NASA / TM—1999-209182

An Automated Code Generator for Three-Dimensional Acoustic Wave Propagation With Geometrically Complex Solid Wall Boundaries

Rodger William Dyson, Jr.
Glenn Research Center, Cleveland, Ohio

June 1999

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the Lead Center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized data bases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA Access Help Desk at (301) 621-0134

- Telephone the NASA Access Help Desk at (301) 621-0390

- Write to:
  NASA Access Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076

NASA / TM—1999-209182

An Automated Code Generator for Three-Dimensional Acoustic Wave Propagation With Geometrically Complex Solid Wall Boundaries

Rodger William Dyson, Jr.
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center

June 1999

Acknowledgments

I wish to express my gratitude and deepest appreciation to Dr. John W. Goodrich for his guidance, counseling, encouragement and for his friendship. He has allowed me to stand on the infrastructure that he has built. In particular, he developed the MESA schemes used in this work and provided most of the numerical analysis expertise required to complete it.

I would like to thank Chuck Putt for his wonderful insights and assistance. His love of books and science helped to keep me focused. His interest in my career was crucial for my current position at NASA. He also originally introduced me to Dr. John Goodrich and made possible our collaborative partnership.

Thanks goes to my father-in-law, George Durham, who originally suggested that I pursue the Ph.D. since I was studying all those textbooks anyway.

I would also like to thank Gary Weegman, Joan Oravec, Dr. Bill Ford, Dennis Huff and the NASA Glenn Research Center for supporting this research with both the funding and the time to complete it.

And I thank my committee members, Dr. Bob Mullen, Dr. Randy Beer, Dr. George Ernst, and my academic advisor, Dr. Chris Papachristou for reviewing this dissertation and taking the time to serve on my committee.

This work, as does all work, stands on the shoulders of giants. I want to thank my Hiram College professors Dr. Jim Case, Dr. Lawrence Becker, Dr. Oberta Slotterbeck, and Dr. Michael Grajek for introducing me to some of those giants.

And finally, I am grateful for Dr. John Adamczyk's physics course at North Olmsted High School. I did not take academics seriously until his no-nonsense approach to science channeled my stubborness into inquisitiveness and introduced my mind to something worth thinking about.

Trade names or manufacturers' names are used in this report for identification only. This usage does not constitute an official endorsement, either expressed or implied, by the National Aeronautics and Space Administration.

Available from

NASA Center for Aerospace Information
7121 Standard Drive
Hanover, MD 21076
Price Code: A16

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22100
Price Code: A16

Contents

List of Figures vii

List of Tables x

1 Introduction to Computational Aeroacoustics 1

1.1 Governing Equations 2

1.1.1 Navier-Stokes Equations 2

1.1.2 The Euler Equations 3

1.1.3 Linearized Euler Equations 4

1.2 CFD vs. CAA 5

1.3 Computational Approaches 8

1.4 Outline of the Thesis 10

2 MESA Propagation Algorithm Development 15

2.1 Spatial Interpolation 16

2.2 Temporal Evolution 21

3 MESA Propagation Algorithm Automation 28

3.1 Spatial Interpolation 29

3.1.1 The Interpolation Problem in 2D 29

3.1.2 Pyramid Representation of Basis Coefficients in 2D 31

3.1.3 Polynomial Ideals and Solving $\mathcal{SA} = \mathcal{Z}$ in 2D 34

3.1.4 Tensor Form Method in 2D 39

3.1.5 The Interpolation Problem in 3D 46

3.1.6 Tensor Form Method in 3D . . . 47
3.2 Temporal Evolution . . . 53
3.2.1 Finite Difference Form Method in 2D . . . 53
3.2.2 Spatial-Temporal Coefficient Form Method in 2D . . . 54
3.2.3 Recursive Tensor Form Method in 2D . . . 61
3.2.4 Cost Comparison of Methods in 2D . . . 65
3.2.5 Finite Difference Form Method in 3D . . . 67
3.2.6 Spatial-Temporal Coefficient Form Method in 3D . . . 69
3.2.7 Recursive Tensor Form Method in 3D . . . 75
3.2.8 Cost Comparison of Methods in 3D . . . 79
3.3 Generating the FORTRAN Propagation Code . . . 81

4 Wall Boundary Mapping in Two-Dimensions 87
4.1 Introduction . . . 87
4.2 Definitions and Approaches . . . 89
4.3 Stencil Constraint Tree . . . 94
4.3.1 Building the Tree . . . 98
4.4 Recursive Boxes . . . 103
4.5 Symmetries and Simplifications . . . 106
4.6 Unique Mappings . . . 109
4.6.1 Mapping the S8 Cases . . . 112
4.6.2 Mapping the S7 Cases . . . 114
4.6.3 Handling the Degenerate Cases . . . 114

5 Solving Near Boundary Grid Points 119
5.1 Lagrangian Form VS. Multidimensional Taylor Series Form . . . 120
5.1.1 Forming the Interpolant With Multidimensional Taylor Series . . . 120
5.1.2 Forming the Interpolant with Lagrangian Polynomials . . . 125
5.1.3 Forming the Interpolant with Hermitian Polynomials . . . 129
5.1.4 Insuring Consistent Linear Systems . . . 133
5.2 Systematic Stencil Selection . . . 135
5.2.1 Maximize Interior Information . . . 135

5.3 Isolated VS. Implicit VS. Recycled Fill Point Solution 136
5.3.1 Isolated Method 140
5.3.2 Implicit Method 140
5.3.3 Recycled Method 141
5.4 Step-by-Step Demonstration of Mapping and Solving the Fill Points 142
5.5 Generating the FORTRAN Wall Boundary Input File 145

6 Extension to Parallel Computational Domain 147
6.1 Domain Decomposition 147
6.2 Message Passing 150
6.3 Synchronous Communication 153
6.4 Asynchronous Communication 156
6.5 Generating the FORTRAN Parallel Propagation Code 157

7 Numerical Results 158
7.1 Two-Dimensional Problems 159
7.1.1 Bi-Periodic Open Domain up to 29^{th} order accuracy 159
7.1.2 Rotated Box at 2^{nd} order accuracy 162
7.1.3 Circle at 2^{nd} order accuracy 165
7.1.4 Unrotated Box up to 11^{th} order accuracy 168
7.1.5 Complex Geometry Demonstration Mappings 169
7.2 Three-Dimensional Problems 171
7.2.1 Tri-Periodic Domain up to 27^{th} order accuracy 172
7.3 Parallel Scalability Studies 173
7.3.1 Bi-Periodic Open Domain up to 21^{st} order accuracy 175

8 Conclusions and Future Research 178
8.1 Summary 178
8.1.1 Scientific Developments in the Thesis 178
8.1.2 Applications of the Scientific Developments 179
8.2 Conclusions 180
8.3 Future Work 181

A Data from Numerical Experiments **182**

A.1 Unrotated Box Numerical Data 182

A.2 Parallel Scalability Study Data 187

B Mathematica Source Code For Acoustics Problems With Wall Boundaries **196**

B.1 Code Generation System Overview 196

B.1.1 Input Parameters 197

B.1.2 Mathematica Modules 199

B.1.3 FORTRAN Subroutines 199

B.1.4 Outputs 201

B.2 Master File – doall2d.geom 202

B.3 Tensor Form of Spatial Interpolation File – tf2d.tfelp 207

B.4 Temporal Evolution Using Recursive Tensor Form File – ce2d.tfelp 219

B.5 Create All The Administrative Files – su2d.geom 227

B.6 Wall Boundary Calculation File – ma2d 250

Bibliography **335**

List of Figures

3.1 Square region Ω of area $((N-1)\times h)^2$ with known data points indicated with dots 30

3.2 Minimal cross-derivative pyramid representation showing diamond sub-structure for $N=2$ and $D=4$ 32

3.3 Pyramid representation showing line sub-structure for $N=2$ and $D=4$ 33

3.4 Maximal Pyramid representation showing stencil evaluation sequence and derivative assignments for case $N=2$ and $D=4$ 35

3.5 Matrix $\mathcal{S}$ with grid size h, N=2 and D=1, solved by lines 36

3.6 Matrix $\mathcal{S}$ with grid size h, N=2 and D=1 solved by diamonds 36

3.7 Pseudocode for correct ordering of vector $\mathcal{Z}$ 38

3.8 Spatial interpolant origin must be at center of stencil 43

3.9 Stencil Local Grid Point Coordinate System (i,j) 43

3.10 Loop to compute the S terms in 2D with even stencil dimensions 44

3.11 Loop to compute the S terms in 2D with odd stencil dimensions 44

3.12 Intermediate Derivative Information Storage Locations For c2d2 MESA 45

3.13 Loop to compute the S2 terms in 2D 46

3.14 Spatial Coefficient Mneumonic for 3D with $N=2$ and $D=1$ 47

3.15 Loop to compute the S terms in 3D with even dimensioned stencils 49

3.16 Loop to compute the S terms in 3D with odd dimensioned stencils 49

3.17 Loop to compute S2 terms in 3D for an even dimensioned stencil 51

3.18 Loop to compute S2 terms in 3D for an odd dimensioned stencil 51

3.19 Loop to compute S3 terms in 3D 52

3.20 S,S2, and S3 Storage Locations For 3D $2\times 2\times 2$ stencil 53

3.21 Evolving all variables using the shifted data, 2D case 60

3.22 Loop to compute all spatial-temporal coefficients in 2D 63
3.23 Loop for advancing all variables with Horner's method in 2D 64
3.24 Evolving all the variables by shifting the data, 3D case 74
3.25 Loop to compute all the spatial-temporal coefficients in 3D 77
3.26 Loop for advancing all variables with Horner's method in 3D 79

4.1 One-Dimensional Wave Equation Boundary Treatments 90
4.2 Two-Dimensional Wave Equation Boundary Treatments 93
4.3 Sample Mapping For Fill Points with 2×2 or 3×3 stencils: Box Rotated $\frac{\pi}{4}$ Case 95
4.4 Staggered Grid with C2oD MESA scheme 96
4.5 Stencil Constraint Tree Branch Numbering Scheme 98
4.6 Stencil Grid Position Labels for N=2, N=3, and N=5 99
4.7 Stencil Constraint Tree, N=2, Assume Position 4 is an Interior Grid Point 99
4.8 First Node Expansion of Stencil Constraint Tree 100
4.9 Unpruned Stencil Constraint Tree 102
4.10 Stencil Grid Box Position Labels for N=3 104
4.11 Box Recursion Method: 8 Neighboring Box Cases 106
4.12 Recursively Collecting 2×2 Sub-Stencil Configurations 107
4.13 No Wrap Assumption: 8 Cases 109
4.14 5×5 stencil configurations under S8 and S7 assumptions 111
4.15 Too Many Collinear Grid Points When Unaligned Boundary Points Need Mapped 111
4.16 Sub 3×3 stencil symmetry under S8 and S7 assumptions 112
4.17 S8 Symmetrical Mapping: 18 Cases 113
4.18 S7 Symmetrical Mapping: 40 Cases 115
4.19 Degenerate Case: No S7 or S8 matching case 117
4.20 All Possible S7 Stencil Configurations with Fill at Center : 16 Cases 117
4.21 All Possible S8 Stencil Configurations with Fill at Center : 49 Cases 117
4.22 Degenerate Case: One S8 matching case 118
4.23 Substituting Non-Degenerate Cases 2 and 3 to Remove Degenerate Case 1 118

5.1 Stable Sample Mapping: Box Rotated $\frac{\pi}{8}$ Case 121
5.2 Unstable Sample Mapping: Box Rotated $\frac{\pi}{8}$ Case 122

5.3 Stable Sample Mapping: Box Rotated $\frac{\pi}{8}$ Case 123
5.4 Stable Sample Mapping: Box Unrotated Case 127
5.5 Sample Mapping Ordered by Minimal Interior Dependency: Box Rotated $\frac{\pi}{8}$ Case 137
5.6 Sample Mapping Ordered by Minimal Interior Dependency – Small 2×2 Stencil: Box Rotated $\frac{\pi}{8}$ Case 138
5.7 Sample Mapping Ordered by Maximal Interior Dependency – Small 2×2 Stencil: Box Rotated $\frac{\pi}{8}$ Case 139
5.8 5×5 stencil with fill point in center 142

6.1 Solving bi-periodic open domain Linearized Euler Equations with MESA 152
6.2 Nearest Neighbor Communication 154
6.3 Synchronous Communication, Implicit Corner Exchange 155

7.1 Rotated Boxes and Circle Low Resolution Cases 160
7.2 Maximum Absolute Error at time=10, with convection Mx=My=1 162
7.3 Maximum Absolute Error at time=10, with convection Mx=My=1 163
7.4 Maximum Absolute Error at time=10, with convection Mx=My=1 163
7.5 Unrotated box grid resolution studies, no convection, time=10 169
7.6 Annular Duct Fill Point Mapping in 2D, iun=8 170
7.7 Airfoil Cascade Grid Point Labeling in 2D, iun=8 170
7.8 Airfoil Cascade Fill Point Mapping in 2D, iun=8 171
7.9 Maximum Absolute Error at time=10, with convection Mx=My=1 in 3D 174
7.10 Scalability Performance to time=10, with convection Mx=My=1, iun=8 176
7.11 Scalability Performance to time=10, with convection Mx=My=1, iun=16 177
7.12 Scalability Performance to time=10, with convection Mx=My=1, iun=32 177

B.1 Overview of Code Generation System 198

List of Tables

3.1 Cost comparison ($\log_{10}$ multiplies per grid point) of 2D methods 68

3.2 Cost comparison ($\log_{10}$ multiplies per grid point) of 3D methods 82

7.1 Maximum Absolute Error of 2D Algorithms at time=10,100,1000, 1^{st} - 9^{th} order 164

7.2 Maximum Absolute Error of 2D Algorithms at time=10,100,1000, 11^{th} - 19^{th} order 165

7.3 Maximum Absolute Error of 2D Algorithms at time=10,100,1000, 21^{st} - 29^{th} order 166

7.4 Maximum Error in p at t=10, c3o0 scheme applied to box rotated by α 166

7.5 Energy Ratio (should be 1) at t=10, c3o0 scheme applied to box rotated by α . . 167

7.6 Maximum Error in p at t=10,100,1000, c3o0 scheme applied to circle 168

7.7 Energy Ratio (should be 1) at t=10,100,1000, c3o0 scheme applied to circle . . . 168

7.8 Maximum Absolute Error of 3D Algorithms at time=10, 100, 1^{st} - 9^{th} order . . . 174

7.9 Maximum Absolute Error of 3D Algorithms at time=10, 100, 11^{th} - 19^{th} order . 174

7.10 Maximum Absolute Error of 3D Algorithms at time=10, 100, 21^{st} - 27^{th} order . 175

A.1 Maximum Error in p at t=1, 10, 100, 1000, c2o1 scheme applied to unrotated box 183

A.2 Maximum Error in p at t=1, 10, 100, 1000, c2o2 scheme applied to unrotated box 184

A.3 Maximum Error in p at t=1, 10, 100, 1000, c2o3 scheme applied to unrotated box 185

A.4 Maximum Error in p at t=1, 10, 100, 1000, c2o4 scheme applied to unrotated box 185

A.5 Maximum Error in p at t=1, 10, 100, 1000, c2o5 scheme applied to unrotated box 186

A.6 Scalability of Even Stenciled 2D Algorithms, c2o1 - c2o5, iun=8 188

A.7 Scalability of Even Stenciled 2D Algorithms, c2o7 - c2o9, iun=8 189

A.8 Scalability of Even Stenciled 2D Algorithms, c4o0 - c4o4, iun=8 190

A.9 Scalability of Odd Stenciled 2D Algorithms, c3o0 - c15o0, iun=8 191

A.10 Scalability of Even Stenciled 2D Algorithms, c2o1 - c2o5, iun=16 192

A.11 Scalability of Even Stenciled 2D Algorithms, c2o7 - c2o9, iun=16 193
A.12 Scalability of Odd Stenciled 2D Algorithms, c3o0 - c15o0, iun=16 194
A.13 Scalability of Even Stenciled 2D Algorithms, c4o0 - c4o4, iun=16 195
A.14 Scalability of Even Stenciled 2D Algorithms, c2o1 - c2o3, iun=32 195

Chapter 1

Introduction to Computational Aeroacoustics

Computational Aeroacoustics (CAA) is a relatively new and rapidly growing field of research that combines the traditional disciplines of Aeroacoustics and Computational Fluid Dynamics (CFD). It may be defined as the direct calculation of all aspects of sound generation and propagation from the underlying differential or integral equations describing fluid motion [48]. CAA can be applied to fields such as aeronautics, medicine, automotive engineering and architectural design. The reduction of aircraft noise is one significant aeronautical application for CAA [98]. The United States has established a national goal of reducing noise levels from the levels of today's subsonic aircraft by a factor of two within 10 years, and by a factor of four within 20 years. These noise reduction goals are important because air travel demand is expected to triple over the next 20 years, and because loud aircraft will not be permitted to land at many airports [3].

CAA is best used by an engineer or a scientist as a tool for analysis that compliments theoretical and experimental techniques. A major capability offered by CAA is the simulation of linear problems. A solution linearized about a known base flow of the unsteady Euler equations with suitable wall and artificial boundary treatments can provide useful predictions of the noise propagated from aircraft propulsion systems. Perhaps the greatest potential for CAA is the solution of the non-linear problem of sound generation. The direct computation of aerodynamic sound generation permits a very detailed look at any flow quantity, and the mechanisms of

sound generation can be explored at a fundamental level [70]. This dissertation is restricted to providing tools that can be applied to linear problems. Much of this work on methods for linear problems will be directly usable or extendible to the nonlinear case, which will be considered in later work.

1.1 Governing Equations

The linearized Euler equations in three space dimensions have been utilized extensively in this work. Their development begins with the fundamental physical laws of conservation of mass, momentum, and energy. With the addition of thermodynamic relations, these fundamental laws are used to derive the Navier-Stokes equations, which give a complete description of viscous flow phenomena [9]. The Navier-Stokes equations are a nonlinear parabolic system, and do not have a general solution [5]. Restrictive assumptions can be added to these viscous flow equations, and a variety of other systems can be derived from them. Brief presentations will be made of the Navier-Stokes equations, and of the Euler and linearized Euler equations [62].

1.1.1 Navier-Stokes Equations

The equations for flows in a compressible medium govern the sound generation and propagation in a fluid flow. The Navier-Stokes equations in air are given by Batchelor [9] as:

Conservation of Mass

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_i}{\partial x_i} = \dot{m}, \tag{1.1}$$

Conservation of Momentum

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_i u_j + p_{ij}}{\partial x_i} = \dot{F}_i, \tag{1.2}$$

Equation of State

$$p = p(\rho, S) = \rho R T, \tag{1.3}$$

Energy Equation

$$T\frac{DS}{Dt} = c_p \frac{DT}{Dt} - \frac{\beta T}{\rho}\frac{Dp}{Dt} = \phi + \frac{1}{\rho}\frac{\partial}{\partial x_i}(k\frac{\partial T}{\partial x_i}), \tag{1.4}$$

with

$$p_{ij} = \delta_{ij}p + \mu[-\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i} + \delta_{ij}\frac{2}{3}(\frac{\partial u_k}{\partial x_k})].$$

These equations are used to obtain the pressure p, the density ρ, the velocity components u_i, the temperature T, and the entropy S. Particular flow conditions are specified by the viscosity coefficient μ, the specific heat at constant pressure c_p, the thermal expansion coefficient $\beta = -\frac{1}{\rho}(\frac{\partial \rho}{\partial T})_p$, the dissipation function proportional to the viscosity coefficient ϕ, the thermal conductivity of the fluid k, the gas constant R, the rate of mass introduction per unit volume $\dot{m}$, and the body force components per unit volume F_i. In the case of isentropic flow (ie. with constant entropy), $\frac{p}{\rho^\gamma}$ is constant. In this case, acoustic waves are propagated at the speed of sound c, where

$$c^2(\rho, p) = (\frac{\partial p}{\partial \rho})_S = \frac{\gamma p}{\rho} = \gamma RT,$$

where $\gamma = 1.4$ in air at 80° F.

1.1.2 The Euler Equations

The Euler equations are developed directly from the Navier-Stokes equations under the assumptions that the flow is inviscid with $\mu = 0$, and that the heat transfer terms are negligible. Euler's Equations of fluid motion for an inviscid fluid are:

Continuity Equation

$$\rho_t + \bigtriangledown \cdot (\rho \bar{u}) = 0, \tag{1.5}$$

Conservation of Momentum

$$\bar{u}_t + (\bar{u} \cdot \bigtriangledown)\bar{u} + (\frac{1}{\rho}) \bigtriangledown p = 0, \tag{1.6}$$

Equation of State

$$p = f(\rho), \tag{1.7}$$

where $\bar{u} = (u_1, u_2, u_3)$ is the velocity vector. The general form of the Equation of State is for isentropic flow. The continuity equation is the form of conservation of mass for the Euler equations. Euler's equations are a nonlinear hyperbolic system.

1.1.3 Linearized Euler Equations

The linearized Euler equations are derived from the Euler equations by linearizing with a perturbation around a steady solution. The linearized Euler equations for the isentropic case in three space dimensions can be written as:

$$\frac{\partial u}{\partial t} + U\frac{\partial u}{\partial x} + V\frac{\partial u}{\partial y} + W\frac{\partial u}{\partial z} + \frac{\partial p}{\partial x} = 0, \tag{1.8}$$

$$\frac{\partial v}{\partial t} + U\frac{\partial v}{\partial x} + V\frac{\partial v}{\partial y} + W\frac{\partial v}{\partial z} + \frac{\partial p}{\partial y} = 0, \tag{1.9}$$

$$\frac{\partial w}{\partial t} + U\frac{\partial w}{\partial x} + V\frac{\partial w}{\partial y} + W\frac{\partial w}{\partial z} + \frac{\partial p}{\partial z} = 0, \tag{1.10}$$

$$\frac{\partial p}{\partial t} + U\frac{\partial p}{\partial x} + V\frac{\partial p}{\partial y} + W\frac{\partial p}{\partial z} + \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0, \tag{1.11}$$

where p and (u, v, w) are the disturbance pressure and velocity, and where (U, V, W) is the constant mean convection velocity. This form of the linearized Euler equations is non-dimensionalized in terms of the Mach number or speed of sound. The linearized Euler equations with a constant mean flow are useful for modeling the propagation of an acoustic signal, but require that signal to be specified. The determination of the sound sources from the underlying fluid dynamics will require the use of the Navier-Stokes or nonlinear Euler equations.

The linearized Euler equations in two space dimensions does not have any z derivatives or velocity components, and can be written as:

$$\frac{\partial}{\partial t}\begin{pmatrix} u \\ v \\ p \end{pmatrix} + \begin{pmatrix} U & 0 & 1 \\ 0 & U & 0 \\ 1 & 0 & U \end{pmatrix}\frac{\partial}{\partial x}\begin{pmatrix} u \\ v \\ p \end{pmatrix} + \begin{pmatrix} V & 0 & 0 \\ 0 & V & 1 \\ 0 & 1 & V \end{pmatrix}\frac{\partial}{\partial y}\begin{pmatrix} u \\ v \\ p \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{1.12}$$

It is not possible to simultaneously diagonalize both of the coefficient matrices in equation 1.12, and reduce the system to separate decoupled systems. The linearized Euler equations in two or three space dimensions are inherently multidimensional, with wave propagation along characteristic surfaces. This property of the multidimensional systems is significantly different from the linearized system in one space dimension, where the linearized Euler equations may be decoupled and solved by the Method of Characteristics. The MESA technique for algorithm development (see Chapter 2) generalizes the method of characteristics by using exact local propagators, and

correctly incorporates multidimensional wave propagation along characteristic surfaces.

1.2 CFD vs. CAA

In many aeroacoustic problems, the energy levels of the unsteady flow fluctuations and of the sound perturbation can differ by from 2 to 5 orders of magnitude. This requires CAA algorithms to have very high accuracy for resolving the sound and the fluid flow. The range of the human ear is 20 Hz - 20 KHz, with peak sensitivity near 2KHz. This requires CAA algorithms to be able to accurately propagate a wide band of frequencies. CFD has typically been interested in steady-state solutions, and has developed methods with high spatial accuracy. But acoustic waves have both a wavelength in space and a frequency in time, and this requires CAA algorithms to have high accuracy in both space and time. CFD grids are often solution-adaptive to provide the correct resolution in regions of varying gradients, but grid stretching or irregularity can distort an acoustic wave if standard CFD algorithms are used. This requires CAA algorithms to have unusually good accuracy when dealing with the geometry of complex objects or the details of complicated flows. All of these considerations imply that CAA requires numerical algorithms with significantly greater capabilities than standard CFD methods.

Dissipation and dispersion are two properties of finite difference discretizations that are commonly used to describe and compare finite difference methods. In order to illustrate these properties, consider approximating a time derivative with a first-order forward difference, or forward Euler differencing:

$$\frac{\triangle F}{\triangle t} = \frac{F(t+\Delta t) - F(t)}{\Delta t} \approx \frac{dF}{dt}(t). \tag{1.13}$$

For the normal or Fourier mode $F(t) = e^{i\omega t}$ with frequency ω, the first order forward Euler discretization is

$$\frac{\triangle F}{\triangle t} = \frac{e^{i\omega t}(e^{i\omega \Delta t} - 1)}{\Delta t} = iwe^{i\omega t}\left(\frac{e^{i\omega\frac{\Delta t}{2}} - e^{-i\omega\frac{\Delta t}{2}}}{2i}\right)\left(\frac{e^{i\omega\frac{\Delta t}{2}}}{\omega\frac{\Delta t}{2}}\right). \tag{1.14}$$

But $\sin(\omega t) = \frac{e^{i\omega t} - e^{-i\omega t}}{2i}$, so that for the normal mode F

$$\frac{\triangle F}{\triangle t} = i\omega e^{i\omega t} \left(\frac{\sin(\frac{\omega \Delta t}{2})}{\frac{\omega \Delta t}{2}} \right) e^{i\omega \frac{\Delta t}{2}} = \frac{dF}{dt} \left(\frac{\sin(\frac{\omega \Delta t}{2})}{\frac{\omega \Delta t}{2}} \right) e^{i\omega \frac{\Delta t}{2}}. \tag{1.15}$$

The Euler discretization reduces the amplitude of the derivative by the dissipation factor $\frac{\sin\theta}{\theta}$, and shifts its phase by the dispersive term $e^{i\theta}$, where $\theta = \frac{\omega \Delta t}{2}$. Higher frequency waves will be distorted more than lower frequency waves [48]. The error introduced by a discretization will depend upon the differencing that is used, but CFD methods generally have significant dispersion and dissipation errors. Many CFD methods actually rely upon introducing significant error in the form of artificial damping, sometimes introduced to overcome dispersive errors, or to force convergence of a solution. Typical acoustic applications demand highly accurate simulations, with very small dissipation and dispersion errors relative to the levels that have been acceptable in standard CFD practice.

Most CFD algorithms are developed in an essentially piecemeal way by choosing various finite difference discretizations for the separate derivative terms in equations that are to be simulated. The separate discretizations are chosen to work together to form a complete algorithm, but the basic process tends to begin with a consideration of the separate derivative terms. The essential issue, however, is not to approximate a particular derivative, but to approximate the solution of a system of equations. The wave dynamics of a system are defined by the governing partial differential equations, and it is useful to compare the solution of the original continuum problem with the computed results obtained from a full discretization in both time and space. Solutions are composed of superposed waves of different frequencies, and the separate waves can travel with different speeds. The wave speed dependency upon the wave frequency is described by the dispersion relation of the governing equation. Consider the one dimensional scalar linear wave equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0, \tag{1.16}$$

where c is constant. The general form of the solution for this equation is

$$u(x,t) = u_o(x - ct), \tag{1.17}$$

where $u_o(x) = u(x,0)$ is the initial data at $t = 0$. A Fourier mode solution of a partial differential

equation has the general form

$$u(x,t) = \alpha e^{i(\omega t + \beta x)}, \tag{1.18}$$

where ω is the frequency of the solution, and β is the wave number, which is related to the wave length Λ by $\beta\Lambda = 2\pi$. In the case of the linear wave equation, the general form of the solution requires that the frequency and wave number of a Fourier mode satisfy the dispersion relation

$$\omega = -c\beta, \tag{1.19}$$

so that a Fourier mode for the linear wave equation has the form

$$u(x,t) = \alpha e^{i\beta(x-ct)}, \tag{1.20}$$

with wave number β and frequency $\omega = -c\beta$. There are dispersion relationships for solutions of both continuum and discrete systems, and these relationships can be compared. Significant effort has been expended to reduce discretization errors by increasing the agreement between the dispersion relationships of the continuum system and the discrete approximation by developing Compact Differencing methods in CFD [69], and the Dispersion Relationship Preserving methods in CAA [105]. In general, CFD methods have difficulty providing accurate phase speeds for modal frequencies greater than $\frac{\pi}{2}$.

The wave equation in a quiescent medium is isotropic, and propagates waves at the same speed in all directions. A central difference scheme that is employed on a rectangular grid will propagate wave information in the x-direction at speed $\frac{\Delta x}{\Delta t}$, in the y-direction at speed $\frac{\Delta y}{\Delta t}$, and in the diagonal direction at speed

$$\frac{\sqrt{(\Delta x)^2 + (\Delta y)^2}}{2\Delta t}. \tag{1.21}$$

This wave speed distortion is further magnified when computing simulations in three dimensions. Standard CFD techniques introduce this type of isotropy error which depends upon the orientation of the propagating wave fronts relative to the discretization grid. In particular, CFD methods that depend upon dimensional splitting can introduce significant dissipation in order to glue together a multidimensional solution from separate intermediate one dimensional solutions [74]. The accuracy required for CAA simulations, and the inherently multidimensional

nature of wave propagation require CAA algorithms that have low levels of isotropy error.

The standard CFD approach to handling curved wall surfaces is to map the physical domain into a rectangular computational domain with the curved surface mapped into a plane boundary, or to use unstructured grids [106]. These are not the best methods for aeroacoustic problems, since they introduce inhomogeneities into the governing equations which could cause unintended acoustic refraction and scattering. Methods for handling objects with complex structures or flows with complicated local features continues to be an important area of CFD research, and is even more significant for CAA.

1.3 Computational Approaches

The numerical simulation of wave propagation has a history, and various computational methods have been developed. Each approach has a unique set of properties, and it is difficult to defend the idea that one method is best for all applications. A practical criterion for comparing algorithms and their code implementations is the amount of effort that they require to produce a desired result. It has been shown that within a given class of numerical methods, higher order methods tend to be more efficient [62]. Methods which are used in CAA for linear problems include:

MacCormack Methods These algorithms are standard CFD methods, with dimensional splitting, and significant dissipation for stability. The original algorithm is second order in time and space [75]. An early variant is second order in time and fourth order in space [41] Recent work includes the development of higher order variants, including algorithms which are fourth order in both time and space [118] and algorithms which are fourth order in time and sixth order in space [53]. MacCormack methods are readily available, and have been used for CAA simulations.

Compact Difference Compact Difference methods have been widely used in CFD but are more recent than other standard CFD methods, such as the MacCormack methods. Compact Differencing methods use global spline approximations to obtain local derivative data, with some of the possible order of accuracy of the spatial interpolation sacrificed to obtain better phase accuracy [69]. Compact Differencing methods are defined by an approach

to spatial interpolation, and tend to use a Runge-Kutta type of time stepping method, generally with fourth order accuracy in time. These methods can accurately propagate relatively high frequency waves, compared to standard finite difference methods, and have been used successfully in CAA simulations [19].

DRP Dispersion Relation Preserving methods are based on the fact that the wave propagation properties of a system are implicit in the dispersion relation of the system [105]. Like Compact Differencing methods, DRP methods sacrifice potential order of accuracy for improved phase speed accuracy, and they use multistep time stepping methods of the Runge-Kutta type. DRP methods sacrifice accuracy in time stepping as well as spatial interpolation in order to optimize phase speed accuracy over a wide range of normal mode frequencies [105], [55]. DRP methods are similar to Compact Differencing methods, but have been used especially for CAA simulations.

The chief difficulty with all of these methods is that they are not efficiently accurate enough for CAA simulations. The focus of this dissertation has been to produce automation tools for developing a new type of algorithm for CAA. Many algorithm realizations have been produced in both two and three dimensions. All of the algorithms of this new type are explicit single step finite difference methods with the same order of accuracy in both time and space. This new type of method uses central stencils, and has both dissipative and dispersive realizations, with from the first to the 29^{th} order accuracy in space and time in two dimensions and three dimensions. Higher order realizations are possible but not useful on today's computers due to roundoff error (64 bit precision). A particular subclass of this new type of algorithm has exceptional phase accuracy, or high resolution.

1.4 Outline of the Thesis

As discussed earlier, it is desirable for both economic and environmental reasons to reduce the noise emissions from commercial aircraft. Using a combination of theoretical, experimental and computational approaches, many advances in noise reduction have been achieved. If the sources of noise can be located, it may be possible to further reduce noise emissions by modifying the propulsion system. However, finding the sources of noise generation in a turbofan propulsion system requires a computational tool that has sufficient fidelity to simulate steep gradients in the flow field and sufficient efficiency to run on today's computer systems.

To meet those needs, the Modified Expansion Solution Approximation (MESA) series of explicit finite-difference schemes were developed by Dr. John Goodrich to provide spectral-like resolution with extraordinary efficiency [33], [34], [35], [36], [37], [40]. The accuracy of these methods can, in theory, be arbitrarily high in both space and time, without the significant inefficiences of Runge-Kutta based schemes. These methods were originally developed in one and two dimensions up to 11^{th} order accuracy but were extended in this work to three dimensions with up to 29^{th} order accuracy. The essential idea behind the MESA schemes is to approximate the solution of the partial differential equations instead of approximating the individual derivative terms of the governing equations. The MESA schemes use multidimensional spatial interpolation and the constructive procedure in the proof of the Cauchy-Kovalevsky theorem [30] to develop a local series approximation to the solution of the partial differential equation system in both space and time. The recursive Cauchy-Kovalevsky procedure has been used by Harten et.al. [50] to produce a local third order method. The high resolution MESA methods use Hermitian interpolation and propagate the spatial derivatives as well as the solution variables of interest. Hermitian interpolation is widely applied to the solution of partial differential equations, as in Collatz [18], and has been used recently by Takewaki et.al. [103] and Yabe [124] to develop the third order Cubic-Interpolated Pseudoparticle schemes which use and propagate first spatial derivatives.

