabla\hat{\eta}|$, $|\nabla\hat{\zeta}|$ are the unit normals of the coordinate surfaces, $\frac{|\nabla\xi|}{J}$, $\frac{|\nabla\eta|}{J}$, $\frac{|\nabla\zeta|}{J}$ are the areas of the coordinate surfaces, p is the pressure, h_0 is the stagnation enthalpy per unit mass, and the components of the velocity in generalized coordinates are given by

$$\begin{pmatrix} \bar{U} \\ \bar{V} \\ \bar{W} \end{pmatrix} = \begin{pmatrix} \hat{\xi}_x & \hat{\xi}_y & \hat{\xi}_z \\ \hat{\eta}_x & \hat{\eta}_y & \hat{\eta}_z \\ \hat{\zeta}_x & \hat{\zeta}_y & \hat{\zeta}_z \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix}, \quad (6.1.6)$$

The viscous flux vectors $\hat{F}_v$, $\hat{G}_v$, $\hat{H}_v$ can all be expressed by the same basic form. Taking $\hat{F}_v$ as an example, the result is

$$\hat{F}_v = \frac{|\nabla\xi|}{J}\begin{pmatrix} -\rho_1\hat{V}_{1_1} \\ -\rho_2\hat{V}_{2_1} \\ \vdots \\ \vdots \\ -\rho_N\hat{V}_{N_1} \\ \hat{\xi}_x\tau_{xx}+\hat{\xi}_y\tau_{xy}+\hat{\xi}_z\tau_{xz} \\ \hat{\xi}_x\tau_{yx}+\hat{\xi}_y\tau_{yy}+\hat{\xi}_z\tau_{yz} \\ \hat{\xi}_x\tau_{zx}+\hat{\xi}_y\tau_{zy}+\hat{\xi}_z\tau_{zz} \\ k_{n_1}(\nabla\xi\cdot\nabla T_{n_1})-\rho_1 e_{n_1}\hat{V}_{1_1} \\ \vdots \\ k_{n_M}(\nabla\xi\cdot\nabla T_{n_M})-\rho_M e_{n_M}\hat{V}_{M_1} \\ \hat{\Theta} \end{pmatrix}, \tag{6.1.7}$$

where the heat flux contribution is

$$\hat{\Theta} = (u\hat{F}_{v_{N+1}} + v\hat{F}_{v_{N+2}} + w\hat{F}_{v_{N+3}})\frac{J}{|\nabla\xi|} + k(\nabla\xi\cdot\nabla T). \tag{6.1.8}$$

and $\hat{V}_s$ stands for the species diffusion velocity written in generalized coordinates, whose components are obtained from their Cartesian counterparts by a formula similar to (6.1.6). Similar results apply for the other space directions, when $\hat{\xi}$ is replaced by $\hat{\eta}$ and $\hat{\zeta}$ respectively, and the proper components of $\hat{V}_s$ are selected.

The shear stresses can be given in tensor notation as

$$\tau_{x_i x_j} = \mu(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k}), \qquad i,j = 1,2,3, \tag{6.1.9}$$

where μ is the viscosity coefficient and δ_{ij} is the Kroneker delta function.

The space derivatives $\partial(\cdot)/\partial x_i$, for $i = 1,2,3$, in equation (6.1.9) are in Cartesian coordinates and need to be converted to generalized coordinates via the chain rule

$$\begin{aligned}\frac{\partial(\cdot)}{\partial x_1} &= |\nabla\xi|\hat{\xi}_x\frac{\partial(\cdot)}{\partial\xi} + |\nabla\eta|\hat{\eta}_x\frac{\partial(\cdot)}{\partial\eta} + |\nabla\zeta|\hat{\zeta}_x\frac{\partial(\cdot)}{\partial\zeta}, \\ \frac{\partial(\cdot)}{\partial x_2} &= |\nabla\xi|\hat{\xi}_y\frac{\partial(\cdot)}{\partial\xi} + |\nabla\eta|\hat{\eta}_y\frac{\partial(\cdot)}{\partial\eta} + |\nabla\zeta|\hat{\zeta}_y\frac{\partial(\cdot)}{\partial\zeta}, \\ \frac{\partial(\cdot)}{\partial x_3} &= |\nabla\xi|\hat{\xi}_z\frac{\partial(\cdot)}{\partial\xi} + |\nabla\eta|\hat{\eta}_z\frac{\partial(\cdot)}{\partial\eta} + |\nabla\zeta|\hat{\zeta}_z\frac{\partial(\cdot)}{\partial\zeta}.\end{aligned} \tag{6.1.10}$$

In order to close the system of equations given in equation (6.1.0) it is necessary to have an equation of state. The state relationship of the pressure to the specific internal energy occurs implicitly through the temperature. For a given chemical composition, internal energy and non-equilibrium energy, the temperature must be evaluated from

$$e = \sum_{i=1}^{N} \frac{\rho_i}{\rho} \Big[\int_{T_{ref}}^{T} c_{v_i}(\tau)\, d\tau + h_{f_i} \Big] + \sum_{i=1}^{M} \frac{\rho_i}{\rho} e_{n_i} \, , \tag{6.1.11}$$

where c_{v_i} is the specific heat at constant volume for species i. Iterative procedures are then used to solve for T. However, if only the translational and rotational contributions to the internal energy are considered, then T can be found directly. Once T is found, the pressure is determined from Dalton's law as

$$p = \sum_{i=1}^{N} \rho_i R_i T \tag{6.1.12}$$

where R_i is the gas constant for species i.

6.2 Simplified Governing Equations

For certain problems in CFD it is not necessary to physically model all of the capabilities discussed in the previous section. Therefore, it is important when writing a CFD code to maintain modularity. This allows the user to model those aspects of a problem deemed important and not waist CPU time modeling phenomenon which do not apply to the problem at hand. This section discusses various possible simplifications to the governing equations.

A typical approximation consists in neglecting viscous contributions except in one space direction, normal to a solid surface[34]. The resulting equations are known as Thin-Layer Navier-Stokes equations[35]. When this approximation is used for the

ξ direction, the viscous flux terms in the ξ direction now become

$$\hat{F}_v = \frac{|\nabla\xi|^2}{J}\begin{pmatrix} -\rho_1 \hat{V}_{1_1}/|\nabla\xi| \\ -\rho_2 \hat{V}_{2_1}/|\nabla\xi| \\ \vdots \\ \vdots \\ -\rho_N \hat{V}_{N_1}/|\nabla\xi| \\ \mu u_\xi + \mu\hat{\xi}_x(\hat{\xi}_x u_\xi + \hat{\xi}_y v_\xi + \hat{\xi}_z w_\xi)/3 \\ \mu v_\xi + \mu\hat{\xi}_y(\hat{\xi}_x u_\xi + \hat{\xi}_y v_\xi + \hat{\xi}_z w_\xi)/3 \\ \mu w_\xi + \mu\hat{\xi}_z(\hat{\xi}_x u_\xi + \hat{\xi}_y v_\xi + \hat{\xi}_z w_\xi)/3 \\ k_{n_1}\partial T_{n_1}/\partial\xi - \rho_1 e_{n_1}\hat{V}_{1_1}/|\nabla\xi| \\ \vdots \\ k_{n_M}\partial T_{n_M}/\partial\xi - \rho_M e_{n_M}\hat{V}_{M_1}/|\nabla\xi| \\ \hat{\Theta} \end{pmatrix}, \qquad (6.2.1)$$

where the heat flux contribution in generalized coordinates is

$$\hat{\Theta} = k\frac{\partial T}{\partial\xi} + \mu\frac{\partial(u^2+v^2+w^2)/2}{\partial\xi} + \mu\frac{\bar{U}}{3}(\hat{\xi}_x\frac{\partial u}{\partial\xi} + \hat{\xi}_y\frac{\partial v}{\partial\xi} + \hat{\xi}_z\frac{\partial w}{\partial\xi}). \qquad (6.2.2)$$

It is also possible to compute the thin-layer contributions in more than one direction which might be considered for duct flows that have multiple walls. In this case, a second viscous flux vector would be calculated having the same form as $\hat{F}_v$ above with the appropriate substitutions. The cross-derivative terms for these two directions can then either be added or neglected at the user's discretion.

A form of the Thin-Layer Navier-Stokes equations that is used for space marching is called Parabolized Navier-Stokes[36]. Since the Thin-Layer equations are elliptic, a technique developed by Vigneron[37] is used to "parabolize" the equations by removing the negative eigenvalue associated with the pressure gradient in subsonic regions. The resulting inviscid flux vector $\hat{F}$ is

$$\hat{F} = \frac{|\nabla\xi|}{J} \begin{pmatrix} \rho_1 \bar{U} \\ \rho_2 \bar{U} \\ \vdots \\ \vdots \\ \rho_N \bar{U} \\ \rho \bar{U} u + \omega \hat{\xi}_x p \\ \rho \bar{U} v + \omega \hat{\xi}_y p \\ \rho \bar{U} w + \omega \hat{\xi}_z p \\ \rho_1 e_{n_1} \bar{U} \\ \vdots \\ \rho_M e_{n_M} \bar{U} \\ \rho h_0 \bar{U} \end{pmatrix}, \tag{6.2.3}$$

where

$$\omega = \frac{\sigma \gamma M_\xi^2}{1 + (\gamma - 1) M_\xi^2} \tag{6.2.4}$$

with the maximum permissible value of $\omega = 1$. The value σ is a safety factor (usually $= 0.95$) used to compensate for the non-linearity of the Navier-Stokes equations.

Another simplification that can be performed separately or in conjunction with assumptions concerns the thermodynamic model. For the full Navier-Stokes equations of (6.1.1), setting the number of non-equilibrium energy equations to zero ($M = 0$) will result in a one-temperature model with finite-rate chemistry. The chemical model can also be simplified by simply dropping the source term W from equation (6.1.1). This results in a frozen flow calculation where the chemical composition remains constant over time.

The simplest possible case is the perfect gas Euler equations described in section 3.1. They can be easily obtained by performing the above simplifications to the thermodynamic and chemical models and by further neglecting the viscosity, thermal conductivity, and diffusion effects. These added approximations are equivalent to dropping the viscous fluxes $\hat{F}_v$, $\hat{G}_v$ and $\hat{H}_v$ and setting the number of species to one ($N = 1$).

6.3 Reynolds-Averaged Navier-Stokes Equations

Up until now, only laminar viscous effects have been considered. The Reynolds-Averaged Navier-Stokes equations model turbulence by averaging the Reynolds stresses over a finite period of time. With this approximation, turbulence quantities can be calculated from the mean flow variables.

Turbulence is approximated through the use of algebraic and two-equation eddy viscosity models. The total dynamic viscosity and thermal conductivity can then be expressed in terms of their laminar and turbulent contributions as

$$\mu = \mu_l + \mu_t\,, \qquad k = k_l + k_t\,, \tag{6.3.1}$$

where the turbulent thermal conductivity is

$$k_t = \frac{\mu_t c_p}{Pr_t}\,. \tag{6.3.2}$$

A binary diffusion coefficient has been used here to model mass diffusion of each species. Using a constant Lewis number, Le, the diffusion coefficient can also be broken down into laminar and turbulent contributions as

$$D_{12} = \frac{Le}{\rho}\left(\frac{\mu_l}{Pr_l} + \frac{\mu_t}{Pr_t}\right). \tag{6.3.3}$$

Two-equation turbulence models add an additional term to the momentum flux. The term has the magnitude of $\frac{2}{3}\rho K$ where K is the mass-averaged turbulent kinetic energy. The numerical treatment of this term is identical to the treatment of the pressure gradient.

6.3.1 Algebraic Turbulence Modeling

The eddy viscosity is approximated by the algebraic model of Baldwin and Lomax[38] with the modifications of Degani and Schiff[39] for 3D vorticity. The origins of this model stem from Van Driest[40]. In this model, two separate calculations are performed for the eddy viscosity μ_t based on the inner and outer regions. The final value of μ_t is then taken to be the inner region value up until this value is equal to the outer region value, at which point the outer region value is used.

The inner region value for μ_t is calculated from the Prandtl-Van Driest formulation

$$\mu_t = \rho l^2 |\omega| \tag{6.3.4}$$

where ω is the vorticity and the mixing length l is given by

$$l = \kappa y(1 - e^{(\frac{-y^+}{A^+})}) \tag{6.3.5}$$

The distance normal to the wall is y, $\kappa = 0.4$, $A^+ = 26$, and

$$y^+ = \frac{\sqrt{\rho_w \tau_w y}}{\mu_w} \tag{6.3.6}$$

The outer region for μ_t is calculated from

$$\mu_t = 0.0168 C_{cp} \rho F_{wake} F_{kleb}(y) \tag{6.3.7}$$

where $F_{kleb}(y)$ is the Klebanoff intermittency factor, $C_{cp} = 1.6$, and

$$F_{wake} = min(y_{max} F_{max} \;\; C_{wk} y_{max} V_{dif} / F_{max}) \tag{6.3.8}$$

The difference in the maximum and minimum velocity magnitudes in the profile is V_{dif}, $C_{wk} = 1.0$, and the quantities y_{max} and F_{max} are determined from

$$F(y) = y|\omega|(1 - e^{(\frac{-y^+}{A^+})}) \tag{6.3.9}$$

as the values at the first maximum of $F(y)$ away from the wall.