The complexity of coding the original form of the MESA schemes was, however, very high, resulting in code that could not compile or took so long to write in FORTRAN that they were rendered impractical. Three algebraically equivalent forms of the MESA schemes were implemented and compared in this work in an attempt to find an optimal algorithmic implementation

form. We call these three algorithmic forms the Finite Difference, the Spatial-Temporal, and the Recursive Tensor forms. The Finite Difference algorithm form calculates new solution values as linear combinations of the old solution values which are known on the stencil. The Spatial-Temporal algorithm form calculates new solution values as combinations of interpolation coefficients which are obtained from the known data on the stencil, and which approximate partial derivatives of the solution variables. The Recursive-Tensor algorithm form uses Tensor Product spatial interpolation and Cauchy-Kovalevsky recursion for obtaining time derivatives. These algorithmic forms are all mathematically equivalent realizations of the MESA methods, but not equivalent with respect to FLOP counts. The Recursive Tensor form of the MESA schemes has the advantage of being simple to code and compile, and it was found to be the relatively more efficient form with respect to FLOP count per grid point per time step when high accuracy MESA algorithms are used on small stencils, particularly for higher spatial dimensions. A code generation tool was developed and written in Mathematica that automatically develops particular implementations of the MESA schemes, in any of the three possible algorithms forms, and then produces FORTRAN codes for the MESA scheme. This code generator creates all the software necessary to numerically solve the linearized Euler equations on Cartesian grids in two or three dimensional spatial domains. This tool is capable of generating algorithms of any order accuracy, though 20^{th} order accuracy appears to be the highest accuracy that is useful while restricted to using 64 bit precision computer hardware. The code generator can create software for a second order MESA scheme for two dimensional spatial domains with Cartesian grids and embedded objects with complex shapes. The ability to treat complex geometric objects with higher than second order accuracy on a Cartesian grid remains an open research problem in numerical analysis.

A method was developed for treating solid wall boundaries with arbitrary piecewise smooth shapes on Cartesian grids in two spatial dimensions, with up to 11^{th} order accuracy on grid aligned boundaries, and with up to 2^{nd} order accuracy on generalized irregular boundaries. The values of the grid points near the solid wall boundaries are found by evaluating a spatial interpolant that is simultaneously consistent with the given wall boundary conditions and the known data on the neighboring interior grid points. A mapping has been developed which insures that a consistent spatial interpolant can be found for each grid point near a boundary. Lagrangian-Hermitian forms of the spatial interpolants were shown to be the most efficient forms

when using a Cartesian grid. In addition, the numerical stability of the MESA schemes with wall boundaries depends upon the order in which the spatial interpolants are evaluated and a procedure for the proper selection of the spatial interpolants stencil domains was developed.

Finally, an automated method for parallelizing these approaches on large scale parallel computers is presented. This method is an extension of the code generation software written in Mathematica and it creates code which uses the message passing interface (MPI) standard for the parallelization. Numerical experiments with the automatically generated parallelized codes for the MESA methods have shown nearly perfect scalability up to 256 processors if a modest minimal load per processor is maintained by scaling the test problem with the number of processors.

All these features are combined to form a turnkey code generation tool in Mathematica. If the list of parametric curves commonly found in CAD files is provided, this tool can then automatically simulate the acoustical physics by replacing the traditionally labor intensive tasks of grid generation, algorithm development, FORTRAN coding, and wall boundary treatments with automated procedures. The results of the automatically generated codes were validated in several ways. First, the results were compared with the earlier results obtained by Dr. John Goodrich in one dimension up to 11^{th} order accuracy and in two dimensions with up to 5^{th} order accuracy. Second, the numerical solution was compared to the exact solution of the test problems while increasing the grid resolution, confirming the order of accuracy of each MESA method. Third, each FORTRAN subroutine has an analogous Mathematica module which enables the validation of individual subroutines. The results from the automatically generated parallel code were validated by comparing them to the serial results, and by comparing them to the known analytical solution, for up to 23^{rd} order accuracy. Several orders of magnitude difference in the efficiency of the methods were observed between the lowest order and higher order methods using Hermitian data. This is consistent with the earlier empirical results of Goodrich [37] and the theoretical studies of Kreiss and Oliger [63].

A brief description of each chapter follows:

Chapter 2 will discuss the MESA scheme as it was originally developed by Dr. John Goodrich. In particular, the scheme can be divided into two processes. First, a multidimensional polynomial spatial interpolant is found that is locally consistent with the known data on a given stencil. Second, this locally defined analytical interpolant is advanced in time using the Cauchy-

Kovalesvky process [126], which generates the time derivatives that are needed for a Taylor series expansion in time.

Chapter 3 will discuss the automation of the three alternative forms of the MESA schemes: the Finite Difference, the Spatial-Temporal, and the Recursive Tensor forms. Each of these forms will produce the same numerical results, but will vary in their execution efficiency and code complexity. A cost analysis comparing these methods is shown as well.

Chapter 4 will provide the foundation necessary for solving near boundary grid points. In particular, an efficient data structure for representing all possible stencils on a Cartesian grid and an algorithm for efficiently constructing it are shown. For small stencils, it is possible to significantly reduce the set of stencil configurations to under 70 cases by making some simplifying assumptions and applying rotational and line symmetry. With this small set of stencils, it is relatively simple to find spatial interpolants for all near boundary grid points by mapping each near boundary grid point to the boundary. The spatial interpolant is consistent with the wall boundary conditions at these mapped locations on the boundary. To ensure linear consistency, the mapping must ensure that more than 3 grid points (mapped or interior) are never on a line that is parallel to a coordinate axis. A complete mapping for 3×3 stencils in two spatial dimensions is shown as well.

Chapter 5 discusses a method for finding a spatial interpolant that is simultaneously consistent with the interior grid points and the wall boundary conditions at the mapped boundary points. In particular, by forming the polynomial interpolant into its Hermitian-Lagrangian form, the number of unknowns reduces to the small set of data elements contained within the near boundary grid points. It was possible to create stable 2^{nd} order methods in complex domains by carefully selecting the stencil domains for each spatial interpolant in a way that maximizes the use of interior grid point information.

In **chapter 6** the code generator that solves the open domain problem in two dimensions is extended to the parallel domain. The computational domain is divided using domain decomposition. Messages are passed between nodes with the Message Passing Interface (MPI) using an asynchronous communication implementation. Excellent scalability was achieved for nearly all the MESA schemes with the small, higher order Hermitian methods showing the best parallel efficiency.

Chapter 7 provides the numerical results for open domain problems in two and three spatial

dimensions using MESA methods that are up to 29^{th} order accuracy. It was found that some of these algorithms are actually more powerful than the computer floating point hardware (64 bit precision). The performance of the method for handling complex geometric shapes in a Cartesian grid was tested in a box that was rotated at many orientations relative to the Cartesian grid, and in a circular geometry which had a Bessel function analytical solution.

Final conclusions are drawn in **chapter 8** and directions for future research are given.

The **appendices** provide the numerical data from the results of chapter 7 and they provide an example Mathematica code for the two dimensional FORTRAN code generator which includes wall boundaries.

Throughout this work, each MESA scheme when applied in D spatial dimensions will be denoted by cSoO in which S represents the size of the stencil in one dimension and O represents the depth of data on each grid point. In particular, there are $(D+1)(O+1)^D$ data elements per grid point. For example, c2o2 in two spatial dimensions, represents the MESA scheme which has a 2×2 stencil with $3(2+1)^2 = 27$ data elements per grid point.

Chapter 2

MESA Propagation Algorithm Development

The MESA method for algorithm development is presented in this chapter, along with a brief discussion of properties of algorithms developed with this method in two space dimensions (see [34, 35, 37]). The acronym MESA stands for Modified Expansion Solution Approximation, and the method is based upon the use of Cauchy-Kowaleskaya expansions [126] for locally approximating the solution of a system of partial differential equations. The MESA method can be viewed as a two stage process of local interpolation to known data, and then time evolution or propagation of the local spatial interpolant. These two algorithm stages are tied together by the use of Cauchy-Kowaleskaya expansions for locally approximating the solution in space and time, or equivalently, for obtaining time derivatives in terms of space derivatives. Data is typically known at a single time level, so that the numerical algorithms generally are explicit. The time evolution is with local Cauchy-Kowaleskaya expansions, so that exact propagators are possible for linear constant coefficient systems. If an exact propagator is used for time evolution, then the properties of particular algorithm realizations depend upon the local spatial interpolants that are used to supply initial data for the propagator. In particular, all of the explicit algorithms which use exact propagators have the same order of accuracy in both space and time, and they all correctly incorporate wave propagation along characteristic surfaces for multidimensional systems. Local polynomial interpolants are used to approximate the local data

surface in multiple space dimensions, and the Method of Undetermined Coefficients is used to obtain the local expansion coefficients. This approach is equivalent to using multidimensional Taylor series expansions for the spatial interpolation, but it does not restrict either the location or nature of the data that is used, as long as the linear system for the expansion coefficients is solvable. Two types of interpolants are used, with two distinct classes of algorithms as a result. The first type of interpolant is ordinary multidimensional Lagrangian polynomial interpolation with known solution data given on the local stencil. The second type of interpolation is Hermitian interpolation, and uses data from the solution and some of its derivatives. Algorithm realizations have been developed in one, two, and three space dimensions with from first to twenty-ninth order accuracy in both space and time, and using data from the solution up to its fourteenth derivatives. In one, two and three space dimensions, the algorithms with the most unusual properties have been developed using Hermitian interpolation. An emphasis of this dissertation work has been the extension of these algorithms to three space dimensions, and the automation of their creation and code implementation in both two and three space dimensions. Significant issues were the comparison of the efficiency of the various algorithms, and an assessment of how difficult they are to create and code, even with automated tools. These algorithms are actually quite simple to implement if a particular approach is used as discussed later.

2.1 Spatial Interpolation

Multidimensional polynomial interpolation is used for the algorithms that have been developed with the MESA method. If $f(x, y)$ is a function with a two dimensional domain, then a general order O polynomial interpolation form can be written as

$$f(x,y) \approx \sum_{i,j=0}^{O} a(i,j) x^i y^j . \tag{2.1}$$

Note that the interpolation is defined in local coordinates, and assumes the change of coordinates $(x, y) = (x - x_c, y - y_c)$, where (x_c, y_c) is the center of the expansion in global coordinates. Note also that for $i, j = 0, 1, \ldots, O$,

$$a(i,j) = \frac{1}{i!j!} \frac{\partial^{i+j} f}{\partial x^i \partial y^j} . \tag{2.2}$$

The interpolation coefficients $a(i,j)$ are obtained by the Method of Undetermined Coefficients, with the constraint equations obtained from expanding the interpolant to represent known data on a specified stencil. There is some flexibility in choosing the data type and stencil layout, subject to stability constraints, and in modifying the form of the local interpolant, subject to order of accuracy constraints. The multidimensional interpolant includes the two separate order O single dimensional interpolants

$$f(x,0) \approx \sum_{i=0}^{O} a(i,0)x^i, \quad f(0,y) \approx \sum_{j=0}^{O} a(0,j)y^j, \tag{2.3}$$

for expansion about (x_c, y_c) in x and y, respectively. The cross-derivative expansion coefficients are necessary for multidimensional approximation, and include derivative terms from order 2 to order $2O$. It has been observed in the past that cross-derivative terms improve stability, isotropy, and accuracy [34]. Symmetric multidimensional interpolants are advantageous in algorithms for the linearized Euler equations because these equations propagate information from every direction along characteristic surfaces.

Biquadratic interpolation on a uniform mesh is a simple example of two dimensional polynomial interpolation, and has been used for previous work with the MESA method. In this case,

$$f(x,y) \approx \sum_{i,j=0}^{2} a(i,j)x^i y^j. \tag{2.4}$$

If a central 3×3 square stencil is used, with expansion about the central stencil point and

function data given at each point, then the constraint equations are

$$\begin{aligned}
f(+h,+h) &= a_{00} + a_{10}h + a_{20}h^2 + (a_{01} + a_{11}h + a_{21}h^2)h + (a_{02} + a_{12}h + a_{22}h^2)h^2, \\
f(+h,0) &= a_{00} + a_{10}h + a_{20}h^2 \\
f(+h,-h) &= a_{00} + a_{10}h + a_{20}h^2 - (a_{01} + a_{11}h + a_{21}h^2)h + (a_{02} + a_{12}h + a_{22}h^2)h^2, \\
f(0,+h) &= a_{00} + a_{01}h + a_{02}h^2, \\
f(0,0) &= a_{00} \\
f(0,-h) &= a_{00} - a_{01}h + a_{02}h^2, \\
f(-h,+h) &= a_{00} - a_{10}h + a_{20}h^2 + (a_{01} - a_{11}h + a_{21}h^2)h + (a_{02} - a_{12}h + a_{22}h^2)h^2, \\
f(-h,0) &= a_{00} - a_{10}h + a_{20}h^2 \\
f(-h,-h) &= a_{00} - a_{10}h + a_{20}h^2 - (a_{01} - a_{11}h + a_{21}h^2)h + (a_{02} - a_{12}h + a_{22}h^2)h^2,
\end{aligned} \tag{2.5}$$

where the function values on the left are assumed known, and the expansion coefficients $a_{ij} = a(i,j)$ are to be determined. Note that a uniform mesh size $h = \triangle x = \triangle y$ is assumed, and that this is not a necessary restriction. Note also that the biquadratic interpolant has the fourth order a_{22} term. For any polynomial interpolant, the system of equations for the expansion coefficients can be written in the general form

$$\mathcal{S}\mathcal{A} = \mathcal{Z}, \tag{2.6}$$

where $\mathcal{Z}$ is the vector of function values known on the interpolation stencil, $\mathcal{A}$ is the vector of unknown expansion coefficients, and $\mathcal{S}$ is the matrix of expansion data from the form of the interpolant and the geometry of the points in the stencil. Any multidimensional interpolant must have a set of coefficient equations that can be solved, and this is the essential constraint that limits the choice of interpolants, stencil geometries, and expansion points.

Multidimensional polynomial interpolation has some flexibility in the choice of the stencil that is used, with a concomitant variation in the form of the interpolant. A biquartic interpolant can be written as

$$f(x,y) \approx \sum_{i,j=0}^{4} a(i,j) x^i y^j, \tag{2.7}$$

with function data specified on the twenty-five points of a 5×5 stencil, and with various possibilities for the expansion center. A different possible fourth order interpolant in two space

dimensions is

$$\begin{aligned} f(x,y) \quad \approx \quad & (a_{00} + a_{10}x + a_{20}x^2 + a_{30}x^3 + a_{40}x^4) \\ & +(a_{01} + a_{11}x + a_{21}x^2 + a_{31}x^3 + a_{41}x^4)y \\ & +(a_{02} + a_{12}x + a_{22}x^2 + a_{32}x^3 + a_{42}x^4)y^2 \\ & +(a_{03} + a_{13}x + a_{23}x^2)y^3 \\ & +(a_{04} + a_{14}x + a_{24}x^2)y^4. \end{aligned} \tag{2.8}$$

This interpolant could be used with expansion about the center of a twenty-one point stencil that is obtained by dropping the four corner points of a 5×5 square stencil. In general, there are more choices for interpolants and stencils as the order of the interpolant increases. In the context of algorithm development for the linearized Euler equations, a single step algorithm of order O in both space and time requires a spatial interpolant that includes all cross derivatives through order O.

Multidimensional polynomial interpolation also has flexibility in the choice of the data that is assumed to be known at each grid point. Hermitian polynomial interpolation uses various derivative values as data, with constraint equations from values of the function and its derivatives. As an example, consider a 2×2 stencil with four grid points located at the corners of a square, and with data for f, f_x, f_y, and f_{xy} at each grid point. Note that there are sixteen degrees of freedom of known data, with four along each grid line in either the x or y direction. A suitable third order interpolant for this choice of grid and data is

$$f(x,y) \approx \sum_{i,j=0}^{3} a(i,j) x^i y^j . \tag{2.9}$$

For this local approximation of f, second order interpolants for f_x and f_y are given by

$$\begin{aligned} \tfrac{\partial f}{\partial x}(x,y) \quad &\approx \quad \textstyle\sum_{i=1,j=0}^{3} i a(i,j) x^{i-1} y^j , \\ \tfrac{\partial f}{\partial y}(x,y) \quad &\approx \quad \textstyle\sum_{i=0,j=1}^{3} j a(i,j) x^i y^{j-1}, \end{aligned} \tag{2.10}$$

and a first order interpolant for f_{xy} is given by

$$\frac{\partial^2 f}{\partial x \partial y}(x,y) \approx \sum_{i,j=1}^{3} ij a(i,j) x^{i-1} y^{j-1}. \tag{2.11}$$

In this case, with expansion about the stencil center at the point in the middle of the stencil square, the constraint equations for obtaining the expansion coefficients with function data are,

$$\begin{aligned}
f(+\tfrac{h}{2},+\tfrac{h}{2}) &= a_{00}+a_{10}\tfrac{h}{2}+a_{20}\tfrac{h^2}{4}+a_{30}\tfrac{h^3}{8}+(a_{01}+a_{11}\tfrac{h}{2}+a_{21}\tfrac{h^2}{4}+a_{31}\tfrac{h^3}{8})\tfrac{h}{2} \\
&\quad +(a_{02}+a_{12}\tfrac{h}{2}+a_{22}\tfrac{h^2}{4}+a_{32}\tfrac{h^3}{8})\tfrac{h^2}{4}+(a_{03}+a_{13}\tfrac{h}{2}+a_{23}\tfrac{h^2}{4}+a_{33}\tfrac{h^3}{8})\tfrac{h^3}{8}, \\
f(+\tfrac{h}{2},-\tfrac{h}{2}) &= a_{00}+a_{10}\tfrac{h}{2}+a_{20}\tfrac{h^2}{4}+a_{30}\tfrac{h^3}{8}-(a_{01}+a_{11}\tfrac{h}{2}+a_{21}\tfrac{h^2}{4}+a_{31}\tfrac{h^3}{8})\tfrac{h}{2} \\
&\quad +(a_{02}+a_{12}\tfrac{h}{2}+a_{22}\tfrac{h^2}{4}+a_{32}\tfrac{h^3}{8})\tfrac{h^2}{4}-(a_{03}+a_{13}\tfrac{h}{2}+a_{23}\tfrac{h^2}{4}+a_{33}\tfrac{h^3}{8})\tfrac{h^3}{8}, \\
f(-\tfrac{h}{2},+\tfrac{h}{2}) &= a_{00}-a_{10}\tfrac{h}{2}+a_{20}\tfrac{h^2}{4}-a_{30}\tfrac{h^3}{8}+(a_{01}-a_{11}\tfrac{h}{2}+a_{21}\tfrac{h^2}{4}-a_{31}\tfrac{h^3}{8})\tfrac{h}{2} \\
&\quad +(a_{02}-a_{12}\tfrac{h}{2}+a_{22}\tfrac{h^2}{4}-a_{32}\tfrac{h^3}{8})\tfrac{h^2}{4}+(a_{03}-a_{13}\tfrac{h}{2}+a_{23}\tfrac{h^2}{4}-a_{33}\tfrac{h^3}{8})\tfrac{h^3}{8}, \\
f(-\tfrac{h}{2},-\tfrac{h}{2}) &= a_{00}-a_{10}\tfrac{h}{2}+a_{20}\tfrac{h^2}{4}-a_{30}\tfrac{h^3}{8}-(a_{01}-a_{11}\tfrac{h}{2}+a_{21}\tfrac{h^2}{4}-a_{31}\tfrac{h^3}{8})\tfrac{h}{2} \\
&\quad +(a_{02}-a_{12}\tfrac{h}{2}+a_{22}\tfrac{h^2}{4}-a_{32}\tfrac{h^3}{8})\tfrac{h^2}{4}-(a_{03}-a_{13}\tfrac{h}{2}+a_{23}\tfrac{h^2}{4}-a_{33}\tfrac{h^3}{8})\tfrac{h^3}{8},
\end{aligned} \tag{2.12}$$

and three sets of four equations with similar sign patterns are obtained from the derivative data, with the data for f_x leading to the four constraints

$$\begin{aligned}
f_x(\pm\tfrac{h}{2},\mp\tfrac{h}{2}) &= a_{10}\pm 2a_{20}\tfrac{h}{2}+3a_{30}\tfrac{h^2}{4}\mp(a_{11}\pm 2a_{21}\tfrac{h}{2}+3a_{31}\tfrac{h^2}{4})\tfrac{h}{2} \\
&\quad +(a_{12}\pm 2a_{22}\tfrac{h}{2}+3a_{32}\tfrac{h^2}{4})\tfrac{h^2}{4}\mp(a_{13}\pm 2a_{23}\tfrac{h}{2}+3a_{33}\tfrac{h^2}{4})\tfrac{h^3}{8},
\end{aligned} \tag{2.13}$$

the data for f_y leading to the four constraint equations

$$\begin{aligned}
f_y(\pm\tfrac{h}{2},\mp\tfrac{h}{2}) &= +(a_{01}+a_{11}\tfrac{h}{2}+a_{21}\tfrac{h^2}{4}+a_{31}\tfrac{h^3}{8}) \\
&\quad \mp 2(a_{02}+a_{12}\tfrac{h}{2}+a_{22}\tfrac{h^2}{4}+a_{32}\tfrac{h^3}{8})\tfrac{h}{2}+3(a_{03}+a_{13}\tfrac{h}{2}+a_{23}\tfrac{h^2}{4}+a_{33}\tfrac{h^3}{8})\tfrac{h^2}{4},
\end{aligned} \tag{2.14}$$

and the data for f_{xy} leading to the four constraint equations

$$\begin{aligned}
f_{xy}(\pm\tfrac{h}{2},\mp\tfrac{h}{2}) &= +(a_{11}\pm 2a_{21}\tfrac{h}{2}+3a_{31}\tfrac{h^2}{4}) \\
&\quad \mp 2(a_{12}\pm 2a_{22}\tfrac{h}{2}+3a_{32}\tfrac{h^2}{4})\tfrac{h}{2}+3(a_{13}\pm 2a_{23}\tfrac{h}{2}+3a_{33}\tfrac{h^2}{4})\tfrac{h^2}{4}.
\end{aligned} \tag{2.15}$$

These sixteen equations for the sixteen unknown coefficients are nonsingular, and can be represented in the general form of Equation (2.6).

Polynomial interpolation is expensive in multiple dimensions [31], and other interpolants such as piecewise polynomials [26] are frequently preferred. For the purpose of providing local data approximations in algorithms for the linearized Euler equations, the local order of accuracy of the interpolation is extremely important, since the properties of the MESA type of algorithms are derived from the interpolant. In an earlier design, the interpolating coefficients were computed only once for a particular stencil, and then used repeatedly throughout the spatial grid at every time step, so that the derivation of the interpolants and the computation of the coefficients was amortized over much use. Later, a method analogous to divided differences [89] was found that for higher-order MESA schemes is actually more efficient and is discussed later. Furthermore, the use of local polynomial expansions grounds the MESA algorithm development process in the tradition of finite difference methods.

Posing and solving the interpolation problem is eased considerably by automation with the use of Gröebner bases for mathematical ideals formed by polynomials [21]. In an earlier design this automation solved the entire multidimensional interpolation problem symbolically with the disadvantage of creating large equations, especially in three dimensions. Later, the use of Gröebner bases solutions was necessary only for one-dimensional interpolants. This improvement reduced equation size from millions of lines of code to dozens in the 29^{th} order three-dimensional case. A better method will be described later based an tensor products. The general problem of multidimensional interpolation is an active area of research and will be discussed further in chapter 5.

2.2 Temporal Evolution

Partial differential equations are categorized as elliptic, parabolic, or hyperbolic depending on the higher order derivative terms. Systems of partial differential equations are categorized in the same way. Solutions to equations of each category have unique properties, and numerical methods best suited for their simulation generally reflect these properties. The linearized Euler equations are of hyperbolic type, with propagating waves as solutions.

In one space dimension, the linearized Euler equations are a hyperbolic system, with

$$\begin{aligned} \frac{\partial u}{\partial t} + U\frac{\partial u}{\partial x} + \frac{\partial p}{\partial x} &= 0, \\ \frac{\partial p}{\partial t} + U\frac{\partial p}{\partial x} + \frac{\partial u}{\partial x} &= 0, \end{aligned} \tag{2.16}$$

where u and p are the velocity and pressure of the acoustic disturbance, and where U is the constant mean convection velocity. This system is written in nondimensional form, with U given in terms of the speed of sound, or as a Mach number. The one dimensional system can be diagonalized in terms of the Riemann variables $\omega_1 = u + p$, and $\omega_2 = u - p$. The evolution equations for u and p are added and subtracted to give the equations for ω_1 and ω_2, with

$$\begin{aligned} \frac{\partial \omega_1}{\partial t} + (U+1)\frac{\partial \omega_1}{\partial x} &= 0, \\ \frac{\partial \omega_2}{\partial t} + (U-1)\frac{\partial \omega_2}{\partial x} &= 0. \end{aligned} \tag{2.17}$$

The general solution of these equations is

$$\omega_1(x,t) = \omega_1(x-(U+1)t), \quad \omega_2(x,t) = \omega_2(x-(U-1)t), \tag{2.18}$$

where $\omega_1(x,0) = \omega_1(x)$ and $\omega_2(x,0) = \omega_2(x)$ are the initial data for ω_1 and ω_2. Notice that the solution for ω_1 is constant along the line $x-(U+1)t = \eta$, and that the solution for ω_2 is constant along the line $x-(U-1)t = \xi$. These two lines are the characteristics for the one dimensional system, and information is propagated along them with the characteristic velocities $U+1$ and $U-1$, respectively. Each Riemann variable solution is determined entirely by the data for that solution variable that is propagated along its own characteristic. The solutions for u and p are obtained as linear combinations of the solutions for ω_1 and ω_2, so that u and p are determined by data from both ω_1 and ω_2 that is propagated along each charatcteristic. This analysis and solution process is called the Method of Characteristics, and it is used for the development of numerical methods.

In two or three space dimensions, the linearized Euler equations are hyperbolic but they cannot be transformed into a related system that can be decomposed into separate equations, they are nondiagonalizable. In two or three space dimensions, information is not propagated

from a finite number of directions along characteristic lines, but from every direction along characteristic surfaces. It is commonly observed that a stone thrown into still water creates a ripple or wave that is an expanding circle. If this expanding circular ripple is viewed in the two dimensions of the water surface, and simultaneously in the third dimension of time, then the location of the expanding wave disturbance is seen as a cone. This cone is the characteristic surface for the wave dynamics of the problem. In this example, the wave front is expanding in a circle, which shows the transmission of information in every direction away from the signal source at the center of the disturbance. The inverse view is from a particular point as information arrives to evolve the solution in time. Just as information is radiated in every direction by the governing dynamics, it is also received from every direction. Wave dynamics for nondiagonalizable systems in more than one space dimension are inherently multiple dimensional, and algorithms for their simulation can usefully incorporate this property. This is not the normal practice, but it is fundamental to the MESA method for algorithm development.

The problems typically encountered in acoustic simulations are purely initial value or initial boundary value problems. The purely initial value or Cauchy problem assumes an infinite domain with no surface boundaries, and the data for the problem is given in the form of initial values specified throughout the domain at a particular time. Initial boundary value problems include surface boundaries, and require both initial data throughout the domain and data specified throughout time on the boundary surfaces. Except at or next to a boundary surface, both types of problems can be seen locally in space and time as Cauchy problems. The Cauchy-Kowaleskaya Theorem gives conditions that guarantee a local solution for Cauchy problems [126] . If the governing system has analytic coefficients, and if the initial data is analytic and is defined on an analytic surface in space and time, then a local analytic solution will exist. Polynomials are analytic, so a linear constant coefficient system certainly has analytic coefficients. Any data on a discrete grid can be viewed locally as analytic, since it can be interpolated locally with polynomials. A flat hyperplane defined in space and time by a constant time value is certainly analytic. Consequently, if a local polynomial interpolant at a fixed time on a given stencil is taken as initial data, then the linearized Euler equations will have a local analytic solution in space and time.

An analytic function can be represented as a convergent power series, and the Cauchy-Kowaleskaya Theorem shows how a local series solution can be constructed. The key issue

in developing power series solutions for Cauchy problems is to modify a general Taylor series expansion by using the governing equations to transform time derivatives into space derivatives. The linearized Euler equations give the first time derivatives in terms of first space derivatives. In two space dimensions

$$\frac{\partial u}{\partial t} = -U\frac{\partial u}{\partial x} - V\frac{\partial u}{\partial y} - \frac{\partial p}{\partial x}, \tag{2.19}$$

$$\frac{\partial v}{\partial t} = -U\frac{\partial v}{\partial x} - V\frac{\partial v}{\partial y} - \frac{\partial p}{\partial y}, \tag{2.20}$$

$$\frac{\partial p}{\partial t} = -U\frac{\partial p}{\partial x} - V\frac{\partial p}{\partial y} - \frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}. \tag{2.21}$$

Higher order derivatives with time differentiation follow from these equations, such as

$$\frac{\partial^2 u}{\partial t \partial x} = -U\frac{\partial^2 u}{\partial x^2} - V\frac{\partial^2 u}{\partial y \partial x} - \frac{\partial^2 p}{\partial x^2}, \tag{2.22}$$

and

$$\begin{aligned}
\tfrac{\partial^2 u}{\partial t^2} &= -U\tfrac{\partial^2 u}{\partial x \partial t} - V\tfrac{\partial^2 u}{\partial y \partial t} - \tfrac{\partial^2 p}{\partial x \partial t} \\
&= -U\tfrac{\partial}{\partial x}\left(-U\tfrac{\partial u}{\partial x} - V\tfrac{\partial u}{\partial y} - \tfrac{\partial p}{\partial x}\right) \\
&\quad -V\tfrac{\partial}{\partial y}\left(-U\tfrac{\partial u}{\partial x} - V\tfrac{\partial u}{\partial y} - \tfrac{\partial p}{\partial x}\right) \\
&\quad -\tfrac{\partial}{\partial x}\left(-U\tfrac{\partial p}{\partial x} - V\tfrac{\partial p}{\partial y} - \tfrac{\partial u}{\partial x} - \tfrac{\partial v}{\partial y}\right) \\
&= (U^2+1)\tfrac{\partial^2 u}{\partial x^2} + 2UV\tfrac{\partial^2 u}{\partial x \partial y} + V^2\tfrac{\partial^2 u}{\partial y^2} + \tfrac{\partial^2 v}{\partial x \partial y} + 2U\tfrac{\partial^2 p}{\partial x^2} + 2V\tfrac{\partial^2 p}{\partial x \partial y},
\end{aligned} \tag{2.23}$$

with similar forms for other derivatives if they are needed, and where U and V are constants. The procedure of transforming time derivatives into space derivatives by means of the governing system of equations has been applied before in the development of numerical methods [50]. Note that the mixed xt and the second order t derivatives are both expressed as combinations of second order spatial derivatives. For a first order linear system with constant coefficients such as the linearized Euler equations, the terms with time derivatives are expressed as purely spatial derivatives with the same total order. As a consequence of this, if the initial data is a finite order polynomial, then the order of the time derivative terms that occur in a local analytic solution can only be as high as the highest order pure space derivative terms. This implies that if the

initial data is a local finite order polynomial, then the local convergent power series solution is a finite degree polynomial in space and time. Polynomial closed form exact solutions are possible for the linearized Euler equations in multiple space dimensions.

Consider now the development of an algorithm for the linearized Euler equations in two space dimensions with a 3×3 central stencil. The biquadratic interpolant 2.4 is used to approximate u, v, and p on the stencil. Recall that the biquadratic expansion includes the fourth order coefficient that corresponds to the $\frac{\partial^4}{\partial x^2 \partial y^2}$ derivative. Consequently, up to fourth order time derivatives are possible for the local analytic solution with this initial data. The general solution can be found by the Cauchy-Kowaleskaya procedure, or in the following equivalent manner. Consider local expansions in space and time with the form

$$\begin{aligned} p_a(x,y,t) &= \sum_{i,j=0}^{2} \sum_{k=0}^{4-(i+j)} p(i,j,k) x^i y^j z^k, \\ u_a(x,y,t) &= \sum_{i,j=0}^{2} \sum_{k=0}^{4-(i+j)} u(i,j,k) x^i y^j z^k, \\ v_a(x,y,t) &= \sum_{i,j=0}^{2} \sum_{k=0}^{4-(i+j)} v(i,j,k) x^i y^j z^k, \end{aligned} \tag{2.24}$$

where time terms up to fourth order are included. The terms with $k = 0$ correspond to space derivatives, and their values are obtained from interpolating the data on the stencil. The linearized Euler equations are applied to the expansion forms 2.24, resulting in three expressions in x, y, and t. Since a solution form is being sought, the equations are forced to be satisfied uniformly in all three variables. This requirement results in relationships that define the expansion coefficients with $k \neq 0$ in terms of the coefficients from the spatial interpolant, and results in an exact local solution in space and time, just as if the Cauchy-Kowaleskaya procedure had been used. Because the local approximations in space and time are exact solutions to the linearized Euler equations, they correctly incorporate the wave dynamics of the system. This includes propagation along characteristic surfaces as long as the entire base of the characteristic surface is inside the footprint of the stencil for the spatial interpolation. The entire local approximation in space and time is not needed in order to calculate the evolution of the solution at the stencil center. A solution value at the new time level in the stencil center can be obtained by evaluating the local space and time approximations at $(x, y, t) = (0, 0, k)$ in local coordinates, where $k = \triangle t$

is the time step size. If spatial interpolation is with the biquadratic 2.4, then the resulting exact propagator algorithm is

$$
\begin{aligned}
p_{i,j}^{n+1} &= pa(0,0,k) \\
&= p_{00} \\
&\quad +k(-M_y p_{01} - M_x p_{10} - u_{10} - v_{01}) \\
&\quad +k^2((1+M_y^2)p_{02} + M_x M_y p_{11} + (1+M_x^2)p_{20} \\
&\qquad +M_y u_{11} + 2M_x u_{20} + 2M_y v_{02} + M_x v_{11}) \\
&\quad +k^3(-(M_x + M_x M_y^2)p_{12} - (M_y + M_x^2 M_y)p_{21} \\
&\qquad -(\tfrac{1}{3} + M_y^2)u_{12} - 2M_x M_y u_{21} - 2M_x M_y v_{12} - (\tfrac{1}{3} + M_x^2)v_{21}) \\
&\quad +k^4((\tfrac{1}{3} + M_x^2 + M_y^2 + M_x^2 M_y^2)p_{22} \\
&\qquad +(\tfrac{2}{3}M_x + 2M_x M_y^2)u_{22} + (\tfrac{2}{3}M_y + 2M_x^2 M_y)v_{22}),
\end{aligned}
$$

$$
\begin{aligned}
u_{i,j}^{n+1} &= ua(0,0,k) \\
&= u_{00} \\
&\quad +k(-p_{10} - M_y u_{01} - M_x u_{10}) \\
&\quad +k^2(M_y p_{11} + 2M_x p_{20} + {M_y}^2 u_{02} + M_x M_y u_{11} + (1+{M_x}^2)u_{20} + \tfrac{1}{2}v_{11}) \\
&\quad +k^3(-(\tfrac{1}{3} + {M_y}^2)p_{12} - 2M_x M_y p_{21} \\
&\qquad -M_x {M_y}^2 u_{12} - (M_y + {M_x}^2 M_y)u_{21} - M_y v_{12} - M_x v_{21}) \\
&\quad +k^4((\tfrac{2}{3}M_x + 2M_x {M_y}^2)p_{22} + (\tfrac{1}{6} + {M_y}^2 + {M_x}^2 {M_y}^2)u_{22} + 2M_x M_y v_{22}),
\end{aligned}
\tag{2.25}
$$

$$
\begin{aligned}
v_{i,j}^{n+1} &= va(0,0,k) \\
&= v_{00} \\
&\quad +k(-p_{01} - M_y v_{01} - M_x v_{10}) \\
&\quad +k^2(2M_y p_{02} + M_x p_{11} + \tfrac{1}{2}u_{11} + (1+M_y^2)v_{02} + M_x M_y v_{11} + M_x^2 v_{20}) \\
&\quad +k^3(-2M_x M_y p_{12} - (\tfrac{1}{3} + M_x^2)p_{21} \\
&\qquad -M_y u_{12} - M_x u_{21} - (M_x + M_x M_y^2)v_{12} - M_x^2 M_y v_{21}) \\
&\quad +k^4((\tfrac{2}{3}M_y + 2M_x^2 M_y)p_{22} + 2M_x M_y u_{22} + (\tfrac{1}{6} + M_x^2 + M_x^2 M_y^2)v_{22}).
\end{aligned}
$$

Notice that algorithm 2.25 has the form of a time expansion with terms up to k^4. A second order Taylor series expansion in time is obtained if algorithm 2.25 is truncated by dropping the

$O[k^3]$ and $O[k^4]$ terms. The second order time expansion with a symmetric 3×3 central stencil can be viewed as a two dimensional Lax-Wendroff method [68]. The Lax-Wendroff method does not include the higher order expansion terms that are required to properly propagate the local second order multidimensional spatial interpolant. If both methods are written in the form of Finite Difference algorithms, as weighted sums of the data values on the stencil, then they will both require the same number of multiplies and adds.