6.3.2 Two-Equation Turbulence Modeling

Two-equation turbulence modeling utilizes two additional differential equations to specify the eddy viscosity[41,42,43]. The first equation determines the velocity scale, and the second determines the length scale, and the eddy viscosity is then calculated as a function of the two variables introduced in the differential equations. A general form of the two equation model similar to the one used by Coakley[41] was developed in this work and can be written as

$$\begin{aligned}\frac{\partial}{\partial t}(\rho s_i) + \frac{\partial}{\partial x}&\left[\rho u s_i - (\mu + \frac{\mu_t}{\mathrm{Pr}_i})\frac{\partial s_i}{\partial x}\right]\\ +\frac{\partial}{\partial y}&\left[\rho v s_i - (\mu + \frac{\mu_t}{\mathrm{Pr}_i})\frac{\partial s_i}{\partial y}\right]\\ +\frac{\partial}{\partial z}&\left]\rho w s_i - (\mu + \frac{\mu_t}{\mathrm{Pr}_i})\frac{\partial s_i}{\partial z}\right] \quad = \quad \phi_i\end{aligned} \tag{6.3.10}$$

where the constants Pr_i are the turbulent Prandtl numbers, and ϕ_i are the source terms. The difference between this model and the Coakley model is the form of the source terms. This model allows for the inclusion of several models given by Brankovic[42], and was developed with the intension of maintaining efficient, modular coding. The source terms are given here by

$$\begin{aligned}\phi_1 &= c_1[\mu_t(\mathcal{L} + c_2) - \frac{2}{3}\rho K\mathcal{D}]\\ \phi_2 &= c_3[\mu_t(c_4\mathcal{L}(\frac{s_2}{s_1}) + c_6) + c_5\rho\epsilon\mathcal{D}]\end{aligned} \tag{6.3.11}$$

The mass-averaged tubulent kinetic energy is $K = (\rho + \bar{\rho'})u_i''u_i''/2\rho$, while ϵ represent the rate of dissipation of turbulent kinetic energy. The strain rate invariant $\mathcal{L}$ is calculated from

$$\begin{aligned}\mathcal{L} = 2\frac{\partial u}{\partial x}\frac{\partial u}{\partial x} &+ \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\frac{\partial u}{\partial y} + \left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right)\frac{\partial u}{\partial z}\\ + \left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)\frac{\partial v}{\partial x} &+ 2\frac{\partial v}{\partial y}\frac{\partial v}{\partial y} + \left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right)\frac{\partial v}{\partial z}\\ + \left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right)\frac{\partial w}{\partial x} &+ \left(\frac{\partial w}{\partial y} + \frac{\partial v}{\partial z}\right)\frac{\partial w}{\partial y} + 2\frac{\partial w}{\partial z}\frac{\partial w}{\partial z}\end{aligned} \tag{6.3.12}$$

and the divergence of the velocity $\mathcal{D}$ is

$$\mathcal{D} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \tag{6.3.13}$$

It is important for convergence to treat the source terms ϕ_i implicitly. The vector Φ can be expanded in time by

$$\Phi^{n+1} = \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix}^{n+1} = \Phi^n + \left[\frac{\partial \Phi}{\partial \hat{S}}\right]^n \Delta \hat{S} \tag{6.3.14}$$

where

$$\Phi = \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix} \qquad \hat{S} = \begin{bmatrix} \rho s_1 \\ \rho s_2 \end{bmatrix} \qquad S = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \tag{6.3.15}$$

The Jacobian matrix can be calculated from

$$\left[\frac{\partial \Phi}{\partial \hat{S}}\right]^n = \left[\frac{\partial \Phi}{\partial S}\right]^n \left[\frac{\partial S}{\partial \hat{S}}\right]^n \qquad \left[\frac{\partial S}{\partial \hat{S}}\right]^n = \begin{bmatrix} \frac{1}{\rho} & 0 \\ 0 & \frac{1}{\rho} \end{bmatrix} \tag{6.3.16}$$

where the elements of $\left[\frac{\partial \Phi}{\partial S}\right]$ are given by

$$\begin{aligned} \frac{\partial \phi_1}{\partial s_i} &= \frac{\partial c_1}{\partial s_i}\phi_1 + c_1\left[\frac{\partial \mu_t}{\partial s_i}(\mathcal{L} + c_2) - \frac{2}{3}\rho\frac{\partial K}{\partial s_i}\mathcal{D} + \frac{\partial c_2}{\partial s_i}\mu_t\right] \\ \frac{\partial \phi_2}{\partial s_i} &= \frac{\partial c_3}{\partial s_i}\phi_2 + c_3\left[\mu_t\left\{\mathcal{L}\left[\frac{\partial c_4}{\partial s_i}(\frac{s_2}{s_1}) + c_4\frac{\partial}{\partial s_i}(\frac{s_2}{s_1})\right] + \frac{\partial c_6}{\partial s_i}\right\} + \right. \\ &\qquad \left. \frac{\partial \mu_t}{\partial s_i}\left[c_4\mathcal{L}(\frac{s_2}{s_1}) + c_6\right] + \frac{\partial c_5}{\partial s_i}\rho\epsilon\mathcal{D} + c_5\rho\frac{\partial \epsilon}{\partial s_i}\mathcal{D}\right] \end{aligned} \tag{6.3.17}$$

The values for c_{1-6} and their derivatives are chosen depending on the specific model desired. This generalized form of the equations allows for the easy coding of additional models, and it also gives a consistent method for comparing various two-equation models. The details of seven different models are given in Appendix A.

In order to use this model in the computer code GASP, it is necessary to convert the governing equations into generalized coordinates. Although straightforward in principle, this is not a trivial task. The final results of this derivation are given in Appendix B. Appendix B also gives a brief discussion of the time integration scheme implemented, and Appendix C describes the upwind treatment of the convective terms for this two-equation turbulence model.

VII. Time Integration on Structured Meshes

In order to obtain a steady-state solution, the spatially discretized Navier Stokes equations must be integrated in time. Three time integration methods were investigated and coded by the author for the Navier-Stokes code GASP, namely : an m-stage explicit Runge-Kutta method, an implicit LU decomposition in a plane, and an implicit Approximate Factorization (AF) in a plane. Both of the implicit schemes use relaxation in the remaining direction. The LU and AF schemes have the ability to reuse the factored Jacobian matrix.

7.1 Explicit Runge-Kutta Time Integration

An m-stage Runge-Kutta scheme advances the solution in time using[24]

$$\begin{aligned}
Q^{(0)} &= Q^{(n)} \\
Q^{(1)} &= Q^{(0)} + \alpha_1 \frac{\Delta t}{V} R(Q^{(0)}) \\
Q^{(2)} &= Q^{(0)} + \alpha_2 \frac{\Delta t}{V} R(Q^{(1)}) \\
&\vdots \qquad \vdots \\
Q^{(m)} &= Q^{(0)} + \alpha_m \frac{\Delta t}{V} R(Q^{(m-1)}) \\
Q^{(n+1)} &= Q^{(m)}
\end{aligned} \tag{7.1.1}$$

where

$$R(Q) = -\frac{\partial(\hat{F} - \hat{F}_v)}{\partial \xi} - \frac{\partial(\hat{G} - \hat{G}_v)}{\partial \eta} - \frac{\partial((\hat{H} - \hat{H}_v))}{\partial \zeta} + W \tag{7.1.2}$$

with the standard weighting coefficients

$$\alpha_i = \frac{1}{m - i + 1} \tag{7.1.3}$$

and V is the cell volume ($V = 1/J$). Convergence to the steady-state is also accelerated by using the maximum permissible time step for each individual cell in the flow field, as dictated by local stability analysis.

7.2 Implicit Time Integration

The LU decomposition and AF methods utilize the Euler implicit time integration which can be represented in delta form as

$$V\frac{\Delta Q}{\Delta t} = R^{n+1} \tag{7.2.1}$$

where $\Delta Q = Q^{n+1} - Q^n$. The equation can be linearized as

$$V\frac{\Delta Q}{\Delta t} = R^n + \frac{\partial R^n}{\partial Q}\Delta Q \tag{7.2.2}$$

Equation (7.2.2) can then be rewritten as the linear system

$$A\Delta Q = R^n \tag{7.2.3}$$

with

$$A = \frac{V}{\Delta t}I + \frac{\partial}{\partial \xi}\frac{\partial(\hat{F}-\hat{F}_v)}{\partial Q} + \frac{\partial}{\partial \eta}\frac{\partial(\hat{G}-\hat{G}_v)}{\partial Q} + \frac{\partial}{\partial \zeta}\frac{\partial(\hat{H}-\hat{H}_v)}{\partial Q} - \frac{\partial W}{\partial Q} \tag{7.2.4}$$

where A contains block matrices of size $(N+M+4)$ x $(N+M+4)$.

7.2.1 LU Decomposition in a Plane

A global LU decomposition method would attempt to solve the linear system exactly. For general three-dimensional problems, however, the storage requirements are too restrictive. Therefore, an LU decomposition is performed in a plane utilizing a banded LU solver described in the next section, and terms from the ξ direction are relaxed through a non-linear update of the residual. Equation (7.2.3) in this case becomes

$$\bar{A}\Delta Q = R(Q^n, Q^{n+1}) \tag{7.2.5}$$

where

$$\begin{aligned}\bar{A} = &\frac{V}{\Delta t}I + \frac{\partial(\hat{F}-\hat{F}_v)^+}{\partial Q} - \frac{\partial(\hat{F}-\hat{F}_v)^-}{\partial Q}\\ &+ \frac{\partial}{\partial \eta}\frac{\partial(\hat{G}-\hat{G}_v)}{\partial Q} + \frac{\partial}{\partial \zeta}\frac{\partial(\hat{H}-\hat{H}_v)}{\partial Q} - \frac{\partial W}{\partial Q}\end{aligned} \tag{7.2.6}$$

On a forward sweep through planes in the ξ direction, the $I-1$ off-diagonal terms are accounted for in the non-linear residual update while the $I+1$ off-diagonal terms are relaxed and vice versa on a reverse i sweep.

The solution procedure involves factoring the linearized coefficient matrix $\bar{A}$ into the product of a lower triangular and upper triangular matrix (L and U) such that

$$[LU]\Delta Q = R(Q^n, Q^{n+1}) \tag{7.2.7}$$

and then solving the system $L\Delta Q^* = R(Q^n, Q^{n+1})$ by forward substitution and $U\Delta Q = \Delta Q^*$ by backward substitution.

The reuse capability involves freezing the LU decomposition in time and performing only forward and backward substitutions to advance the solution vector Q. This can be simply represented by

$$\Delta Q^{n+k} = (U^{-1}L^{-1})^n R(Q^{n+k}, Q^{n+k+1}) \tag{7.2.8}$$

where n represents the time at which the LU is frozen, and $n+k$ is the current time step.

The formulation of this LU decomposition scheme is very similar to the one developed for the unstructured solver. One conceptual difference is the relaxation of the ξ direction. There is also a difference related to the spatial discretization which results in the Jacobian matrix having a variable bandwidth instead of a fixed bandwidth. The application of the unstructured LU decomposition and other time integration schemes to three dimensions will be discussed in more detail in chapter IX.

7.2.2 Approximate Factorization in a Plane

The Approximate Factorization (AF) method in a plane can be written as

$$\left(D + \frac{\partial}{\partial\eta}\frac{\partial(\hat{G}-\hat{G}_v)}{\partial Q}\right) D^{-1} \left(D + \frac{\partial}{\partial\zeta}\frac{\partial(\hat{H}-\hat{H}_v)}{\partial Q}\right) \Delta Q = R(Q^n, Q^{n+1}) \quad (7.2.9)$$

where

$$D = \left(\frac{V}{\Delta t} I + \frac{\partial(\hat{F}-\hat{F}_v)^+}{\partial Q} - \frac{\partial(\hat{F}-\hat{F}_v)^-}{\partial Q} - \frac{\partial W}{\partial Q}\right) \quad (7.2.10)$$

Since each factor only contains derivatives in one direction, they can be put into block tri-diagonal form. The matrix D contains only block diagonal elements because the ξ off-diagonal terms are relaxed in the same manner as they were for the LU case using a non-linear update of the residual. The vector ΔQ is then acquired through the two step procedure

$$\begin{aligned} \Delta Q^* &= \left(D + \frac{\partial}{\partial\eta}\frac{\partial(\hat{G}-\hat{G}_v)}{\partial Q}\right)^{-1} R(Q^n, Q^{n+1}) \\ \Delta Q &= \left(D + \frac{\partial}{\partial\zeta}\frac{\partial(\hat{H}-\hat{H}_v)}{\partial Q}\right)^{-1} D\Delta Q^* \end{aligned} \quad (7.2.11)$$

The AF scheme also takes advantage of the reuse capability by freezing the LU decomposition of each of the factored matrices in time similar to the technique described for the LU method.

7.2.3 Storage and Vectorization for LU and AF

Both the LU decomposition and the AF methods utilize a highly vectorized banded solver in order to save storage and CPU time. The solver stores a matrix $A(j,k)$ with *ndiag* super/sub diagonals in an array B by the rule $A(j,k)$ is stored in $B(ndiag + 1 + j - k, k)$; that is, the jth column of A goes to jth column of B and the main diagonal of A is then the $ndiag + 1$ row of B with each super and subdiagonal on the rows above and below that one in B as shown in Fig. 39.

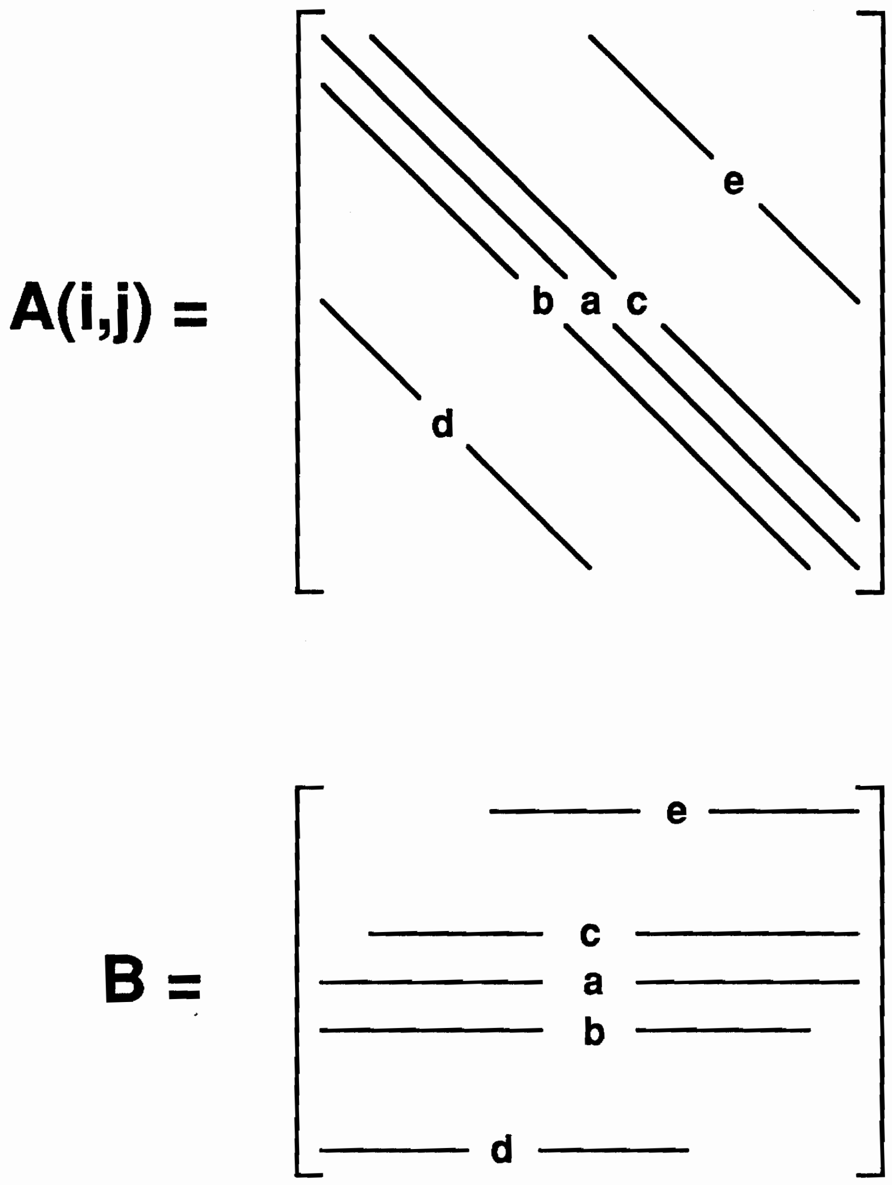


Figure 39) Flat storage of matrix for structured LU and AF solvers

For LU in a plane of dimension $jdim$ x $kdim$ with $(N + M + 4)$ conserved variables, this results in a matrix that requires storage of $(2 * ndiag + 1) * (jdim * kdim) * (N + M + 4)$ where $ndiag = (jdim + 1) * (N + M + 4) - 1$ for the $jdim$ bandwidth case, or $ndiag = (kdim + 1) * (N + M + 4) - 1$ for the $kdim$ bandwidth case.

For AF in a plane, the first step of equation (7.2.11) is done with a $jdim$ bandwidth, and the second is done with a $kdim$ bandwidth so that in both cases the matrix is in block tri-diagonal form and therefore requires the amount of storage given for the LU method but with $ndiag = 2 * (N + M + 4) - 1$. Hence for large $jdim$ and $kdim$, AF in a plane requires far less storage than LU in a plane.

In terms of vectorization, the banded matrix solver vectorizes over the length of the dot product involved in the LU decomposition phase and the forward/backward substitution phases. In addition, for the AF algorithm, since the η and ζ directions are decoupled, each factor in equation (7.2.11) can be divided into independent sections of length $jdim$ and $kdim$ respectively. The LU decomposition can then be further vectorized over these sections resulting in substantial CPU savings.

7.3 Space Marching

Space marching is a very powerful and efficient method for obtaining steady state solutions to many supersonic and hypersonic problems in CFD. In order to utilize space marching, it is necessary that all of the characteristic waves in the streamwise direction are positive so that information is only passed downstream. Because of this it is possible to solve one streamwise plane at a time using only the information given from upstream locations where the solution is already known. As a result, the solution process is broken down from solving one large domain into solving a series of much smaller domains (streamwise planes)[36,44].

In implementing a space marching algorithm, many authors drop the time dependent term and solve the steady state equations directly. An alternative approach given by Walters[44] maintains the time dependent term and iterates for the solution on each plane. This method has several advantages which include: (1) it is possible to calculate higher order streamwise fluxes, (2) there are no stability problems related to streamwise grid spacing, (3) the algorithm is so similar to a global iteration algorithm that both methods can be easily incorporated into the same code.

VIII. Results for Three Dimensional Structured Solver

This section describes the solutions for three problems: a Mach 1.7 supersonic, inviscid analytic forebody, a Mach 1.38 supersonic, inviscid, axisymmetric nozzle with hydrogen-air chemistry, a Mach 14.1 hypersonic flow over a 15° wedge, and a Mach 0.3 subsonic viscous flat plate. The purpose of the analytic forebody calculation is to demonstrate the ability and efficiency of space marching a three-dimensional problem. The nozzle calculation displays the finite-rate chemistry capability. The 15° wedge demonstrates the ability to calculate flows in the hypersonic regime, while the flat plate problem shows a variety of viscous capabilities of the three-dimensional structured solver.

8.1 Analytic Forebody

This analytic forebody problem has been studied experimentally[45] as well as numerically[46]. The problem consists of computing the three-dimensional, inviscid, Mach 1.7 supersonic flow over the cockpit region of a high performance aircraft as shown in Fig. 40. Figure 41 shows the grid used for this calculation which was 31 x 31 x 41 (39,401 grid points), thus, there are 40 i-planes to space march.

The CPU time on a Cray-YMP supercomputer to run this problem was only 36 seconds using the Approximate Factorization in a plane time integration scheme. Pressure contours in the top and bottom planes of symmetry are shown in Fig. 42. The surface pressure distribution in the planes of symmetry are compared to experiment in Fig. 43.

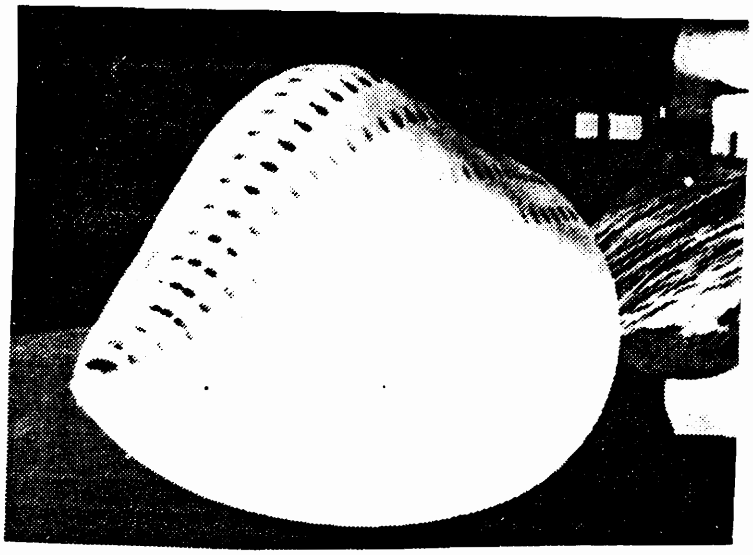


Figure 40) Analytic forebody experimental configuration

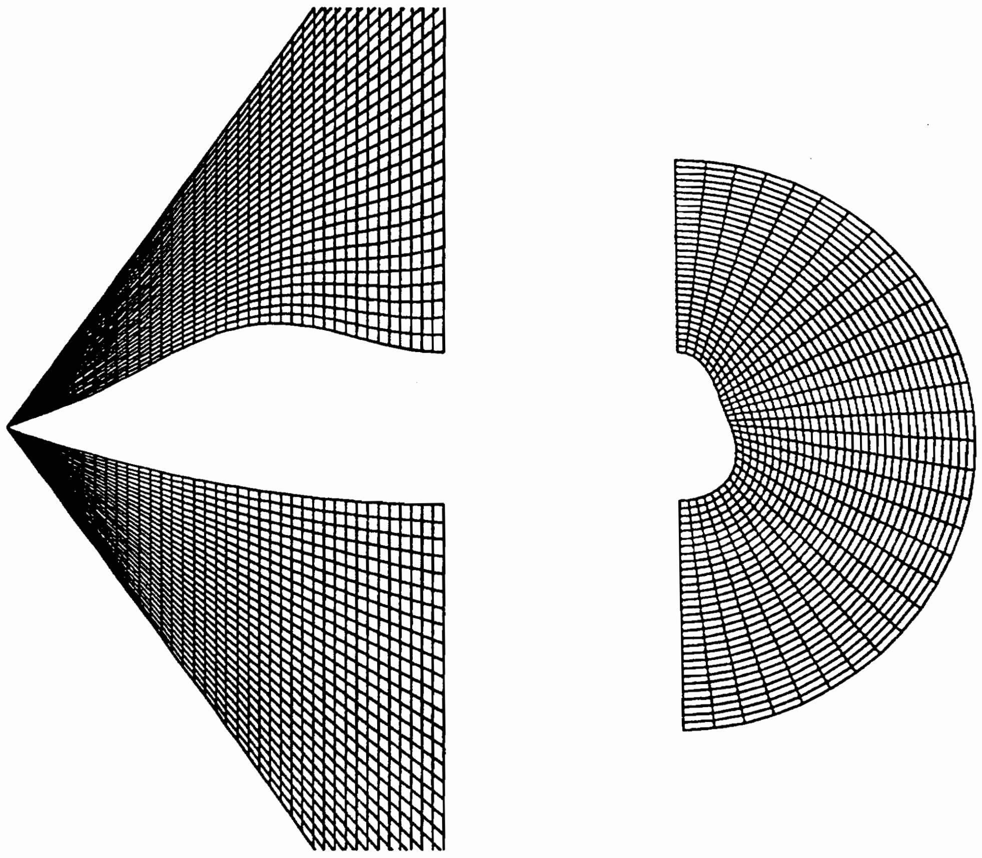


Figure 41) Symmetry plane and crossflow plane grid for analytic forebody

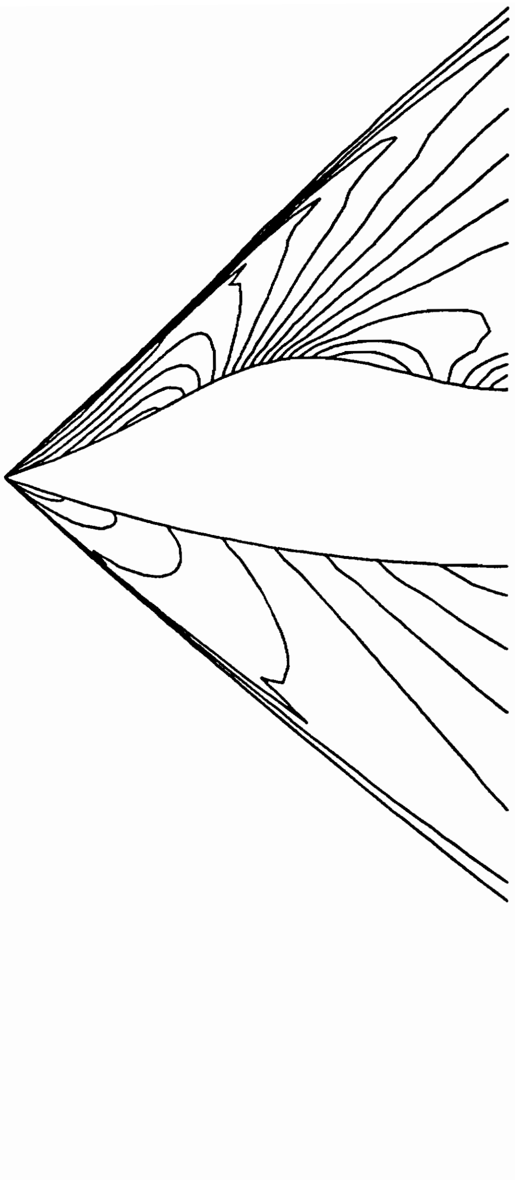


Figure 42) Higher order pressure contours in symmetry plane for analytic forebody, $M_\infty = 1.7$, $\alpha = 0^o$

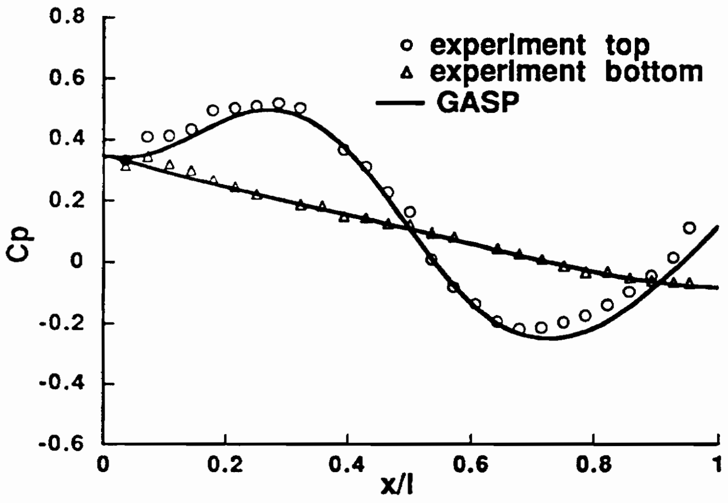


Figure 43) Surface pressure comparison in symmetry plane of analytic forebody calculation with experiment, $M_{\infty} = 1.7$, $\alpha = 0^{o}$

8.2 Supersonic Nozzle with Hydrogen-Air Chemistry

This problem calculates a Mach 1.38 supersonic flow through an axisymmetric nozzle, sketched in Fig. 44, with chemically reacting hydrogen and air at a free stream temperature of 1884.3^oK. The hydrogen-air chemical model used is given by Rogers and Chinitz[47]. The species for this model are: N_2, O_2, H_2, OH, and H_2O and the reactions are given as

$$H_2 + O_2 + N_2 \rightarrow 2OH + N_2$$

$$2OH + H_2 + N_2 \rightarrow 2H_2O + N_2$$

The grid for this problem contains 81 x 11 x 2 grid points. It was converged using AF (Approximate Factorization) in a plane. Volume averaged value of the temperature are compared in Fig. 45 with the calculations of Drummond[48].

8.3 Holden's 15^o Ramp

Holden and Moselle[49] measured skin friction, heat transfer, and pressure distributions on a flat plate with an inclined wedge. The case considered here is the 15^o ramp case in which the flow is laminar and remains attached at the test conditons of $M_\infty = 14.1, T_\infty = 72.2^oK., Re/m = 2.369 \times 10^5$. A sketch of this problem is shown in Fig. 46. The grid for this case is 2 x 51 x 51. The boundary condition on the ramp is no-slip with a prescribed wall temperature of 297^oK.

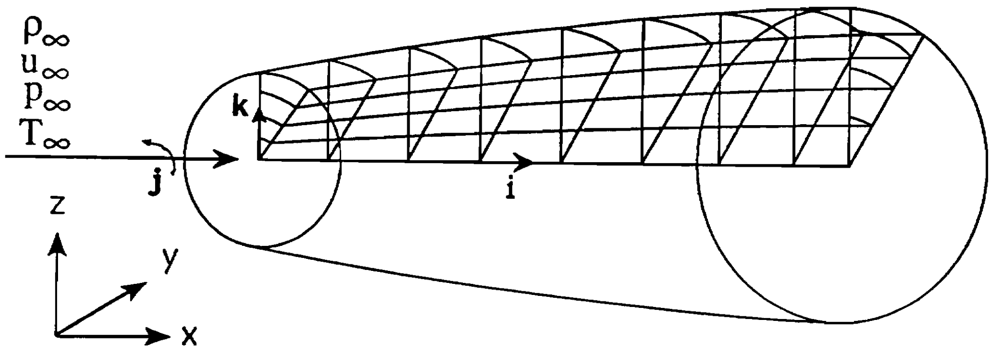


Figure 44) Axisymmetric supersonic combustion nozzle configuration

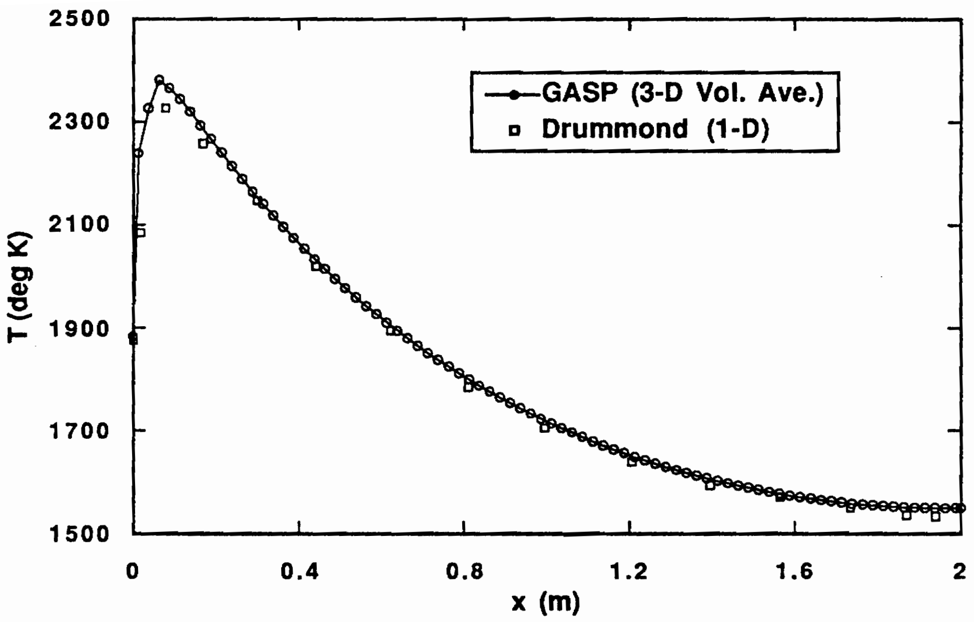


Figure 45) Volume averaged temperature comparison with results from Drummond,et al.[48]