The particular algorithm form 2.25 is an example which shows an application of the general MESA method which can also be extended to arbitrary orders of accuracy with different interpolants. If local polynomial interpolants are used with just the solution data assumed known on a symmetric multidimensional stencil, then the resulting numerical algorithms are stable if [34].

$$\lambda = \frac{k}{h} \leq \frac{1}{1 + \max\{|U|, |V|\}}. \tag{2.26}$$

Algorithms with exact propagators can also use Hermitian interpolants, but the resulting algorithms are unstable if central stencils are used. This problem can be avoided if alternating grids are used, with a time separation of $\frac{k}{2}$, and a spatial offset of $\frac{h}{2}$ in both x and y. For these Hermitian methods on staggered grids, the data at each grid level is a real solution, and the numerical algorithms have the same stability limits as the methods which use only the solution data on central stencils. The numerical methods that have been derived for one or two spatial dimensions with Hermitian interpolation have shown spectral like accuracy [35, 37].

Chapter 3

MESA Propagation Algorithm Automation

It is necessary to automate the development of higher order numerical algorithms using the MESA method because of the growth in development complexity as the order of accuracy increases. The efficiency of the higher order algorithms compensates for their complexity.

Perhaps one reason why commonly used numerical schemes are limited in their accuracy in real applications is that higher order schemes and their boundary treatments are too complicated to develop without automated assistance.

For example, the leading computational aeroacoustics algorithms typically use Runge-Kutta methods for computing time derivatives. However, deriving high-order Runge-Kutta methods is no easy task. The first difficulty is in finding the so-called order conditions, which are nonlinear equations. The second difficulty is in solving these equations; There is generally no unique solution, and many heuristics and simplifying assumptions are usually made. In addition, there is the problem of combinatorial explosion. For a twelfth-order method there are 7813 order conditions [123].

A strength of the MESA method for algorithm development is its natural adaptability to the use of computer algebra systems. The method provides order and structure which can be automated. The symbolic differentiation and polynomial system solvers of computer algebra software packages (Mathematica is used in this work) are some of the basic features the automation of

the MESA method depends upon. Mathematica also provides the capabilities typically associated with LISP (LISt Processing language) in which lists of symbols may be data, functions, or both simultaneously. This duality of symbol definition enables the automated FORTRAN code generation utilized in this dissertation.

This chapter will address the automation of the MESA propagation scheme without wall boundaries. The balance of this dissertation will build upon this capability by addressing the incorporation of wall boundaries with complex geometries into the computational domain.

3.1 Spatial Interpolation

The first of two steps required for the MESA scheme discussed in the previous chapter is solving the spatial interpolation problem. Two methods for its automatic solution will be discussed, the General Form and the Tensor Form. In three dimensions spatial interpolation using the General Form was a failure; it simply took too much time using today's technology. It was possible to generate a c2o2 algorithm in about a week on a fast workstation but the number of equations overwhelmed all compilers, even the CRAY's FORTRAN90 compiler. Fortunately, the Tensor Form not only greatly simplfies the solution of the spatial coefficients, but it is actually more efficient for the higher-order algorithms. However, the techniques developed for the General Form are useful for understanding the Tensor Form and a full discussion of its development and application will be presented. In addition, both method's extensions to 3D are discussed.

3.1.1 The Interpolation Problem in 2D

The osculating polynomial interpolation problem solved in this work is defined for an $N \times N$ grid region (see figure 3.1) with data at each grid point that contain the following scalar data up to order 2D at each grid point:

$$\frac{\partial^{i+j} f}{\partial x^i \partial y^j} \forall i, j : (i, j = 0, 1, \ldots, D). \tag{3.1}$$

where $f(x, y) \in \mathcal{C}^{2D}$ is defined in Ω and known at the grid points of the stencil in figure 3.1.

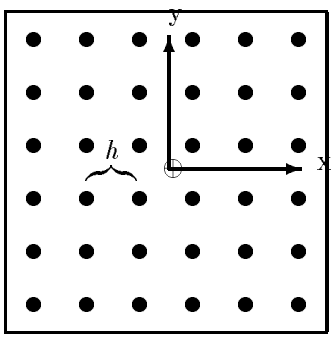

Figure 3.1: Square region Ω of area $((N-1)\times h)^2$ with known data points indicated with dots

The form of the local interpolating function defined over each stencil domain will be:

$$f(x,y) = \sum_i \sum_j a(i,j) x^i y^j \tag{3.2}$$

The coefficients $a(i,j)$ are determined by solving the linear system formed by the list of interpolators 3.2 evaluated at each grid point in the stencil. This linear system was discussed in section 2.1 and is represented as:

$$\mathcal{SA} = \mathcal{Z} \tag{3.3}$$

The unknown coefficient vector $\mathcal{A}$ needs to be solved symbolically. Each coefficient will be a linear combination of stencil data from vector $\mathcal{Z}$. The vector $\mathcal{Z}$ contains algebraic variables as opposed to numerical data. This information will be used later to develop a finite difference scheme for time advancing the stencil's center grid point primitive variables and their spatial derivatives. Since this same stencil will be applied to every grid point in the domain, the actual data values will change; Hence it is necessary to symbolically solve for the vector $\mathcal{A}$.

The osculating polynomial interpolation system 3.3 can be solved in two ways, depending upon the stencil's properties. If the stencil is on an irregular grid then it is necessary to use a less efficient General Form described next which when arranged into a particular form, can be more efficiently solved. The algebraic representations of these more efficient forms are necessarily complicated by notation and do not provide insight into this discussion. A clearer understanding is provided using the pyramid mneumonic discussed next. If the stencil is on a uniform Cartesian grid however, the far more efficient Tensor Form method of spatial interpolation can be used.

But first, the General method will be addressed as this provides insight into the Tensor Form method to be discussed later.

3.1.2 Pyramid Representation of Basis Coefficients in 2D

It is possible to represent the coefficients $a(i,j)$ of equation 3.2 in a pyramid structure that resembles Pascal's triangle like that in figure 3.2. Each pair of numbers in the pyramid corresponds to the pair (i,j). The figure shows the pyramid number 22 corresponds to coefficient a(2,2). Numbers larger than 9 may be represented with hexadecimal digits or some larger number system. In figure 3.2, the first number of the pairs starts at 0 at the top of the pyramid and increases at each level by proceeding down and to the left. The second number in the pairs increase down and to the right. Each horizontal slice of the pyramid contains coefficients representing binomial basis terms of the same order. For example, the level designated by $D = 1$ represents a(3,0), a(2,1), a(1,2), and a(0,3) – all of which represent 3^{rd} order derivatives and multiply a 3^{rd} order binomial term $a(i,j)x^i y^j$ for all $i,j \geq 0$ such that $i + j = 3$.

It is also a property of this pyramid that the set of binomial basis functions represented by the pyramid are of overall minimal order. For example, it would be possible to replace the pair 30 with A0 which represents coefficient a(10,0) and still have a consistent linear system 3.3. But then a gap would form in the pyramid and extra multiplies would be incurred since the interpolant now has the term $a(10,0)x^{10}$ which has ten multiplies instead of the three in the previous case. Also, the time advance of the primitive variables using the MESA scheme requires the a(3,0) spatial coefficient.

In figure 3.2 the coefficients form diamonds (diamond-group) of N^2 elements shown as a dashed diamond. The MESA scheme's stencil dimensions will determine the size of the diamonds in the figure. If the coefficients represented by the numbered pairs contained within a diamond-group are selected and placed in a vector $\mathcal{A}_d$. Then it is possible to symbolically solve these coefficients independently of the other coefficients. The coefficients will be linear combinations of the other coefficients in the pyramid and linear combinations of the stencil data of vector $\mathcal{Z}$ in the right hand side of linear system equation 3.3.

An even smaller set of coefficients can be solved at a time using the ordering shown in figure 3.3. This case is for a 2×2 stencil, with minimal cross-derivatives, and all minimal derivative terms up to the 4^{th} order on the stencil. The numbered pairs (coefficients) are

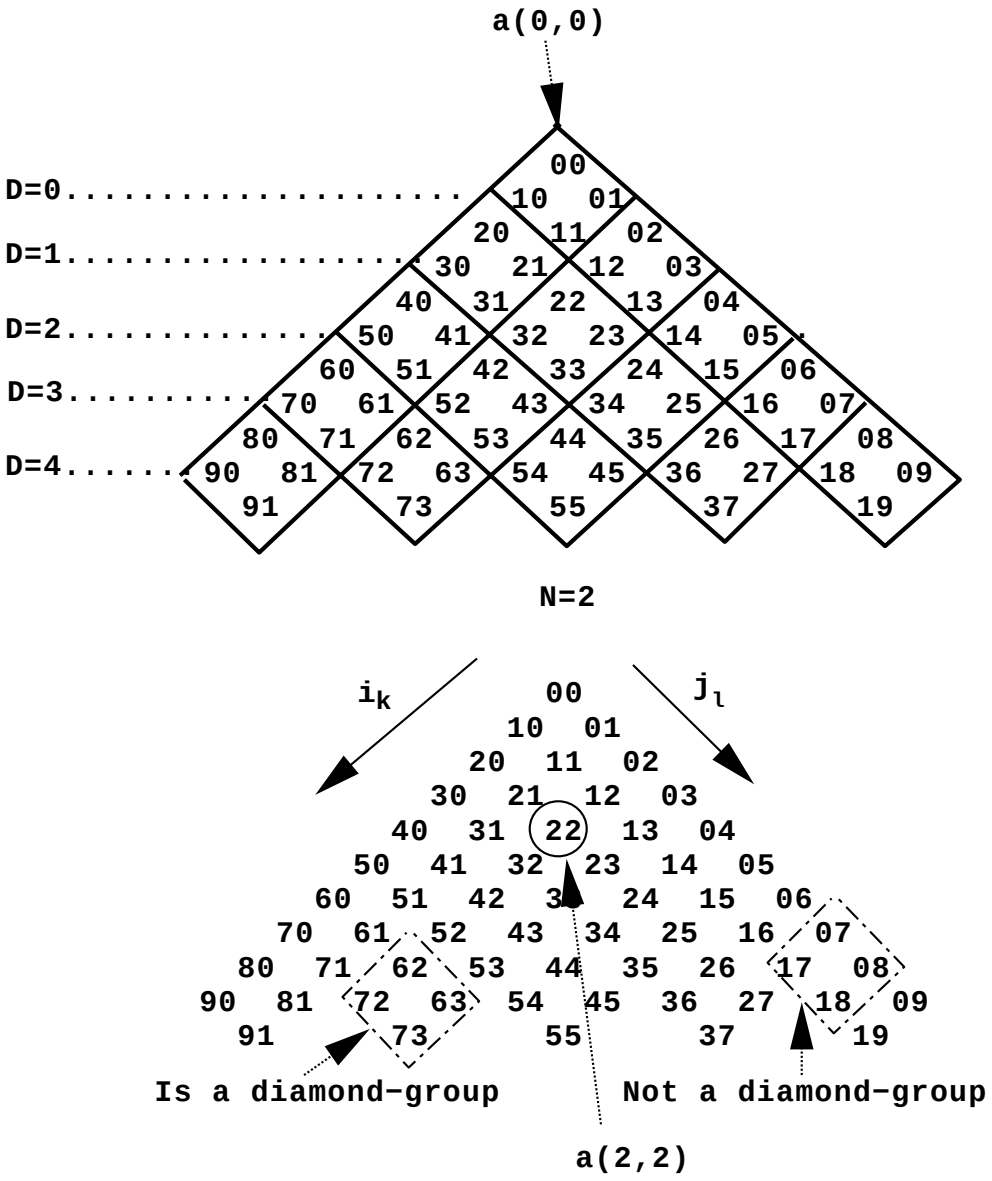

Figure 3.2: Minimal cross-derivative pyramid representation showing diamond sub-structure for $N = 2$ and $D = 4$

00
10 01
20 11 **02**
30 21 **12** 03
40 31 **22** 13 **04**
50 41 **32** 23 **14** 05
60 51 **42** 33 **24** 15 **06**
70 61 **52** 43 **34** 25 **16** 07
80 71 **62** 53 **44** 35 **26** 17 **08**
90 81 **72** 63 **54** 45 **36** 27 **18** 09
91 73 55 37 19

Figure 3.3: Pyramid representation showing line sub-structure for $N = 2$ and $D = 4$

selected from this pyramid in groups of two, by following the darker line from the top to the bottom left in which the right number of each pair is equal to 0. Then following the light line in which the right number of each pair is equal to 1 from top to bottom left. Repeating this for the lines containing pairs with right numbers equal to 2, 3, 4, 5, 6, 7, 8, and 9, in that order. In this manner, all the unknown coefficients may be solved in 30 groups of 2 which require the solution of 30 simple 2×2 matrix systems. Compared to the fifteen 4×4 matrix systems from diamond-groupings, and compared to the one 60×60 matrix systems with no grouping, it is vastly more efficient to use line grouping when symbolically solving the linear system of polynomial equations 3.3.

To obtain better accuracy and isotropy, all the cross-derivative terms are used as shown in figure 3.4. Again, line grouping is used for efficiency. Define the vector $\mathcal{A}$ in equation 3.3 by using the coefficients in the order found by traversing the pyramid in line groups. Now, with $\mathcal{A}$ defined, the matrix $\mathcal{S}$ will be completely defined when the vector $\mathcal{Z}$ is defined. Figure 3.4 provides another mneumonic for ordering $\mathcal{Z}$. The smaller square underneath the now familiar coefficient pyramid (square in this case), provides derivative information (referred to as the derivative pyramid). Each diamond in the coefficient pyramid is linked to the diamond of the same topological location in the derivative pyramid. For example, in the case shown in figure 3.4, there are 25 diamonds in both pyramids (squares). The diamond in the coefficient pyramid containing numbered pairs 42,43,52, and 53, corresponds to the diamond in the derivative pyramid containing the single numbered pair 21. This correspondence will be used to define vector $\mathcal{Z}$. The first number of each numbered pair in the derivative pyramid represents the order of a derivative with respect to the x-axis direction; And the second number of the pair represents the order of a derivative

with respect to the y-axis direction.

The stencil corresponding to $N = 2$ is also shown in figure 3.4. The numbers in the stencil's grid points show the ordering in which the spatial interpolants 3.2 are evaluated. It is the combination of the sequence of the stencil locations and the line groups within the derivative pyramid which determine the precise ordering of the elements in vector $\mathcal{Z}$. If N were larger, the number of elements in the diamonds of the coefficient pyramid and the stencil size would increase. But the derivative pyramid would still contain a single numbered pair per diamond. The larger stencil would still use row-major ordering starting with the bottom row in the bottom left corner of the stencil. The total number of diamonds would not change with larger N, but will increase as D is increased. Larger N and/or larger D both result in more unknown coefficients, a(i,j). The number of coefficients (or degrees of freedom) determines the accuracy of the scheme, regardless of the actual size of N or D. For example, $N = 2$ and $D = 4$ will produce the same accuracy spatial interpolant as $N = 4$ and $D = 2$. However, smaller N results in easier spatial interpolant solutions and larger D results in numerical schemes with better resolution properties. But, very large D values of ten or more can result in significant roundoff error on computer systems with 64-bit precision. This effect is well known when computing divided differences [89].

3.1.3 Polynomial Ideals and Solving $\mathcal{SA} = \mathcal{Z}$ in 2D

The linear system of polynomial equations 3.3 for the interpolation problem can be written in the form

$$I =< \mathcal{SA} - \mathcal{Z} > \subset \mathcal{C}[x, y, z], \tag{3.6}$$

where $\mathcal{C}[x, y, z]$ is the space of polynomials in (x, y, z) with complex coefficients, and where I is the ideal defined by the Hermitian polynomial expansions associated with each grid point. The ideal I has a set of Gröebner basis polynomials that can be used to solve for $\mathcal{A}$. It has been shown that the construction of a Gröebner basis using Buchberger's algorithm for an ideal generated by polynomials of degree less than or equal to d can involve polynomials of degree up to 2^{2^d} [76]. The complexity of constructing the Gröebner basis can be dramatically affected by the ordering of the independent variables [21]. An optimization in the formulation of the Hermitian polynomial interpolation problem dramatically reduces its complexity.

The order of accuracy of the two-dimensional Hermitian polynomial interpolator in this work

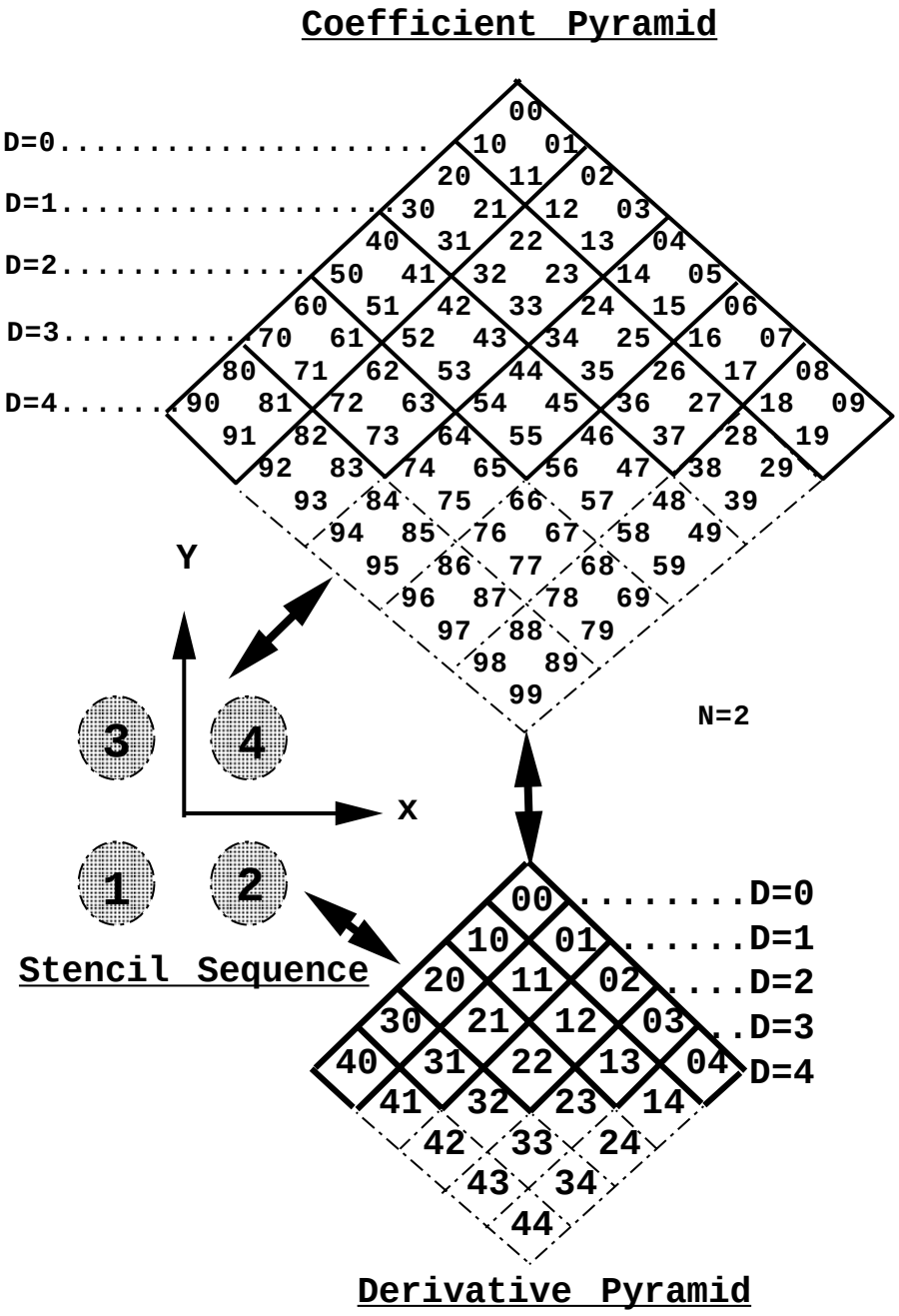

Figure 3.4: Maximal Pyramid representation showing stencil evaluation sequence and derivative assignments for case $N = 2$ and $D = 4$

$$
\begin{array}{cccccccccccc}
1 & \frac{-h}{2} & \frac{h^2}{4} & \frac{-h^3}{8} & \frac{-h}{2} & \frac{h^2}{4} & \frac{-h^3}{8} & \frac{h^4}{16} & \frac{h^2}{4} & \frac{-h^3}{8} & \frac{-h^3}{8} & \frac{h^4}{16} \\
1 & \frac{h}{2} & \frac{h^2}{4} & \frac{h^3}{8} & \frac{-h}{2} & \frac{-h^2}{4} & \frac{-h^3}{8} & \frac{-h^4}{16} & \frac{h^2}{4} & \frac{h^3}{8} & \frac{-h^3}{8} & \frac{-h^4}{16} \\
0 & 1 & -h & \frac{3h^2}{4} & 0 & \frac{-h}{2} & \frac{h^2}{2} & \frac{-3h^3}{8} & 0 & \frac{h^2}{4} & 0 & \frac{-h^3}{8} \\
0 & 1 & h & \frac{3h^2}{4} & 0 & \frac{-h}{2} & \frac{-h^2}{2} & \frac{-3h^3}{8} & 0 & \frac{h^2}{4} & 0 & \frac{-h^3}{8} \\
1 & \frac{-h}{2} & \frac{h^2}{4} & \frac{-h^3}{8} & \frac{h}{2} & \frac{-h^2}{4} & \frac{h^3}{8} & \frac{-h^4}{16} & \frac{h^2}{4} & \frac{-h^3}{8} & \frac{h^3}{8} & \frac{-h^4}{16} \\
1 & \frac{h}{2} & \frac{h^2}{4} & \frac{h^3}{8} & \frac{h}{2} & \frac{h^2}{4} & \frac{h^3}{8} & \frac{h^4}{16} & \frac{h^2}{4} & \frac{h^3}{8} & \frac{h^3}{8} & \frac{h^4}{16} \\
0 & 1 & -h & \frac{3h^2}{4} & 0 & \frac{h}{2} & \frac{-h^2}{2} & \frac{3h^3}{8} & 0 & \frac{h^2}{4} & 0 & \frac{h^3}{8} \\
0 & 1 & h & \frac{3h^2}{4} & 0 & \frac{h}{2} & \frac{h^2}{2} & \frac{3h^3}{8} & 0 & \frac{h^2}{4} & 0 & \frac{h^3}{8} \\
0 & 0 & 0 & 0 & 1 & \frac{-h}{2} & \frac{h^2}{4} & \frac{-h^3}{8} & -h & \frac{h^2}{2} & \frac{3h^2}{4} & \frac{-3h^3}{8} \\
0 & 0 & 0 & 0 & 1 & \frac{h}{2} & \frac{h^2}{4} & \frac{h^3}{8} & -h & \frac{-h^2}{2} & \frac{3h^2}{4} & \frac{3h^3}{8} \\
0 & 0 & 0 & 0 & 1 & \frac{-h}{2} & \frac{h^2}{4} & \frac{-h^3}{8} & h & \frac{-h^2}{2} & \frac{3h^2}{4} & \frac{-3h^3}{8} \\
0 & 0 & 0 & 0 & 1 & \frac{h}{2} & \frac{h^2}{4} & \frac{h^3}{8} & h & \frac{h^2}{2} & \frac{3h^2}{4} & \frac{3h^3}{8}
\end{array}
\tag{3.4}
$$

Figure 3.5: Matrix $\mathcal{S}$ with grid size h, N=2 and D=1, solved by lines

$$
\begin{array}{cccccccccccc}
1 & \frac{-h}{2} & \frac{-h}{2} & \frac{h^2}{4} & \frac{h^2}{4} & \frac{-h^3}{8} & \frac{-h^3}{8} & \frac{h^4}{16} & \frac{h^2}{4} & \frac{-h^3}{8} & \frac{-h^3}{8} & \frac{h^4}{16} \\
1 & \frac{h}{2} & \frac{-h}{2} & \frac{-h^2}{4} & \frac{h^2}{4} & \frac{h^3}{8} & \frac{-h^3}{8} & \frac{-h^4}{16} & \frac{h^2}{4} & \frac{h^3}{8} & \frac{-h^3}{8} & \frac{-h^4}{16} \\
1 & \frac{-h}{2} & \frac{h}{2} & \frac{-h^2}{4} & \frac{h^2}{4} & \frac{-h^3}{8} & \frac{h^3}{8} & \frac{-h^4}{16} & \frac{h^2}{4} & \frac{-h^3}{8} & \frac{h^3}{8} & \frac{-h^4}{16} \\
1 & \frac{h}{2} & \frac{h}{2} & \frac{h^2}{4} & \frac{h^2}{4} & \frac{h^3}{8} & \frac{h^3}{8} & \frac{h^4}{16} & \frac{h^2}{4} & \frac{h^3}{8} & \frac{h^3}{8} & \frac{h^4}{16} \\
0 & 0 & 1 & \frac{-h}{2} & -h & \frac{h^2}{2} & \frac{3h^2}{4} & \frac{-3h^3}{8} & 0 & 0 & \frac{h^2}{4} & \frac{-h^3}{8} \\
0 & 0 & 1 & \frac{h}{2} & -h & \frac{-h^2}{2} & \frac{3h^2}{4} & \frac{3h^3}{8} & 0 & 0 & \frac{h^2}{4} & \frac{h^3}{8} \\
0 & 0 & 1 & \frac{-h}{2} & h & \frac{-h^2}{2} & \frac{3h^2}{4} & \frac{-3h^3}{8} & 0 & 0 & \frac{h^2}{4} & \frac{-h^3}{8} \\
0 & 0 & 1 & \frac{h}{2} & h & \frac{h^2}{2} & \frac{3h^2}{4} & \frac{3h^3}{8} & 0 & 0 & \frac{h^2}{4} & \frac{h^3}{8} \\
0 & 1 & 0 & \frac{-h}{2} & 0 & \frac{h^2}{4} & 0 & \frac{-h^3}{8} & -h & \frac{3h^2}{4} & \frac{h^2}{2} & \frac{-3h^3}{8} \\
0 & 1 & 0 & \frac{-h}{2} & 0 & \frac{h^2}{4} & 0 & \frac{-h^3}{8} & h & \frac{3h^2}{4} & \frac{-h^2}{2} & \frac{-3h^3}{8} \\
0 & 1 & 0 & \frac{h}{2} & 0 & \frac{h^2}{4} & 0 & \frac{h^3}{8} & -h & \frac{3h^2}{4} & \frac{-h^2}{2} & \frac{3h^3}{8} \\
0 & 1 & 0 & \frac{h}{2} & 0 & \frac{h^2}{4} & 0 & \frac{h^3}{8} & h & \frac{3h^2}{4} & \frac{h^2}{2} & \frac{3h^3}{8}
\end{array}
\tag{3.5}
$$

Figure 3.6: Matrix $\mathcal{S}$ with grid size h, N=2 and D=1 solved by diamonds

is based upon the size N of the stencil in one dimension, and the highest order D of the derivative terms on the stencil in one dimension. The order of the spatial interpolator polynomial will be

$$N(D+1)-1, \tag{3.7}$$

where it is assumed that all single dimension derivatives up to order D and various combinations of multiple-dimension (mixed or cross) derivatives up to order $2D$ are included as data at each grid point. For example, an eleventh order scheme ($N = 4, D = 2$) is possible on a square 4×4 stencil with solution data and first, second, third and fourth order (cross) derivatives of the solution data at each grid point. In particular, a single grid point would contain 9 pieces of information, namely:

$$\frac{\partial^{D_m} f(x,y)}{\partial x^r y^s} \; \forall r, s : (r+s = D_m \; and \; D_m < 2) \tag{3.8}$$

or $f(x,y)$, $f_x(x,y)$, $f_{xx}(x,y)$, $f_y(x,y)$, $f_{yy}(x,y)$, $f_{xy}(x,y)$, $f_{xxy}(x,y)$, $f_{xyy}(x,y)$, and $f_{xxyy}(x,y)$.

Solving the entire linear system by directly inverting $\mathcal{S}$ of equation 3.3 exhausts available computer resources for even moderately sized problems. However, by rearranging the interpolant basis matrix $\mathcal{S}$ it is possible to form invertible sub-blocks of size $N \times N$ down the main diagonal that markedly decreases the cost of inverting $\mathcal{S}$. In figure 3.5 an example coefficient matrix $\mathcal{S}$ is configured using line-group ordering; And in figure 3.6 the same matrix $\mathcal{S}$ is configured diamond-groups. The line-group sequence creates a matrix with more zeros in the bottom-left corner of the matrix form for $\mathcal{S}$ than the diamond-group ordering. For even N, the specific ordering for the vector $\mathcal{A}$ of expansion coefficients a(i,j) is:

$$a(0,0), a(1,0), \ldots, a(N(D+1)-1,0), a(0,1), a(1,1), \ldots,$$
$$a(N(D+1)-1,1), a(0,2), \ldots, \ldots, a(N(D+1)-1, N(D+1)-1),$$

The specific ordering for the vector $\mathcal{Z}$ of stencil data is described using the following pseudocode in figure 3.7:

let degree = D
let maxi = $\frac{N}{2}$
let index = 0
for dy = 0 *to* degree *do*
 for j = -maxj *to* maxj *do*
  for dx = 0 *to* degree *do*
   for i = -maxi *to* maxi *do*
    if $(i \neq 0)$ *and* $(j \neq 0)$ *then*
     if $(i < 0)$ *then*
      xcoord=i+1
     else
      xcoord=i
     endif
     if $(j < 0)$ *then*
      ycoord=j+1
     else
      ycoord=j
     endif
     index=index+1
     if N is odd then
      $\mathcal{Z}_{index} = \frac{\partial^{dx+dy} f(x=xcoordh, y=ycoordh)}{\partial^{dx}x \partial^{dy}y}$
     else
      $\mathcal{Z}_{index} = \frac{\partial^{dx+dy} f(x=xcoordh - \frac{|xccord|}{xcoord}\frac{h}{2}, y=ycoordh - \frac{|ycoord|}{ycoord}\frac{h}{2})}{\partial^{dx}x \partial^{dy}y}$
     endif
    endif
   endforloop
  endforloop
 endforloop
endforloop

Figure 3.7: Pseudocode for correct ordering of vector $\mathcal{Z}$

The ordering of the vectors $\mathcal{A}$ and $\mathcal{Z}$ completely determine the form of the matrix $\mathcal{S}$.

The osculating polynomial interpolation function formed will be :

$$f(x,y) = \sum_{j=0}^{N(D+1)-1} \sum_{i=0}^{N(D+1)-1} a(i,j) x^i y^j \tag{3.9}$$

With the system of equations 3.3 formed in the proper order using the line group sequence, it is now possible to efficiently solve the coefficients in $\mathcal{A}$. Starting with the first N equations of the line ordered system 3.3, solve the first N coefficients in $\mathcal{A}$. Then substitute these results into $\mathcal{A}$ and solve the next N coefficients using the next N equations. Proceed in this manner

until all the coefficients are solved for.

After all $N^2(D+1)^2$ coefficients of $\mathcal{A}$ are solved using the above implementation in Mathematica, they will be expressed in the simplest algebraic form as a linear combination of some or all of the elements of $\mathcal{Z}$. Mathematica is developing a parallel version of its software which could be used in solving all line-groups simultaneously. In addition, much research is currently being done in the area of parallel Gröebner basis solution solvers in the field at large. These developments will be useful for some of the irregular grid challenges encountered in complex geometry interpolations.

3.1.4 Tensor Form Method in 2D

The difficulty of the General method just described is that 3D problems are too complicated to solve using current technology, despite the efficiency of using line-group ordering. The length of the algebraic equations produced by the General method for three-dimensional cases can be over a million lines of code. The latter issue can be minimized by combining the spatial interpolant coefficients directly into the full time advance form to reduce the problem to its explicit finite difference form in which the evolved data is a linear combination of the stencil data from the previous time step. In this explicit form, each data element in the stencil has an equation associated with it that is manageable for low order schemes. This simplification only applies to the constant coefficient linearized Euler equations. If the convection velocity is constant then the explicit form reduces to a single constant for each data element in the stencil. This process shifts the burden from the FORTRAN compilers to Mathematica, which can take weeks to produce a single code with explicit finite difference formulations for 3D schemes higher than third order. Moreover, the overall purpose of this research is to solve variable coefficient and nonlinear Navier-Stokes systems; Therefore a better process is required.

The Tensor Form method never attempts to provide an explicit equation for each spatial coefficient. Instead, the solution of the coefficients is expressed as a set of DO loops in FORTRAN that must be executed at each stencil. In this form, it is considerably easier to compile the resulting code and it improves performance since it reuses derivative information in much the same manner as Newton's Interpolatory Divided-Difference algorithm [15]. This improvement is discussed later and it's advantage in three-dimensional applications is demonstrated.

The key idea of the Tensor Form method is to interpolate in only one-dimension at a time.

The function and its derivatives orthogonal to the interpolated direction are interpolated first to the mid-point of the stencil. Then these interpolated values are reused in another one-dimensional interpolation in the next dimension. Finally, if its three dimensional problem, one additional interpolation is performed that reuses the second interpolation results. At no time is more than one-dimension interpolated.

For example, define a one-dimensional interpolating function of order O as :

$$f(x) = \sum_{i=0}^{O} a(i,0)x^i \tag{3.10}$$

and let (x,y)=(0,0) be the center of the stencil.

In 2D, the order of accuracy, O, is determined by the number of data elements along one row of the stencil minus one (counting only those elements whose derivatives are in the same direction as the interpolation). A 2×2 stencil has 2 grid points in either the vertical or horizontal rows. In a horizontal row using the $c2o2$ MESA scheme, f, f_x, f_{xx} are the data elements at each grid point whose derivatives are in the direction of interpolation.

In this case the MESA scheme is 5^{th} order, $(2\times3)-1=5$, and therefore so is equation 3.10.

The $a(i,0)$ are solved in equation 3.10 using computer algebra as described earlier, and can be solved in line-groups as well. An additional simplification is to divide the system into two separate systems, one containing the even and the other containing the odd derivative coefficients. For example, the fifth order case can remove all the odd spatial coefficients terms, (the a(i,0) terms when i is odd), by observing $f(x)+f(-x)$ removes the odd derivatives. Similarly, $\frac{\partial f(x)}{\partial x} - \frac{\partial f(-x)}{\partial x}$ will remove the even terms. Each system is then solved independently for the a(i,0) coefficients In practice, 14^{th} order or higher systems need to be separated into odd and even sub-systems and these sub-systems need solved using line-group ordering.

After the linear system is solved, the spatial coefficients from equation 3.10, $a(i,0)$, will be a linear combination of the function, f(x) and its x-derivatives evaluated at the grid points in one row of the stencil. If the origin is at the center of the stencil then

$$a(i,0) = \frac{1}{i!}\frac{\partial^i f(x)}{\partial x^i} \tag{3.11}$$

The function f(x) in equation 3.10 can represent any function including the y or z-derivatives

of f(x), (f_y, f_{zz}). It is a one-dimensional function that could be applied in any row of the stencil to interpolate f(x) anywhere within that row. Let $f^{dx,dy}(i,j)$ represent the derivative $\frac{\partial^{dx+dy} f}{\partial x^{dx} y^{dy}}$ of the function f at grid point (i,j) in figure 3.9. And assume the function being interpolated in equation 3.10 is $\frac{\partial^{dy} f}{\partial y^{dy}}$. Then for the third order $c2o1$ MESA scheme, the one-dimensional interpolant's spatial coefficients are found to be:

$$\begin{aligned}
a(0,0) &= \frac{f^{(0,dy)}(0,j)}{2} + \frac{f^{(0,dy)}(1,j)}{2} + \frac{h\, f^{(1,dy)}(0,j)}{8} - \frac{h\, f^{(1,dy)}(1,j)}{8} \\
a(1,0) &= \frac{-3\, f^{(0,dy)}(0,j)}{2\,h} + \frac{3\, f^{(0,dy)}(1,j)}{2\,h} - \frac{f^{(1,dy)}(0,j)}{4} - \frac{f^{(1,dy)}(1,j)}{4} \\
a(2,0) &= \frac{-f^{(1,dy)}(0,j)}{2\,h} + \frac{f^{(1,dy)}(1,j)}{2\,h} \\
a(3,0) &= \frac{2\, f^{(0,dy)}(0,j)}{h^3} - \frac{2\, f^{(0,dy)}(1,j)}{h^3} + \frac{f^{(1,dy)}(0,j)}{h^2} + \frac{f^{(1,dy)}(1,j)}{h^2}
\end{aligned} \tag{3.12}$$

where $f^{(dx,dy)}(i,j) = \frac{\partial^{dx+dy} f(x_i,y_j)}{\partial x^{dx} \partial y^{dy}}$. Notice that the spatial coefficients are related to the derivatives of the function as:

$$a(dx,0) = \frac{1}{dx!} \frac{\partial^{dx} f(0,y_j)}{\partial x^{dx}}. \tag{3.13}$$

The (i,j) coordinate system used to determine grid point locations depends upon the parity of the stencil's dimension in one row. Odd dimensioned stencils (3×3,5×5) have a grid point in the center of its stencil and the origin of the (i,j) coordinate system is located in the center of the stencil. Even dimensioned stencils (2×2,4×4) do not have a grid point in the center and so the (i,j) origin is the closest grid point down and to the left of the center of the stencil as shown on the right side of figure 3.9. In both cases however, the spatial interpolants coordinate system's origin is in the center of the stencil.

It is important that the local coordinate system be defined such that its origin is at the center of the stencil. To see this, consider the following one-dimensional interpolation example. Assume there are four collinear grid points labeled, fll, fl, fr, and frr respectively. And we will use a third order interpolant

$$f(x) = \sum_{i=0}^{3} a(i) x^i. \tag{3.14}$$

If we define the center of the 4 point one-dimensional stencil as the origin (the left stencil in

figure 3.8), then the a(i) coefficients are defined as:

$$a(0) = \frac{9\,fl - fll + 9\,fr - frr}{16} \tag{3.15}$$

$$a(1) = \frac{-(27\,fl - fll - 27\,fr + frr)}{24\,h} \tag{3.16}$$

$$a(2) = \frac{-(fl - fll + fr - frr)}{4\,h^2} \tag{3.17}$$

$$a(3) = \frac{-(-3\,fl + fll + 3\,fr - frr)}{6\,h^3} \tag{3.18}$$

Compare this to the new coefficients when the origin is on the fl grid point as in the right stencil of figure 3.8, the a(i) coefficients then become:

$$a(0) = fl \tag{3.19}$$

$$a(1) = \frac{-(3\,fl + 2\,fll - 6\,fr + frr)}{6\,h} \tag{3.20}$$

$$a(2) = \frac{-(2\,fl - fll - fr)}{2\,h^2} \tag{3.21}$$

$$a(3) = \frac{-(-3\,fl + fll + 3\,fr - frr)}{6\,h^3} \tag{3.22}$$

When either of these formulations are evaluated at the center of the stencil, they produce the same interpolated solutions. However, the Tensor Product method requires that the spatial coefficients be directly related to the actual data on the grid. This happens when the center of the stencil is used as the local coordinate system's origin. For example, the second derivative of f(x) evaluated at the midpoint of the stencil with the origin at stencil center is:

$$\frac{\partial^2 f(0)}{\partial x^2} = 2a(2) \tag{3.23}$$

And in general,

$$a(i) = \frac{\partial^i f(0)}{\partial x^i}\frac{1}{i!} \tag{3.24}$$

However, when the origin is at grid point fl as shown in the right stencil of figure 3.8, the second derivative becomes a linear combination of the spatial coefficients:

$$\frac{\partial^2 f(h/2)}{\partial x^2} = 2\,a(2) + 3\,h\,a(3) \tag{3.25}$$

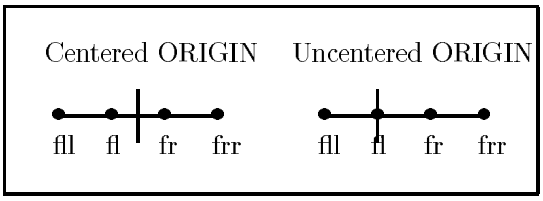

Figure 3.8: Spatial interpolant origin must be at center of stencil

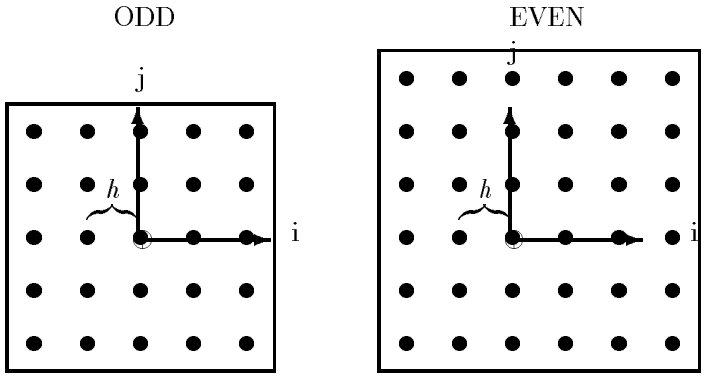

Figure 3.9: Stencil Local Grid Point Coordinate System (i,j)

And it is not possible, using this formulation, to simply substitute function values with spatial coefficients– a key requirement of the Tensor Form method.

Notice that the variables dy and j are undefined in equation 3.12. Changing variable dy corresponds to creating a new one-dimensional interpolation function for interpolating $\frac{\partial^{dy} f}{\partial y^{dy}}$ in the x-direction. Reusing the spatial interpolant coefficients in this manner avoids the need to use time consuming Gröebner basis solvers for each function being interpolated.

In addition, another advantage of the Tensor form of the equations is they may be put into a DO loop in FORTRAN. The y-derivative terms may then be interpolated in the x-direction without recalculating the symbolic form of the spatial coefficients. Since the spatial coefficients a(i,0) are restricted to the one-dimensional case, they are not nearly as complicated as the two and three-dimensional spatial interpolant coefficients tend to be. This results in simple to design FORTRAN code consisting of relatively small equations.

Let s(iindex,dy,j) denote the spatial coefficient a(iindex,0) at row j when the function being interpolated is $\frac{\partial^{dy} f}{\partial y^{dy}}$. Perform the simple DO loop for an even dimensioned stencil:

```
Do[
Do[
Do[
s[iindex,dy,j]=a(iindex,0)
,iindex,0,O]
,dy,0,D]
,j,1-(N/2),N/2];
```

Figure 3.10: Loop to compute the S terms in 2D with even stencil dimensions

or for an odd dimensioned stencil:

```
Do[
Do[
Do[
s(iindex,dy,j)=a(iindex,0)
,iindex,0,O]
,dy,0,D]
,j,-IntegerPart(N/2),IntegerPart(N/2)];
```

Figure 3.11: Loop to compute the S terms in 2D with odd stencil dimensions

and all x-derivatives of function f(x) at the center of the stencil on each row is determined at the S locations as shown in figure 3.12. The variable s(iindex,dy,j) is equal to $\frac{1}{iindex!}\frac{\partial^{iindex+dy} f(0,j)}{\partial x^{iindex} y^{dy}}$.

At this point, we still need the a(i,j) spatial coefficients from equation 3.9 at the center of the stencil (x,y)=(0,0). To get this information, we will now interpolate the data at the S locations to the S2 location in figure 3.12 using another one-dimensional interpolation.

We now create a one-dimensional interpolant function in the y-direction:

$$f(y) = \sum_{j=0}^{O} a(0,j) y^j \tag{3.26}$$

using the same procedures as was used to solve the unknown spatial coefficients in equation 3.10. The unknown spatial coefficients $a(0,j)$ in equation 3.26 correspond to the right most diagonal starting at the top and going down and to the right in the pyramid 3.4. For the c2o1 MESA

EVEN

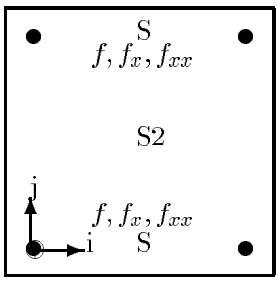

Figure 3.12: Intermediate Derivative Information Storage Locations For c2d2 MESA

scheme, the spatial coefficients are defined as:

$$\begin{aligned}
a(0,0) &= \frac{f^{(dx,0)}(i,0)}{2} + \frac{f^{(dx,0)}(i,1)}{2} + \frac{h\, f^{(dx,1)}(i,0)}{8} - \frac{h\, f^{(dx,1)}(i,1)}{8} \\
a(0,1) &= \frac{-3\, f^{(dx,0)}(i,0)}{2\, h} + \frac{3\, f^{(dx,0)}(i,1)}{2\, h} - \frac{f^{(dx,1)}(i,0)}{4} - \frac{f^{(dx,1)}(i,1)}{4} \\
a(0,2) &= \frac{-f^{(dx,1)}(i,0)}{2\, h} + \frac{f^{(dx,1)}(i,1)}{2\, h} \\
a(0,3) &= \frac{2\, f^{(dx,0)}(i,0)}{h^3} - \frac{2\, f^{(dx,0)}(i,1)}{h^3} + \frac{f^{(dx,1)}(i,0)}{h^2} + \frac{f^{(dx,1)}(i,1)}{h^2}
\end{aligned} \tag{3.27}$$

These coefficients correspond to the y-derivatives of the function f:

$$a(0, jindex) = \frac{1}{jindex!} \frac{\partial^{dx+jindex} f(x_i, 0)}{\partial x^{dx} y^{jindex}} \tag{3.28}$$

Notice that the variables dx and i are undefined and provide the same functions and benefits in the y-direction as discussed for variables dy and j in the x-direction. However, the functions on the right hand side of these equations are already known and are substituted with $s(iindex, dy, j)$ terms from the first set of DO loops the functions f and s are related by:

$$f^{(iindex,dy)}(i,j) = s(iindex, dy, j)(iindex!) \tag{3.29}$$

Substituting with $s(iindex, dy, j)$ instead of with $s(iindex, dy, j)(iindex!)$ has the effect of dividing the equations 3.27 by $iindex!$. This process actually circumvents division roundoff errors which can seriously degrade the very high accuracy MESA schemes.

In two dimensions, the variable i is constant since only the center of the stencil needs derivative information and the previous DO loops put the (f, f_x, f_{xx}) derivative information of the functions $f = f, f_y, f_{yy}$ in the center column at the S locations of figure 3.12.

Now it is possible to solve for all the spatial coefficients in the pyramid using another simple set of DO loops that uses the s(iindex,dy,j) information. The substitution into equation 3.27 is done in Mathematica. Substituting one symbol for another is an important capability of Mathematica and in this case significantly simplifies the solution of the two-dimensional spatial coefficients, a(i,j), in equation3.9.

This final DO loop uses the one-dimensional interpolation in the y-direction to interpolate all the required spatial coefficients at the center of the stencil.

Do[
Do[
s2(iindex,jindex)=a(0,jindex) with $f^{(derx,dery)}(i,j)$ substituted with s(derx,dery,j)
,jindex,0,O]
,iindex,0,O];

Figure 3.13: Loop to compute the S2 terms in 2D

The s2(iindex,jindex) terms correspond to the a(iindex,jindex) terms in equation 3.9. Notice that the problem of finding the spatial coefficients has been reduced to two loops in which one-dimensional interpolations are performed in first the x-direction, and then the y-direction.

3.1.5 The Interpolation Problem in 3D

The 3D interpolation problem is analogous to that described in section 3.1.1. The osculating polynomial is defined on an $N \times N \times N$ grid region with data at each grid point that contain the following scalar data up to order 3D at each grid point:

$$\frac{\partial^{i+j+k} f}{\partial x^i \partial y^j \partial z^k} \forall i, j, k : (i, j, k = 0, 1, \ldots, D). \tag{3.30}$$

The form of the local interpolating function defined over each stencil domain will be:

$$f(x, y, z) = \sum_i \sum_j \sum_k a(i, j, k) x^i y^j z^k \tag{3.31}$$

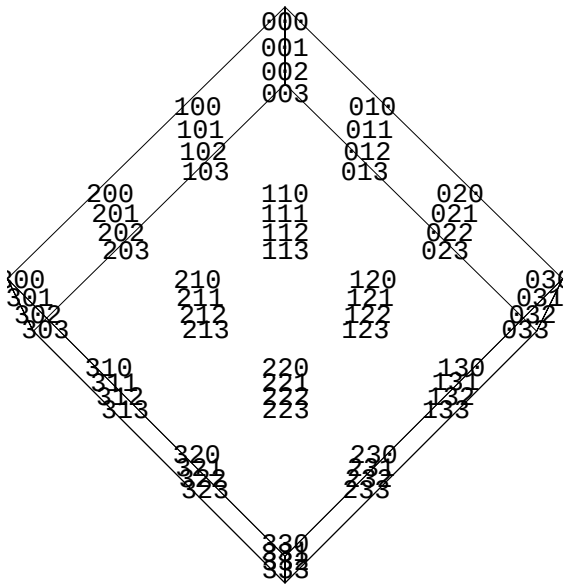

Figure 3.14: Spatial Coefficient Mneumonic for 3D with $N = 2$ and $D = 1$

The coefficients $a(i, j, k)$ are determined by solving the linear system formed by evaluating equation 3.31 at each grid point in the stencil for all data elements.

The representation of the spatial coefficients in three dimensions can be drawn similarly to the pyramid in figure 3.4 as a cube in figure 3.14. Notice the top plane of the cube corresponds to the two-dimensional spatial coefficients for the c2o1 MESA scheme, if all the cross-derivative terms are included. It is still possible to solve these spatial coefficients in sub-groupings as was done in the two-dimensional case. The top plane of the cube in figure 3.14 of coefficients is solved in line-groups first, then the next plane's line-groups, and so forth.

The three-dimensional spatial interpolation problem requires the Tensor Form method since the General method is slow, complicated, and results in equations that are too lengthy for compilation.

3.1.6 Tensor Form Method in 3D

The Tensor Form method provides an efficient procedure for solving the 3D spatial interpolant problem, and produces equations simple enough for today's FORTRAN90 compilers. The Tensor Form requires stencil data to be collinear in the x and y-directions though nonuniform spacing is permitted. Since only Cartesian grids are used in this work, the Tensor Form method is ideal. For problems involving complex geometry however, it is necessary to revert back to the General

method and solve in line-groups. However, it appears it may be possible to symbolically solve all possible cases in a preprocessing step. If this occurs, then a long wait for its General method solution may be acceptable since it will need to be computed only once. In addition, recent developments in parallel Mathematica, parallel Gröebner basis solvers, and parallel computers hold the promise of quickly solving the three-dimensional geometry cases with the General method.

The Tensor Form method in three dimensions uses three sets of loops, one per dimension. The computational savings arises from reusing the data from previous loops and interpolating only in one-dimension at a time. The savings are significant in three-dimensions since a large amount of data is reused and the three-dimensional spatial coefficients, a(i,j,k), in equation 3.31 are solved in one-dimensional slices.

As in the two-dimensional case, define a one-dimensional interpolating function of order O as :

$$f(x) = \sum_{i=0}^{O} a(i,0,0)x^i \tag{3.32}$$

with its coordinate system's origin always at the center of the stencil.

In three dimensional problems, the order O is the number of data elements along one row of the stencil cube in the x-direction that are not a y-derivative or z-derivative, minus one. A $2\times2\times2$ stencil has 2 grid points in a row and f, f_x, f_{xx} are the data elements at each grid point for a c2o2 MESA scheme that contain no y or z-derivatives. The two and three dimensional problems reduce to the same simple set of equations in one-dimension and it is because of this that the Tensor Form method is so successful in 3D (ie. The equation complexity does not change, only the number of loops).

For the c2o1 third order MESA scheme in three-dimensions, the spatial coefficients are solved using Gröebner basis' and Buchberger's algorithm in Mathematica. They are found to be:

$$\begin{aligned}
a(0,0,0) &= \frac{f^{(0,dy,dz)}(0,j,k)}{2} + \frac{f^{(0,dy,dz)}(1,j,k)}{2} + \frac{h\,f^{(1,dy,dz)}(0,j,k)}{8} - \frac{h\,f^{(1,dy,dz)}(1,j,k)}{8} \\
a(1,0,0) &= \frac{-3\,f^{(0,dy,dz)}(0,j,k)}{2\,h} + \frac{3\,f^{(0,dy,dz)}(1,j,k)}{2\,h} - \frac{f^{(1,dy,dz)}(0,j,k)}{4} - \frac{f^{(1,dy,dz)}(1,j,k)}{4} \\
a(2,0,0) &= \frac{-f^{(1,dy,dz)}(0,j,k)}{2\,h} + \frac{f^{(1,dy,dz)}(1,j,k)}{2\,h} \\
a(3,0,0) &= \frac{2\,f^{(0,dy,dz)}(0,j,k)}{h^3} - \frac{2\,f^{(0,dy,dz)}(1,j,k)}{h^3} + \frac{f^{(1,dy,dz)}(0,j,k)}{h^2} + \frac{f^{(1,dy,dz)}(1,j,k)}{h^2}
\end{aligned} \tag{3.33}$$

This is essentially the same form as in the two-dimensional case previously shown.

The local right-handed (i,j,k) coordinate system used is analogous to the (i,j) coordinate system used in 2D in figure 3.9 and represents grid point locations. Odd stencils have their (i,j,k) origin at the center of the stencil and even stencils have an (i,j,k) origin at the closest grid point to the center of the stencil in the direction of descending i,j, and k coordinates. The spatial interpolants origin is however, always at the center of the stencil whether it has an odd or even dimension.

The variables dy,dz,j,and k are undefined in equation 3.33. Assigning values to these is equivalent to creating an interpolation function in the x-direction across the stencil for interpolating the function $\frac{\partial^{dy+dz} f}{\partial y^{dy} \partial z^{dz}}$ along the line intersected by the j and k planes.

In a step analogous to the Tensor Form method in 2D, let s(iindex,dy,dz,j,k) denote the spatial coefficient a(iindex,0,0) in equation 3.32 at the center of the row formed from the intersection of plane j and plane k when the function being interpolated is $\frac{\partial^{dy+dz} f}{\partial y^{dy} \partial z^{dz}}$.

Perform the following set of loops to assign all the s variables for an even dimensioned stencil:

```
Do[Do[ Do[ Do[ Do[
s(iindex,dy,dz,j,k)=a(iindex,0,0)
,iindex,0,O]
,dy,0,D]
,dz,0,D]
,j,1-(N/2),N/2]
,k,1-(N/2),N/2];
```

Figure 3.15: Loop to compute the S terms in 3D with even dimensioned stencils

And for an odd dimensioned stencil use:

```
Do[ Do[ Do[ Do[ Do[
s[iindex,dy,dz,j,k]=a(iindex,0,0)
,iindex,0,O]
,dy,0,D]
,dz,0,D]
,j,-IntegerPart[N/2],IntegerPart[N/2]];
,k,-IntegerPart[N/2],IntegerPart[N/2]];
```

Figure 3.16: Loop to compute the S terms in 3D with odd dimensioned stencils

After that first set of loops, all the interpolations in the x-direction are completed. Now, using that information (location S in figure 3.20), we will interpolate the data at S in the j-direction shown in the figure. Again, define a one-dimensional interpolating function of order O in the y direction as:

$$f(y) = \sum_{j=0}^{O} a(0,j,0) y^j \tag{3.34}$$

This system can be solved using Gröebner basis again or since it will have the same form as the x-direction interpolator, it is possible to symbolically replace the x-direction terms into y-direction terms using a simple shift of indices. This symbolic replacement can be easily performed in Mathematica. Again, the equations are kept simple since they are one-dimensional; This advantage is significant in 3D since it permits fast creation and compilation of the FORTRAN code.

For the c2o1 third order MESA scheme in 3D, the spatial coefficients for the y-direction interpolant are:

$$\begin{aligned}
a(0,0,0) &= \frac{f^{(dx,0,dz)}(i,0,k)}{2} + \frac{f^{(dx,0,dz)}(i,1,k)}{2} + \frac{h\, f^{(dx,1,dz)}(i,0,k)}{8} - \frac{h\, f^{(dx,1,dz)}(i,1,k)}{8} \\
a(0,1,0) &= \frac{-3\, f^{(dx,0,dz)}(i,0,k)}{2\,h} + \frac{3\, f^{(dx,0,dz)}(i,1,k)}{2\,h} - \frac{f^{(dx,1,dz)}(i,0,k)}{4} - \frac{f^{(dx,1,dz)}(i,1,k)}{4} \\
a(0,2,0) &= \frac{-f^{(dx,1,dz)}(i,0,k)}{2\,h} + \frac{f^{(dx,1,dz)}(i,1,k)}{2\,h} \\
a(0,3,0) &= \frac{2\, f^{(dx,0,dz)}(i,0,k)}{h^3} - \frac{2\, f^{(dx,0,dz)}(i,1,k)}{h^3} + \frac{f^{(dx,1,dz)}(i,0,k)}{h^2} + \frac{f^{(dx,1,dz)}(i,1,k)}{h^2}
\end{aligned} \tag{3.35}$$

Since these correspond to a one-dimensional interpolant, they are again essentially the same form as the first set of spatial coefficients used in the i-direction of figure 3.20.

Notice that the variables dx,dz,i, and k are undefined in equation 3.35. Assigning values to these is equivalent to creating an interpolation function in the y-direction across the stencil for interpolating the function $\frac{\partial^{dx+dz} f}{\partial x^{dx}\partial z^{dz}}$. along the line formed by the intersection of the i-plane and k-plane. However, the functions, $f^{(dx,dy,dz)}(i,j,k)$ on the right-hand side of these equations are already determined from the first loop and stored at the S locations of figure 3.20 .

They are defined by:

$$f^{(dx,dy,dz)}(i,j,k) = s(dx,dy,dz,j,k)(dx!)(dy!) \tag{3.36}$$

Substituting s(dx,dy,dz,j,k) for $f^{dx,dy,dz}(i,j,k)$ has the effect of dividing these equations by $(dx!)(dy!)$.

Let s2(iindex,jindex,dz,k) denote the spatial coefficient a(iindex,jindex,0) at the center of plane k when the function being interpolated is $\frac{\partial^{dx+dz} f}{\partial x^{dx} \partial z^{dz}}$. along the line in the j-direction of figure 3.20. Perform the following loops to compute the spatial coefficients at location S2 on each k plane for an even dimensioned stencil:

```
Do[ Do[ Do[ Do[
s2(iindex,jindex,dz,k)=a(0,jindex,0)  with  f^(derx,dery,derz)(i,j,k)  substituted  by
    s(derx,dery,derz,j,k)
,iindex,0,O]
,jindex,0,O]
,dz,0,degree]
,k,1-(N/2),N/2];
```

Figure 3.17: Loop to compute S2 terms in 3D for an even dimensioned stencil

Or perform this set of loops for an odd dimensioned stencil:

```
Do[ Do[ Do[ Do[
s2(iindex,jindex,dz,k)=a(0,jindex,0)  with  f^(derx,dery,derz)(i,j,k)  substituted  by
    s(derx,dery,derz,j,k)
,iindex,0,O]
,jindex,0,O]
,dz,0,degree]
,k,-IntegerPart[N/2],IntegerPart[N/2]];
```

Figure 3.18: Loop to compute S2 terms in 3D for an odd dimensioned stencil

At this point, the spatial coefficients a(i,j,0) for the full 3D spatial interpolant 3.31 are known at the center of each plane k (indicated by S2 in figure 3.20) and will be used in one final set of loops to find all a(i,j,k) at the center of the stencil. This is accomplished by using a one-dimensional interpolant of order O in the z-direction.

$$f(z) = \sum_{k=0}^{O} a(0,0,k) z^k \tag{3.37}$$

This can be solved using Gröebner basis or symbolic modification of the coefficients from the x-direction interpolant's coefficients using Mathematica. For the c2o1 third order MESA scheme in 3D, the spatial coefficients are:

$$
\begin{aligned}
a(0,0,0) &= \frac{f^{(dx,dy,0)}(i,j,0)}{2} + \frac{f^{(dx,dy,0)}(i,j,1)}{2} + \frac{h\, f^{(dx,dy,1)}(i,j,0)}{8} - \frac{h\, f^{(dx,dy,1)}(i,j,1)}{8} \\
a(0,0,1) &= \frac{-3\, f^{(dx,dy,0)}(i,j,0)}{2\,h} + \frac{3\, f^{(dx,dy,0)}(i,j,1)}{2\,h} - \frac{f^{(dx,dy,1)}(i,j,0)}{4} - \frac{f^{(dx,dy,1)}(i,j,1)}{4} \\
a(0,0,2) &= \frac{-f^{(dx,dy,1)}(i,j,0)}{2\,h} + \frac{f^{(dx,dy,1)}(i,j,1)}{2\,h} \\
a(0,0,3) &= \frac{2\, f^{(dx,dy,0)}(i,j,0)}{h^3} - \frac{2\, f^{(dx,dy,0)}(i,j,1)}{h^3} + \frac{f^{(dx,dy,1)}(i,j,0)}{h^2} + \frac{f^{(dx,dy,1)}(i,j,1)}{h^2}
\end{aligned}
\tag{3.38}
$$

The variables dx,dy,i, and j are undefined. Assigning values to these is equivalent to creating an interpolation function in the z-direction across the stencil for interpolating the function $\frac{\partial^{dx+dy} f}{\partial y^{dy} \partial z^{dz}}$ However, the functions, $f^{(dx,dy,dz)}(i,j,k)$ on the right-hand side of these equations are already determined in the previous loop and stored at the S2 locations of figure 3.20.

They are defined to be:

$$
f^{(dx,dy,dz)}(i,j,k) = s2(dx,dy,dz,k)(dx!dy!dz!) \tag{3.39}
$$

Substituting $s2(dx,dy,dz,k)$ for $f^{(dx,dy,dz)}(i,j,k)$ in equations 3.38 has the effect of dividing them by $(dx!dy!dz!)$, which is the required form for the spatial coefficients in equation 3.31.

Let s3(iindex,jindex,kindex) denote the spatial coefficient a(iindex,jindex,kindex) at the center of the stencil when the function being interpolated in the k-direction is $\frac{\partial^{dx+dy} f}{\partial x^{dx} \partial y^{dy}}$.

Perform the following loops to compute all the three-dimensional spatial coefficients at location S3, the center of the stencil,

Do[Do[Do[
s3[iindex,jindex,kindex]=a(0,0,kindex) with $f^{(derx,dery,derz)}(i,j,k)$ substituted by
s2(derx,dery,derz,k)
,kindex,0,O]
,jindex,0,O]
,iindex,0,O];

Figure 3.19: Loop to compute S3 terms in 3D

Simply substitute $s3(i,j,k)$ for $a(i,j,k)$ in equation 3.31 and the 3D interpolation problem is completed.

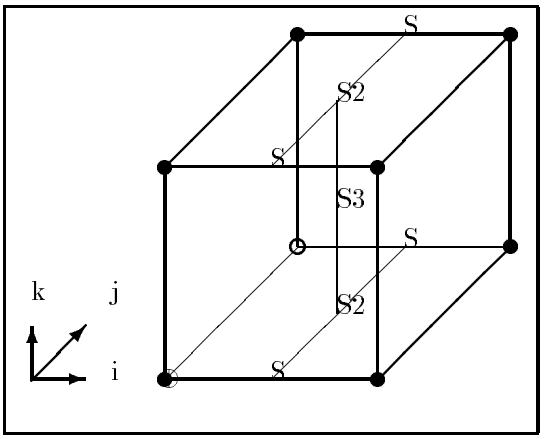

Figure 3.20: S,S2, and S3 Storage Locations For 3D $2 \times 2 \times 2$ stencil

3.2 Temporal Evolution

With the spatial coefficients from equations 3.9 and 3.31 known, the second step of the MESA schemes is the development of a local propagator that evolves the solution to the next time step. For the constant coefficient case of the linearized Euler equations, it is possible to develop a propagator that propagates the waves exactly along the characteristic surfaces originating from the stencil. Three equivalent procedures for implementing this locally defined exact propagator are discussed. The Finite Difference Form is the the most expensive to create and compile but is most efficient to execute in some cases. The Spatial Coefficient Form is easier to create and compile but is the least efficient to execute. The Recursive Tensor Form is the simplest to create, compile and the most efficient to execute on small high resolution stencils in 2D and most cases in 3D. A cost comparison of the methods is made later and shown in tables 3.1 and 3.2.

3.2.1 Finite Difference Form Method in 2D

The Finite Difference form expresses each evolution variable as a linear combination of all the data on the stencil from a previous time step. It results in a simple single-step explicit finite difference scheme. Each data element on the stencil has an associated coefficient for each variable that's evolved. This is achieved by using Mathematica to symbolically simplify the exact propagator form from a linear combination of the spatial coefficients to a linear combination of the stencil data. For example, in equation 2.25 the exact propagator form for the MESA

c3o0 2D case is shown as a linear combination of the spatial coefficients. The spatial coefficient indicated by v_{22} at the end of the first equation 2.25 for $p_{i,j}^{n+1}$ is defined in terms of data elements on the stencil using the techniques of section 3.1 as:

$$\begin{aligned} v_{22} = \quad & (v_{i-1,j-1}^n - 2\,v_{i-1,j+0}^n + v_{i-1,j+1}^n - \\ & 2\,v_{i+0,j-1}^n + 4\,v_{i+0,j+0}^n - 2\,v_{i+0,j+1}^n + \\ & v_{i+1,j-1}^n - 2\,v_{i+1,j+0}^n + v_{i+1,j+1}^n)/(4\,h^4) \end{aligned} \tag{3.40}$$

Each of the spatial coefficients is expanded similarly resulting in a very large evolution equation in terms of stencil data elements. This process is too expensive in 3D. While it is possible to use this process to create 2D codes, the large set of equations (one at each data element in the stencil times the number of data elements at a grid point) are difficult to create and compile. If the convection field is constant, then the set of equations reduce to a set of constants which can be compiled quickly, but this still puts an enormous strain on Mathematica and may take weeks to solve low order 3D problems. A cost analysis (see table 3.1) shows the explicit Finite-Difference Form is not the most efficient to execute in the cases of interest anyway.

3.2.2 Spatial-Temporal Coefficient Form Method in 2D

The Spatial-Temporal Coefficient Form uses the exact propagator as a linear combination of the spatial coefficients. The General Form of the exact local propagator for the primitive variables will be:

$$\begin{aligned} p(x,y,t) &= \sum_{k}^{2(O)} \sum_{j}^{O} \sum_{i}^{\min(O,2(O)-j-k)} cp(i,j,k) x^i y^j t^k \\ u(x,y,t) &= \sum_{k}^{2(O)} \sum_{j}^{O} \sum_{i}^{\min(O,2(O)-j-k)} cu(i,j,k) x^i y^j t^k \\ v(x,y,t) &= \sum_{k}^{2(O)} \sum_{j}^{O} \sum_{i}^{\min(O,2(O)-j-k)} cv(i,j,k) x^i y^j t^k \end{aligned} \tag{3.41}$$

where $cp(i,j,k), cu(i,j,k), cv(i,j,k)$ are arrays of spatial-temporal coefficients. The coefficients with k=0, $(cp(i,j,0), cu(i,j,0), cv(i,j,0))$ are defined using the techniques of section 3.1 and represent the spatial coefficients of each primitive variable.

The conditions for an exact propagator on these equations is:

$$\begin{aligned} p^{(0,0,1)}(x,y,t) + my\,p^{(0,1,0)}(x,y,t) + v^{(0,1,0)}(x,y,t) + mx\,p^{(1,0,0)}(x,y,t) + u^{(1,0,0)}(x,y,t) &= 0 \\ u^{(0,0,1)}(x,y,t) + my\,u^{(0,1,0)}(x,y,t) + p^{(1,0,0)}(x,y,t) + mx\,u^{(1,0,0)}(x,y,t) &= 0 \\ v^{(0,0,1)}(x,y,t) + p^{(0,1,0)}(x,y,t) + my\,v^{(0,1,0)}(x,y,t) + mx\,v^{(1,0,0)}(x,y,t) &= 0 \end{aligned} \tag{3.42}$$

where mx and my is the convection velocity in the x and y directions respectively.