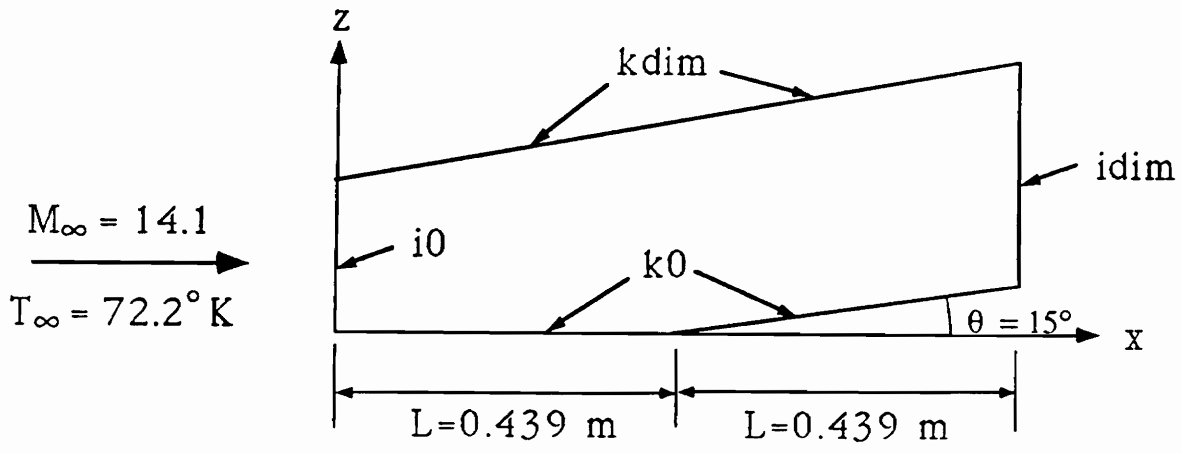


Figure 46) Sketch of Holden's 15° Ramp, $M_\infty = 14.1$

In this problem, as may occur with many space-marching problems, the initial time step may have to be lower than that required on subsequent planes. For this case, the first two planes were run at a Courant number of .2. The first plane ran 180 iterations, met the tolerance for re-using the LU decompostions on the 90th iteration and took 2.366 seconds. The second plane ran 198 iterations, met the tolerance for re-using the LU decompostions on the 62nd iteration and took 1.92 seconds of execution time. Eventhough the second plane took more overall iterations, it tokk less CPU time because more of the iterations re-used the LU decomposition which requires less computational effort.

Next, this case was restarted on plane 3 and marched to plane 50 with an increased Courant number of 5. Figure 47 is a plot of the number of iterations required to converge each plane. The initial enforcement of the no-slip boundary conditions, the subsequent development of the boundary layer, and the lower time step, combine to result in the large number of iterations at the first two stations. The iterations then rapidly decrease, due to both the larger time step and the fact that the initialization from the previous plane improves rapidly at first. The beginning of the ramp can then be clearly seen at the i=26 location where the iterations again rapidly rise, only to start decreasing again as the remainder of the ramp is marched. This brings out a point which is simply that the physics, geometry, and numerics are coupled. When the physics of the flowfield are rapidly changing, so will be the residual, when the geometry changes (as is the case here), the physics get effected and so do the numerics.

Skin friction, pressure, and heat transfer on the surface are compared with the experimental values in Fig. 48. The results of the GASP calculation compare well with expermient for this configuration.

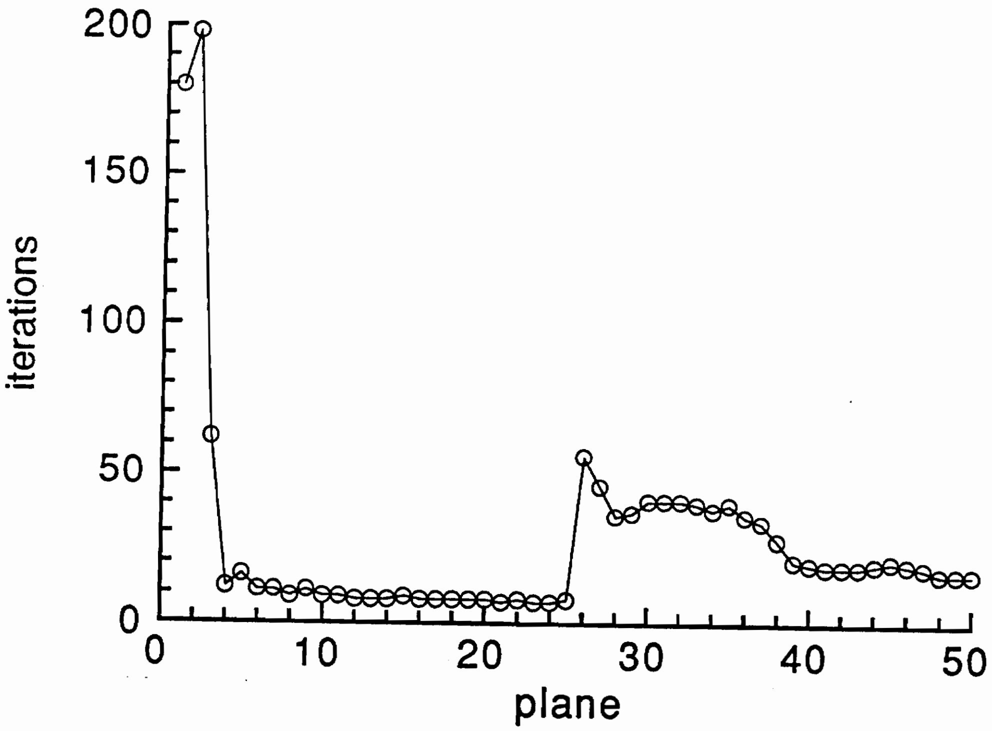
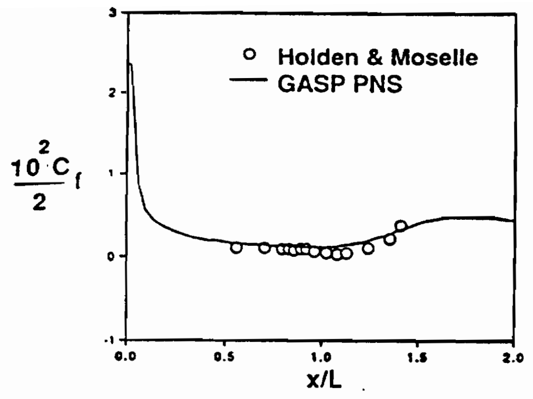
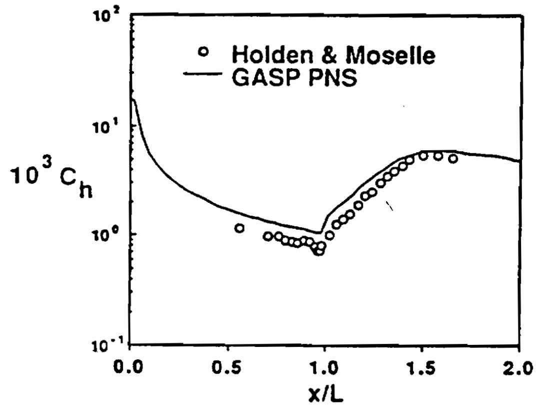


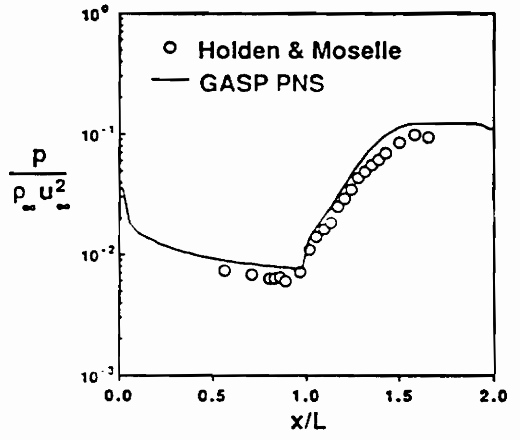
Figure 47) Number of iterations versus the station number for Holden's 15° Ramp, $M_\infty = 14.1$



(a)



(b)



(c)

Figure 48) (a) Skin friction, (b) heat transfer, (c) pressure distribution comparison with experiment for Holden's 15° Ramp, $M_\infty = 14.1$

8.4 Viscous Flow over Flat Plate

Both laminar and turbulent viscous flows over a flat plate were run. The same size grid (2 x 61 x 31) and stretching was used for the calculations where 31 represents the number of streamwise locations, and 61 is the number of grid lines in the direction normal to the plate.

The first run was a subsonic laminar flow, $M_\infty = 0.3$. The velocity profiles at stations $x/L = \frac{1}{3}, \frac{2}{3}, 1$ are plotted in similarity coordinates and compared to a Blasius profile in Fig. 49[50]. As expected, all of the profiles fall on top of the Blasius solution. The purpose of this calculation was simply to demonstrate that the laminar viscous terms were correct before moving on to a more complicated turbulent computation.

The second run was a subsonic turbulent calculation, $M_\infty = 0.3$. This calculation was performed with both the Baldwin-Lomax[38] algebraic and Lam-Bremhorst[41] two-equation eddy viscosity models. Because of the numerical difficulty of starting a two-equation model calculation from the free stream, the Lam Bremhorst model solution was initialized with the results obtained from the algebraic model solution using the eddy viscosity and the mixing length to calculate K and ϵ. Another possible solution to this start-up problem is to initialize the two-equation model using K and ϵ profiles. However, obtaining realistic profiles for a given problem is not always a trivial task. Both the algebraic and two-quation model results are compared in Fig. 50(a) in a u^+ versus y^+ plot with the law of the wall ($u^+ = y^+$ for $y^+ < 10$) and with the log region ($u^+ = 5.6 log(y^+) + 4.9$ for $10 < y^+ < 500$)[50]. Figures 50(b) and 50(c) show K^+ versus y^+ and ϵ^+ versus y^+ plots respectively, where $K^+ = K/u^{*2}$ and $\epsilon^+ = \epsilon\nu/u^{*4}$ with $u^* = \sqrt{c_f/2}$. These results are consistent with those found by Patel, et al[43].

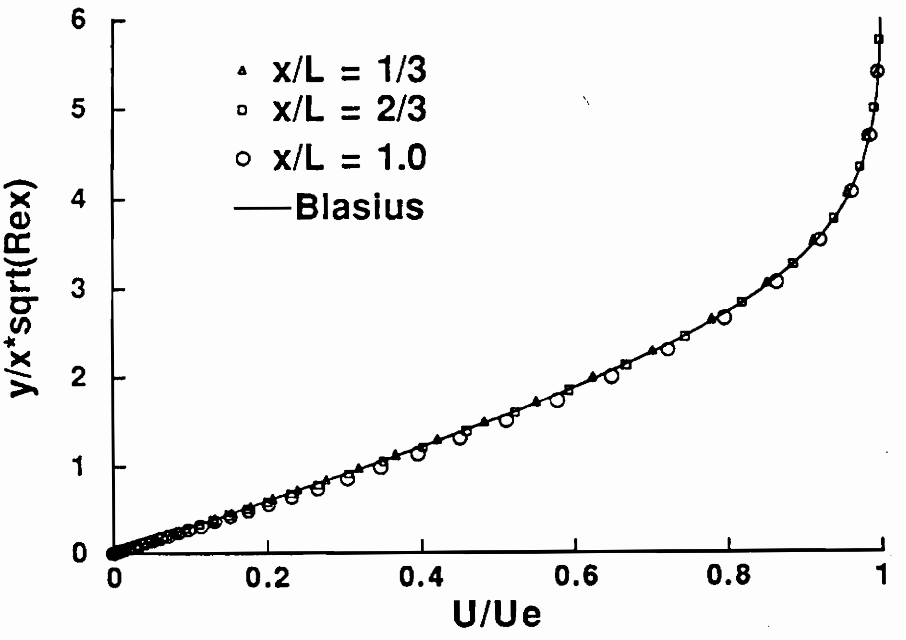
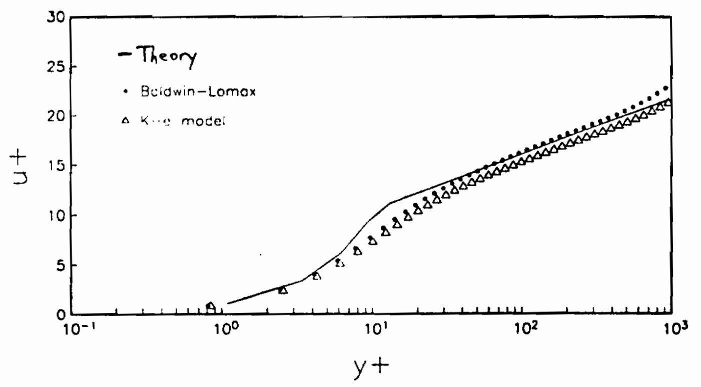
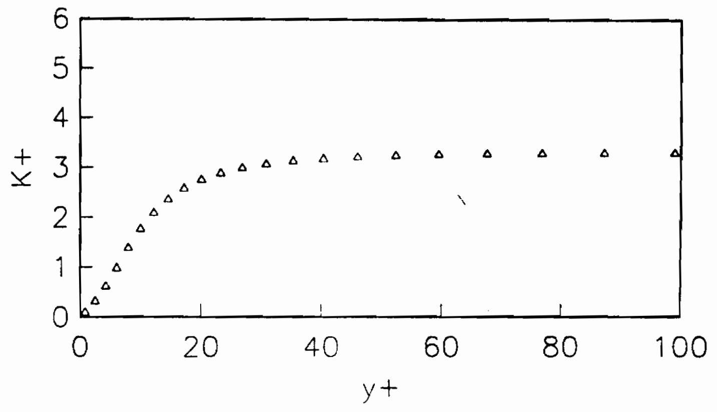


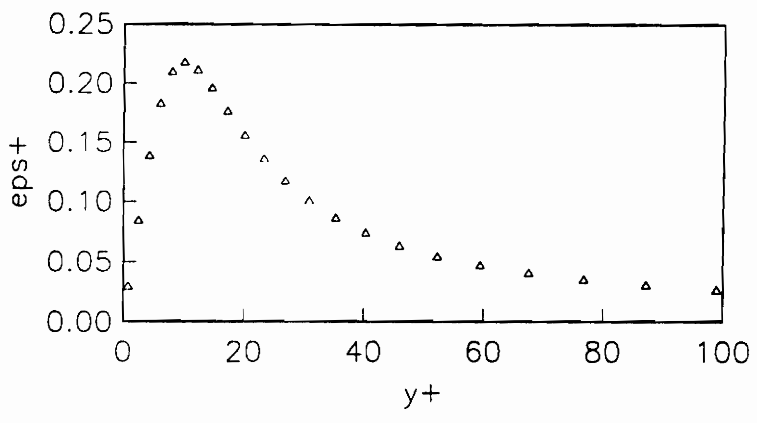
Figure 49) Similarity velocity profiles for subsonic laminar flat plate compared with Blasius[50], $M_\infty = 0.3$



(a)



(b)



(c)

Figure 50) (a) u^+ vs. y^+, (b) K^+ vs. y^+, (c) ϵ^+ vs. y^+ for subsonic turbulent flat plate, $M_\infty = 0.3$

IX. Unstructured Applications in Three Dimensions

The intention of this section is to discuss how to combine the capabilities of the two-dimensional unstructured solvers and the capabilities of the three-dimensional structured flow solver GASP into an efficient three-dimensional unstructured flow solver.