Solving these equations for all values of x, y, and t results in expressing the spatial-temporal coefficients as a linear combination of the purely spatial coefficients. For example, the c2o1 MESA scheme in 2D with an exact local propagator requires the following relationship between the spatial-temporal coefficient and the purely spatial coefficients:

$$\begin{aligned} cp(1,0,5) \;=\; & \left(-3\,my - 9\,mx^2\,my - 3\,my^3 - 3\,mx^2\,my^3\right)\, cp(3,3,0) + \\ & \left(-6\,mx\,my - 6\,mx\,my^3\right)\, cu(3,3,0) + \\ & \left(-\frac{3}{5} - 3\,mx^2 - 3\,my^2 - 9\,mx^2\,my^2\right)\, cv(3,3,0) \end{aligned} \tag{3.43}$$

Letting ($h = \delta x = \delta y$) be the grid spacing and ($lam = \frac{\triangle t}{\triangle x}$) be the CFL time-step fraction, and using the relationship in equation 3.43, and expressing the other spatial-temporal coefficients as a linear combination of the purely spatial coefficients, produces the following MESA algorithm for the pressure variable $p_{i,j}^{n+1} =:$

$$\begin{aligned} & cp(0,0,0) - \\ & h\,lam\,my\,cp(0,1,0) + \\ & \left(h^2\,lam^2 + h^2\,lam^2\,my^2\right)\, cp(0,2,0) + \\ & \left(-3\,h^3\,lam^3\,my - h^3\,lam^3\,my^3\right)\, cp(0,3,0) - \\ & h\,lam\,mx\,cp(1,0,0) + \\ & h^2\,lam^2\,mx\,my\,cp(1,1,0) + \\ & \left(-\left(h^3\,lam^3\,mx\right) - h^3\,lam^3\,mx\,my^2\right)\, cp(1,2,0) + \\ & \left(3\,h^4\,lam^4\,mx\,my + h^4\,lam^4\,mx\,my^3\right)\, cp(1,3,0) + \\ & \left(h^2\,lam^2 + h^2\,lam^2\,mx^2\right)\, cp(2,0,0) + \end{aligned}$$

$$
\begin{aligned}
&\left(-\left(h^3\,lam^3\,my\right)-h^3\,lam^3\,mx^2\,my\right)\ cp(2,1,0)+\\
&\left(\frac{h^4\,lam^4}{3}+h^4\,lam^4\,mx^2+h^4\,lam^4\,my^2+h^4\,lam^4\,mx^2\,my^2\right)\ cp(2,2,0)+\\
&\left(-\left(h^5\,lam^5\,my\right)-3\,h^5\,lam^5\,mx^2\,my-h^5\,lam^5\,my^3-h^5\,lam^5\,mx^2\,my^3\right)\ cp(2,3,0)+\\
&\left(-3\,h^3\,lam^3\,mx-h^3\,lam^3\,mx^3\right)\ cp(3,0,0)+\\
&\left(3\,h^4\,lam^4\,mx\,my+h^4\,lam^4\,mx^3\,my\right)\ cp(3,1,0)+\\
&\left(-\left(h^5\,lam^5\,mx\right)-h^5\,lam^5\,mx^3-3\,h^5\,lam^5\,mx\,my^2-h^5\,lam^5\,mx^3\,my^2\right)\ cp(3,2,0)+\\
&\left(3\,h^6\,lam^6\,mx\,my+3\,h^6\,lam^6\,mx^3\,my+3\,h^6\,lam^6\,mx\,my^3+h^6\,lam^6\,mx^3\,my^3\right)\ cp(3,3,0)-\\
&h\,lam\,cu(1,0,0)+\\
&h^2\,lam^2\,my\,cu(1,1,0)+\\
&\left(\frac{-\left(h^3\,lam^3\right)}{3}-h^3\,lam^3\,my^2\right)\ cu(1,2,0)+\\
&\left(h^4\,lam^4\,my+h^4\,lam^4\,my^3\right)\ cu(1,3,0)+\\
&2\,h^2\,lam^2\,mx\,cu(2,0,0)-\\
&2\,h^3\,lam^3\,mx\,my\,cu(2,1,0)+\\
&\left(\frac{2\,h^4\,lam^4\,mx}{3}+2\,h^4\,lam^4\,mx\,my^2\right)\ cu(2,2,0)+\\
&\left(-2\,h^5\,lam^5\,mx\,my-2\,h^5\,lam^5\,mx\,my^3\right)\ cu(2,3,0)+\\
&\left(-\left(h^3\,lam^3\right)-3\,h^3\,lam^3\,mx^2\right)\ cu(3,0,0)+\\
&\left(h^4\,lam^4\,my+3\,h^4\,lam^4\,mx^2\,my\right)\ cu(3,1,0)+\\
&\left(\frac{-\left(h^5\,lam^5\right)}{5}-h^5\,lam^5\,mx^2-h^5\,lam^5\,my^2-3\,h^5\,lam^5\,mx^2\,my^2\right)\ cu(3,2,0)+\\
&\left(\frac{3\,h^6\,lam^6\,my}{5}+3\,h^6\,lam^6\,mx^2\,my+h^6\,lam^6\,my^3+3\,h^6\,lam^6\,mx^2\,my^3\right)\ cu(3,3,0)-\\
&h\,lam\,cv(0,1,0)+\\
&2\,h^2\,lam^2\,my\,cv(0,2,0)+\\
&\left(-\left(h^3\,lam^3\right)-3\,h^3\,lam^3\,my^2\right)\ cv(0,3,0)+\\
&h^2\,lam^2\,mx\,cv(1,1,0)-\\
&2\,h^3\,lam^3\,mx\,my\,cv(1,2,0)+\\
&\left(h^4\,lam^4\,mx+3\,h^4\,lam^4\,mx\,my^2\right)\ cv(1,3,0)+\\
&\left(\frac{-\left(h^3\,lam^3\right)}{3}-h^3\,lam^3\,mx^2\right)\ cv(2,1,0)+
\end{aligned}
$$

$$\left(\frac{2\,h^4\,lam^4\,my}{3}+2\,h^4\,lam^4\,mx^2\,my\right)\,cv(2,2,0)+$$
$$\left(\frac{-\left(h^5\,lam^5\right)}{5}-h^5\,lam^5\,mx^2-h^5\,lam^5\,my^2-3\,h^5\,lam^5\,mx^2\,my^2\right)\,cv(2,3,0)+$$
$$\left(h^4\,lam^4\,mx+h^4\,lam^4\,mx^3\right)\,cv(3,1,0)+$$
$$\left(-2\,h^5\,lam^5\,mx\,my-2\,h^5\,lam^5\,mx^3\,my\right)\,cv(3,2,0)+$$
$$\left(\frac{3\,h^6\,lam^6\,mx}{5}+h^6\,lam^6\,mx^3+3\,h^6\,lam^6\,mx\,my^2+3\,h^6\,lam^6\,mx^3\,my^2\right)\,cv(3,3,0) \tag{3.44}$$

Notice that the number of spatial coefficients used in this linear combination is equal to the number of data elements in the stencil. Each spatial coefficient has an associated equation on its left hand side just as occurs in the finite difference form, except only one coefficient is assigned here regardless of the number of evolving variables. Let ppkp(i,j),ppku(i,j), and ppkv(i,j) respectively represent the left hand side equation of the terms involving $cp(i,j,0)$, $cu(i,j,0)$, and $cv(i,j,0)$ in equation 3.44.

It is possible to then express the pressure variable's MESA scheme as:

$$p_{i,j}^{n+1}=\sum_{i=0}^{O}\sum_{j=0}^{O}(ppkp(i,j)cp(i,j,0)+ppku(i,j)cu(i,j,0)+ppkv(i,j)cv(i,j,0)) \tag{3.45}$$

A helpful mneumonic is the following matrix form:

$$\begin{array}{cccc}
ppkp(0,0)*cp(0,0,0) & ppkp(0,1)*cp(0,1,0) & ppkp(0,2)*cp(0,2,0) & ppkp(0,3)*cp(0,3,0) \\
ppkp(1,0)*cp(1,0,0) & ppkp(1,1)*cp(1,1,0) & ppkp(1,2)*cp(1,2,0) & ppkp(1,3)*cp(1,3,0) \\
ppkp(2,0)*cp(2,0,0) & ppkp(2,1)*cp(2,1,0) & ppkp(2,2)*cp(2,2,0) & ppkp(2,3)*cp(2,3,0) \\
ppkp(3,0)*cp(3,0,0) & ppkp(3,1)*cp(3,1,0) & ppkp(3,2)*cp(3,2,0) & ppkp(3,3)*cp(3,3,0) \\
\hline
ppku(0,0)*cu(0,0,0) & ppku(0,1)*cu(0,1,0) & ppku(0,2)*cu(0,2,0) & ppku(0,3)*cu(0,3,0) \\
ppku(1,0)*cu(1,0,0) & ppku(1,1)*cu(1,1,0) & ppku(1,2)*cu(1,2,0) & ppku(1,3)*cu(1,3,0) \\
ppku(2,0)*cu(2,0,0) & ppku(2,1)*cu(2,1,0) & ppku(2,2)*cu(2,2,0) & ppku(2,3)*cu(2,3,0) \\
ppku(3,0)*cu(3,0,0) & ppku(3,1)*cu(3,1,0) & ppku(3,2)*cu(3,2,0) & ppku(3,3)*cu(3,3,0) \\
\hline
ppkv(0,0)*cv(0,0,0) & ppkv(0,1)*cv(0,1,0) & ppkv(0,2)*cv(0,2,0) & ppkv(0,3)*cv(0,3,0) \\
ppkv(1,0)*cv(1,0,0) & ppkv(1,1)*cv(1,1,0) & ppkv(1,2)*cv(1,2,0) & ppkv(1,3)*cv(1,3,0) \\
ppkv(2,0)*cv(2,0,0) & ppkv(2,1)*cv(2,1,0) & ppkv(2,2)*cv(2,2,0) & ppkv(2,3)*cv(2,3,0) \\
ppkv(3,0)*cv(3,0,0) & ppkv(3,1)*cv(3,1,0) & ppkv(3,2)*cv(3,2,0) & ppkv(3,3)*cv(3,3,0)
\end{array} \tag{3.46}$$

in which to produce the pressure at the next time step, each of these elements in the matrix form must be evaluated. A similar matrix mneumonic is formed for the u and v velocity variables.

Unlike the explicit finite-difference form of these equations, evolving the derivative of the pressure p_x, will actually require fewer of these matrix elements. The $ppkp(i,j)$, $ppku(i,j)$, $ppkv(i,j)$ coefficients do not depend upon the dimension variables x or y and do not change therefore when the derivative of equation 3.45 with respect to x or y is taken. And, the $cp(i,j,0)$, $cu(i,j,0)$, $cv(i,j,0)$ can simply be shifted in the matrix because of relation 2.2. This is seen by taking the x-derivative of p:

$$\frac{\partial p_{i,j}^{n+1}}{\partial x} = \sum_{i=0}^{O}\sum_{j=0}^{O}(ppkp(i,j)cp(i{+}1,j,0){+}ppku(i,j)cu(i{+}1,j,0){+}ppkv(i,j)cv(i{+}1,j,0)) \tag{3.47}$$

The new mneumonic matrix for p_x becomes:

$$\begin{array}{llll}
ppkp(0,0)*cp(1,0,0) & ppkp(0,1)*cp(1,1,0) & ppkp(0,2)*cp(1,2,0) & ppkp(0,3)*cp(1,3,0) \\
ppkp(1,0)*cp(2,0,0) & ppkp(1,1)*cp(2,1,0) & ppkp(1,2)*cp(2,2,0) & ppkp(1,3)*cp(2,3,0) \\
ppkp(2,0)*cp(3,0,0) & ppkp(2,1)*cp(3,1,0) & ppkp(2,2)*cp(3,2,0) & ppkp(2,3)*cp(3,3,0) \\
ppkp(3,0)*0 & ppkp(3,1)*0 & ppkp(3,2)*0 & ppkp(3,3)*0 \\
\hline
ppku(0,0)*cu(1,0,0) & ppku(0,1)*cu(1,1,0) & ppku(0,2)*cu(1,2,0) & ppku(0,3)*cu(1,3,0) \\
ppku(1,0)*cu(2,0,0) & ppku(1,1)*cu(2,1,0) & ppku(1,2)*cu(2,2,0) & ppku(1,3)*cu(2,3,0) \\
ppku(2,0)*cu(3,0,0) & ppku(2,1)*cu(3,1,0) & ppku(2,2)*cu(3,2,0) & ppku(2,3)*cu(3,3,0) \\
ppku(3,0)*0 & ppku(3,1)*0 & ppku(3,2)*0 & ppku(3,3)*0 \\
\hline
ppkv(0,0)*cv(1,0,0) & ppkv(0,1)*cv(1,1,0) & ppkv(0,2)*cv(1,2,0) & ppkv(0,3)*cv(1,3,0) \\
ppkv(1,0)*cv(2,0,0) & ppkv(1,1)*cv(2,1,0) & ppkv(1,2)*cv(2,2,0) & ppkv(1,3)*cv(2,3,0) \\
ppkv(2,0)*cv(3,0,0) & ppkv(2,1)*cv(3,1,0) & ppkv(2,2)*cv(3,2,0) & ppkv(2,3)*cv(3,3,0) \\
ppkv(3,0)*0 & ppkv(3,1)*0 & ppkv(3,2)*0 & ppkv(3,3)*0
\end{array} \tag{3.48}$$

The zero terms occur since for the c2o1 MESA scheme the higher order derivatives are zero again using the relation 2.2,

$$\begin{aligned}
cp(i>3,j>3,0) &= 0 \\
cu(i>3,j>3,0) &= 0 \\
cv(i>3,j>3,0) &= 0
\end{aligned} \tag{3.49}$$

In general, the higher order derivative evolution variables require less work.

The shifting process is efficiently implemented in the loop shown in figure 3.21 that evolves all the variables:

```
Do[Do[
psum=0;
usum=0;
vsum=0;
Do[
Do[
mfac=((iindex+dx)!*(jindex+dy)!)/((iindex!)*(jindex!));
psum = psum+(mfac*ppkp(iindex,jindex) * cp(iindex+dx,jindex+dy,0)) ;
psum = psum+(mfac*ppku(iindex,jindex) * cu(iindex+dx,jindex+dy,0)) ;
psum = psum+(mfac*ppkv(iindex,jindex) * cv(iindex+dx,jindex+dy,0)) ;
usum = usum+(mfac*uukp(iindex,jindex) * cp(iindex+dx,jindex+dy,0)) ;
usum = usum+(mfac*uuku(iindex,jindex) * cu(iindex+dx,jindex+dy,0)) ;
usum = usum+(mfac*uukv(iindex,jindex) * cv(iindex+dx,jindex+dy,0)) ;
vsum = vsum+(mfac*vvkp(iindex,jindex) * cp(iindex+dx,jindex+dy,0)) ;
vsum = vsum+(mfac*vvku(iindex,jindex) * cu(iindex+dx,jindex+dy,0)) ;
vsum = vsum+(mfac*vvkv(iindex,jindex) * cv(iindex+dx,jindex+dy,0)) ;
,iindex,0,O-dx]
,jindex,0,O-dy];
p^{n+1}_{dx,dy}=psum;
u^{n+1}_{dx,dy}=usum;
v^{n+1}_{dx,dy}=vsum;
,dx,0,D];
,dy,0,D];
```

Figure 3.21: Evolving all variables using the shifted data, 2D case

The factorial is introduced for higher order derivatives and simply multiplies each element of the mneumonic matrix and is a consequence of equation 2.2. Also, notice that the inner loop size decreases as dx and dy are increased. This corresponds to removing a column or multiple rows from matrix 3.46 for each derivative of p. Compare this to the Finite Difference method in which the number of elements in the matrix remains the same for successive derivatives. Also, the $ppkp(i,j), ppku(i,j), ppkv(i,j)$ left hand side equations only need to be computed for the primitive variables and are reused for subsequent derivatives of the primitive variables; While this significantly reduces the number of equations required to be compiled by a factor of $3(D+1)^2$ compared to the Finite Difference method. This is still too many equations and takes too long to create and compile the FORTRAN code for high accuracy 2D and all 3D schemes. Despite the fewer sets of equations, the Spatial-Temporal Form is less efficient because it needs to recompute the spatial coefficients at every stencil.

3.2.3 Recursive Tensor Form Method in 2D

The Recursive Tensor form method eliminates most of the complexity of creating and compiling the FORTRAN code since only the one-dimensional equations created in the 2D or 3D spatial interpolation Tensor Form method are used. This is accomplished by developing a recursive form of the time advance that can be expressed in a simple loop.

The General Form of the exact local propagator in equation 3.41 has coefficients defined similarly to the spatial coefficients of equation 2.2, except the time dimension is added:

$$\begin{aligned} cp(i,j,k) &= \frac{1}{i!j!k!}\frac{\partial^{i+j+k}p}{\partial x^i \partial y^j \partial t^k} \\ cu(i,j,k) &= \frac{1}{i!j!k!}\frac{\partial^{i+j+k}u}{\partial x^i \partial y^j \partial t^k} \\ cv(i,j,k) &= \frac{1}{i!j!k!}\frac{\partial^{i+j+k}v}{\partial x^i \partial y^j \partial t^k} \end{aligned} \tag{3.50}$$

Now substitute i+a,j+b,and k+c for i,j, and k, respectively, into these equations 3.50. Since p,u, and v are analytic functions the mixed derivatives may be permuted and therefore, these equations after the substitution may be written as:

$$\begin{aligned} cp(i+a,j+b,k+c) &= \left(\frac{1}{(i+a)!(j+b)!(k+c)!}\right)\frac{\partial^{a+b+c}\left(\frac{\partial^{i+j+k}p}{\partial x^i \partial y^j \partial t^k}\right)}{\partial x^a \partial y^b \partial t^c} \\ cu(i+a,j+b,k+c) &= \left(\frac{1}{(i+a)!(j+b)!(k+c)!}\right)\frac{\partial^{a+b+c}\left(\frac{\partial^{i+j+k}u}{\partial x^i \partial y^j \partial t^k}\right)}{\partial x^a \partial y^b \partial t^c} \\ cv(i+a,j+b,k+c) &= \left(\frac{1}{(i+a)!(j+b)!(k+c)!}\right)\frac{\partial^{a+b+c}\left(\frac{\partial^{i+j+k}v}{\partial x^i \partial y^j \partial t^k}\right)}{\partial x^a \partial y^b \partial t^c} \end{aligned} \tag{3.51}$$

And then use the basic definition 3.50 to replace the primitive variable derivative terms in equation 3.51 with the spatial-temporal coefficients (cp(i,j,k),cu(i,j,k),cv(i,j,k)).

With these procedures, an expression for the derivatives of the spatial-temporal coefficients of equation 3.41 is found in terms of lower order spatial coefficients. The basic recursive relation

is:

$$\begin{aligned}
\frac{\partial^{a+b+c} cp(i,j,k)}{\partial x^a \partial y^b \partial t^c} &= \frac{(i+a)!(j+b)!(k+c)!}{i!j!k!} cp(i+a,j+b,k+c) \\
\frac{\partial^{a+b+c} cu(i,j,k)}{\partial x^a \partial y^b \partial t^c} &= \frac{(i+a)!(j+b)!(k+c)!}{i!j!k!} cu(i+a,j+b,k+c) \\
\frac{\partial^{a+b+c} cv(i,j,k)}{\partial x^a \partial y^b \partial t^c} &= \frac{(i+a)!(j+b)!(k+c)!}{i!j!k!} cv(i+a,j+b,k+c)
\end{aligned} \tag{3.52}$$

All recurrence relations need a starting condition and this is found from the development of an exact propagator for the linear Euler equations in equation 3.42. Notice that the basic condition equations 3.42 contain only first order derivatives. Therefore the factorial terms in equations 3.50 become unity. The basic condition equations 3.42 can now be rewritten in terms of the spatial-temporal coefficients by simple substitution:

$$\begin{aligned}
cp(0,0,1) + my\, cp(0,1,0) + cv(0,1,0) + mx\, cp(1,0,0) + cu(1,0,0) &= 0 \\
cu(0,0,1) + my\, cu(0,1,0) + cp(1,0,0) + mx\, cu(1,0,0) &= 0 \\
cv(0,0,1) + cp(0,1,0) + my\, cv(0,1,0) - mx\, cv(1,0,0) &= 0
\end{aligned} \tag{3.53}$$

In addition, the derivatives of these equations are also zero.

$$\begin{aligned}
\frac{\partial^{a+b+c}}{\partial x^a y^b z^c} \left(cp(0,0,1) + my\, cp(0,1,0) + cv(0,1,0) + mx\, cp(1,0,0) + cu(1,0,0) \right) &= 0 \\
\frac{\partial^{a+b+c}}{\partial x^a y^b z^c} \left(cu(0,0,1) + my\, cu(0,1,0) + cp(1,0,0) + mx\, cu(1,0,0) \right) &= 0 \\
\frac{\partial^{a+b+c}}{\partial x^a y^b z^c} \left(cv(0,0,1) + cp(0,1,0) + my\, cv(0,1,0) - mx\, cv(1,0,0) \right) &= 0
\end{aligned} \tag{3.54}$$

It is now possible to construct a recursive definition of the spatial-temporal coefficients that generates all the required coefficients. The derivatives in equation 3.54 can now be replaced with products of spatial-temporal coefficients and factorials using equation 3.52. After putting the coefficient term with the highest t-derivative on the left side of the equations the following recurrence relation is found:

$$
\begin{aligned}
cp(a,b,c) &= \\
&\frac{-((1+a)(mx\,cp(1+a,b,-1+c)+cu(1+a,b,-1+c)))-(1+b)(my\,cp(a,1+b,-1+c)+cv(a,1+b,-1+c))}{c} \\
cu(a,b,c) &= \\
&\frac{-((1+b)my\,cu(a,1+b,-1+c))-(1+a)(cp(1+a,b,-1+c)+mx\,cu(1+a,b,-1+c))}{c} \\
cv(a,b,c) &= \\
&\frac{-((1+b)(cp(a,1+b,-1+c)+my\,cv(a,1+b,-1+c)))-(1+a)mx\,cv(1+a,b,-1+c)}{c}
\end{aligned}
\tag{3.55}
$$

Since the left hand side is always a t-derivative of one higher order, it is possible to do the following loop to compute all the spatial-temporal coefficient relationships:

```
Do[Do[Do[
cp[i,j,k]= (-((1 + i)*(mx*cp(1 + i,j,-1 + k) + cu(1 + i,j,-1 + k))) -
(1 + j)*(my*cp(i,1 + j,-1 + k) + cv(i,1 + j,-1 + k)))/k
cu[i,j,k]= (-((1 + j)*my*cu(i,1 + j,-1 + k)) -
(1 + i)*(cp(1 + i,j,-1 + k) + mx*cu(1 + i,j,-1 + k)))/k
cv[i,j,k]= (-((1 + j)*(cp(i,1 + j,-1 + k) + my*cv(i,1 + j,-1 + k))) -
(1 + i)*mx*cv(1 + i,j,-1 + k))/k ,
i,0,Min[O,2*O-k-j] ],
j,0,O ],
k,1,2*O ];
```

Figure 3.22: Loop to compute all spatial-temporal coefficients in 2D

This method does not produce any equations; therefore, the FORTRAN code is easy to create and compile.

After executing that loop, all the spatial-temporal coefficients are known. The evolving variables may be now be time advanced. According to the General Form of the exact local propagator in equation 3.41, only $p(0,0,k)$ is needed since we are using centered difference

schemes and the origin is at the center of the stencil. Therefore, equation 3.41 reduces to:

$$\begin{aligned} p(0,0,t) &= \sum_{k}^{2(O)} cp(0,0,k)t^k \\ u(0,0,t) &= \sum_{k}^{2(O)} cu(0,0,k)t^k \\ v(0,0,t) &= \sum_{k}^{2(O)} cv(0,0,k)t^k \end{aligned} \tag{3.56}$$

Derivatives of these primitive variables require higher-order spatial-temporal coefficients. For example, p_x is:

$$p_x(0,0,t) = \sum_{k}^{2(O)} cp(1,0,k)t^k \tag{3.57}$$

Higher order derivatives introduce factorial terms and must be accounted for. For example, the General Form for the primitive variable's derivatives is:

$$\frac{\partial^{a+b} p(0,0,t)}{\partial x^a y^b} = \sum_{k}^{2(O)} \frac{1}{a!b!} cp(a,b,k)t^k \tag{3.58}$$

This form may be more efficiently computed using the well-known Horner's method [15]. The loop in figure 3.23 efficiently advances all the variables on a stencil using Horner's method:

```
DO[DO[
factterm=fac(dx)*fac(dy)
psum=0.0; usum=0.0; vsum=0.0
DO[
psum=physicaltstep*((factterm * cp(dx,dy,kindex))+psum)
,kindex,2*O,1,-1]
DO[
usum=physicaltstep*((factterm * cu(dx,dy,kindex))+usum)
,kindex,2*O,1,-1]
DO[
vsum=physicaltstep*((factterm * cv(dx,dy,kindex))+vsum)
,kindex,2*O,1,-1]
p^{n+1}_{dx,dy}(i,j)=psum+(cp(dx,dy,0)*factterm)
u^{n+1}_{dx,dy}(i,j)=usum+(cu(dx,dy,0)*factterm)
v^{n+1}_{dx,dy}(i,j)=vsum+(cv(dx,dy,0)*factterm)
,dx,0,D]
,dy,0,D]
```

Figure 3.23: Loop for advancing all variables with Horner's method in 2D

Note that all the spatial-temporal coefficients (cp(i,j,k),cu(i,j,k),cv(i,j,k) are solved in the loop in figure 3.22 first, and that they were computed in an efficient recursive manner by reusing the data from other lower order spatial-temporal coefficients. This efficient reuse of data increases as the number of data elements on a grid point increases and the number of spatial dimensions increases.

3.2.4 Cost Comparison of Methods in 2D

The various approaches to implementing the MESA scheme will now be compared for efficiency. Fortunately, the most efficient method for the applications of interest are also the easiest to create and compile.

In 2D, there are $3(D+1)^2$ data elements per grid point where D is the maximum order derivative term. The number of evolution equations is also $3(D+1)^2$ since all data elements on a grid point need to be propagated using the MESA scheme. A stencil will have $3(N^2)(D+1)^2$ data elements where N is the number of grid points in a row. Since each evolved data element using the Finite Difference Form is a linear combination of all the data contained in the stencil, the number of multiplies required per stencil to advance all the data at the center of the stencil is:

$$9\,(1+D)^4\,N^2 \tag{3.59}$$

For the constant coefficient linearized Euler equations, the only additional cost is evaluating the coefficients of the data elements once at the beginning, but this cost is quickly amortized and therefore will be ignored here.

The Spatial-Temporal Form evolution method can use either the General Form or the Tensor Form of the spatial interpolation. The Tensor Form and General Form are comparable in efficiency when the General Form is solved in line-groups , but the General Form requires too many equations and will not be considered further.

Calculating the cost is a simple matter of counting the number of multiplies performed after all the spatial coefficients are determined. The Tensor Form of spatial interpolation has the following cost to compute the interpolated data at location S in figure 3.12 is bounded by

$$\approx \sum_{j=1-(N/2)}^{N/2} \sum_{dy=0}^{D} \sum_{iindex=0}^{O} O+1 = (1+D)\;N\,(1+O)^2 \tag{3.60}$$

In addition, computing the y-interpolated data at location S2 in figure 3.12 is bounded by the following cost:

$$\approx \sum_{iindex=0}^{O} \sum_{jindex=0}^{O} O+1 = (1+O)^3 \tag{3.61}$$

And the cost of evolving all the stencil data using the Spatial-Temporal Form is:

$$\sum_{dy=0}^{D} \sum_{dx=0}^{D} \sum_{jindex=0}^{O-dy} \sum_{iindex=0}^{O-dx} 21 = \frac{21\,(1+D)^2\,(D-2\,(1+O))^2}{4} \tag{3.62}$$

, assuming the factorials are computed and stored once at the beginning.

If the Recursive Tensor form of the propagator is used, then in addition to the cost of determining the spatial coefficients, with costs 3.60 and 3.61, the cost for determining the spatial-temporal coefficients is:

$$\sum_{k=1}^{2O} \sum_{j=0}^{O} \sum_{i=0}^{O} 15 = 30\,O\,(1+O)^2 \tag{3.63}$$

and the cost for using those spatial-temporal coefficients to evolve all the data elements in the center of the stencil is:

$$\sum_{dy=0}^{D} \sum_{dx=0}^{D} 4 + 3(4O) = 4\,(1+D)^2\,(1+3\,O) \tag{3.64}$$

A cost comparison of these three approaches in two-dimensions is shown in table 3.1. The first column represents the width of the stencil in grid points. The second column is the data depth or equivalently the maximum order derivative in a given direction on a grid point of the stencil.

The third column displays the cost using the Finite Difference Form. In this form, it is assumed that the convection velocity is constant so that a single constant multiplies each data element of the stencil. Most of the computational work is done in Mathematica to reduce it to this form. Time evolution is then accomplished with a simple linear combination of the stencil data.

The fourth column displays the cost of using the Spatial-Temporal Form. In this form it is assumed that the spatial coefficients are found using the Tensor Product form.

The fifth column displays the cost of using the Recursive-Temporal Form in which the spatial coefficients are found using the Tensor Product Form and the time advance is accomplished with

a recursive DO-Loop formulation.

The last column identifies which method is superior for the given MESA algorithm.

Notice in table 3.1 that the Finite Difference form is the best method in most cases, except for the cases which are most useful for this dissertation. The small high-order stencils simplify wall boundary solutions as will be discussed later.

Next, we will discuss the extension of these ideas into three spatial dimensions.

3.2.5 Finite Difference Form Method in 3D

The Finite Difference form expresses each evolution variable as a linear combination of all the data on the stencil from a previous time step as discussed in section 3.2.1. It results in a simple single-step explicit finite difference scheme. Each data element on the stencil has an associated coefficient for each variable that's evolved. This is achieved by using Mathematica to symbolically simplify the exact propagator form from a linear combination of the spatial coefficients to a linear combination of the stencil data. For example, the exact propagator form for the MESA c3o0 3D case for $p^{n+1}_{i,j,k}$ is a linear combination of 159 3D spatial coefficients, $(cp(i,j,k,s), cu(i,j,k,s), cv(i,j,k,s), cw(i,j,k,s))$. The spatial coefficients for $p^{n+1}_{i,j,k}$ are defined in terms of data elements on the stencil using the techniques of section 3.1. The v_{222} is defined as:

$$
\begin{aligned}
cv(2,2,2) = \\
&(v^n_{i-1,j-1,k-1} - 2\,v^n_{i-1,j-1,k+0} + v^n_{i-1,j-1,k+1} - 2\,v^n_{i-1,j+0,k-1} + \\
&4\,v^n_{i-1,j+0,k+0} - 2\,v^n_{i-1,j+0,k+1} + v^n_{i-1,j+1,k-1} - 2\,v^n_{i-1,j+1,k+0} + \\
&v^n_{i-1,j+1,k+1} - 2\,v^n_{i+0,j-1,k-1} + 4\,v^n_{i+0,j-1,k+0} - 2\,v^n_{i+0,j-1,k+1} + \\
&4\,v^n_{i+0,j+0,k-1} - 8\,v^n_{i+0,j+0,k+0} + 4\,v^n_{i+0,j+0,k+1} - 2\,v^n_{i+0,j+1,k-1} + \\
&4\,v^n_{i+0,j+1,k+0} - 2\,v^n_{i+0,j+1,k+1} + v^n_{i+1,j-1,k-1} - 2\,v^n_{i+1,j-1,k+0} + \\
&v^n_{i+1,j-1,k+1} - 2\,v^n_{i+1,j+0,k-1} + 4\,v^n_{i+1,j+0,k+0} - 2\,v^n_{i+1,j+0,k+1} + \\
&v^n_{i+1,j+1,k-1} - 2\,v^n_{i+1,j+1,k+0} + v^n_{i+1,j+1,k+1})/8\,h^6
\end{aligned}
\tag{3.65}
$$

The other 158 spatial coefficients are expanded similarly resulting in a very large evolution equation in terms of stencil data elements. This process is too expensive for three-dimensional

N	D	Finite Difference (FD)	Spatial-Temporal (ST)	Recursive Tensor (RT)	BEST
2	0	1.5563	2.	2.18184	FD
2	1	2.76042	3.06333	3.23754	FD
2	2	3.46479	3.7124	3.80672	FD
2	3	3.96454	4.18241	4.20063	FD
2	4	4.35218	4.55145	4.50243	FD
2	5	4.66891	4.85543	4.74719	FD
2	6	4.93669	5.11393	4.95312	FD
2	7	5.16866	5.33884	5.13087	RT
2	8	5.37327	5.53789	5.28725	RT
2	9	5.5563	5.71644	5.42684	RT
2	10	5.72187	5.87832	5.5529	RT
2	11	5.87303	6.02637	5.66783	RT
2	12	6.01208	6.16279	5.77344	RT
2	13	6.14081	6.28926	5.87112	RT
2	14	6.26067	6.40714	5.96199	RT
2	15	6.37278	6.51752	6.04693	RT
3	0	1.90849	2.38561	2.79379	FD
4	0	2.15836	2.66652	3.20629	FD
4	1	3.36248	3.75959	4.17073	FD
4	2	4.06685	4.42037	4.71767	FD
4	3	4.5666	4.89672	5.10153	FD
4	4	4.95424	5.26975	5.39759	FD
4	5	5.27097	5.57647	5.63866	FD
4	6	5.53875	5.83698	5.84199	FD
4	7	5.77072	6.06341	6.01783	FD
4	8	5.97533	6.26367	6.17274	FD
4	9	6.15836	6.44319	6.31116	FD
4	10	6.32393	6.60587	6.43628	FD
4	11	6.47509	6.7546	6.55043	FD
4	12	6.61414	6.89159	6.65537	FD
4	13	6.74287	7.01855	6.75249	FD
4	14	6.86273	7.13686	6.84287	RT
4	15	6.97484	7.24763	6.92739	RT
5	0	2.35218	2.8893	3.51878	FD
6	0	2.51055	3.07482	3.77056	FD
6	1	3.71466	4.16331	4.71198	FD
6	2	4.41903	4.82142	5.25231	FD
6	3	4.91878	5.29611	5.63303	FD
6	4	5.30643	5.66801	5.92727	FD
6	5	5.62315	5.97393	6.16713	FD
6	6	5.89094	6.23383	6.36962	FD
6	7	6.1229	6.45979	6.54483	FD
6	8	6.32752	6.65967	6.69925	FD
6	9	6.51055	6.83888	6.83729	FD
6	10	6.67612	7.0013	6.96209	FD

Table 3.1: Cost comparison ($\log_{10}$ multiplies per grid point) of 2D methods

applications. The large set of equations (one at each data element in the stencil times the number of data elements at a grid point) are difficult to create and compile. A cost analysis (see table 3.2 shows the explicit Finite-Difference Form is not the most efficient to execute in the cases of interest.

3.2.6 Spatial-Temporal Coefficient Form Method in 3D

The Spatial-Temporal Coefficient Form uses the exact propagator as a linear combination of the spatial coefficients. The General Form of the exact local propagator for the primitive variables will be:

$$p(x,y,z,t) = \sum_{s}^{3(O)} \sum_{k}^{O} \sum_{j}^{O} \sum_{i}^{\min(O,3(O)-i-j-k)} cp(i,j,k,s) x^i y^j z^k t^s \tag{3.66}$$

$$u(x,y,z,t) = \sum_{s}^{3(O)} \sum_{k}^{O} \sum_{j}^{O} \sum_{i}^{\min(O,3(O)-i-j-k)} cu(i,j,k,s) x^i y^j z^k t^s \tag{3.67}$$

$$v(x,y,z,t) = \sum_{s}^{3(O)} \sum_{k}^{O} \sum_{j}^{O} \sum_{i}^{\min(O,3(O)-i-j-k)} cv(i,j,k,s) x^i y^j z^k t^s \tag{3.68}$$

$$w(x,y,z,t) = \sum_{s}^{3(O)} \sum_{k}^{O} \sum_{j}^{O} \sum_{i}^{\min(O,3(O)-i-j-k)} cw(i,j,k,s) x^i y^j z^k t^s \tag{3.69}$$

where $cp(i,j,k,s)$, $cu(i,j,k,s)$, $cv(i,j,k,s)$, and $cw(i,j,k,s)$ are arrays of spatial-temporal coefficients. The coefficients with s=0, ($cp(i,j,k,0)$, $cu(i,j,k,0)$, $cv(i,j,k,0)$, $cw(i,j,k,0)$) are defined using the the techniques of section 3.1 and represent the spatial interpolation coefficients of each primitive variable.

The conditions for an exact propagator to the linearized Euler equations using the General

Form equations 3.66 is:

$$
\begin{aligned}
& p^{(0,0,0,1)}(x,y,z,t) + mz\, p^{(0,0,1,0)}(x,y,z,t) + \\
& w^{(0,0,1,0)}(x,y,z,t) + my\, p^{(0,1,0,0)}(x,y,z,t) + \\
& v^{(0,1,0,0)}(x,y,z,t) + mx\, p^{(1,0,0,0)}(x,y,z,t) + \\
& u^{(1,0,0,0)}(x,y,z,t) && = \quad 0 \\
& u^{(0,0,0,1)}(x,y,z,t) + mz\, u^{(0,0,1,0)}(x,y,z,t) + \\
& my\, u^{(0,1,0,0)}(x,y,z,t) + p^{(1,0,0,0)}(x,y,z,t) + \\
& mx\, u^{(1,0,0,0)}(x,y,z,t) && = \quad 0 \\
& v^{(0,0,0,1)}(x,y,z,t) + mz\, v^{(0,0,1,0)}(x,y,z,t) + \\
& p^{(0,1,0,0)}(x,y,z,t) + my\, v^{(0,1,0,0)}(x,y,z,t) + \\
& mx\, v^{(1,0,0,0)}(x,y,z,t) && = \quad 0 \\
& w^{(0,0,0,1)}(x,y,z,t) + p^{(0,0,1,0)}(x,y,z,t) + \\
& mz\, w^{(0,0,1,0)}(x,y,z,t) + my\, w^{(0,1,0,0)}(x,y,z,t) + \\
& mx\, w^{(1,0,0,0)}(x,y,z,t) && = \quad 0
\end{aligned}
\tag{3.70}
$$

where mx, my, and mz is the convection velocity in the x,y, and z directions respectively.

Solving these equations for all values of dimension variables x,y,z and t results in expressing the spatial-temporal coefficients as a linear combination of the purely spatial coefficients. For example, the c2o1 MESA scheme in 3D with an exact local propagator requires the following

relationship between the spatial-temporal coefficient and the purely spatial coefficients:

$$\begin{aligned} cp(1,0,0,2) = {} & \left(1+mz^2\right)\ cp(1,0,2,0) + \\ & my\, mz\, cp(1,1,1,0) + \\ & \left(1+my^2\right)\ cp(1,2,0,0) + \\ & 2\, mx\, mz\, cp(2,0,1,0) + \\ & 2\, mx\, my\, cp(2,1,0,0) + \\ & \left(3+3\, mx^2\right)\ cp(3,0,0,0) + \\ & 2\, mz\, cu(2,0,1,0) + \\ & 2\, my\, cu(2,1,0,0) + \\ & 6\, mx\, cu(3,0,0,0) + \\ & mz\, cv(1,1,1,0) + \\ & 2\, my\, cv(1,2,0,0) + \\ & 2\, mx\, cv(2,1,0,0) + \\ & 2\, mz\, cw(1,0,2,0) + \\ & my\, cw(1,1,1,0) + \\ & 2\, mx\, cw(2,0,1,0) \end{aligned} \tag{3.71}$$

Using that relationship and expressing the other spatial-temporal coefficients as a linear combination of the purely spatial coefficients, produces the MESA algorithm for the pressure variable $p_{i,j,k}^{n+1}$ as a linear combination of the spatial interpolation coefficients similar to equation 3.44, but much larger in 3D. The number of spatial coefficients used in this linear combination is equal to the number of data elements in the 3D stencil. Each spatial coefficient has an associated equation on its left hand side just as occurs in the finite difference form, except only one coefficient is assigned per data element regardless of the number of evolving variables. Let ppkp(i,j,k),ppku(i,j,k), ppkv(i,j,k), and ppkw(i,j,k) respectively represent the left hand side equation of the terms involving cp(i,j,k,0),cu(i,j,k,0), cv(i,j,k,0) and cw(i,j,k,0) analogous to the 2D case.

It is possible to then express the pressure variable's MESA scheme as:

$$p_{i,j,k}^{n+1} = \sum_{i=0}^{O}\sum_{j=0}^{O}\sum_{k=0}^{O} \begin{array}{l} (ppkp(i,j,k)cp(i,j,k,0) + \\ ppku(i,j,k)cu(i,j,k,0) + \\ ppkv(i,j,k)cv(i,j,k,0) + \\ ppkw(i,j,k)cw(i,j,k,0) \end{array} \tag{3.72}$$

The matrix mneumonic 3.46 used in two-dimensions becomes a tensor or cube-shape in three-dimensions. To evolve the pressure to the next time step, each of the elements in the mneumonic form must be evaluated. A similar mneumonic is formed for the u,v and w velocity variables.

Unlike the explicit finite-difference form of these equations, evolving the derivative of the pressure p_x, will actually require fewer of these mneumonic elements. The $ppkp(i,j,k)$, $ppku(i,j,k)$, $ppkv(i,j,k)$, and $ppkw(i,j,k)$ coefficients do not depend upon the dimension variables x,y or z and do not change therefore when the derivative of equation 3.72 with respect to x,y, or z is taken. And, the $cp(i,j,k,0)$, $cu(i,j,k,0)$, $cv(i,j,k,0)$, and $cw(i,j,k,0)$ can simply be shifted, as was done in the two-dimensional case, in the tensor cube. This is seen by taking the x-derivative of p:

$$\frac{\partial p_{i,j,k}^{n+1}}{\partial x} = \sum_{i=0}^{O}\sum_{j=0}^{O}\sum_{k=0}^{O} \begin{array}{l} (ppkp(i,j,k)cp(i+1,j,k,0) + \\ ppku(i,j,k)cu(i+1,j,k,0) + \\ ppkv(i,j,k)cv(i+1,j,k,0) + \\ ppkw(i,j,k)cw(i+1,j,k,0)) \end{array} \tag{3.73}$$

The new mneumonic 3.48 for p_x in 2D becomes a cube in 3D in which each zero represents a whole role of zeros into the page.

The zero terms occur since the c2o1 MESA scheme's higher order derivatives are zero and

under the derivative interpretation, the spatial coefficients satisfy:

$$cp(i > 3, j > 3, k > 3, 0) = 0 \tag{3.74}$$

$$cu(i > 3, j > 3, k > 3, 0) = 0 \tag{3.75}$$

$$cv(i > 3, j > 3, k > 3, 0) = 0 \tag{3.76}$$

$$cw(i > 3, j > 3, k > 3, 0) = 0 \tag{3.77}$$

In general, the higher order derivatives of the evolution variables require considerably less work in three-dimensions than the finite-difference method because many of the terms become zero and therefore do not need to be multiplied.

The shifting process is efficiently implemented in the loop shown in figure 3.24 that evolves all the variables in the center of a stencil:

```
Do[Do[Do[
psum=0;
usum=0;
vsum=0;
wsum=0;
Do[
Do[
Do[
mfac=((iindex+dx)!*(jindex+dy)!*(kindex+dz)!)/((iindex!)*(jindex!)*(kindex!))
psum=psum+(mfac*ppkp(iindex,jindex,kindex)*cp(iindex+dx,jindex+dy,kindex+dz,0))
psum=psum+(mfac*ppku(iindex,jindex,kindex)*cu(iindex+dx,jindex+dy,kindex+dz,0))
psum=psum+(mfac*ppkv(iindex,jindex,kindex)*cv(iindex+dx,jindex+dy,kindex+dz,0))
psum=psum+(mfac*ppkw(iindex,jindex,kindex)*cw(iindex+dx,jindex+dy,kindex+dz,0))
usum=usum+(mfac*uukp(iindex,jindex,kindex)*cp(iindex+dx,jindex+dy,kindex+dz,0))
usum=usum+(mfac*uuku(iindex,jindex,kindex)*cu(iindex+dx,jindex+dy,kindex+dz,0))
usum=usum+(mfac*uukv(iindex,jindex,kindex)*cv(iindex+dx,jindex+dy,kindex+dz,0))
usum=usum+(mfac*uukw(iindex,jindex,kindex)*cw(iindex+dx,jindex+dy,kindex+dz,0))
vsum=vsum+(mfac*vvkp(iindex,jindex,kindex)*cp(iindex+dx,jindex+dy,kindex+dz,0))
vsum=vsum+(mfac*vvku(iindex,jindex,kindex)*cu(iindex+dx,jindex+dy,kindex+dz,0))
vsum=vsum+(mfac*vvkv(iindex,jindex,kindex)*cv(iindex+dx,jindex+dy,kindex+dz,0))
vsum=vsum+(mfac*vvkw(iindex,jindex,kindex)*cw(iindex+dx,jindex+dy,kindex+dz,0))
wsum=wsum+(mfac*wwkp(iindex,jindex,kindex)*cp(iindex+dx,jindex+dy,kindex+dz,0))
wsum=wsum+(mfac*wwku(iindex,jindex,kindex)*cu(iindex+dx,jindex+dy,kindex+dz,0))
wsum=wsum+(mfac*wwkv(iindex,jindex,kindex)*cv(iindex+dx,jindex+dy,kindex+dz,0))
wsum=wsum+(mfac*wwkw(iindex,jindex,kindex)*cw(iindex+dx,jindex+dy,kindex+dz,0))
,iindex,0,O-dx]
,jindex,0,O-dy];
,kindex,0,O-dz];
p^{n+1}_{dx,dy,dz}=psum;
u^{n+1}_{dx,dy,dz}=usum;
v^{n+1}_{dx,dy,dz}=vsum;
w^{n+1}_{dx,dy,dz}=wsum;
,dx,0,D];
,dy,0,D];
,dz,0,D];
```

Figure 3.24: Evolving all the variables by shifting the data, 3D case

The factorial is introduced for higher order derivatives and simply multiplies each element of the mneumonic matrix as in the two-dimensional case. Also, notice that the inner loop size decreases as dx,dy and dz are increased. This corresponds to removing a vertical plane or multiple horizontal planes from the tensor cube mneumonic for each derivative of p. The

$ppkp(i,j,k), ppku(i,j,k), ppkv(i,j,k), ppkw(i,j,k)$ left hand side equations only need to be computed for the primitive variables and are reused for subsequent derivatives of the primitive variables. While this significantly reduces the number of equations required to be compiled by a factor of $4(D+1)^3$ compared to the Finite Difference method, this is still too many equations and takes too long to create and compile the FORTRAN code for high accuracy 2D and all 3D schemes. Despite the fewer sets of equations, the Spatial-Temporal Form is less efficient than the Finite Difference method because it needs to recompute the coefficients at every stencil. The Finite Difference form has constant coefficients assigned to each element of the stencil; they do not change with position.

3.2.7 Recursive Tensor Form Method in 3D

The Recursive Tensor Form method eliminates most of the complexity of creating and compiling the FORTRAN code since only one-dimensional interpolation equations are used. This is a significant advantage in three dimensions. The three-dimensional spatial Tensor Form method is used to calculate the spatial coefficients. The time evolution formulation is determined in the same manner as the two-dimensional case by developing a recurrence relation for the spatial-temporal coefficients.

The basic relation utilized is:

$$\begin{aligned}
\frac{\partial^{a+b+c+d} cp(i,j,k,s)}{\partial x^a \partial y^b \partial z^c \partial t^s} &= \frac{(i+a)!(j+b)!(k+c)!(s+d)!}{i!j!k!s!} cp(i+a, j+b, k+c, s+d) \\
\frac{\partial^{a+b+c+d} cu(i,j,k,s)}{\partial x^a \partial y^b \partial z^c \partial t^s} &= \frac{(i+a)!(j+b)!(k+c)!(s+d)!}{i!j!k!s!} cu(i+a, j+b, k+c, s+d) \\
\frac{\partial^{a+b+c+d} cv(i,j,k,s)}{\partial x^a \partial y^b \partial z^c \partial t^s} &= \frac{(i+a)!(j+b)!(k+c)!(s+d)!}{i!j!k!s!} cv(i+a, j+b, k+c, s+d) \\
\frac{\partial^{a+b+c+d} cw(i,j,k,s)}{\partial x^a \partial y^b \partial z^c \partial t^s} &= \frac{(i+a)!(j+b)!(k+c)!(s+d)!}{i!j!k!s!} cw(i+a, j+b, k+c, s+d)
\end{aligned} \tag{3.78}$$

which is derived in a manner analogous to the 2D discussion of section 3.2.3.

The equations 3.78 provide a relationship between the derivatives of the spatial-temporal coefficients and can be used to recursively derive all other higher order spatial-temporal coefficients when used in conjunction with the basic condition equations 3.79.

As in the two-dimensional case, notice that the following basic condition equations contain

only first order derivatives.

$$\begin{aligned}
cp(0,0,0,1) &= -mz\,cp(0,0,1,0) - my\,cp(0,1,0,0) - mx\,cp(1,0,0,0) - \\
&\quad cu(1,0,0,0) - cv(0,1,0,0) - cw(0,0,1,0) \qquad (3.79)\\
cu(0,0,0,1) &= -cp(1,0,0,0) - mz\,cu(0,0,1,0) - my\,cu(0,1,0,0) - mx\,cu(1,0,0,0)\\
cv(0,0,0,1) &= -cp(0,1,0,0) - mz\,cv(0,0,1,0) - my\,cv(0,1,0,0) - mx\,cv(1,0,0,0)\\
cw(0,0,0,1) &= -cp(0,0,1,0) - mz\,cw(0,0,1,0) - my\,cw(0,1,0,0) - mx\,cw(1,0,0,0)
\end{aligned}$$

These equations can now be used to find all other spatial-temporal coefficients by taking spatial derivatives of equations 3.79. In particular, the following is true:

$$\begin{aligned}
&\frac{\partial^{a+b+c}}{\partial x^a y^b z^c}(cp(0,0,0,1) + (mz\,cp(0,0,1,0)) + my\,cp(0,1,0,0) + mx\,cp(1,0,0,0) + cu(1,0,0,0) + \\
&\qquad cv(0,1,0,0) + cw(0,0,1,0)) = 0\\
&\frac{\partial^{a+b+c}}{\partial x^a y^b z^c}(cu(0,0,0,1) + cp(1,0,0,0) + mz\,cu(0,0,1,0) + my\,cu(0,1,0,0) + mx\,cu(1,0,0,0)) = 0\\
&\frac{\partial^{a+b+c}}{\partial x^a y^b z^c}(cv(0,0,0,1) + cp(0,1,0,0) + mz\,cv(0,0,1,0) + my\,cv(0,1,0,0) + mx\,cv(1,0,0,0)) = 0\\
&\frac{\partial^{a+b+c}}{\partial x^a y^b z^c}(cw(0,0,0,1) + cp(0,0,1,0) + mz\,cw(0,0,1,0) + my\,cw(0,1,0,0) + mx\,cw(1,0,0,0)) = 0
\end{aligned} \qquad (3.80)$$

Now applying the basic relation equations 3.78 to these rewritten basic condition equation 3.80 and putting the term with highest t-derivative on the left provides the following

relations:

$$
\begin{aligned}
cp(a,b,c,d) &= -((1+c)\ mz\ cp(a,b,1+c,-1+d)+(1+b)\ my\ cp(a,1+b,c,-1+d)+ \\
&\quad (1+a)\ mx\ cp(1+a,b,c,-1+d)+(1+a)\ cu(1+a,b,c,-1+d)+ \\
&\quad (1+b)\ cv(a,1+b,c,-1+d)+(1+c)\ cw(a,b,1+c,-1+d))/d \qquad (3.81) \\
cu(a,b,c,d) &= -((1+a)\ cp(1+a,b,c,-1+d)+(1+c)\ mz\ cu(a,b,1+c,-1+d)+ \\
&\quad (1+b)\ my\ cu(a,1+b,c,-1+d)+(1+a)\ mx\ cu(1+a,b,c,-1+d))/d \\
cv(a,b,c,d) &= -((1+b)\ cp(a,1+b,c,-1+d)+(1+c)\ mz\ cv(a,b,1+c,-1+d)+ \\
&\quad (1+b)\ my\ cv(a,1+b,c,-1+d)+(1+a)\ mx\ cv(1+a,b,c,-1+d))/d \\
cw(a,b,c,d) &= -((1+c)\ cp(a,b,1+c,-1+d)+(1+c)\ mz\ cw(a,b,1+c,-1+d)+ \\
&\quad (1+b)\ my\ cw(a,1+b,c,-1+d)+(1+a)\ mx\ cw(1+a,b,c,-1+d))/d
\end{aligned}
$$

Since the left hand side is always a t-derivative of one higher order, it is possible to do the loop in figure 3.25 to compute all the spatial-temporal coefficient relationships:

```
Do[Do[Do[Do[
cp[i,j,k,s]=-(((1 + k)*mz*cp(i,j,1 + k,-1 + s) +
(1 + j)*my*cp(i,1 + j,k,-1 + s) +
(1 + i)*mx*cp(1 + i,j,k,-1 + s) + (1 + i)*cu(1 + i,j,k,-1 + s) +
(1 + j)*cv(i,1 + j,k,-1 + s) + (1 + k)*cw(i,j,1 + k,-1 + s))/s)
cu[i,j,k,s]=-(((1 + i)*cp(1 + i,j,k,-1 + s) + (1 + k)*mz*cu(i,j,1 + k,-1 + s) +
(1 + j)*my*cu(i,1 + j,k,-1 + s) + (1 + i)*mx*cu(1 + i,j,k,-1 + s))/s)
cv[i,j,k,s]=-(((1 + j)*cp(i,1 + j,k,-1 + s) + (1 + k)*mz*cv(i,j,1 + k,-1 + s) +
(1 + j)*my*cv(i,1 + j,k,-1 + s) + (1 + i)*mx*cv(1 + i,j,k,-1 + s))/s) cw[i,j,k,s]=-(((1
    + k)*cp(i,j,1 + k,-1 + s) + (1 + k)*mz*cw(i,j,1 + k,-1 + s) +
(1 + j)*my*cw(i,1 + j,k,-1 + s) + (1 + i)*mx*cw(1 + i,j,k,-1 + s))/s)
,i,0,Min[O,3*O-k-j-i]]
,j,0,O]
,k,0,O]
,s,1,3*O];
```

Figure 3.25: Loop to compute all the spatial-temporal coefficients in 3D

This method does not produce any more equations, making the FORTRAN code easy to create and compile.

With the spatial-temporal coefficients known, the primitive variables and their spatial derivatives may be now be time advanced. Referring to the General Form of the exact local propagator

in equation 3.66, only $p(0,0,0,t)$ is needed since we are using centered difference schemes and we only wish to time advance the data at the center of the stencil. Therefore, those equations reduce to:

$$p(0,0,0,t) = \sum_{s}^{3(O)} cp(0,0,0,s)t^s \tag{3.82}$$

$$u(0,0,0,t) = \sum_{s}^{3(O)} cu(0,0,0,s)t^s \tag{3.83}$$

$$v(0,0,0,t) = \sum_{s}^{3(O)} cv(0,0,0,s)t^s \tag{3.84}$$

$$w(0,0,0,t) = \sum_{s}^{3(O)} cv(0,0,0,s)t^s \tag{3.85}$$

Derivatives of these primitive variables require higher-order spatial-temporal coefficients. For example, p_x is:

$$p_x(0,0,0,t) = \sum_{s}^{3(O)} cp(1,0,0,s)t^s \tag{3.86}$$

Higher order derivatives introduce factorial terms and must be accounted for. For example, the General Form for the primitive variable's derivatives is:

$$\frac{\partial^{a+b+c} p(0,0,0,t)}{\partial x^a y^b z^c} = \sum_{s}^{3(O)} \frac{1}{a!b!c!} cp(a,b,c,s)t^s \tag{3.87}$$

These equations are in a simple Taylor series form in time and may be more efficiently computed using the well-known Horner's method [15].

The loop in figure 3.26 efficiently advances all the variables on a stencil using Horner's method :

```
DO[DO[DO[
factterm=fac(dx)*fac(dy)*fac(dz)
psum=0.0; usum=0.0; vsum=0.0; wsum=0.0
DO[
psum=physicaltstep*((factterm * cp(dx,dy,dz,sindex))+psum)
,sindex,3*O,1,-1]
DO[
usum=physicaltstep*((factterm * cu(dx,dy,dz,sindex))+usum)
,sindex,3*O,1,-1]
DO[
vsum=physicaltstep*((factterm * cv(dx,dy,dz,sindex))+vsum)
,sindex,3*O,1,-1]
DO[
wsum=physicaltstep*((factterm * cw(dx,dy,dz,sindex))+wsum)
,sindex,3*O,1,-1]
p^{n+1}_{dx,dy,dz}(i,j,k)=psum+(cp(dx,dy,dz,0)*factterm)
u^{n+1}_{dx,dy,dz}(i,j,k)=usum+(cu(dx,dy,dz,0)*factterm)
v^{n+1}_{dx,dy,dz}(i,j,k)=vsum+(cv(dx,dy,dz,0)*factterm)
w^{n+1}_{dx,dy,dz}(i,j,k)=wsum+(cw(dx,dy,dz,0)*factterm)
,dx,0,D]
,dy,0,D]
,dz,0,D]
```

Figure 3.26: Loop for advancing all variables with Horner's method in 3D

Note that all the spatial-temporal coefficients ($cp(i,j,k,s), cu(i,j,k,s), cv(i,j,k,s)$) spatial-temporal are solved in the first set of loops 3.25, then the primitive variables are advanced in the next set of loops. In neither case are extensive equations required. In fact, the size of the equations are constant for all MESA schemes in both of these loops. Only the spatial coefficients require equations to be generated in Mathematica, but these are limited to relatively small one-dimensional interpolation equations.

3.2.8 Cost Comparison of Methods in 3D

The three methods for implementing the MESA suite of algorithms will now be compared for efficiency in terms of multiplication operation counts per grid point. Recall that in two dimensions, the Recursive Tensor Form was the best choice for only a small subset of MESA algorithms– fortunately, they were the algorithms of most interest in this work. Fortunately,

again, the easy to create and compile Recursive Tensor form method also turns out to be successful in three dimensions.

In three dimensions, there are $4(D+1)^3$ data elements per grid point. There will be this same number of evolution equations since all the $4(N^3)(D+1)^3$ data elements need to be propagated using the MESA scheme. Since each evolved data element using the Finite Difference Form is a linear combination of all the stencil's data, the number of multiplies required per stencil is:

$$16\,(1+D)^6\,N^3 \tag{3.88}$$

For the constant coefficient linearized Euler equations, the only additional cost is evaluating the coefficients of the data elements once at the beginning, but this cost is quickly amortized and therefore is negligible.

The Spatial-Temporal Form evolution method can use either the General Form or the Tensor Form of the spatial interpolation. The Tensor Form and General Form are comparable in efficiency when the General Form is solved in line-groups. But the General Form requires too many lengthy, intractable equations and will not be considered further. The Tensor Form of spatial interpolation has the following cost to compute the x-interpolated data at location S in figure 3.20 is bounded by:

$$\approx \sum_{k=1-(N/2)}^{N/2} \sum_{j=1-(N/2)}^{N/2} \sum_{dz=0}^{D} \sum_{dy=0}^{D} \sum_{iindex=0}^{O} O+1 = (1+D)^2\,N^2\,(1+O)^2 \tag{3.89}$$

And the cost to compute the y-interpolated data at location S2 in figure 3.20 is bounded by:

$$\approx \sum_{iindex=0}^{O} \sum_{jindex=0}^{O} \sum_{dz=0}^{D} \sum_{k=1-(N/2)}^{N/2} O+1 = (1+D)\;N\,(1+O)^3 \tag{3.90}$$

Finally, the cost to compute the z-interpolated data at location S3 in figure 3.20 is bounded by:

$$\approx \sum_{iindex=0}^{O} \sum_{jindex=0}^{O} \sum_{kindex=0}^{O} O+1 = (1+O)^4 \tag{3.91}$$

The cost to evolve the primitive variables and their spatial derivatives using the Spatial-

Temporal Form is:

$$\sum_{dz=0}^{D}\sum_{dy=0}^{D}\sum_{dx=0}^{D}\sum_{kindex=0}^{O-dz}\sum_{jindex=0}^{O-dy}\sum_{iindex=0}^{O-dx} 37 = \frac{-37\,(1+D)^3\,(D-2\,(1+O))^3}{8} \tag{3.92}$$

assuming the factorials are computed and stored once at the beginning.

The costs of evolving those same variables using the Recursive Tensor Form is divided between two sets of loops. One loop is for determining the spatial-temporal coefficients and its cost is bounded by:

$$\sum_{s=1}^{3O}\sum_{k=0}^{O}\sum_{j=0}^{O}\sum_{i=0}^{O} 34 = 102\,O\,(1+O)^3 \tag{3.93}$$

The other loop uses those coefficients to evolve the variables and its cost is bounded by:

$$\sum_{dz=0}^{D}\sum_{dy=0}^{D}\sum_{dx=0}^{D} 24O+6 = 6\,(1+D)^3\,(1+4\,O) \tag{3.94}$$

As was done for the two-dimensional case, a table comparing the efficiencies of the three implementations of the MESA scheme is shown in table 3.2. The columns have the same meanings as before in table 3.1. An interesting result here is that the Recursive Tensor Form becomes the best algorithm much sooner in three-dimensions. Notice, however, that a $4\times4\times4$ stencil still requires at least a 23^{rd} order algorithm to be competitive with the Finite-Difference Form. At this level of accuracy, the effects of 64-bit precision dominate the calculation and yields this algorithm ineffective. Therefore, only the $2\times2\times2$ stencil actually benefits from the Recursive Tensor Form. Fortunately, this happens to be the best stencil to use in complex geometries and is used extensively in this work.

3.3 Generating the FORTRAN Propagation Code

The methods of the last section will implement the MESA propagation scheme applied to the constant coefficient linearized Euler equations. The Recursive-Tensor method is the most efficient when small stencils are used and small stencils synergistically work well with the needs of problems involving complex geometries. Generating FORTRAN code is simplified with the Recursive-Tensor form since even the 21^{st} order accuracy method's equations are under 20 lines long in 2D and 3D. If large multidimensional arrays are used it is possible to write many of

N	D	Finite Difference (FD)	Spatial-Temporal (ST)	Recursive Tensor (RT)	BEST
2	0	2.10721	2.53656	2.95134	FD
2	1	3.91339	4.12901	4.32172	FD
2	2	4.96994	5.10979	5.06985	FD
2	3	5.71957	5.82125	5.58994	RT
2	4	6.30103	6.37985	5.98934	RT
2	5	6.77612	6.83975	6.31373	RT
2	6	7.1778	7.23063	6.58691	RT
2	7	7.52575	7.5705	6.82288	RT
2	8	7.83267	7.87116	7.03058	RT
2	9	8.10721	8.14071	7.21606	RT
2	10	8.35557	8.38499	7.38363	RT
2	11	8.5823	8.60833	7.53644	RT
2	12	8.79087	8.81404	7.67689	RT
2	13	8.98398	9.00469	7.80682	RT
2	14	9.16376	9.18235	7.92771	RT
2	15	9.33193	9.34867	8.04072	RT
3	0	2.63548	3.09412	3.7638	FD
4	0	3.0103	3.49638	4.31027	FD
4	1	4.81648	5.13724	5.57892	FD
4	2	5.87303	6.1436	6.30283	FD
4	3	6.62266	6.87016	6.81204	FD
4	4	7.20412	7.43859	7.20525	FD
4	5	7.67921	7.90538	7.52565	RT
4	6	8.08089	8.30134	7.79605	RT
4	7	8.42884	8.64511	8.02996	RT
4	8	8.73576	8.94885	8.23608	RT
4	9	9.0103	9.2209	8.42031	RT
4	10	9.25866	9.46724	8.58686	RT
4	11	9.48539	9.69232	8.73883	RT
4	12	9.69396	9.89951	8.87857	RT
4	13	9.88707	10.0914	9.0079	RT
4	14	10.0668	10.2702	9.12826	RT
4	15	10.235	10.4375	9.24082	RT
5	0	3.30103	3.81291	4.72409	FD
6	0	3.53857	4.07482	5.05757	FD
6	1	5.34475	5.7096	6.30172	FD
6	2	6.4013	6.71792	7.01865	FD
6	3	7.15093	7.44667	7.52455	FD
6	4	7.73239	8.01686	7.91584	FD
6	5	8.20748	8.48501	8.23498	FD
6	6	8.60916	8.88204	8.50449	RT
6	7	8.95711	9.22668	8.73774	RT
6	8	9.26403	9.53111	8.94334	RT
6	9	9.53857	9.80374	9.12717	RT
6	10	9.78693	10.0506	9.29339	RT

Table 3.2: Cost comparison ($\log_{10}$ multiplies per grid point) of 3D methods

these codes by hand as simple FORTRAN DO loops. However, these large multidimensional arrays are difficult to keep in cache and can be less efficient than using multiple DO loops on arrays with fewer dimensions. It is possible to use Mathematica as a precompiler by subdividing large multidimensional arrays into many smaller arrays to take advantage of the RISC cache-based architectures. This leads to more complicated FORTRAN code, but since the computer is writing it and the code is easily compiled–this is a successful approach.

Also, it is necessary for testing purposes to provide initial conditions to all the variables and this can result in many equations. For example, in 3D the c2o10 MESA scheme has 5324 data elements at each grid point. Each data element needs an initial condition assigned to it. This would therefore require 5324 equations be developed and implemented in FORTRAN. This tedious procedure is best accomplished with automatic code generation. Indeed, the ability to quickly generate the many modestly complicated codes in this dissertation was necessary for the grid studies performed; And it has resulted in higher personal productivity.

The idea of generating FORTRAN code using a symbolic manipulator is not new to this work. Most of the computer algebra packages provide facilities for converting an equation into C or FORTRAN and more recently C++. The challenge in this dissertation has been to reduce the complexity of the equations so that they could be coded and compiled. The Recursive Tensor Form implementation and the Tensor Form spatial interpolator combine to provide relatively simple codes; the 21^{st} order c2o10 3D MESA scheme can be translated into a fully independent FORTRAN code in less than 5 minutes and compiled in about the same time.

A key benefit of automatic code generation is the ease of improving the FORTRAN code's efficiency and the ease of incrementally adding complexity to the FORTRAN code. Since floating point operations are significantly faster than memory accesses in modern computer systems, one way to improve performance is to organize memory accesses to minimize cache memory misses. The process of rewriting a FORTRAN code consisting of a few complexly dimensioned DO loops to very many less complex DO loops to improve efficiency can be time-consuming; but doing the same activity within Mathematica is a relatively simple task. Also, since the forms of the one-dimensional equations in the Tensor form are the same in each dimension (only the indices switch), it is convenient to use a Mathematica rule to create the other equations rather than solving the equations anew each time. The testing and development of these algorithms could not have been done without the automation as it permitted fast and error-free testing of each

algorithm.

The actual process of automatically generating FORTRAN code is simply an effort in pattern matching and rules application and is more of an art than a recipe, much like programming. The artificial intelligence language LISP is clearly a foundation for the Mathematica language. The unified theme of representing everything as a symbol (data,function,equation,etc.) permits flexible and intuitive programming that is invaluable in the code generation process. Procedural, Functional, String-based, Rule-based, and List-based programming are possible in Mathematica and used in this work. When creating a FORTRAN code, the FORTRAN is essentially output from Mathematica as text. But, creating that text is made possible by assigning different meanings to each part of the text in Mathematica. For example, in Mathematica it is possible to represent an equation as a simple symbol. FORTRAN needs the actual algebraic form and so the simple symbol representing the equation is expanded into its fullest algebraic form. The same equation form may be used slightly differently within the same FORTRAN code and if there is a pattern to it, then a rule can be created to implement it. The representation of complex relationships via symbols also aids in debugging code since when fully expanded into FORTRAN code much of the higher structure is lost. For example, a single sign error is difficult to detect but easy to do when many algebraic equations are being coded into C or FORTRAN; the automation prevents many of these occurrences.

Each FORTRAN subroutine has a Mathematica Module assigned to it. The shell of the subroutine is simply stored as text in the module. By simultaneously writing the program in Mathematica and FORTRAN, it is possible to eliminate many errors as well as confirm the code generation process. The main body of the FORTRAN subroutine can vary significantly depending upon which MESA algorithm is used; But in all cases, all changes to the main body of a particular FORTRAN subroutine is effected by ONLY its partnered module in Mathematica. The goal is to have the entire program run in Mathematica and FORTRAN. Many of the computer algebra packages provide a compile option in which it will convert its modules into FORTRAN. This only works with numerical modules; it does not incorporate the symbolic capabilities necessary for this work.

Typical coding challenges that needed to be resolved were:

Mathematica Mathematica had a bug in it on SGI systems that resulted in core dumps for larger Gröebner basis problems; a work around was to apply line-group solutions or gen-

erate the FORTRAN code on a SUN workstation where the problem does not exist. Of course, this error was fixed in Mathematica 3.0.2, at the time this work was nearly completed. The Recursive-Tensor Form method only experienced this problem for methods higher than twenty-ninth order.

Fortran The Fortran compilers on SUN, SGI, and IBM workstations have built-in continuation line limits, and limits on the complexity of the equations that they will parse. They will sometimes compile large formulas, but the compiled statements may fail to run during execution. The symbol tables are also easily exceeded. Many of these limitations can be handled with compile line options. FORTRAN90 in free form permits 39 continuation lines, and in fixed form only 19 continuation lines [2]. These limits can prove too limiting for the long formulas in higher order multidimensional algorithms; a work-around is to let Mathematica breakup the equations into smaller subequations that are added together in FORTRAN. Fortunately, this can be done in this work and the length of the equations are minimized using the Recursive Tensor form.

Unix Unix shell commands like "sed" and "awk" have file size limitations on certain systems; a work-around is to keep the files small by doing pattern translation in Mathematica. This problem is also avoided by using the Recursive Tensor form.

Computational Trade-offs must be made between increasing the work of Mathematica, and increasing the work of the Fortran compiler, and increasing the work of executing the Fortran code. Fortunately, the propagation code using the Recursive Tensor and Spatial Tensor is simple enough for today's technology. Unfortunately, those methods cannot be applied to general geometry problems, in which case, the General Form of spatial interpolation must be used.

Memory Forming a Gröebner basis for the ideal in equation 3.6 can consume all memory unless variable reordering and matrix decomposition are performed first as discussed in section 3.1.

Many checks are included in the automation process, but there is still a good deal of faith involved in assuming that Mathematica is working properly for the larger problems. It is possible to check the results from Mathematica with results from a competing computer algebra program

such as Maple or Macsyma, but this has not yet been done since the numerical results show good agreement with the analytical solutions in chapter 7. Despite these validation attempts, unexpected problems can develop. For example, Mathematica 3.0 uses a different Fortran code output format than does Mathematica 2.2. The outputs are essentially the same except the 3.0 version automatically puts a decimal at the end of each integer whereas the 2.2 version does not. This difference generally is not noticed except when two integers are divided. Without the decimal points, integer division results in a truncation of the decimal information. Since the development of the MESA algorithm is automated and the equations are lengthy, this problem went undetected at first since the initial lower order test cases did not have integer division in their equations. These examples indicate the difficulties that can occur due to software and hardware limitations. Since the systems are not perfect, it is important to automate the code generation process to minimize the effects of human mistakes and maximize the detection of system errors. The cumulative effect of the many random areas in which things can go wrong is why a recipe is not possible for the creation of code.