One of the problems faced with large three-dimensional problems is storage. In GASP, this problem is address by treating the third dimension or i direction differently than the first two directions. All of the variables are stored by i-planes, and the time integration is performed one i-plane at a time. The i-direction fluxes and diagonal contributions to the Jacobian are still considered as usual, but the off-diagonal i-direction contributions to the Jacobian are treated through relaxation in the form of a non-linear update of the residual. Therefore, to solve any given i-plane it is only necessary to have the information from five i-planes (to construct higher order fluxes), and to store and solve an LU decomposition on a single two-dimensional plane. In order to allow maximum flexibility, GASP allows the user to store as many i-planes as possible or as few as five.

GASP implements storage management through the use of binary C input/output. Binary C is both portable and highly efficient. Binary C can point to a location in the middle of a file and read or write only the information it needs, so it is very fast.

Implementing a similar strategy for an unstructured solver would appear quite difficult since there is no streamwise structure to the grid. One possible solution was developed by McGrory[51]. This method maintained a structured grid in the streamwise direction by requiring adjacent unstructured crossflow planes to have the same number of cells and connectivity. When it is no longer possible to maintain this criteria due to changes in streamwise geometry, a zonal interpolation of the variables is performed to pass information onto the new grid. This method can become quite cumbersome in terms of grid generation for problems which require many zonal interpolations.

As a result, an alternate proposal is given which maintains a partially structured streamwise grid which can be generated robustly with a general three-dimensional unstructured grid generator. Figure 51 shows a streamwise two-dimensional slice of a typical grid that might be generated using this technique. The streamwise domain is broken down into sections which could be thought of as i-planes (in a structured sense). The storage and time integration can then be performed (as in the case of the structured solver) one i-plane at a time. The only conceptual difference would be that instead of computing on a single two-dimensional plane at a time for a structured solver, this problem would involve computing a small three-dimensional unstructured section.

All of the time integration schemes for the two-dimensional unstructured solvers could be easily applied to these streamwise sections. This sectioning technique will allow for implementation of space marching as discussed in section 7.4 for the structured solver. All of the chemical and thermodynamic modeling routines of the structured solver were written in terms of a general vector length and without the need for logical (i.e. structured) indexing; therefore, these routines also apply directly to an unstructured solver. The viscous routines, on the other hand, require gradient calculations which depend on the mesh. These routines would need to be reformulated for an unstructured solver. The best method for computing these terms on unstructured meshes is a topic of current research.

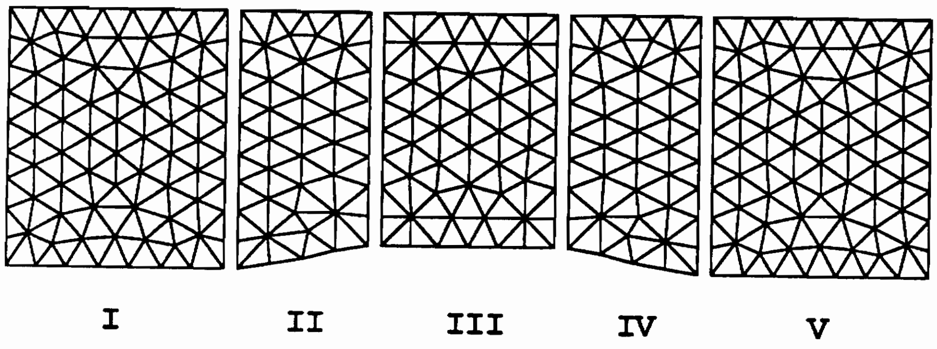


Figure 51) Grid for supersonic nozzle broken into streamwise sections

The only other major source of additional work involved with developing an unstructured flow solver is computing and storing the connectivity information. In the case of streamwise sectioning, this would involve information relating elements to other elements in the same section as is done for the two-dimensional case, and it would also require additional information relating the elements on the boundary of one section to the elements on the boundary of the adjacent section. Much of this connectivity information is computed during the grid generation process and can be stored there and passed to the flow solver.

X. Conclusions

The goal of this research is to lay a foundation for the development of a three-dimensional unstructured solver capable of obtaining a grid converged solution for any general CFD problem with automatic mesh adaptation. The main elements of the solver would include a grid generator, a flow solver, and an interactive graphics package to interface between the user, grid generator, and flow solver.

Three-dimensional unstructured grid generation for a general domain has already been achieved by several authors[10,11]. Improving the robustness and adding capabilities such as generating unstructured meshes which could be utilized by a space marching algorithm are topics of current research by McGrory[50].

A model two-dimensional interactive graphics package has been developed in this work. Many improvements upon this model need to be made to make it useful as a general interface controller. One important capability that could be added is to allow user input of variables, boundary conditions, and overall setup of the problem. A more sophisticated windowing system and increased output capabilities could also be added. Although these are conceptually simple tasks, implementing these changes is a tedious and time consuming project, and hence was left for a later time.

The main focus of this work is the development of the flow solver. The two-dimensional unstructured research relates directly to the efficiency of the flow solver while the three-dimensional research involves the development of a memory efficient generalized solver capable of computing a wide range of physical problems.

Several time integration schemes for unstructured meshes were explored. The sparse matrix solvers GMRES and CHEBYCODE showed promising results in terms of CPU efficiency especially as the mesh size increased which is critical when considering applying these algorithms in three dimensions. Runge-Kutta with residual smoothing is attractive from a code developer point of view because it is very easy to code, requires little memory, and yields reasonable convergence rates. The line Gauss-Seidel type relaxation method can be improved and would then require further testing to determine its utility. The line Gauss-Seidel method for structured meshes is certainly a useful algorithm, thus it is likely that its unstructured conterpart could yield similar performance. The use of grid adaptation was also shown to produce more accurate results in less CPU time. Grid adaptation will become even more important in reducing the number of cells needed to obtain an accurate solution in three-dimensions.

The three-dimensional structured flow solver GASP is capable of solving a wide range of physical problems. The advanced chemistry and thermodynamic modeling is needed to compute sophisticated problems in combustion and hypersonics. The extension of these modeling routines from the structured code GASP to an unstructured code is straightforward. The viscous terms, on the other hand, present more of a problem. The method for determining a length scale for an algebraic turbulence model on unstructured meshes is not clear. However, two-equation turbulence modeling only requires a new procedure for gradient calculations on unstructured meshes. This is certainly possible and is a topic of current research. The storage structure of GASP allows for the computation of large three-dimensional problems and can be applied to an unstructured solver through streamwise sectioning of the domain as discussed in chapter IX. Streamwise sectioning also makes space marching possible for an unstructured solver which is a very efficient method for computing supersonic and hypersonic problems in CFD.

At this point, many of the tools have been developed. Work on accuracy issues on unstructured meshes, improved numerical and physical models will continue to be the subject of ongoing investigations.

References

1. Cinnella, P., “Flux-Split Algorithms for Flows with Nonequilibrium Chemistry and Thermodynamics ”, Ph.D. Thesis, Dept. of Aerospace and Ocean Engineering, Virginia Polytechnic Institute and State University, 1989.
2. Grossman, B. and Cinnella, P., “The Development of Flux-Split Algorithms for Flows with Non-Equilibrium Thermodynamics and Chemical Reactions”, AIAA Paper No. 88-3595-CP, 1988.
3. Grossman, B. and Cinnella, P., “Upwind Methods for Flows with Non-Equilibrium Chemistry and Thermodynamics”, *Third International Conference on Numerical Combustion*, Antibes, 1989.
4. Grossman, B. and Cinnella, P.,“Flux-Split Algorithms for Flows with Nonequilibrium Chemistry and Vibrational Relaxation”, *Journal of Computational Physics*, Vol. 88, No. 1, 1990.
5. Barth, T.J. and Jespersen D.C., “The Design and Application of Upwind Schemes on Unstructured Meshes”, AIAA Paper 89–0336, January 1989.
6. Löhner, R. , “Improved Adaptive Refinement Strategies for Finite Element Aerodynamic Computations”, AIAA Paper 86–0499, January 1986.
7. Mavriplis, D. and Jameson, A., “Multigrid Solution of the Euler Equations on Unstructured and Adaptive Meshes”, ICASE Report 87-53, July 1987.
8. Whitaker, D.L., “Two-Dimensional Euler Computations on a Triangular Mesh Using an Upwind, Finite-Volume Scheme.”, Ph.D. Thesis, Dept. of Aerospace and Ocean Engineering, Virginia Polytechnic Institute and State University, 1988.
9. Stoufflet, B., Périaux, J., Fezoui, F., Dervieux, A., “Numerical Simulation of 3-D Hypersonic Euler Flows Around Space Vehicles using Adapted Finite-Elements”, AIAA Paper 87–0560, 1987.
10. Morgan, K., Peraire, J., Thareja, R.R., and Stewart, J.R., “An Adaptive Finite Element Scheme for the Euler and Navier-Stokes Equations”, AIAA Paper 87-1172, June 1987.
11. Merriam, M.L., “A Fast Robust Algorithm For Delaunay Triangulation”, NASA Ames Research Center, October 1989.

12. Riggins, D.W., Walters, R.W. and Pelletier, D., "The Use of Direct Solvers for Compressible Flow Computations", AIAA Paper 88–0229, January 1988.

13. Saad, Y. and Schultz, M. H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems", *Siam J. Sci. Stat. Comput.*, Vol. 7, No. 3, July 1986, pp. 856-869.

14. Ashby, S. A., "CHEBYCODE: A FORTRAN Implementation of Manteuffel's Adaptive Chebyshev Algorithm", Dept. of Computer Science, University of Illinois at Urbana-Champaign, Report No. UIUCDCS-R-85-1203, May 1985.

15. Van Leer, B.,"Flux-vector Splitting for the Euler equations", *Lecture Notes in Physics*, Vol. 170, ISBN 0-387-11948-5, Springer-Verlag, 1982.

16. Roe, P.L., "Characteristic Based Schemes for the Euler Equations", *Annual Review of Fluid Mechanics*, Vol. 18, 1986, pp.337-365.

17. Peyret, R., Taylor, T.D., *Computational Methods for Fluid Flow*, pp. 112-115, Springer-Verlag, New York, 1983.

18. Gudonov, S.K., "Finite-Difference Method for Numerical Computation of Discontinuous Solutions of the Equations of Fluid Dynamics", *Mat. Sbornik*, Vol. 47, 1959, pp. 271-306. (In Russian).

19. Steger, J. L. and Warming, R. F., "Flux Vector Splitting of the Inviscid Gasdynamic Equations with Applications to Finite-Difference Methods", *Journal of Computational Physics*, Vol. 40, 1981.

20. Van Albada, G.D., Van Leer, B. and Roberts Jr., W.W., "A Comparative Study of Computational Methods in Cosmic Gas Dynamics", *Astron. Astrophys.*, Vol. 108, 1982, pp. 76-84.

21. Concus, P. and Golub, G., "A Generalized Conjugate Gradient Method for Nonsymmetric Systems of Linear Equations", *Lecture Notes in Economic and Mathematical Systems*, Vol. 134, (R. Glowinski and J. R. Lions, Eds.), 1976.

22. Elman, H. C., "Preconditioned Conjugate Gradient Methods for Nonsymmetric Systems of Linear Equations", *Advances in Computer Methods for Partial Differential Equations-IV*, R. Vichnevetsky and R. S. Stepleman, Editors, 1981.

23. Axelsson, O., "Conjugate Gradient Type Methods for Partial Differential Equations", *Lin. Alg. Appl.*, 29 (1980), pp. 1-16.

24. Jameson, A., Schmidt, W., and Turkel, E., "Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time Stepping Schemes", AIAA Paper 81-1259,1981.

25. Turkel, E. and Van Leer, B.,"Flux-Vector Splitting and Runge-Kutta Methods for the Euler Equations", ICASE Report No. 84-27, NASA CR 172415, June 1984.

26. Mavriplis, D. and Jameson, A., "Multigrid Solution of the Two-Dimensional Euler Equations on Unstructured Triangular Meshes", AIAA Paper 87-0353, January 1987.

27. Duff, I.S, Erisman, A.M., and Reid, J.K., *Direct Methods for Sparse Matrices*, Oxford University Press, New York, 1986.

28. Wigton, L.B., Yu, N.J., and Young, D.P., "GMRES Acceleration of Computational Fluid Dynamic Codes", AIAA 7^{th} Computational Fluid Dynamics Conference, Cincinnati, Ohio, July 1985.

29. Dongarra,J.J., Moler, C.B¿, Bunch, J.R., and Stewart, G.W., *LINPACK User's Guide*,SIAM,1979.

30. Manteuffel, T. A., "An Iterative Method for Solving Nonsymmetric Linear Systems with Dynamic Estimation of Parameters", Ph.D. Thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1975.

31. Elman, H.C., Saad, Y., and Saylor, P.E., "Hybrid Chebyshev Krylov Subspace Algorithm for Solving Nonsymmetric Systems of Linear Equations", SIAM J. Sci. Stat. Comput., Vol. 7, No. 3, July 1986, pp. 840-855.

32. Manteuffel, T.A., "Adaptive Procedure for Estimation of Parameters for the Nonsymmetric Tchebychev Iteration", Numer. Math, 31 (1978), pp.187-208

33. "Numerical Methods for the Computation of Inviscid Transonic Flows with Shock Waves - A GAMM Workshop", *Notes on Numerical Fluid Mechanics*, Vol. 3.

34. Anderson, D. A., Tannehill, J. C. and Pletcher, R. H., *Computational Fluid Mechanics and Heat Transfer*, ISBN 0-89116-471-5, Hemisphere Publishing Corporation, 1984.

35. Pulliam, T. H. and Steger, J. L., "On Implicit Finite-Difference Simulations of Three Dimensional Flow", AIAA Paper No. 78-10, 1978.

36. Reu, T., "Techniques for Compressible Flow Calculations on Multi-Zone Grids", Ph.D. Thesis, Dept. of Aerospace and Ocean Engineering, Virginia Polytechnic Institute and State University, 1988.

37. Vigneron, Y.C., Rakich, J.V., and Tannehill, J.C., "Calculation of Supersonic Viscous Flow over Delta Wings and Sharp Subsonic Leading Edges", AIAA paper 78-1137, July, 1978.

38. Baldwin, B. S. and Lomax, H., "Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows", AIAA Paper No. 78-257, 1978.

39. Degani, D. and Schiff, L. B., "Computation of Supersonic Viscous Flows Around Pointed Bodies at Large Incidence", AIAA Paper No. 83-0034, 1983.ISBN 0-13-329334-3, Prentice-Hall Inc., 1984.

40. Van Driest, E.R., "On Turbulent Flow Near a Wall", *J. Aerosp. Sci.*, Vol. 23, pp. 1007-1012, 1956a.

41. Coakley, T.J., "Turbulence Modeling Methods for the Compressible Navier-Stokes Equations", AIAA 16th Fluid and Plasma Dynamics Conference, Danvers, Mass., July 1983

42. Brankovic, A. and Stowers, S.T., " Review of Low Reynolds Number Turbulence Models for Complex Internal Separated Flows", AIAA/ASME/SAE/ASEE 24th Joint Propulsion Conference, Boston, Mass., July, 1988.

43. Patel, V.C., Rodi, W., and Scheuerer, G., "Turbulence Models for Near-Wall and Low Reynolds Number Flows: A Review", AIAA Journal,Vol. 23, No. 9, pp. 1308-1319.

44. Walters, R.W. and Dwoyer, D.L., "An Efficient Iteration Strategy for the Solution of the Euler Equations", AIAA 85-1529, July 1985.

45. Townsend, J.C., Howell, D.T., Collins, I.K., and Hayes, C., "Surface Pressure Data on a Series of Analytic Forebodies at Mach Numbers from 1.70 to 4.50 and Combined Angles of Attack and Sideslip", NASA Technical Memorandum 80062, June 1979.

46. Thomas, J. L., Van Leer, B., and Walters, R. W., "Implicit Flux-Split Schemes for the Euler Equations", AIAA 18th Fluid Dynamics and Plasma Dynamics and Lasers Conference, Cincinnati, Ohio, July 1985.

47. Rogers, R.C. and Chinitz, W., "Using a Global Hydrogen-Air Combustion

Model in Turbulent Reacting Flow Calculations", AIAA Journal, Vol. 21, No. 4, 1983.

48. Drummond, J. P., Hussaini, M. J., and Zang, T. A., "Spectral Methods for Modeling Supersonic Chemically Reacting Flowfields", AIAA Journal, Vol. 24, No. 9, 1986.

49. Holden,

50. Schetz, J. A., *Foundations of Boundary Layer Theory for Momentum, Heat and Mass Transfer*, Perentice Hall, New Jersey, 1984.

51. McGrory, W.D., Walters, R.W., and Lӧner, R., "A Three-Dimensional Space Marching Algorithm for the Solution of the Euler Equations on Unstructured Grids", AIAA Paper 90-0014,January 1990. Also to appear in AIAA Journal in August 1991.

52. McGrory, W.D., "Generalized Spatial Discretization Techniques for Space Marching Algorithms", Ph.D. Thesis, Dept. of Aerospace and Ocean Engineering, Virginia Polytechnic Institute and State University, 1991.

Appendix A

A.1 High Reynolds Two-Equation Turbulence Model ($K - \epsilon$)

Here is a list of the functions for this model

$$s_1 = k\,, \qquad s_2 = \epsilon\,, \qquad \mu_t = \frac{c_\mu \rho s_1^2}{s_2}\,, \qquad k = s_1\,, \qquad \epsilon = s_2\,, \qquad \omega = \frac{s_2}{s_1}$$

$$c_\mu = 0.09\,, \qquad f_\mu = 1.0\,, \qquad c_1 = 1.0\,, \qquad c_2 = -\frac{\omega^2}{c_\mu}$$

$$c_3 = 1.0\,, \qquad c_4 = 1.44\,, \qquad c_5 = -\frac{2}{3}c_4\,, \qquad c_6 = -\left(\frac{1.92}{c_\mu}\right)\omega^3$$

The derivatives with respect to s_1 can be expressed as

$$\frac{\partial f_\mu}{\partial s_1} = 0.0\,, \qquad \frac{\partial \mu_t}{\partial s_1} = \frac{2c_\mu \rho s_1}{s_2}\,, \qquad \frac{\partial \omega}{\partial s_1} = -\frac{s_2}{s_1}$$

$$\frac{\partial k}{\partial s_1} = 1.0\,, \qquad \frac{\partial \epsilon}{\partial s_1} = 0.0\,, \qquad \frac{\partial c_1}{\partial s_1} = 0.0\,, \qquad \frac{\partial c_2}{\partial s_1} = \frac{2}{\omega}c_2\frac{\partial \omega}{\partial s_1}\,,$$

$$\frac{\partial c_3}{\partial s_1} = 0.0\,, \qquad \frac{\partial c_4}{\partial s_1} = 0.0\,, \qquad \frac{\partial c_5}{\partial s_1} = 0.0\,, \qquad \frac{\partial c_6}{\partial s_1} = \frac{3}{\omega}c_6\frac{\partial \omega}{\partial s_1}$$

The derivatives with respect to s_2 can be expressed as

$$\frac{\partial f_\mu}{\partial s_2} = 0.0\,, \qquad \frac{\partial \mu_t}{\partial s_2} = -\frac{c_\mu \rho s_1^2}{s_2^2}\,, \qquad \frac{\partial \omega}{\partial s_1} = \frac{1}{s_1}$$

$$\frac{\partial k}{\partial s_2} = 0.0\,, \qquad \frac{\partial \epsilon}{\partial s_2} = 1.0\,, \qquad \frac{\partial c_1}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_2}{\partial s_2} = \frac{2}{\omega}c_2\frac{\partial \omega}{\partial s_2}\,,$$

$$\frac{\partial c_3}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_4}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_5}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_6}{\partial s_2} = \frac{3}{\omega}c_6\frac{\partial \omega}{\partial s_2}$$

A.2 Lam-Bremhorst Two-Equation Turbulence Model ($K-\epsilon$)

Here is a list of the functions for this model

$$s_1 = k\,, \qquad s_2 = \epsilon\,, \qquad \mu_t = \frac{c_\mu f_\mu \rho s_1^2}{s_2}\,, \qquad k = s_1\,, \qquad \epsilon = s_2\,, \qquad \omega = \frac{s_2}{s_1}$$

$$c_\mu = 0.09\,, \quad f_\mu = \left(1.0 - e^{-0.0165R_y}\right)^2\left(1.0 + \frac{20.5}{R_t}\right), \qquad c_1 = 1.0\,, \quad c_2 = -\frac{\omega^2}{c_\mu f_\mu}$$

$$c_3 = 1.0\,, \quad c_4 = 1.44\left(1.0 + \left[\frac{0.05}{f_\mu}\right]^3\right), \quad c_5 = -\frac{2}{3}c_4\,, \quad c_6 = -\frac{1.92\omega^3\left(1.0 - e^{-R_t^2}\right)}{c_\mu f_\mu}$$

The derivatives with respect to s_1 can be expressed as

$$\frac{\partial R_t}{\partial s_1} = \frac{2s_1}{\nu s_2}\,, \quad \frac{\partial R_y}{\partial s_1} = \frac{\frac{1}{2}s_1^{-\frac{1}{2}}y}{\nu}\,, \qquad \frac{\partial \mu_t}{\partial s_1} = \frac{2c_\mu f_\mu \rho s_1}{s_2} + \frac{c_\mu \rho s_1^2}{s_2}\frac{\partial f_\mu}{\partial s_1}\,, \quad \frac{\partial \omega}{\partial s_1} = -\frac{s_2}{s_1}$$

$$\frac{\partial f_\mu}{\partial s_1} = \frac{-.033}{e^{.0165}}\left(1 - e^{-.0165R_y}\right)\left(1.0 + \frac{20.5}{R_t}\right)\frac{\partial R_y}{\partial s_1} + \left(1 - e^{-.0165R_y}\right)\left(-\frac{20.5}{R_t^2}\right)\frac{\partial R_t}{\partial s_1}$$

$$\frac{\partial k}{\partial s_1} = 1.0\,, \qquad \frac{\partial \epsilon}{\partial s_1} = 0.0\,, \qquad \frac{\partial c_1}{\partial s_1} = 0.0\,, \qquad \frac{\partial c_2}{\partial s_1} = \frac{2}{\omega}c_2\frac{\partial \omega}{\partial s_1} - \frac{1}{f_\mu}c_2\frac{\partial f_\mu}{\partial s_1}$$

$$\frac{\partial c_3}{\partial s_1} = 0.0\,, \qquad \frac{\partial c_4}{\partial s_1} = -1.44\left(\frac{0.075}{f_\mu^2}\right)^2\frac{\partial f_\mu}{\partial s_1}\,, \qquad \frac{\partial c_5}{\partial s_1} = -\frac{2}{3}\frac{\partial c_4}{\partial s_1}$$

$$\frac{\partial c_6}{\partial s_1} = \frac{3}{\omega}c_6\frac{\partial \omega}{\partial s_1} - \frac{1}{f_\mu}c_6\frac{\partial f_\mu}{\partial s_1} - e^{-R_t^2}\left(\frac{3.84\omega^3 R_t}{c_\mu f_\mu}\right)\frac{\partial R_t}{\partial s_1}$$

The derivatives with respect to s_2 can be expressed as

$$\frac{\partial R_t}{\partial s_2} = -\frac{s_1^2}{\nu s_2^2}\,, \quad \frac{\partial R_y}{\partial s_2} = 0.0\,, \quad \frac{\partial \mu_t}{\partial s_2} = -\frac{c_\mu f_\mu \rho s_1^2}{s_2^2} + \frac{c_\mu \rho s_1^2}{s_2}\frac{\partial f_\mu}{\partial s_2}\,, \quad \frac{\partial \omega}{\partial s_2} = \frac{1.0}{s_1}$$

$$\frac{\partial f_\mu}{\partial s_2} = \left(1.0 - e^{-0.0165R_y}\right)\left(-\frac{20.5}{R_t^2}\right)\frac{\partial R_t}{\partial s_2}\,, \qquad \frac{\partial k}{\partial s_2} = 0.0$$

$$\frac{\partial \epsilon}{\partial s_2} = 1.0\,, \qquad \frac{\partial c_1}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_2}{\partial s_2} = \frac{2}{\omega}c_2\frac{\partial \omega}{\partial s_2} - \frac{1}{f_\mu}c_2\frac{\partial f_\mu}{\partial s_2}$$

$$\frac{\partial c_3}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_4}{\partial s_2} = -1.44\left(\frac{0.075}{f_\mu^2}\right)^2\frac{\partial f_\mu}{\partial s_2}\,, \qquad \frac{\partial c_5}{\partial s_2} = -\frac{2}{3}\frac{\partial c_4}{\partial s_2}$$

$$\frac{\partial c_6}{\partial s_2} = \frac{3}{\omega}c_6\frac{\partial \omega}{\partial s_2} - \frac{1}{f_\mu}c_6\frac{\partial f_\mu}{\partial s_2} - e^{-R_t^2}\left(\frac{3.84\omega^3 R_t}{c_\mu f_\mu}\right)\frac{\partial R_t}{\partial s_2}$$

A.3 Chien Two-Equation Turbulence Model ($K - \epsilon$)

Here is a list of the functions for this model

$$s_1 = k\,, \qquad s_2 = \epsilon\,, \qquad \mu_t = \frac{c_\mu f_\mu \rho s_1^2}{s_2}\,, \qquad k = s_1\,, \qquad \epsilon = s_2\,, \qquad \omega = \frac{s_2}{s_1}$$

$$c_\mu = 0.09\,, \qquad f_\mu = 1.0 - e^{-0.0115 y^+}\,, \qquad c_1 = 1.0\,, \qquad c_2 = -\frac{1.0}{c_\mu f_\mu}\left(\omega^2 + \frac{2\nu\omega}{y^2}\right)$$

$$c_3 = 1.\,, \quad c_4 = 1.35\,, \quad c_5 = -\frac{2}{3}c_4\,, \quad c_6 = -\frac{1.0}{c_\mu f_\mu}\left[1.8\omega^3\left(1 - \frac{2}{9}e^{\frac{-R_t^2}{36}}\right) + \frac{2\nu\omega^2 e^{\frac{-y^+}{2}}}{y^2}\right]$$

The derivatives with respect to s_1 can be expressed as

$$\frac{\partial R_t}{\partial s_1} = \frac{2s_1}{\nu s_2}\,, \qquad \frac{\partial R_y}{\partial s_1} = \frac{\frac{1}{2}s_1^{-\frac{1}{2}}y}{\nu}\,, \qquad \frac{\partial \mu_t}{\partial s_1} = \frac{2c_\mu f_\mu \rho s_1}{s_2}\,, \qquad \frac{\partial \omega}{\partial s_1} = -\frac{s_2}{s_1}\,,$$

$$\frac{\partial f_\mu}{\partial s_1} = 0.0\,, \qquad \frac{\partial k}{\partial s_1} = 1.0\,, \qquad \frac{\partial \epsilon}{\partial s_1} = 0.0\,, \qquad \frac{\partial c_1}{\partial s_1} = 0.0\,,$$

$$\frac{\partial c_2}{\partial s_1} = -\frac{1.0}{c_\mu f_\mu}\left(2\omega + \frac{2\nu}{y^2}\right)\frac{\partial \omega}{\partial s_1}\,, \qquad \frac{\partial c_3}{\partial s_1} = 0.0\,, \quad \frac{\partial c_4}{\partial s_1} = 0.0\,, \quad \frac{\partial c_5}{\partial s_1} = -\frac{2}{3}\frac{\partial c_4}{\partial s_1}$$

$$\frac{\partial c_6}{\partial s_1} = -\frac{1.0}{c_\mu f_\mu}\left[5.4\omega^2\left(1.0 - \frac{2}{9}e^{\frac{-R_t^2}{36}}\right) + \frac{4\nu\omega e^{\frac{-y^+}{2}}}{y^2}\frac{\partial \omega}{\partial s_1}\right] - \left(\frac{7.2\omega^3 R_t}{324 c_\mu f_\mu}\right)e^{\frac{-R_t^2}{36}}\frac{\partial R_t}{\partial s_1}$$