The entire FORTRAN code generation software written in Mathematica for 2D problems is approximately 57 KBytes and for 3D problems is approximately 96 KBytes. This software is reused to generate all the 2D biperiodic and 3D triperiodic boundary problems in FORTRAN. The c2o10 MESA scheme FORTRAN code written by the Mathematica software in 2D is approximately 377 KBytes and in 3D is approximately 1459 KBytes.

Chapter 4

Wall Boundary Mapping in Two-Dimensions

The previous chapters described the development of the MESA scheme for solving the linearized Euler equations in time on a Cartesian mesh without solid wall boundaries. The addition of geometrically complex boundaries complicates the solution of the linear system of equations in time. The addition of boundaries can create instabilities in an otherwise stable MESA scheme [100] and reduce its accuracy. In addition, the complexity of formulating the problem in a FORTRAN code can be enormous if not automated [116].

This chapter presents the automated methods necessary for the inclusion of solid wall boundaries. A wall boundary is the locus in computational space which corresponds to an actual physical boundary. Appropriate conditions for the flow at these boundaries are assumed to be known for all time.

4.1 Introduction

Many methods for grid generation have been developed [111]. Most of the methods require considerable human interaction for successful implementation with realistic geometries. It can take months or years to generate a single grid for a real application. A method is presented in the following sections that uses complex but automated MESA schemes on an easily generated

Cartesian grid. The use of Cartesian grids significantly reduces the grid generation effort at the expense of increasing the complexity of the numerical scheme. Fortunately, this complexity is dealt with in an automated manner. Various Cartesian-based grid approaches have been used in the past [115], including recently developed wall boundary conditions with ghost cells for the DRP scheme [106]. Renewed interest in the approach is due to continued difficulties with body-fitted grid generation (structured, unstructured, multiblock, etc.), and the relative ease of automating the generation of Cartesian-based grids. Traditionally, the essential issue for Cartesian-based grids has been either computing and characterizing the geometric intersections between the Cartesian flow field cells and the surface geometry (using a flux formulation), or the development of ghost cells outside the boundary with data values chosen to maintain well-posedness of the finite-difference scheme. For example, a 3D Cartesian mesh is used in solving the finite volume form of the Euler equations in reference [79], and a 2D Cartesian mesh is used to solve the Navier-Stokes equations in [17]. However, a new approach has been successfully implemented in this work that uses mapped fill points inside the boundary. This mapping considerably simplifies the solution of near boundary grid points (fill points) as will be shown later. This mapping is further simplified by using very small stencils, a core capability of the MESA scheme.

Cartesian meshes must be able to vary the level of resolution according to the features of both the geometry and flow field unless an efficient higher accuracy algorithm with very high resolution can be used. When a grid is locally refined, an efficient data structure for nearest grid neighbor calculations is the Alternating Digital Tree [17]. The mesh resolution directly affects the accuracy and the CFL stability constraint; And finer meshes generally require more time steps to advance a solution through time. Local grid refinement can be avoided by increasing the accuracy of the MESA scheme which is used locally. By including more derivative terms at the local grid points, the stencil size remains unchanged and the CFL stability constraint is therefore unaffected. This dissertation has not addressed the issues of adaptive resolution but the special advantages of adaptive algorithm resolution via the MESA approach instead of adaptive mesh resolution are significant and will be developed in future work. In this work, the more general issue of solving complex geometry problems is developed.

Near a boundary surface, the Cartesian mesh will have grid points on either side of the boundary, and the boundary location will generally fall between grid points. This leads to

the well known problem of clipping cells [10], with potentially small cell fragments left inside the computational domain. If the boundary points are used instead of regular grid points, the stability limit on the time step size will in general be lowered. This problem is typically handled by *locally* increasing the grid density and *locally* decreasing the time step size. This results in more computational work and a loss of accuracy in extrapolating from finer to coarser meshes [10, 17]. This dissertation has used the concept of mapped fill cells to maintain the accuracy and the CFL condition across the entire computational domain. Mapped fill cells and their application will be the topic of the remainder of this chapter.

4.2 Definitions and Approaches

Consider a square or cubic uniform Cartesian mesh of grid points. Each grid point is labeled in one of four different ways, either as an interior grid point, a needed fill grid point, an unneeded fill grid point, or a boundary grid point. Next, assume some solid wall geometry is superimposed into this mesh. For example, a square has been used in figure 4.3 in which the interior of the square contains the acoustical perturbations modeled by the linearized Euler equations. Those grid points that fall within or on the edge of a solid object are labeled "boundary grid points". Those grid points within the solid object are not used since no flow travels through the solid boundaries. But boundary grid points on the edges of the object are used for calculating the effect of boundary conditions on the interior grid points. The "interior" grid points are those points which do not require information from a "boundary" grid point for their time advancement (ie. their stencil is completely within the flow field). The remaining grid points are labeled as "needed fill" points, except that do not have an "interior" point adjacent to them, they are labeled as "unneeded fill" points. An unneeded fill point can be seen in figure 4.3 in the corners of the square. The unneeded fill points are not required by any of the interior grid points for their time advancement. The specific labeling of a given Cartesian mesh's grid points depends on the stencil size. In figure 4.3 the stencil size is either (2×2) or (3×3), corresponding to MESA schemes c2oD and c3o0 respectively, with $D \geq 0$.

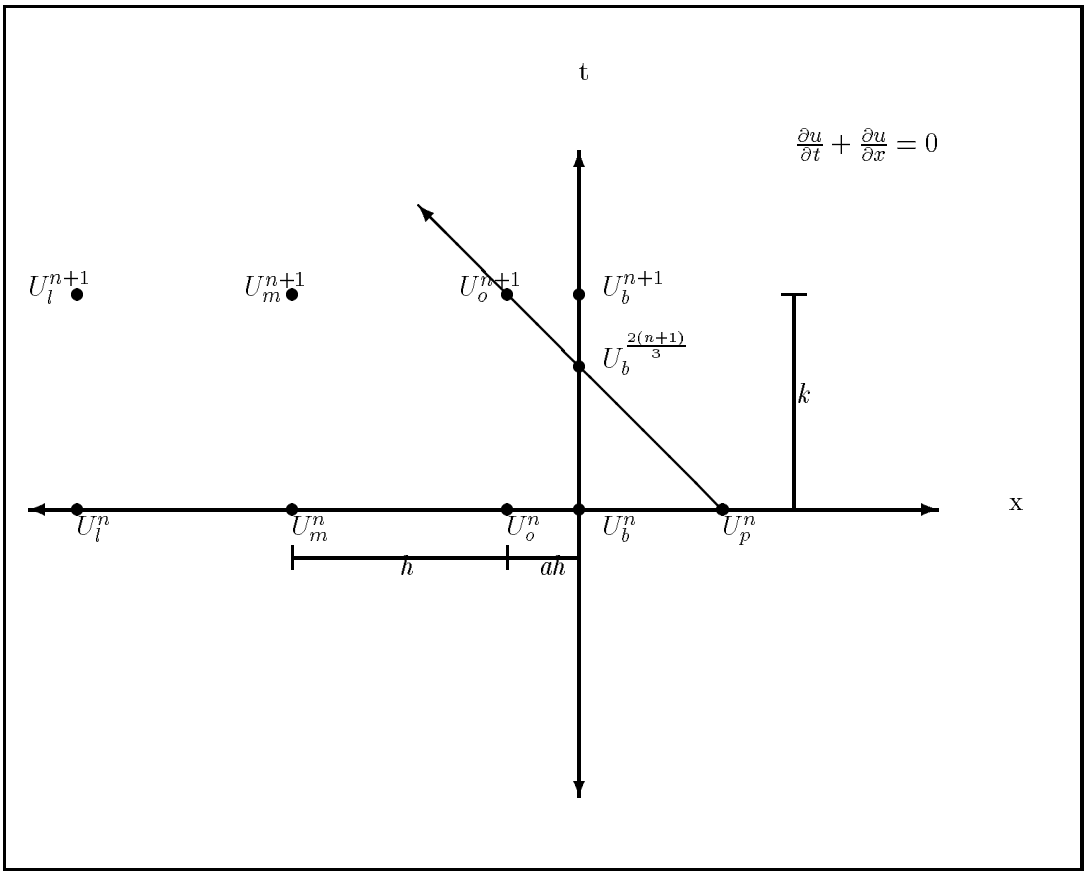

Figure 4.1: One-Dimensional Wave Equation Boundary Treatments

Approaches

Several approaches using a c3o0 MESA central stencil in one and two spatial dimensions shown in figures 4.1 and 4.2 will be explored to motivate the particular approach adopted by this dissertation. The one-dimensional and two-dimensional wave equations will be considered. The new solution value U_o^{n+1} in figure 4.1 can be calculated by the MESA propagation algorithm using the three data values U_m^n, U_o^n, U_p^n, and the data value U_p^n is actually located on the other side of the boundary, so that it is not even defined. The data value U_p^n is said to be located at a "ghost point" outside of the domain for the problem. The boundary condition is given by specifying data along the boundary for all time. The boundary in figure 4.1 is located on the t axis.

Several possible boundary treatments can be considered for this example. The most obvious approach is to use a nonuniform stencil for interpolation, and solve for U_o^{n+1} using the boundary data U_b^n instead of the ghost point data U_p^n. This shortens the distance between U_o^n and the stencil point to its right. It is now necessary to reduce the time step size in order to satisfy the CFL stability condition. Another idea is to use the boundary data that occurs on the characteristic between the boundary and the point where a new solution is desired. This method of calculation should maintain the CFL condition, but it would require knowing the slope of the characteristic curves (and surfaces in higher dimensions), and then computing their intersections with the boundaries. A third idea that was attempted in this work was to use the boundary data U_b^{n+1} at the new time level. This approach eliminated the need to calculate the slopes of the characteristic base curves and surfaces, and it included all the necessary information for calculating a new solution value, but it was not stable for $\lambda = \frac{\Delta t}{\Delta x} < 1$. In fact, stability unexpectedly improved for the c3o0 algorithm by choosing $\lambda \geq 1$. These approaches were not pursued further.

Instead, a reduction in the complexity of the finite-difference scheme for propagating the solution is accomplished by developing an approach that uses a Cartesian grid that is equally spaced throughout the physical domain. One possible way to achieve this is to label U_o^n and U_o^{n+1} as fill points that are not time propagated with the usual MESA scheme. Instead, a linear spatial interpolant is found at each time step that is a function of U_l^n and U_m^n. Then with point U_o^n solved for using that spatial interpolant, it is possible to time advance U_m^n to U_m^{n+1} using

the stencil points U_l^n, U_m^n and U_o^n (using the c3o0 MESA scheme).

Now consider the two dimensional convective wave equation where the same idea of spatially interpolating the fill points introduces the additional complexity of finding a mapping of the fill points to a general wall geometry.

$$\frac{\partial u}{\partial t} + M_x \frac{\partial u}{\partial x} + M_y \frac{\partial u}{\partial y} = 0, \tag{4.1}$$

with general solution

$$u[x, y, t] = \sum_{\alpha=0}^{2} \sum_{\beta=0}^{2} u_{\alpha,\beta} (x - M_x t)^\alpha (y - M_y t)^\beta. \tag{4.2}$$

Consider the second order algorithm (MESA scheme c3o0) on a uniform 3×3 square stencil with solution values but no derivatives at each grid point. A typical problem with a two dimensional boundary surface is illustrated in figure 4.2, where a corner of the bounding surface intrudes into the stencil. The wall boundaries are located on the planes formed by the (x, t) axes and the (y,t) axes in figure 4.2. Notice that three data values on the stencil required for time advancing $U_{o,o}^n$ are at ghost points $(U_{p,p}^n, U_{p,o}^n, U_{p,m}^n)$ on the other side of the boundary. Just as in the previous one dimensional case, it would be desirable to use the interior grid points $(U_{l,m}^n, U_{l,o}^n, U_{l,p}^n, U_{m,m}^n, U_{m,o}^n, U_{m,p}^n)$ to interpolate the values of the three fill points $(U_{o,m}^n, U_{o,o}^n, U_{o,p}^n)$. And then to use these "filled in" fill points to time advance $U_{m,m}^n$,$U_{m,o}^n$, and $U_{m,p}^n$ using the standard MESA techniques. Notice however, that the three advancing points do not have the same stencil, but do share common fill points. The fill points are calculated once for each time step before the interior grid points are advanced.

The polynomial interpolation function for the fill points is determined by solving a linear system formed by the known interior grid points and the boundary conditions applied to each unknown fill point mapped to a boundary. The 3×3 stencil centered on $U_{m,o}^n$ in this example has 6 known data values on interior grid points and three unknown fill points.

The c3o0 MESA scheme is a second order scheme and therefore requires at least second order accuracy in its boundary condition solutions as well. In this simple example, the data values $(U_{b,p}^n, U_{b,o}^n, U_{b,m}^n)$ can be used directly along with the 6 interior grid points to form a second order interpolant. This is achieved by mapping the fill points horizontally to the nearest boundary

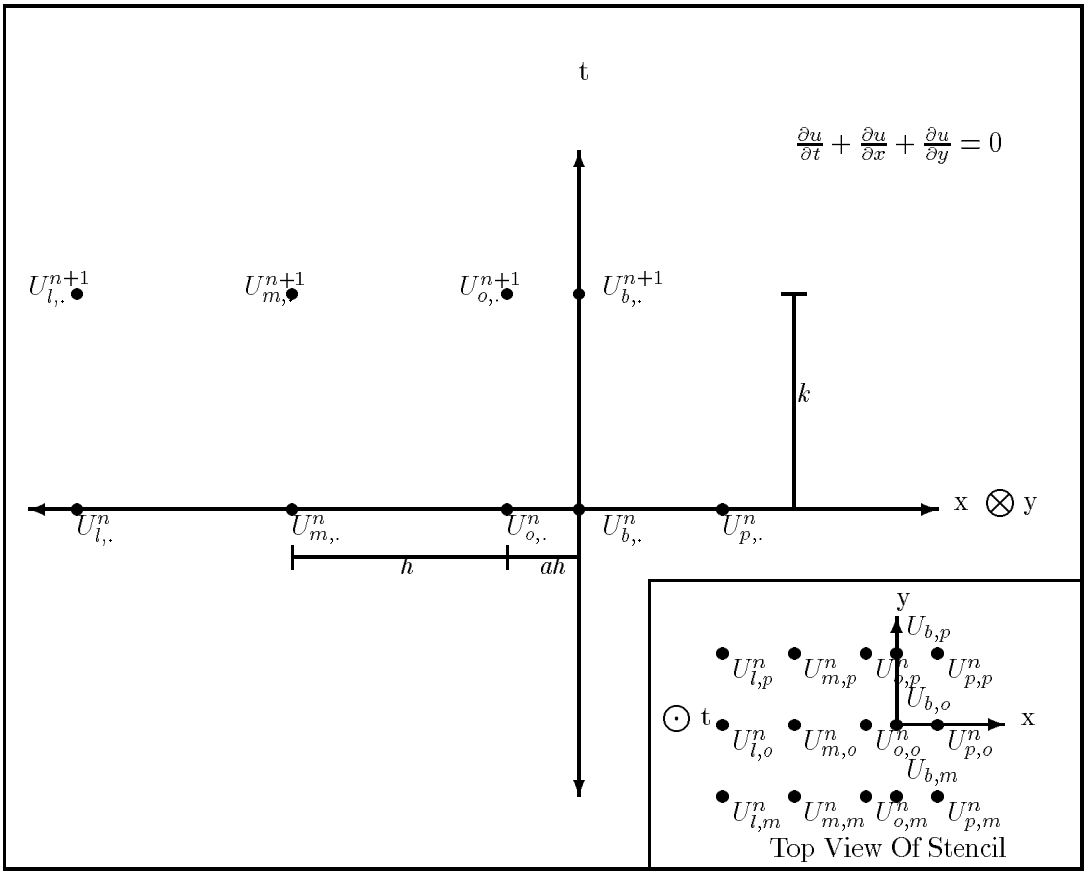

Figure 4.2: Two-Dimensional Wave Equation Boundary Treatments

at which point the boundary condition(s) are applied. If the interpolant function is written as a Lagrangian polynomial, the linear system will require inverting a 3×3 system in this simple wave equation example and is discussed further in chapter 5 (The Lagrangian form will have only 3 unknown coefficients, the fill points). The rank of the linear system will be equal to the number of fill points which will never exceed 7 as shown later.

One requirement to forming a consistent linear system is that never more than 3 stencil points be collinear when using the c3o0 MESA scheme. It is thus important that the mapping of fill points to a boundary of general shape provide this linear independence of stencil point locations, while minimizing the distance from fill point to boundary to maximize accuracy in the spatial interpolant. Therefore, one of the central issues that needed to be solved was how to generate, in an automated way, a mapping for each fill point to a boundary of general geometry that forms linearly consistent systems of interpolants of required accuracy and stability.

A sample mapping for a unit box rotated 45 degrees relative to the grid is presented in figure 4.3. The arrows indicate where the fill point is mapped onto the boundary. The boundary conditions are then applied at these locations and a consistent spatial interpolant is thus generated.

4.3 Stencil Constraint Tree

The problem of finding a mapping for all fill points to the wall boundaries is simplified by first standardizing the dimension size of the stencil to 3. Higher order schemes will be achieved by adding more derivative terms into the current stencil instead of the typical approach of enlarging the stencil. This standardization fits nicely into the c2oD and c3o0 MESA schemes with the selection of D dependent on the accuracy and resolution required for the particular simulation. The c2oD MESA schemes use a staggered grid, two-step process that results in effectively using a 3×3 stencil with artificial dissipation as shown in figure 4.4. The "X" grid points in this figure are evolved first using it's 4 neighboring grid points with a half time-step. Then the center grid point is evolved using the information from the newly computed "X" grid points again with a half time-step. The set of fill grid points is the same for the c2oD and c3o0 MESA schemes. The c2oD scheme's fill points may be interpolated using either a $2(D+1)-1$ order interpolant based upon a 2×2 stencil or the $3(D+1)-1$ order interpolant based upon a 3×3 stencil. The

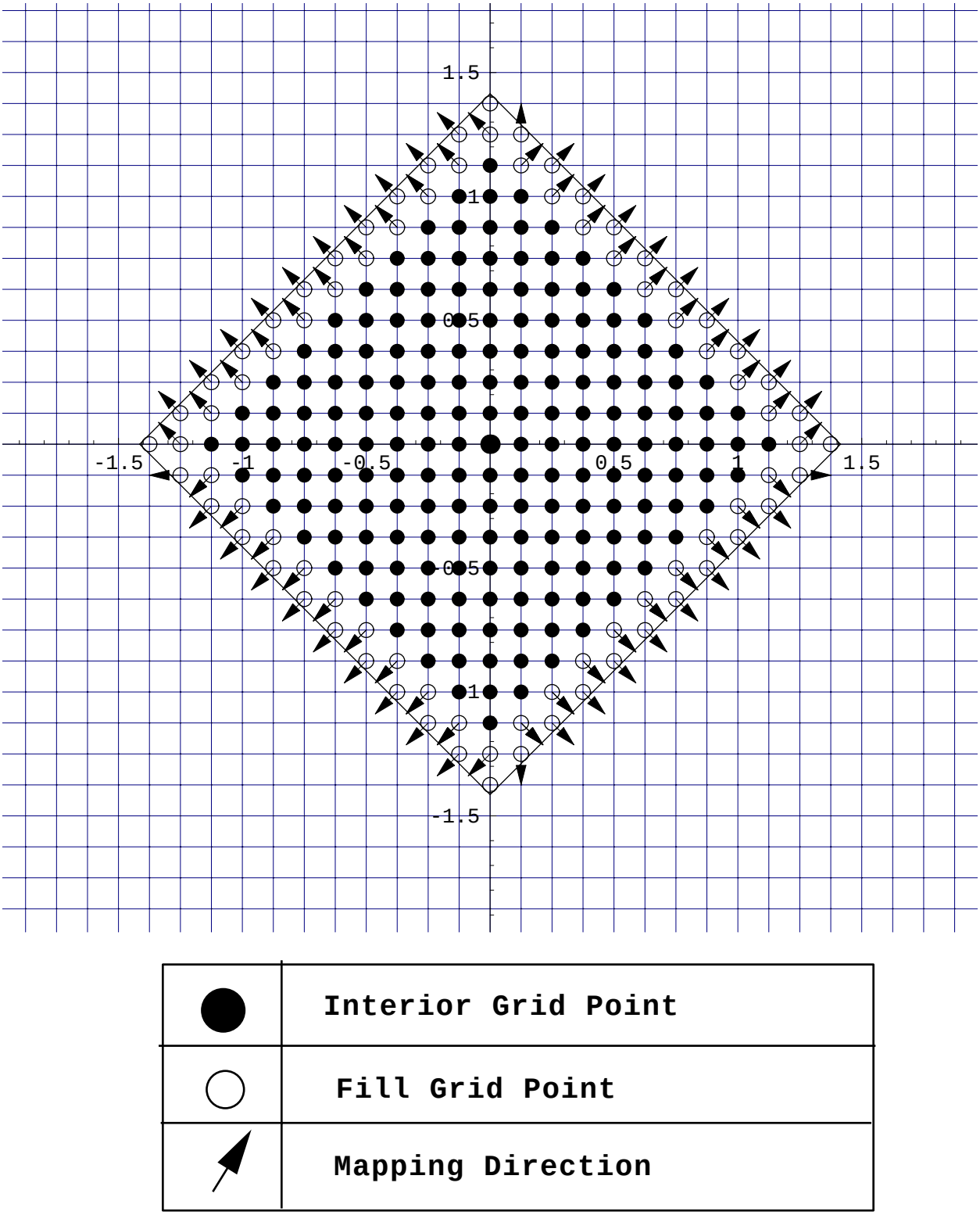

Figure 4.3: Sample Mapping For Fill Points with 2×2 or 3×3 stencils: Box Rotated $\frac{\pi}{4}$ Case

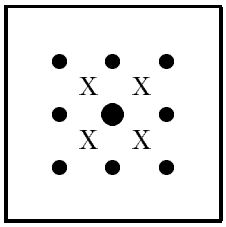

Figure 4.4: Staggered Grid with C2oD MESA scheme

advantage to using the smaller stencil is the simplicity of mapping its fill points to the boundary (figure 5.7). The disadvantage is that it is no longer possible to overlap the stencils–numerical experiments suggest overlap is important to maintaining stability as discussed in chapter 5. In addition, the c3o0 method is the simplest, useful algorithm and was the first method attempted in this work. Therefore, the 3×3 stencil will be dealt with first.

With this standardization (3×3 stencils), it is possible to organize the mapping of the fills into subproblems consisting of 3×3 groups of grid points (corresponding to a particular stencil configuration) in which all the fills in each stencil configuration are mapped to the wall boundary and a local spatial interpolant is developed using information from only the interior data points and mapped boundary locations for each stencil configuration.

Recall that the grid points are identified as "interior", "fill", or "boundary". And the fill points can be further labeled as "needed" or "unneeded". In addition, the boundary points can be further labeled aligned with the wall boundary or unaligned. The principles discussed next apply to any labeling scheme–as many labeling schemes have been used in this work. Each labeling scheme permits a different approach to mapping the fill points. For pedagogical purposes, assume the grid points can be labeled into three categories: interior, fill, or boundary. With no constraints imposed on the grid points in the 3×3 stencil, there are $3^9 = 19,683$ possible stencils configurations. That is, the stencil as a whole may have 19,683 unique manifestations. Each will need a unique mapping for all its fill points, if the stencil configuration contains any. In three dimensions, without natural constraints imposed, the $3 \times 3 \times 3$ stencil has $3^{27} = 7,625,597,484,987$ possible stencil configurations.

Fortunately, there are natural constraints that can be applied that makes this problem tractable.

A method similar to Waltz's procedure for symbolic constraint propagation [120] provides

a way of reducing the possible stencils to a surprisingly manageable set. As is usually the case in artificial intelligence applications, the choice of representation of the problem can assist its solution. In this case, a successful representation of the stencil configurations is in a tree. The tree data structure's leaf nodes are NOT labeled, but it's branches ARE labeled. This data structure efficiently stores all possible stencil configurations in its tree branch structure, and enables a recursive building and pruning algorithm to efficiently find all possible stencil configurations. This tree data structure shall be referred to as the Stencil Constraint Tree.

The locations of the grid points in a stencil are labeled in row-major order starting with the bottom row as shown in figure 4.6. These labels uniquely identify each grid point's location within the stencil. A branch in the stencil constraint tree represents both a grid point's position and its label. The branches are organized into groups of three. Each group represents a particular grid point location. The relative position of the branch within a particular group of three branches determines the grid points label. If the branch is the first branch in its group, it represents an interior point. The second branch in the group represents a fill point. And the third branch in the group represents a boundary point. For example, in figure 4.5 all possible branch numbers for a 2×2 stencil are shown. A branch number of 7 is in position 3 and has label 1. According to figure 4.6, position 3 corresponds to the top-left grid point in the 2×2 stencil. Since it has label 1, it is an interior grid point as well. Similarly, branch number 12 is in position 4 and has label 3. Therefore this branch number corresponds to the top-right grid point being a boundary grid point. This branch numbering scheme can be applied to any size stencil with any number of grid point labels. A given branch A will be connected to other branches B in the tree only if the other branches represent grid points that are adjacent to the grid point represented by branch A. In this way, the stencil constraint tree succinctly represents a given stencil's topology and all it's configurations. An example stencil constraint tree with its branches labeled is shown in figure 4.7.

As discussed, the number of stencil configurations is enormous if all configurations are allowed. Fortunately, the natural constraints in the problem can be exploited to reduce the size of the stencil constraint tree. Some of the natural constraints are:

1. Each interior point must have no boundary types adjacent to it.

2. Each fill point must have a boundary point adjacent to it.

1	2	3	4	Position
1 2 3	1 2 3	1 2 3	1 2 3	Label
1 2 3	4 5 6	7 8 9	10 11 12	Branch

Figure 4.5: Stencil Constraint Tree Branch Numbering Scheme

3. Each boundary point must have no interior types adjacent to it.

Those constraints can be applied during the construction of the stencil constraint tree. For example, in the 2×2 stencil with interior, fill and boundary grid point types, natural constraint number 2 implies that a branch labeled 11 (it's a fill grid point) *MUST* be connected to a branch labeled 3,6, or 9 (a neighboring boundary point). Also, the tree structure itself can be exploited during its construction to form additional natural constraints that significantly reduce its growth. This is an algorithm in which the parents learn from the children to avoid producing too many children!

4.3.1 Building the Tree

Assume we are interested in building a stencil constraint tree for the 2×2 stencil with the assumption that grid position 4 is of interior type and that each grid point is either an interior, fill, or boundary point. Every node will have at most 12 branches as shown in figure 4.5. A completed stencil constraint tree showing all possible stencil configurations with an interior grid point in the top-right position is shown in figure 4.7. The number of leaf nodes on the bottom of the tree equals the number of possible stencil configurations (8 in this case) under the given assumptions.

Notice in figure 4.7 that only the branches are labeled and that it has a tree depth of four, which corresponds to the number of grid points in the stencil In addition, the first level contains only branches corresponding to grid point position 4, the second level contains only grid point position 3 branches, the third level contains only position 2 branches, and the fourth has only position 1 branches. A 3×3 stencil would have a tree depth of nine and a $3 \times 3 \times 3$ stencil would have a stencil constraint tree depth of 27.

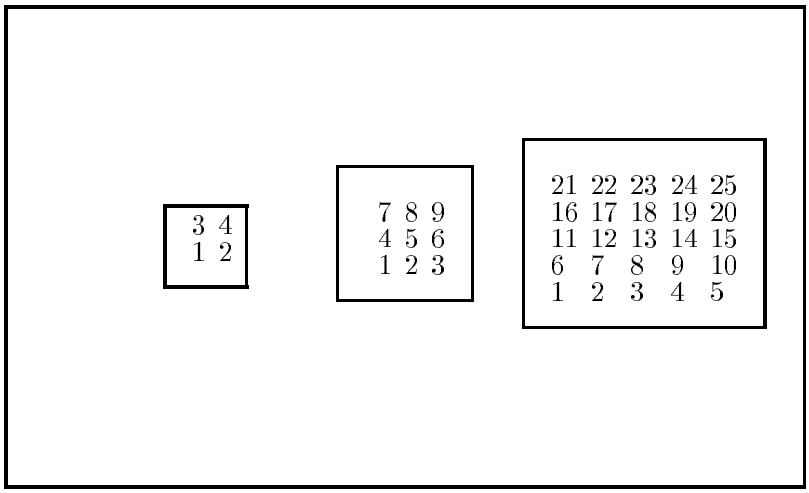

Figure 4.6: Stencil Grid Position Labels for N=2, N=3, and N=5

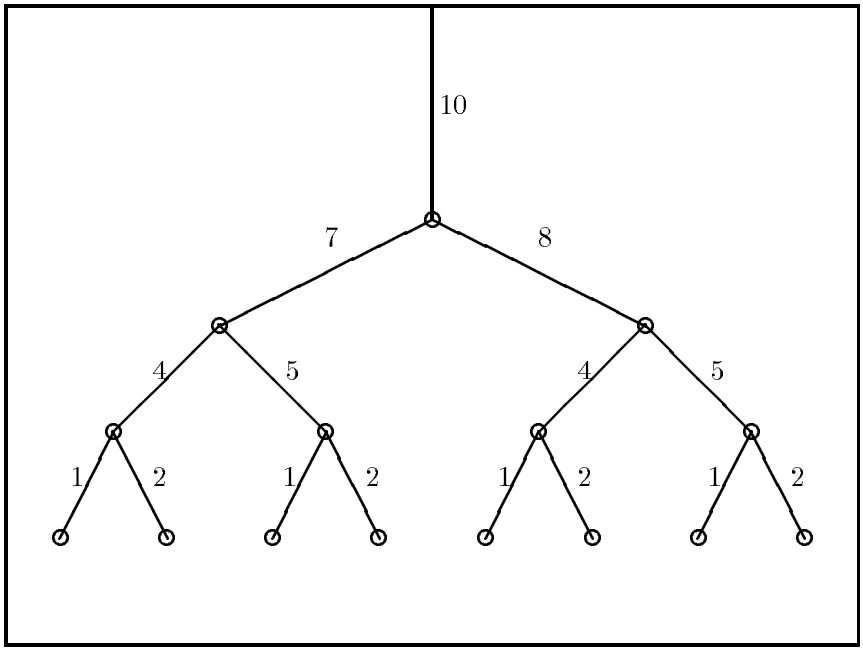

Figure 4.7: Stencil Constraint Tree, N=2, Assume Position 4 is an Interior Grid Point

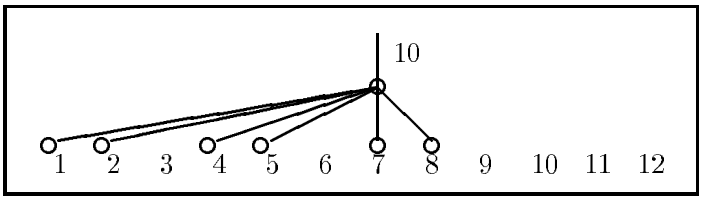

Figure 4.8: First Node Expansion of Stencil Constraint Tree

Building this tree is done by starting at the top of the tree. Branch 10 falls between 10 and 12 making it the fourth group of three branches as shown in figure 4.5, so it corresponds to grid position 4, the top-right of the stencil. Within that group of 3, it is the first branch, so it corresponds to label 1, interior. The first branch represents the given assumption of node 4– that it is of type "interior". Since it is an interior node, it cannot by natural constraint number 1 have neighboring nodes of type "boundary" (label number 3 in figure 4.5). If there were more assumptions, such as the top-left position is a fill point, then an addition branch number 8 would be on the first level of the constraint tree. The current branch number 10 in the top level of the tree constrains the choices of the branches in level 2. In particular, branches 3,6,9,12 are removed from consideration in level 2 of the tree since the boundary points cannot be adjacent to the interior points. In addition, an additional constraint is imposed to prevent repetitive loops in the tree. Specifically, a parent node cannot repeat its own position number or that of its ancestors. Therefore, branches 10, 11, and 12 are also removed from consideration in level 2 of the tree. The first node in figure 4.8 may only expand branches numbered 1,2,4,5,7, and 8. The numbering underneath the nodes in the figure corresponds to the branch's number. Without the natural constraints, all branches (1 through 9) would be expanded.

The figure 4.7 is the completed stencil constraint tree and therefore does not show branches 1,2,4, and 5 since they are later pruned in a post-processing step that removes redundancies.

Branches 1 and 2 correspond to grid point position 1. The children branches (level 3 and 4) underneath these branches (level 2) will not contain grid point position 4 (branches 10,11,12) since the first level determined it already, nor will they contain branches representing position 1 since the second level has determined it. The sub-trees under branches 1 and 2 of level 2 (corresponding to position 1) are recursively constructed using the same procedure for the current parent branch. After these sub-trees are expanded, they contain all possible permutations of

the remaining grid positions (2 and 3). The permutations represented in the sub-trees will be restricted under the assumptions of its ancestors branches, parent and grandparent (level 2 and level 1), in the constraint tree. The branches that were not expanded in these two sub-trees represent impossible labels for grid positions 2 and 3, under the assumption that grid position 4 is an interior point. The set of impossible labels is passed from each child to its parent branch, once the child has completed constructing its sub-tree.

This list of impossible branches is passed to the next child on level 2 that represents another grid position (branch 4 in figure 4.8). Branch 4 on level 2 still has the same parent branch (10) as branches 1 and 2 on the second level. The union of the set of branches contained in the sub-trees of branches 1 and 2 represent all possible permutations of grid positions 2 and 3. It would not be possible for grid position 2 or 3 to be labeled type 3 (boundary) since branches 6 and 9 do not occur in either of the sub-trees under branches 1 and 2 of the second level. Therefore, the list of illegal branches passed to the next child (branch 4, level 2) includes (6,9,10,11,12). Therefore, the branch number 4 on level 2 builds its subtree utilizing constraint information from its left sibling branches. It's parent, branch 10 on level 1 receives this list when the sub-tree under branch 2 on level 2 is completed. The parent branch then passes this information to branch 4 on level 2. If branch 4 were one of the illegal branches, then the parent branch 10 on level 1 would not expand branch 4 on level 2, but instead would skip to branch 5 or the next legal branch.

After branch numbers 4 and 5 have completed constructing their subtrees, additional illegal branches may be found. The subtrees under branches 4 and 5 (on level 2) represent all possible values of grid positions 1 and 3 constrained by the illegal branch list from the branches 1 and 2 on level 2. These additional illegal branches are added to the current list and passed to the next group of branches in level 2 (branches 7 and 8). Note that it is possible to pass the additional illegal branches to the already completed children on the left and prune their subtrees, but this was not needed in this work. Once the children on level 2 have constructed their sub-trees the stencil constraint tree is completely built. All subtrees are built using the same recursive preorder traversal construction with natural constraint propagation among siblings. Each recursive call returns a list of illegal branches that the parent receives from each child. The parent always passes on this list to the next child created.

The stencil constraint tree now contains all possible stencil configurations in its structure as shown in figure 4.9. Each configuration is found by simply traversing the tree from top to bottom.