The derivatives with respect to s_2 can be expressed as

$$\frac{\partial R_t}{\partial s_2} = -\frac{s_1^2}{\nu s_2^2}\,, \qquad \frac{\partial \mu_t}{\partial s_2} = -\frac{c_\mu f_\mu \rho s_1^2}{s_2^2}\,, \qquad \frac{\partial \omega}{\partial s_2} = \frac{1.0}{s_1}\,,$$

$$\frac{\partial f_\mu}{\partial s_2} = 0.0\,, \qquad \frac{\partial k}{\partial s_2} = 0.0\,, \qquad \frac{\partial \epsilon}{\partial s_2} = 1.0\,, \qquad \frac{\partial c_1}{\partial s_2} = 0.0\,,$$

$$\frac{\partial c_2}{\partial s_2} = -\frac{1.0}{c_\mu f_\mu}\left(2\omega + \frac{2\nu}{y^2}\right)\frac{\partial \omega}{\partial s_2}\,, \qquad \frac{\partial c_3}{\partial s_2} = 0.0\,, \quad \frac{\partial c_4}{\partial s_2} = 0.0\,, \quad \frac{\partial c_5}{\partial s_2} = -\frac{2}{3}\frac{\partial c_4}{\partial s_2}$$

$$\frac{\partial c_6}{\partial s_2} = -\frac{1.0}{c_\mu f_\mu}\left[5.4\omega^2\left(1.0 - \frac{2}{9}e^{\frac{-R_t^2}{36}}\right) + \frac{4\nu\omega e^{\frac{-y^+}{2}}}{y^2}\frac{\partial \omega}{\partial s_2}\right] - \left(\frac{7.2\omega^3 R_t}{324 c_\mu f_\mu}\right)e^{\frac{-R_t^2}{36}}\frac{\partial R_t}{\partial s_2}$$

A.4 Nagano-Hisida Two-Equation Turbulence Model ($K - \epsilon$)

Here is a list of the functions for this model

$$s_1 = k\,,\quad s_2 = \epsilon,\quad \mu_t = \frac{c_\mu f_\mu \rho s_1^2}{s_2}\,,\quad k = s_1\,,\quad \epsilon = s_2\,,\quad \omega = \frac{s_2}{s_1}\,,\quad c_\mu = 0.09$$

$$f_\mu = \left(1.0 - e^{\frac{-R_t}{26.5}}\right)^2\,,\quad c_1 = 1.0\,,\quad c_2 = \frac{1}{c_\mu f_\mu}\left[\frac{\nu\omega}{2k^2}\left(\frac{\partial s_1}{\partial y}\right)^2 - \omega^2\right]\,,\quad c_3 = 1.0$$

$$c_4 = 1.45\,,\quad c_5 = -\frac{2}{3}c_4\,,\quad c_6 = \nu\left(1.0 - f_\mu\right)\left(\frac{\partial^2 u}{\partial y^2}\right)^2 - 1.9\omega^3\left(1.0 - 0.3e^{-R_t^2}\right)$$

The derivatives with respect to s_1 can be expressed as

$$\frac{\partial R_t}{\partial s_1} = \frac{2s_1}{\nu s_2}\,,\quad \frac{\partial \mu_t}{\partial s_1} = \frac{2c_\mu f_\mu \rho s_1}{s_2} + \frac{c_\mu \rho s_1^2}{s_2}\frac{\partial f_\mu}{\partial s_1}\,,\quad \frac{\partial \omega}{\partial s_1} = -\frac{s_2}{s_1}$$

$$\frac{\partial f_\mu}{\partial s_1} = -\frac{2.0}{26.5}\left(1.0 - e^{\frac{-R_t}{26.5}}\right)e^{\frac{-R_t}{26.5}}\frac{\partial R_t}{\partial s_1}\,,\quad \frac{\partial k}{\partial s_1} = 1.0\,,\quad \frac{\partial \epsilon}{\partial s_1} = 0.0\,,$$

$$\frac{\partial c_1}{\partial s_1} = 0.\,,\quad \frac{\partial c_2}{\partial s_1} = -\frac{1}{f_\mu}c_2\frac{\partial f_\mu}{\partial s_1} - \frac{1}{c_\mu f_\mu}\left[\frac{\nu\omega}{4s_1^3}\left(\frac{\partial s_1}{\partial y}\right)^2\right] - \frac{1}{c_\mu f_\mu}\left[2\omega - \frac{\nu}{2s_1^2}\left(\frac{\partial s_1}{\partial y}\right)^2\frac{\partial \omega}{\partial s_1}\right]$$

$$\frac{\partial c_3}{\partial s_1} = 0.0\,,\quad \frac{\partial c_4}{\partial s_1} = 0.0\,,\quad \frac{\partial c_5}{\partial s_1} = 0.0$$

$$\frac{\partial c_6}{\partial s_1} = -5.7\omega^2\left(1.0 - 0.3^{-R-t^2}\right) - \nu\left(\frac{\partial^2 u}{\partial y^2}\right)\frac{\partial f_\mu}{\partial s_1} - 1.14R_t\omega^3 e^{-R_t^2}\frac{\partial R_t}{\partial s_1}$$

The derivatives with respect to s_2 can be expressed as

$$\frac{\partial R_t}{\partial s_2} = -\frac{s_1^2}{\nu s_2^2}\,,\quad \frac{\partial R_y}{\partial s_2} = 0.0\,,\quad \frac{\partial \mu_t}{\partial s_2} = -\frac{c_\mu f_\mu \rho s_1^2}{s_2^2} + \frac{c_\mu \rho s_1^2}{s_2}\frac{\partial f_\mu}{\partial s_2}\,,\quad \frac{\partial \omega}{\partial s_2} = \frac{1.0}{s_1}$$

$$\frac{\partial f_\mu}{\partial s_2} = -\frac{2.0}{26.5}\left(1.0 - e^{\frac{-R_t}{26.5}}\right)e^{\frac{-R_t}{26.5}}\frac{\partial R_t}{\partial s_2}\,,\quad \frac{\partial k}{\partial s_2} = 0.0\,,\quad \frac{\partial \epsilon}{\partial s_2} = 1.0\,,$$

$$\frac{\partial c_1}{\partial s_2} = 0.0\,,\quad \frac{\partial c_2}{\partial s_2} = -\frac{1}{f_\mu}c_2\frac{\partial f_\mu}{\partial s_2} - \frac{1}{c_\mu f_\mu}\left[2\omega - \frac{\nu}{2s_1^2}\left(\frac{\partial s_1}{\partial y}\right)^2\frac{\partial \omega}{\partial s_2}\right]$$

$$\frac{\partial c_3}{\partial s_2} = 0.0\,,\quad \frac{\partial c_4}{\partial s_2} = 0.0\,,\quad \frac{\partial c_5}{\partial s_2} = 0.0$$

$$\frac{\partial c_6}{\partial s_2} = -5.7\omega^2\left(1.0 - 0.3^{-R-t^2}\right) - \nu\left(\frac{\partial^2 u}{\partial y^2}\right)\frac{\partial f_\mu}{\partial s_1} - 1.14R_t\omega^3 e^{-R_t^2}\frac{\partial R_t}{\partial s_2}$$

A.5 Jones-Launder Two-Equation Turbulence Model ($K - \epsilon$)

Here is a list of the functions for this model

$$s_1 = k\,, \qquad s_2 = \epsilon\,, \qquad \mu_t = \frac{c_\mu f_\mu \rho s_1^2}{s_2}\,, \qquad k = s_1\,, \qquad \epsilon = s_2\,, \qquad \omega = \frac{s_2}{s_1}$$

$$c_\mu = 0.09\,, \qquad f_\mu = e^{\frac{-3.4}{(1.0+0.02R_t)^2}}\,, \qquad c_1 = 1.0\,, \qquad c_2 = \frac{1}{c_\mu f_\mu}\left(\frac{2\nu\omega}{k}\frac{\partial s_1}{\partial y} - \omega^2\right)$$

$$c_3 = 1.0\,, \quad c_4 = 1.45\,, \quad c_5 = -\frac{2}{3}c_4\,, \quad c_6 = 2\nu\left(\frac{\partial^2 u}{\partial y^2}\right)^2 - \frac{1.9\omega^3}{c_\mu f_\mu}\left(1.0 - \frac{2}{9}e^{\frac{-R_t^2}{36}}\right)$$

The derivatives with respect to s_1 can be expressed as

$$\frac{\partial R_t}{\partial s_1} = \frac{2s_1}{\nu s_2}\,, \qquad \frac{\partial \mu_t}{\partial s_1} = \frac{2c_\mu f_\mu \rho s_1}{s_2} + \frac{c_\mu \rho s_1^2}{s_2}\frac{\partial f_\mu}{\partial s_1}\,, \qquad \frac{\partial \omega}{\partial s_1} = -\frac{s_2}{s_1}$$

$$\frac{\partial f_\mu}{\partial s_1} = 0.136e^{\frac{-3.4}{(1.0+0.02R_t)^2}}(1.0+0.02R_t)^{-3}\frac{\partial R_t}{\partial s_1}\,, \qquad \frac{\partial k}{\partial s_1} = 1.0\,, \qquad \frac{\partial \epsilon}{\partial s_1} = 0.0\,,$$

$$\frac{\partial c_1}{\partial s_1} = 0.0\,, \quad \frac{\partial c_2}{\partial s_1} = -\frac{1}{f_\mu}c_2\frac{\partial f_\mu}{\partial s_1} - \frac{1}{c_\mu f_\mu}\left(\frac{2\nu\omega}{s_1^2}\frac{\partial s_1}{\partial y}\right) - \frac{1}{c_\mu f_\mu}\left(2\omega - \frac{2\nu}{s_1}\frac{\partial s_1}{\partial y}\frac{\partial \omega}{\partial s_1}\right)$$

$$\frac{\partial c_3}{\partial s_1} = 0.0\,, \qquad \frac{\partial c_4}{\partial s_1} = 0.0\,, \qquad \frac{\partial c_5}{\partial s_1} = 0.0$$

$$\frac{\partial c_6}{\partial s_1} = -\frac{5.76\omega^2}{c_\mu f_\mu}\left(1.0 - \frac{2}{9}e^{\frac{-R_t^2}{36}}\right)\frac{\partial \omega}{\partial s_1} - \left(\frac{7.86\omega^3 R_t}{324 c_\mu f_\mu}\right)e^{\frac{-R_t^2}{36}}\frac{\partial R_t}{\partial s_1}$$

The derivatives with respect to s_2 can be expressed as

$$\frac{\partial R_t}{\partial s_2} = -\frac{s_1^2}{\nu s_2^2}\,, \quad \frac{\partial R_y}{\partial s_2} = 0.0\,, \quad \frac{\partial \mu_t}{\partial s_2} = -\frac{c_\mu f_\mu \rho s_1^2}{s_2^2} + \frac{c_\mu \rho s_1^2}{s_2}\frac{\partial f_\mu}{\partial s_2}\,, \qquad \frac{\partial \omega}{\partial s_2} = \frac{1.0}{s_1}$$

$$\frac{\partial f_\mu}{\partial s_2} = 0.136e^{\frac{-3.4}{(1.0+0.02R_t)^2}}(1.0+0.02R_t)^{-3}\frac{\partial R_t}{\partial s_2}\,, \qquad \frac{\partial k}{\partial s_2} = 0.0\,, \qquad \frac{\partial \epsilon}{\partial s_2} = 1.0\,,$$

$$\frac{\partial c_1}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_2}{\partial s_2} = -\frac{1}{f_\mu}c_2\frac{\partial f_\mu}{\partial s_2} - \frac{1}{c_\mu f_\mu}\left(2\omega - \frac{2\nu}{s_1}\frac{\partial s_1}{\partial y}\frac{\partial \omega}{\partial s_2}\right)$$

$$\frac{\partial c_3}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_4}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_5}{\partial s_2} = 0.0$$

$$\frac{\partial c_6}{\partial s_2} = -\frac{5.76\omega^2}{c_\mu f_\mu}\left(1.0 - \frac{2}{9}e^{\frac{-R_t^2}{36}}\right)\frac{\partial \omega}{\partial s_2} - \left(\frac{7.86\omega^3 R_t}{324 c_\mu f_\mu}\right)e^{\frac{-R_t^2}{36}}\frac{\partial R_t}{\partial s_2}$$

A.6 Wilcox - Rubesin Two-Equation Turbulence Model ($K-\omega^2$)

Here is a list of the functions for this model

$$s_1 = k\,, \qquad s_2 = \omega^2\,, \qquad \mu_t = \frac{f_\mu \rho s_1}{s_2^{\frac{1}{2}}}\,, \qquad k = s_1\,, \qquad \epsilon = c\mu s_1 s_2^{\frac{1}{2}}\,, \qquad \omega = s_2^{\frac{1}{2}}$$

$$c_\mu = 0.09\,, \qquad f_\mu = 1.0{-}0.99174e^{-R_t}\,, \qquad c_1 = 1.0\,, \qquad c_2 = -\frac{c_\mu \omega^2}{f_\mu} c_3 = 1.0\,,$$

$$c_4 = \frac{.9\omega^2(1-.99174e^{-\frac{R_t}{2}})}{f_\mu s_1}\,, \quad c_5 = -0.6\,, \quad c_6 = -\left(.15 + \left(\frac{\partial(\frac{q}{\omega})}{\partial y}\right)^2\right)\left(\frac{\omega^4}{f_\mu s_1}\right)$$

The derivatives with respect to s_1 can be expressed as

$$\frac{\partial R_t}{\partial s_1} = \frac{1.0}{c_\mu \nu s_2^{\frac{1}{2}}}\,, \qquad \frac{\partial \mu_t}{\partial s_1} = \frac{f_\mu \rho}{s_2^{\frac{1}{2}}} + \frac{\rho s_1}{s_2^{\frac{1}{2}}}\frac{\partial f_\mu}{\partial s_1}\,, \qquad \frac{\partial \omega}{\partial s_1} = 0.0$$

$$\frac{\partial f_\mu}{\partial s_1} = 0.99174e^{-R_t}\frac{\partial R_t}{\partial s_1}\,, \qquad \frac{\partial k}{\partial s_1} = 1.0\,, \qquad \frac{\partial \epsilon}{\partial s_1} = c_\mu s_2^{\frac{1}{2}}\,, \qquad \frac{\partial c_1}{\partial s_1} = 0.0$$

$$\frac{\partial c_2}{\partial s_1} = -\frac{1}{f_\mu}c_2\frac{\partial f_\mu}{\partial s_1}\,, \quad \frac{\partial c_3}{\partial s_1} = 0.\,, \quad \frac{\partial c_4}{\partial s_1} = \frac{c_4}{s_1} - \frac{1}{f_\mu}c_4\frac{\partial f_\mu}{\partial s_1} + \frac{.45\omega^2(.99174e^{-\frac{R_t}{2}})}{f_\mu s_1}\frac{\partial R_t}{\partial s_1}$$

$$\frac{\partial c_5}{\partial s_1} = 0.0\,, \qquad \frac{\partial c_6}{\partial s_1} = -\frac{1}{f_\mu}c_6\frac{\partial f_\mu}{\partial s_1} - \frac{c_6}{s_1} + \left(\frac{(\frac{\partial s_1}{\partial y})^2}{4s_1^2 s_2} - \frac{(\frac{\partial s_2}{\partial y})^2}{4s_2^3}\right)\left(\frac{\omega^4}{f_\mu s_1}\right)$$

The derivatives with respect to s_2 can be expressed as

$$\frac{\partial R_t}{\partial s_2} = -\frac{\frac{1}{2}s_1}{c_\mu \nu s_2^{\frac{3}{2}}}\,, \qquad \frac{\partial \mu_t}{\partial s_2} = -\frac{f_\mu \rho s_1}{2s_2^{\frac{3}{2}}} + \frac{\rho s_1}{s_2^{\frac{1}{2}}}\frac{\partial f_\mu}{\partial s_2}$$

$$\frac{\partial \omega}{\partial s_2} = \frac{1}{2}s_2^{-\frac{1}{2}}\,, \quad \frac{\partial f_\mu}{\partial s_2} = 0.99174e^{-R_t}\frac{\partial R_t}{\partial s_2}\,, \quad \frac{\partial k}{\partial s_2} = 0.0\,, \quad \frac{\partial \epsilon}{\partial s_2} = \frac{1}{2}c_\mu s_1 s_2^{-\frac{1}{2}}$$

$$\frac{\partial c_1}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_2}{\partial s_2} = -\frac{1}{f_\mu}c_2\frac{\partial f_\mu}{\partial s_2} + \frac{2}{\omega}c_2\frac{\partial \omega}{\partial s_2}\,, \qquad \frac{\partial c_3}{\partial s_2} = 0.0\,,$$

$$\frac{\partial c_4}{\partial s_2} = \frac{2}{\omega}c_4\frac{\partial \omega}{\partial s_2} - \frac{1}{f_\mu}c_4\frac{\partial f_\mu}{\partial s_2} + \frac{0.45\omega^2(0.99174e^{-\frac{R_t}{2}})}{f_\mu s_1}\frac{\partial R_t}{\partial s_2}\,, \qquad \frac{\partial c_5}{\partial s_2} = 0.0$$

$$\frac{\partial c_6}{\partial s_2} = \frac{4}{\omega}c_6\frac{\partial \omega}{\partial s_2} - \frac{1}{f_\mu}c_6\frac{\partial f_\mu}{\partial s_2} + \left(\frac{(\frac{\partial s_1}{\partial y})^2}{4s_1 s_2^2} - \frac{(\frac{\partial s_1}{\partial y})(\frac{\partial s_2}{\partial y})}{s_2^3} + \frac{3s_1(\frac{\partial s_2}{\partial y})^2}{4s_2^4}\right)\left(\frac{\omega^4}{f_\mu s_1}\right)$$

A.7 Coakley Two-Equation Turbulence Model ($q - \omega$)

Here is a list of the functions for this model

$$s_1 = q\,, \qquad s_2 = \omega\,, \qquad \mu_t = \frac{c_\mu f_\mu \rho s_1^2}{s_2}\,, \qquad k = s_1^2\,, \qquad \epsilon = s_2 s_1^2\,, \qquad \omega = s_2$$

$$c_\mu = 0.09\,, \qquad f_\mu = 1.0 - e^{-0.0065 R_y}\,, \qquad c_1 = \frac{1.0}{2s_1}\,, \qquad c_2 = -\frac{\omega^2}{c_\mu f_\mu}$$

$$c_3 = \frac{1.0}{s_1^2}\,, \quad c_4 = \frac{(.405 f_\mu + .045)s_1}{f_\mu}\,, \quad c_5 = -\frac{2}{3}(.405 f_\mu + .045)\,, \quad c_6 = -\frac{.92\omega^3}{c_\mu f_\mu}$$

The derivatives with respect to s_1 can be expressed as

$$\frac{\partial R_y}{\partial s_1} = \frac{y}{\nu}\,, \qquad \frac{\partial \mu_t}{\partial s_1} = \frac{2c_\mu f_\mu \rho s_1}{s_2} + \frac{c_\mu \rho s_1^2}{s_2}\frac{\partial f_\mu}{\partial s_1}\,, \qquad \frac{\partial \omega}{\partial s_1} = 0.0$$

$$\frac{\partial f_\mu}{\partial s_1} = -0.0065 e^{-.0065 R_y}\frac{\partial R_y}{\partial s_1}\,, \qquad \frac{\partial k}{\partial s_1} = 2s_1\,, \qquad \frac{\partial \epsilon}{\partial s_1} = 2s_2 s_1\,,$$

$$\frac{\partial c_1}{\partial s_1} = -\frac{1.0}{s_1^2}\,, \qquad \frac{\partial c_2}{\partial s_1} = -\frac{1}{f_\mu}c_2\frac{\partial f_\mu}{\partial s_1}\,, \qquad \frac{\partial c_3}{\partial s_1} = -\frac{1.0}{2s_1^3}\,,$$

$$\frac{\partial c_4}{\partial s_1} = \frac{c_4}{s_1} - \left(\frac{0.045}{f_\mu^2}\right)\frac{\partial f_\mu}{\partial s_1}\,, \qquad \frac{\partial c_5}{\partial s_1} = -\frac{0.81}{3}\frac{\partial c_4}{\partial s_1}\,, \qquad \frac{\partial c_6}{\partial s_1} = -\frac{1}{f_\mu}c_6\frac{\partial f_\mu}{\partial s_1}$$

The derivatives with respect to s_2 can be expressed as

$$\frac{\partial R_y}{\partial s_2} = 0.0\,, \qquad \frac{\partial \mu_t}{\partial s_2} = -\frac{c_\mu f_\mu \rho s_1^2}{s_2^2}\,, \qquad \frac{\partial \omega}{\partial s_2} = 1.0\,, \qquad \frac{\partial f_\mu}{\partial s_2} = 0.0$$

$$\frac{\partial k}{\partial s_2} = 0.0\,, \qquad \frac{\partial \epsilon}{\partial s_2} = s_1^2\,, \qquad \frac{\partial c_1}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_2}{\partial s_2} = \frac{2}{\omega}c_2\frac{\partial \omega}{\partial s_2}$$

$$\frac{\partial c_3}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_4}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_5}{\partial s_2} = 0.0\,, \qquad \frac{\partial c_6}{\partial s_2} = \frac{3}{\omega}c_6\frac{\partial \omega}{\partial s_2}$$

Appendix B

B.1 Generalized Formulation

The two-equation model of section 6.3 can be written in generalized coordinates as

$$\begin{aligned}\frac{1}{J}\frac{\partial}{\partial t}(\rho s_i) &+ \frac{\partial}{\partial \xi}\frac{|\nabla\xi|}{J}(\rho\bar{u}s_i)^{n+1} + \frac{\partial}{\partial \eta}\frac{|\nabla\eta|}{J}(\rho\bar{v}s_i)^{n+1} + \frac{\partial}{\partial \zeta}\frac{|\nabla\zeta|}{J}(\rho\bar{w}s_i)^{n+1} \\ &- \sum_{m=1}^{3}\frac{\partial}{\partial k}\frac{|\nabla k|}{J}\left[\left(\mu + \frac{\mu_t}{Pr_i}\right)\left[|\nabla\xi|(\nabla\hat{k}\cdot\nabla\hat{\xi})\frac{\partial s_i}{\partial \xi}\right.\right. \\ &\left.\left.+ |\nabla\eta|(\nabla\hat{k}\cdot\nabla\hat{\eta})\frac{\partial s_i}{\partial \eta} + |\nabla\zeta|(\nabla\hat{k}\cdot\nabla\hat{\zeta})\frac{\partial s_i}{\partial \zeta}\right]\right]^{n+1} \\ &= \frac{1}{J}\phi_i^{n+1}\end{aligned} \qquad (B.1.1)$$

with

$$m = 1 \quad k = \xi \quad m = 2 \quad k = \eta \quad m = 3 \quad k = \zeta$$

which can be converted to vector notation as

$$\frac{1}{J}\frac{\partial \hat{S}}{\partial t} = -[\frac{\partial \hat{F}}{\partial \xi} + \frac{\partial \hat{G}}{\partial \eta} + \frac{\partial \hat{H}}{\partial \zeta}]^{n+1} + [\frac{\partial \hat{F}_v}{\partial \xi} + \frac{\partial \hat{G}_v}{\partial \eta} + \frac{\partial \hat{H}_v}{\partial \zeta}]^{n+1} + \frac{1}{J}\phi_i^{n+1} = R^{n+1} \qquad (B.1.2)$$

This equation can be linearized in time to give

$$[\frac{I}{J\Delta t} + \frac{\partial \hat{A}}{\partial \xi} + \frac{\partial \hat{B}}{\partial \eta} + \frac{\partial \hat{C}}{\partial \zeta} - \frac{\partial \hat{A}_v}{\partial \xi} - \frac{\partial \hat{B}_v}{\partial \eta} - \frac{\partial \hat{C}_v}{\partial \zeta} - \frac{\partial \Phi}{\partial \hat{S}}]\Delta\hat{S} = R^n \qquad (B.1.3)$$

where

$$\hat{S} = \begin{bmatrix}\rho s_1 \\ \rho s_2\end{bmatrix} \quad \text{and} \quad \Phi = \begin{bmatrix}\phi_1 \\ \phi_2\end{bmatrix}$$

$$\hat{A} = \frac{\partial \hat{F}}{\partial \hat{S}} \quad \hat{B} = \frac{\partial \hat{G}}{\partial \hat{S}} \quad \hat{C} = \frac{\partial \hat{H}}{\partial \hat{S}}$$

$$\hat{A}_v = \frac{\partial \hat{F}_v}{\partial \hat{S}} \quad \hat{B}_v = \frac{\partial \hat{G}_v}{\partial \hat{S}} \quad \hat{C}_v = \frac{\partial \hat{H}_v}{\partial \hat{S}}$$

Equation (B.1.3) is solved using the approximate factorization time integration scheme discussed in section 7.2.2. The AF scheme was coded for a variable number of unknowns so that it can be applied separately for the Navier-Stokes equations of section 6.1 with $N+M+4$ unknowns and here where there are only two unknowns.

The source term Φ has two quantities, the divergence of the velocity D and the strain rate invariant L, which need to be converted to generalized coordinates to complete the transformation. They can be written as

$$\frac{1}{J}D_m = \frac{|\nabla k|}{J}\left[\nabla\hat{k}\cdot\frac{\partial\vec{u}}{\partial k}\right] \tag{B.1.4}$$

$$\frac{1}{J}L_m = \frac{|\nabla k|}{J}\begin{bmatrix} |\nabla\xi|\left[\left(\nabla\hat{\xi}\cdot\nabla\hat{k}\right)\left(\frac{\partial\vec{u}}{\partial\xi}\cdot\frac{\partial\vec{u}}{\partial k}\right) + \frac{1}{3}\left(\nabla\hat{k}\cdot\frac{\partial\vec{u}}{\partial\xi}\right)\left(\nabla\hat{\xi}\cdot\frac{\partial\vec{u}}{\partial k}\right)\right] + \\ |\nabla\eta|\left[\left(\nabla\hat{\eta}\cdot\nabla\hat{k}\right)\left(\frac{\partial\vec{u}}{\partial\eta}\cdot\frac{\partial\vec{u}}{\partial k}\right) + \frac{1}{3}\left(\nabla\hat{k}\cdot\frac{\partial\vec{u}}{\partial\eta}\right)\left(\nabla\hat{\eta}\cdot\frac{\partial\vec{u}}{\partial k}\right)\right] + \\ |\nabla\zeta|\left[\left(\nabla\hat{\zeta}\cdot\nabla\hat{k}\right)\left(\frac{\partial\vec{u}}{\partial\zeta}\cdot\frac{\partial\vec{u}}{\partial k}\right) + \frac{1}{3}\left(\nabla\hat{k}\cdot\frac{\partial\vec{u}}{\partial\zeta}\right)\left(\nabla\hat{\zeta}\cdot\frac{\partial\vec{u}}{\partial k}\right)\right] \end{bmatrix} \tag{B.1.5}$$

Both of these quantities are coded so that terms which are not relevant to the given approximation are not computed. A good example of this is the Thin-Layer approximation which only considers viscous contributions in the direction normal to a solid surface.

Appendix C

C.1 Upwind First Derivatives

The two-equation turbulence model can be represented in matrix form as

$$\hat{S}_t + A\hat{S}_\xi + B\hat{S}_\eta + C\hat{S}_\zeta + \cdots = \Phi \tag{C.1.1}$$

where

$$\hat{S} = \begin{bmatrix} \rho s_1 \\ \rho s_2 \end{bmatrix} \quad A = \begin{bmatrix} \bar{u} & 0 \\ 0 & \bar{u} \end{bmatrix} \quad B = \begin{bmatrix} \bar{v} & 0 \\ 0 & \bar{v} \end{bmatrix} \quad C = \begin{bmatrix} \bar{w} & 0 \\ 0 & \bar{w} \end{bmatrix}$$

Find eigenvalues λ of A by solving

$$|A - \lambda I| = \begin{vmatrix} \bar{u} - \lambda & 0 \\ 0 & \bar{u} - \lambda \end{vmatrix} = 0 \tag{C.1.2}$$

and find the eigenvectors T by solving

$$[A - \lambda I]T = 0 \tag{C.1.3}$$

using the eigenvalues found in equation (C.1.2). The results for all three direction can be found to be

$$\begin{aligned} \xi: \quad \lambda_{1,2} &= \bar{u}, \bar{u} \quad \text{with} \quad T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \eta: \quad \lambda_{1,2} &= \bar{v}, \bar{v} \quad \text{with} \quad T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \zeta: \quad \lambda_{1,2} &= \bar{w}, \bar{w} \quad \text{with} \quad T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned}$$

Now, it is possible to split the flux into contributions from the positive and negative eigenvalues

$$f = f^+ + f^- \qquad f^+ = T\lambda^+ T^1 \hat{S} \qquad f^- = T\lambda^- T^{-1}\hat{S} \tag{C.1.4}$$

where for the ξ direction

$$\lambda^{\pm} = \frac{(\bar{u}^{\mp}) \pm |\bar{u}^{\mp}|}{2} \quad f^+ = \begin{bmatrix} \rho\lambda^+ s_1 \\ \rho\lambda^+ s_2 \end{bmatrix} \qquad f^- = \begin{bmatrix} \rho\lambda^- s_1 \\ \rho\lambda^- s_2 \end{bmatrix}$$

VITA

The author was born in Parma Heights, Ohio on February, 2 1965. He was raised and received his elementary education on the west coast in Livermore, California. He then moved and attended high school in Media, Pennsylvania. The author entered the Aerospace Engineering program at Virginia Tech in 1983 and received his Bachelor of Science degree with the honor of Magna Cum Laude in 1987. In the fall of 1987 the author entered the graduate program in Aerospace Engineering at Virginia Tech and as of the completion of this work is continuing his studies there.

David C. [illegible]