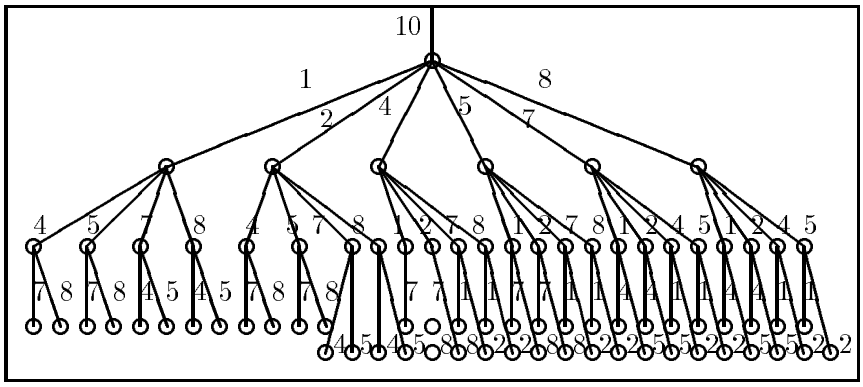

Figure 4.9: Unpruned Stencil Constraint Tree

Each unique path represents a possible stencil configuration. However, the tree as currently constructed has many repetitive solutions. For example, in the 2×2 stencil constraint tree in figure 4.9, the first branch traversed in any path taken will be grid position 4, label 1 (on level 1). It is possible for grid positions 2, 3, and 4 to be labeled 1 (interior) as well without violating the natural constraints. This corresponds to branches (1,4,7,10). According to the unpruned stencil constraint tree there are 6 paths that give the same result: (10,1,4,7),(10,1,7,4),(10,4,7,1), (10,4,1,7), (10,7,1,4), and (10,7,4,1).

This inefficiency is eliminated by pruning the tree as follows. Starting from the top of the tree, remove all branches from the tree that do not correspond to grid position 4 on the first level. In this case, this does nothing since only branch 10 is on the first level. Next, remove all branches from the second level that do not correspond to node 3. This removes branches 1,2,4,and 5 and their subtrees. Repeat this for the rest of the levels. After this, the stencil constraint tree is pruned and shown in figure 4.7. The tree now efficiently represents all stencil configurations. This pruning process requires that the order of grid positions selected for pruning be such that they are topologically connected in the tree. For example, in the 3×3 case grid position 4 is not connected to grid position 6 and so should never be placed together in the pruning list. Failure to do this results in a destroyed stencil constraint tree.

This pruned tree requires only 4 tests to determine if a given stencil configuration is correct by simply traversing the four levels of the tree from top to bottom. If a path from top to bottom matching the stencil configuration does not exist, the stencil configuration is incorrect. The 2×2 stencil with an interior grid point in the top-right has a total of eight possible configurations. If

all eight configurations were stored as arrays, and each grid point was naively tested, it would take 32 tests to reject a given 2×2 stencil. This represents eight times more effort than using the stencil constraint tree. The savings are significant for this simple problem, but are absolutely essential for more complex stencil configurations in three dimensions. For example, a $2 \times 2 \times 2$ stencil in three dimensions requires only 8 tests if the stencil constraint tree is used compared to ($2^8 \times 8 = 2048$) for the array method when it is assumed the corner grid point is of type 1, "interior". Also notice that without natural constraints there are (3^8=6561) permutations of the $2 \times 2 \times 2$ stencil configuration. Employing natural constraints and propagating them in the stencil constraint tree reduces the complexity of mapping the fill points.

The ability to efficiently test all stencils used for the fill points is important. In some cases, the CAD input geometry file may be incorrect, the geometries' curvature may be too steep for the given grid resolution, or a degenerate case has occurred that needs human assistance. By quickly testing all stencil configurations across the entire computational domain, these hard to identify problems are quickly dealt with.

Note that representing the stencil configurations in an undirected graph such as the stencil constraint tree without propagating the natural constraints results in an NP-Complete algorithm. This is because all paths must be traversed such that all nodes are visited as in the Traveling Salesman Problem [20], which is an NP-Complete problem. Fortunately, the number of edges in the stencil constraint tree decreases as the tree is built due to the propagation of the natural constraints.

4.4 Recursive Boxes

The recursive algorithm just described was implemented in Mathematica which is itself designed to operate efficiently with trees. All Mathematica expressions are represented internally as trees [122]. Unfortunately, the recursive stencil constraint tree algorithm described in section 4.3 took too long to run on stencils with widths larger than $N = 2$. Rather than improving performance by implementing this algorithm in a lower level language, an improvement in performance was achieved with a meta-algorithm that made use of the constraint tree algorithm. This "meta-algorithm" recursively calls upon the stencil constraint tree algorithm to solve smaller sub-problems and then collects these results to form a global solution.

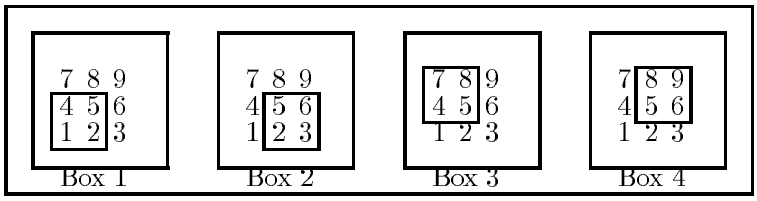

Figure 4.10: Stencil Grid Box Position Labels for N=3

The idea is to divide up the stencil into overlapping 2×2 boxes and to solve these boxes using the constraint tree algorithm just described. The advantage to this is that the pruning process happens after each 2×2 box is solved; This results in a far smaller tree during its construction. The boxes overlap so that the result from one box can restrict the neighboring boxes via the natural constraints of the problem (The natural constraints only apply to neighboring grid points).

For the N=2, 3, and 5 two dimensional stencils there is 1, 4, and 16 overlapping boxes respectively covering the entire stencil domain. In figure 4.10 the four boxes of the 3×3 stencil is shown. The labeling of the 2×2 boxes is done in the same manner as the individual grid positions were labeled for the constraint tree (row major order, left to right, bottom row first, but in overlapped groups of four).

Finding all possible stencil configurations for the 3×3 stencil is achieved by starting with any of the boxes in figure 4.10 and proceeding in a recursive way to solve all the boxes. It is important that the list of boxes are chosen so that each successive box is overlapped by a previous box. In the case of a 3×3 stencil, all four boxes overlap each other and so any ordering is appropriate. A box is solved by simply calling the stencil constraint tree algorithm to find all possible stencil configurations for the 2×2 set of grid points in the box.

Each 2×2 box may be restricted in two ways. First, one of its grid points may be assumed to be of some type (interior, fill, or boundary) in which the constraint tree algorithm can easily find solutions. Second, a diagonally neighboring box will introduce natural constraints as shown in figure 4.11.

For example, solving the 3×3 stencil for all possible configurations in which the center grid point number 5 is a fill point (branch number 14) using the Recursive Box method could proceed as follows. First, the stencil constraint tree algorithm is called with the assumption that grid

point location 4 is of type 2, "Fill" (Note that the local grid location labels are based on the 2×2 stencil relative to each box). All possible configurations are found for box 1 in figure 4.10. Next, for each 2×2 stencil configuration in box 1, the set of permissible stencil configurations for box 2 is then found by again calling the stencil constraint tree algorithm. For each stencil configuration in Box 1, the configurations in Box 2 will be based upon the two grids in Box 1. Its local grid locations 1 and 3 are defined by the values in Box 1 at its local grid locations 2 and 4 (case 7 in figure 4.11). Therefore, in solving Box 2 the constraint tree algorithm is initially given a list of illegal branches as described in section 4.3. The illegal branches are simply those branches representing local grid positions 1 and 3 that are not of the correct type defined by Box 1 (4 illegal branches total). Next, for each 2×2 stencil configuration found for Box 2, the stencil constraint algorithm is applied to Box 3. This time, Boxes 1 and 2 are already defined and will restrict the stencil configurations in Box 3. In particular, Box 3 will have its local grid position numbers 1 and 2 defined by Box 1's local grid position numbers 3 and 4. In addition, box 2 has a defined local grid position number 4 that will restrict the choices of the neighboring grid point in Box 3's local grid position number 4. Therefore, if Box 2 has an interior point in its upper right corner, then box 3 may not have a boundary point in its upper right corner (tree branch 12). When the constraint tree algorithm is called, it will not only have a starting list of illegal branches from the two defined grid points from box 1, it will also be given the list of illegal branches introduced from the effect of Box 3. This situation is represented by cases 4 and 5 in figure 4.11. Next, for each configuration found in Box 3, which was restricted by the definitions and influence of the configurations in Boxes 1 and 2, the stencil constraint algorithm is applied to Box 4 to find all of its permissible stencil configurations. This time, Box 4 will have only 1 undefined grid point (its upper right position 4). The constraint tree for Box 4 will have illegal branches (6 illegal branches) corresponding to the 3 defined grid points. This case can be represented as the superposition of the 3 cases 5, 6, and 7 in figure 4.11. After these procedures, all of the 2×2 stencil configurations in the boxes have been found and the list of all possible 3×3 stencil configurations is gathered by simply unwinding the recursive calls as shown in figure 4.12. The set of 2×2 stencil configurations for Box 4 which was restricted by each set of 2×2 stencil configurations in Boxes 1,2, and 3 of figure 4.10 may be combined to provide a set of 3×3 stencil configurations for the entire stencil. For example, in figure 4.12, each numbered box corresponds to a specific 2×2 stencil configuration for the aprropriate box in figure 4.10.

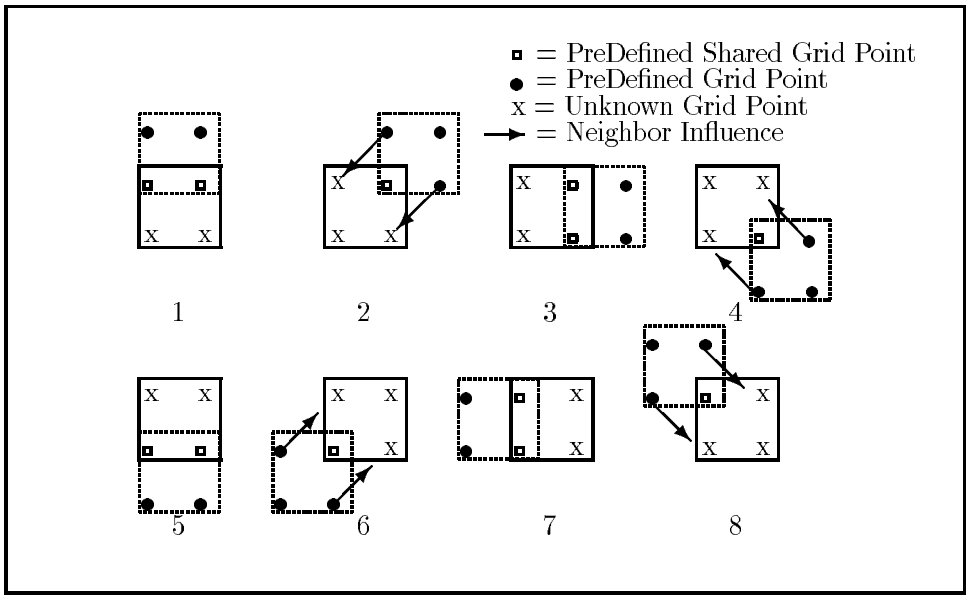

Figure 4.11: Box Recursion Method: 8 Neighboring Box Cases

Across the top row of boxes in figure 4.12 the Boxes 1, 2, 3 and 4 form one complete 3×3 stencil configuration. Now, backtracking one step from Box 4 back to Box 3 and then taking the next path to the second Box number 4, a second 3×3 stencil configuration is found. This process repeats until all possible 3×3 stencil configurations are determined. This process is efficient since the constraint tree algorithm is applied only to small 2×2 stencil sub-problems.

Notice that not all eight possible two-dimensional cases in figure 4.10 were used in solving the 3×3 stencil in this example. The ordering of the boxes and the size of the stencil determine which of the eight Box Recursion method cases are employed. Larger stencils will still be subdivided into smaller, overlapping 2×2 sub-problems.

4.5 Symmetries and Simplifications

With the tools described in the previous two sections, it is possible to examine in more detail the structures of all possible stencils. It is important to quickly reject impossible stencils from consideration so that a systematic mapping may be developed. In most applications, relatively few of all the possible stencil configurations are encountered, but it is still necessary to exhaustively handle all cases. Once all the stencil configurations are known it may be possible to assign a

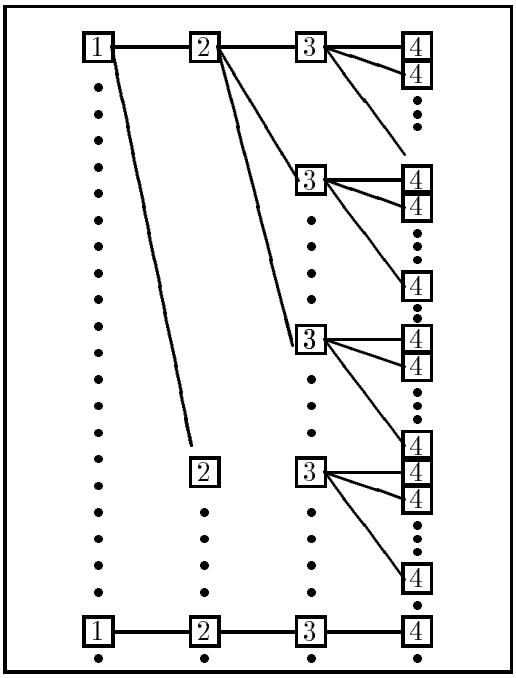

Figure 4.12: Recursively Collecting 2×2 Sub-Stencil Configurations

specific mapping to each case that provides a linearly consistent system for spatial interpolation at each fill point.

One approach to achieve this was to find all possible stencil configurations for a 5×5 stencil in which the only assumption is that the center grid point location 13 was a fill point. Then, for each of these configurations simply map any fill point in grid point locations 7,8,9,12,13,14,17,18, and 19 to the boundary points in grid point locations 1,2,3,4,5,10,15,20,25,24,23,22,21,16,11, and 6. Each fill point needs to be mapped to a unique grid point location and no more than three grid points in the new mapped stencil are permitted to be collinear with respect to the x and y axis. The former can only be satisfied if enough boundary points exist. The latter must be experimentally determined using the tools of the previous sections. Unfortunately, it was found that more fill grid points than boundary grid points occurred in many realizable stencils. If two fills are mapped to the same aligned boundary grid point, it is possible for both points to intersect. This reduces the spatial interpolant's linear system to only 8 degrees of freedom instead of the 9 required for a two dimensional linear system on a 3×3 stencil to be solved. If the boundary grid point is unaligned with the physical boundary then it is possible to map multiple

fills towards the same boundary grid point without an intersection occurring. The number of possible stencil configurations is quite large, even with the natural constraints applied. For example, a 3×3 stencil with the center grid point number 5 assumed to be a fill point has 8,456 legal stencil configurations that need to be mapped. This mapping ideally would only depend on the 3×3 stencil, but to verify this all possible stencils need to be generated. It would take 11 days to compute this larger set. Without constraints imposed there are $2^{24} = 16,777,216$ stencils to consider (each point is a boundary or not). Even with constraints imposed the set is still larger than will actually occur in applications since the outer stencil points are actually constrained by their outside neighbors which are not known without including all 7×7 stencil configurations.

Fortunately, it is possible to reduce the set of possible stencils by applying symmetry and additional assumptions to the stencils. A key assumption is that no stencil will be completely enclosed by solid walls (ie. a stencil will have at least one neighboring and overlapping stencil in the geometry). However, if it is completely enclosed, then locally increase the grid resolution until at least two overlapping stencils fit within the enclosed area. With this assumption, a 3×3 stencil will have an interior grid point on at least one of its edges, and a boundary grid point on the opposite side. The center will always be a fill point. The assumptions are shown in figure 4.13. At least one of those cases in the figure are assumed to occur in every stencil. Assumptions S1, S3, S5, and S7 have 36 possible stencil configurations each (using the Box Recursion method); And assumptions S2, S4, S6 and S8 have 144 configurations each. The fact the odd and even numbered assumptions all have the same number of configurations suggests there is symmetry in this problem. The union of all configurations corresponding to assumptions S7 are in figure 4.20 and those corresponding to assumptions S8 are in figure 4.21. In those figures, the aligned and unaligned boundaries are treated as identical and results in a total of 61 possible stencil configurations. By rotating these stencil configurations by 90 degrees clockwise, then the solution set to assumptions S5 and S6 in figure 4.13 is obtained. Another 90 degree rotation provides the S3 and S4 solutions; And one more 90 degree rotation provides the S1 and S2 solutions. Note that the set of stencil configurations after each rotation has 7 stencil configurations in common with its pre-rotated configurations. This suggests the symmetry is not perfect, but that's fine since the number of stencil configurations is now manageable at 61 cases. The union of stencil configuration sets for all eight assumptions (S1, S2, S3, S4, S5, S6,

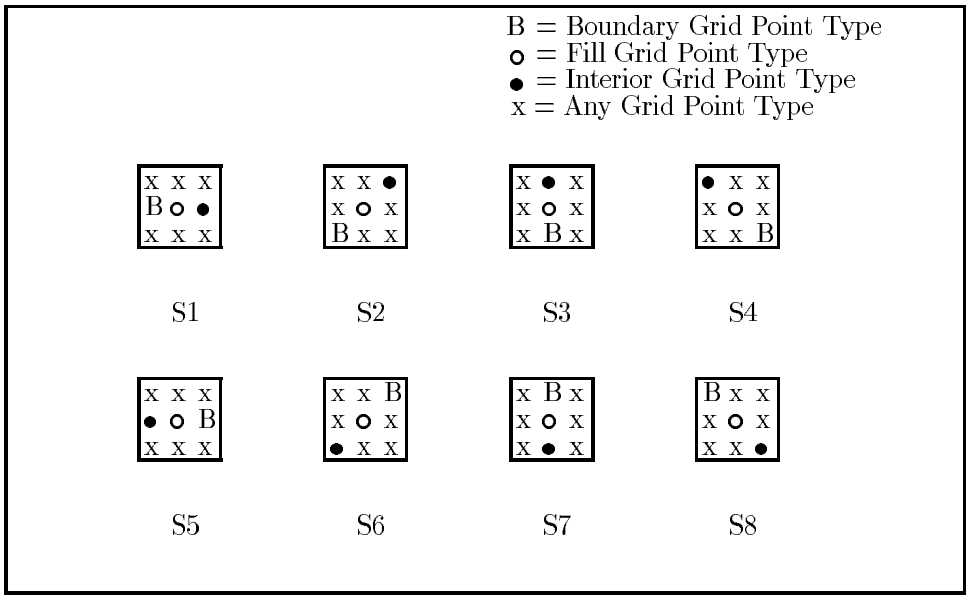

Figure 4.13: No Wrap Assumption: 8 Cases

S7 and S8) has 216 configurations for the 3×3 problem when unaligned and aligned boundaries are considered the same.

4.6 Unique Mappings

With the set of possible 3×3 stencil configurations now minimized, further detailed study of the mapping problem is possible. The mapping problem is to find a set of directions (unit vectors) for each stencil configuration that will intersect with the solid wall edges. The selection of the direction vectors needs to satisfy the following mapping criteria:

1. Maximize the accuracy of the spatial interpolant of the stencil by finding the closest wall boundary for each mapped grid point.

2. Expand outward from the center of the 3×3 stencil to maintain the same CFL conditions for numerical stability.

3. Insure that never more than 3 grid points are collinear after the mapping occurs to provide a linearly consistent system for spatial interpolation.

The assumption that a stencil will not be wrapped by the wall geometry was described in the last section. A consequence of this assumption is that any legal 5×5 stencil with a fill point in the center grid position number 13, will contain a 3×3 sub-stencil that contains no boundary grid points. And, all of the 9 possible sub-stencils (3×3) will contain the center fill point of the 5×5 stencil. In figure 4.14 notice that the S8 case has no "B" points in the bottom-right corner 3×3 sub-stencil. Similarly, the S7 case has no "B" points in the bottom-center 3×3 stencil.

The importance of this is that all grid points can be expanded out to meet the second mapping criteria. The CFL constraint would require a smaller time step if the stencil were made smaller. And, the third criteria could not be assured if the unaligned boundary points were mapped to an aligned position on the wall edge. This is readily seen in figure 4.15 in which a simple line parallel to the x-axis is the only geometry. If the fill point is taken as the center of any 3×3 stencil there will always be 6 collinearly mapped grid points (3 mapped fills and 3 mapped unaligned boundaries). Note that the unaligned boundary points in the figure 4.15 would be mapped to the same location as the fill points. However, if the fill points are considered the center of the 5×5 stencil then assumptions S2, S3, and S4 in figure 4.13 are true. In this case, there will be at least one 3×3 stencil that contains no boundary points and the fills may be safely mapped as shown in the figure 4.15, and the remaining 6 interior grid points do not need to be mapped.

Only the stencil configurations resulting from assumptions S7 and S8 need to be mapped however. All other mappings are deduced by rotating the direction vectors about the center of the 5×5 stencil up to a maximum of 3 times (90 degree rotation each) until the rotated S7 and S8 assumptions align with the particular stencil configuration as discussed in section 4.5.

Each of the sub 3×3 stencils in figure 4.14 has one fill point, two interior grid points and six grid points that are not a boundary type. The remaining 6 grid points may be either a fill point or an interior point. Therefore, there are $2^6 = 64$ stencil configurations that need a mapping under the S7 assumption and another 64 stencil configurations that need a mapping under the S8 assumption. It is possible to simplify this mapping task further by noting the additional local symmetry found in the 3×3 sub-stencils. The S8 sub-stencil is symmetrical across the main diagonal and the S7 sub-stencil is symmetrical down the middle vertical as shown in figure 4.16. Across this line of symmetry, the same set of mappings will occur. It is therefore necessary to only derive a mapping for one-half of the 3×3 sub-stencil. Only 3 grid points are unknown on

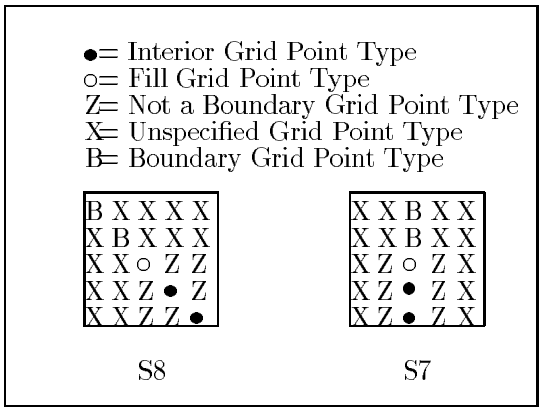

Figure 4.14: 5×5 stencil configurations under S8 and S7 assumptions

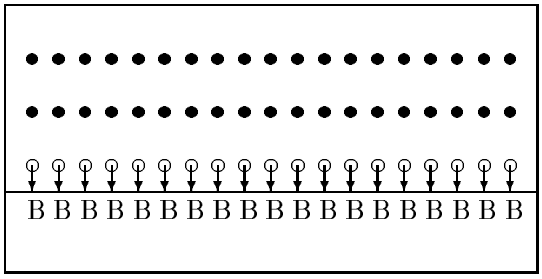

Figure 4.15: Too Many Collinear Grid Points When Unaligned Boundary Points Need Mapped

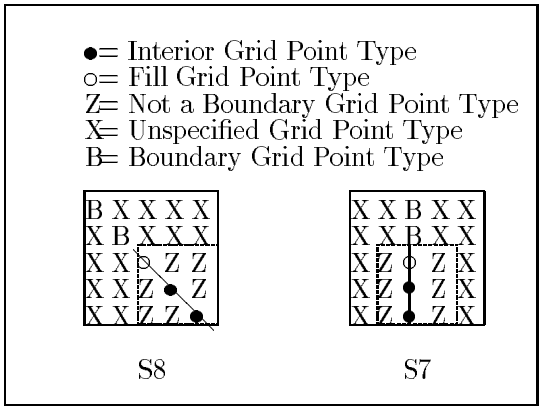

Figure 4.16: Sub 3×3 stencil symmetry under S8 and S7 assumptions

one side of the line of symmetry and therefore at most $2^3 = 8$ configurations need to be mapped per the S7 or S8 assumption. The mapping on other side of the sub-stencil is found by rotating the direction vectors around the line of symmetry.

4.6.1 Mapping the S8 Cases

A mapping will be developed for the upper triangle of the sub 3×3 S8 stencil in figure 4.16 that may be used for the lower triangle as well. Grid position 13 of the 5×5 S8 stencil will always be assigned unit direction vector $(\frac{-1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$. The fixed interior grid point locations 5 and 9 never need to be mapped to the boundary since they already contain the actual data to be used in forming the spatial interpolant. This leaves grid positions 10,14, and 15 (refer to as p10,p14,and p15, respectively) to be mapped. When p14 and p15 are not both a fill, the mapping only depends on p10, p14 and p15 which can be of either interior or fill type (in this case the mapping does not depend upon other grid points since the natural constraints dictate what they already must be). In addition, it is not possible for p10 to be type fill and p15 to be type interior simultaneously due to natural constraints. And, when p10, p14, and p15 are all interior grid points, no mapping is needed. For the cases when p14 and p15 are both fills, it is necessary to use the information from p18, p19, and p20 to determine the mapping. See figure 4.17 for all the mappings of the upper triangle of the 3×3 S8 sub-stencil in figure 4.14. The notation used in figure 4.17 is the same as in figure 4.14. The arrows indicate which direction the fill point is to be mapped. Notice that the first two rows in figure 4.17 depend upon the

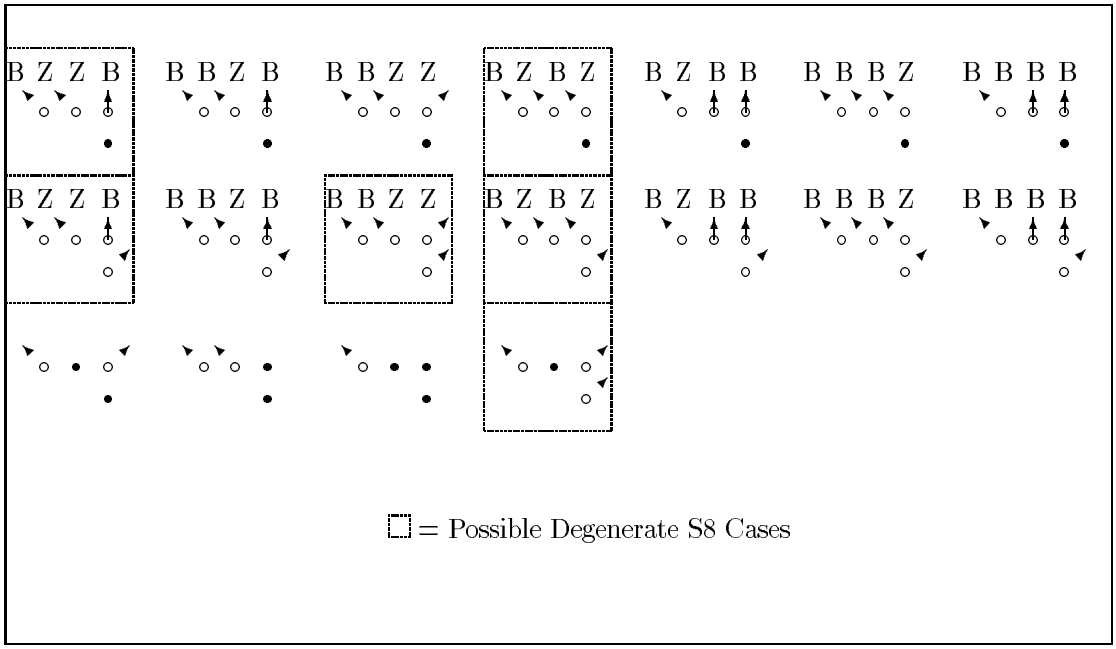

Figure 4.17: S8 Symmetrical Mapping: 18 Cases

type of grid point at p10, p14, 15, p18, p19, and p20. The bottom row only depends upon p10, p14, and p15. Those cases marked degenerate in figure 4.17 require human assistance since they may not work on all geometries. For example, the degenerate case in the top row of figure 4.17 may have a problem with its middle fill point. The other two fill points will definitely map to a nearby wall boundary, but the middle point could conceivably not intersect a wall boundary. Clearly, this is not likely to happen since geometry tends to be continuous and well-structured. It is most likely that the wall boundary grid point is at the p22 or p23 grid point. Rather than increasing the complexity of the mapping scheme at this time for these very rare cases, it seems acceptable to let the human provide occasional assistance. Note that there are 18 mappings for the upper triangle of the 3×3 S8 case. If the local symmetry were not utilized it would be necessary to determine the mappings for $18^2 = 324$ separate stencil configurations. While not intractable in two dimensions, this would become a more serious issue in three-dimensions.

Flipping the 18 mappings in figure 4.17 across the diagonal symmetry line provides the mappings for the other half of the 3×3 sub-stencil.

4.6.2 Mapping the S7 Cases

A mapping is found for the left side of the sub 3×3 S7 stencil in figure 4.16 using the same procedures as for the S8 case just discussed. However, the line of symmetry is now vertical instead of diagonal. Grid position 13 of the 5×5 S7 stencil will always be assigned unit direction vector $(0, 1)$ since by the S7 assumption a boundary grid point is directly above it. The fixed interior grid point locations p3 and p8 never need to be mapped since they are interior grid points. This leaves grid point locations p2, p7, and p12 to be mapped. See figure 4.18 for all the mappings of these grid points under the S7 assumption. The S7 assumption has over twice as many cases as the S8 assumption (40 vs. 18 cases respectively). As in the S8 cases, using the local symmetry saves the effort of mapping the $(40^2 = 1600)$ cases that would occur if symmetry were not used. Again, significant simplifications in the mapping problem occur in three-dimensions when symmetry is applied.

4.6.3 Handling the Degenerate Cases

The degenerate cases that are outlined in figure 4.18 and figure 4.17 are designed to never map more than 3 grid points collinearly onto a coordinate axis, but the solid wall may not be close enough to the mapped fill points along the direction of the unit direction vector, or may not exist at all. This could result in an interpolation function of lower accuracy than required or simply not be correct. The direction of the arrows in degenerate cases are selected with an educated guess but could be made better by including information outside of the 5×5 stencil or using information not directly adjacent to the S8 sub-stencil in figure 4.14, but inside the 5×5 stencil. The degenerate cases may be removed by increasing grid resolution to smooth out the crevices and bumps.

There are some situations in which the S7 or S8 no wrap assumption is not valid. Shown in figure 4.19 is a geometry in which neither the S7 nor S8 assumption is true for the case when the Z's are fills and the center of the 5×5 stencil is at the larger circle. It is characterized by little bumps no bigger than the grid spacing which occur in a corner area. Fortunately, CAD systems represent their curves with smooth parametric descriptions that will not typically have tiny bumps. This would be one case in which human assistance is necessary, or a relaxation of the S7 and S8 assumptions is needed. Since these cases are so rare, human assistance seems

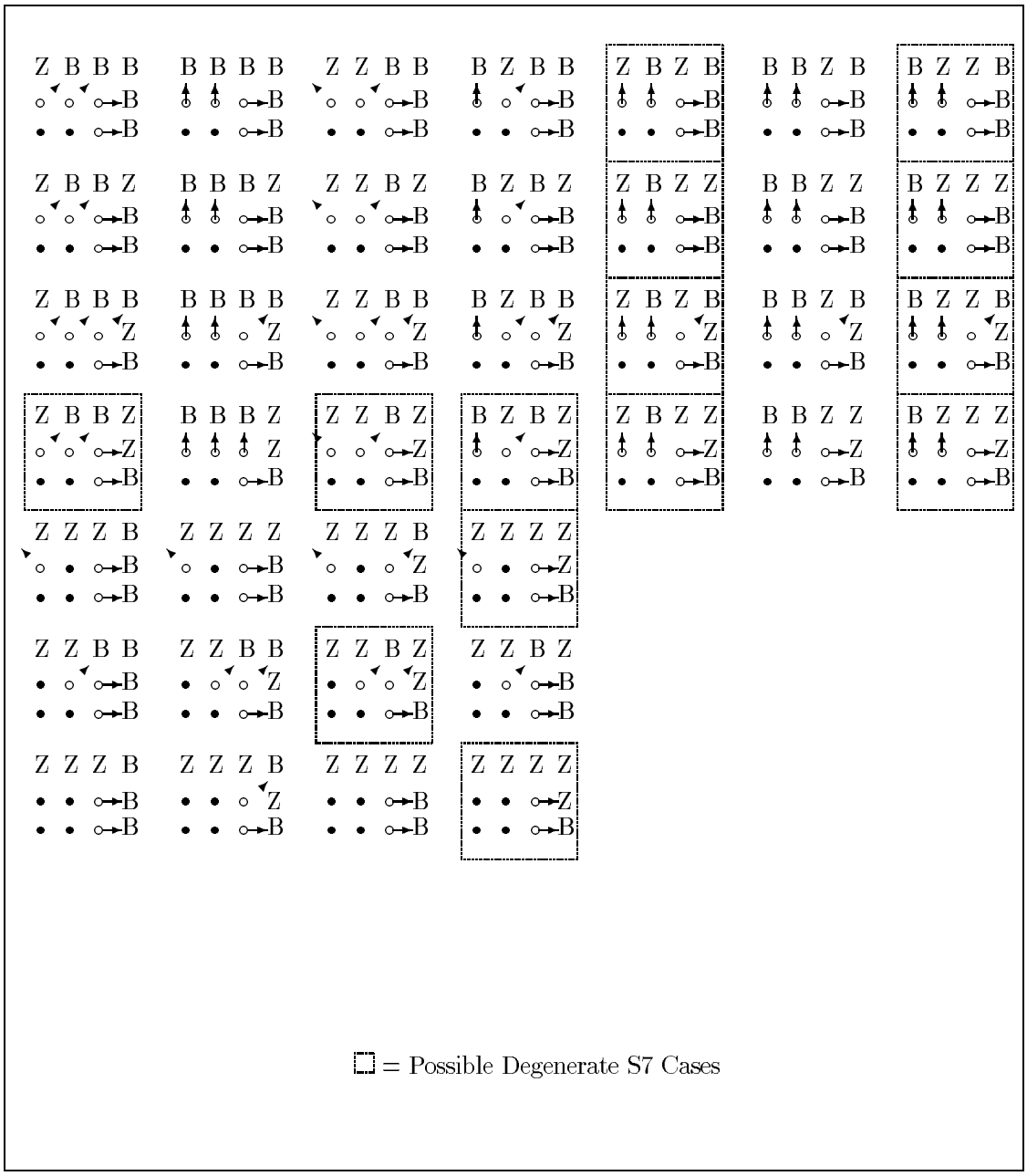

Figure 4.18: S7 Symmetrical Mapping: 40 Cases

more efficient. Or, these rare cases may also simply be included with the other mappings but this was not necessary in this work. All of these special cases may be calculated using the Stencil Constraint Tree algorithm and Recursive Boxes algorithm discussed previously.

If the degenerate cases are included in the acceptable mapping set but with a limit to the arrow size imposed, then acceptably accurate interpolations could still be automatically generated. Simply restricting the maximum arrow mapping length to 2 or 3 grid spacings will permit the degenerate cases to function correctly, albeit less accurately. In figure 4.22 is the previously discussed difficult case of figure 4.19 in which the highlighted fill point has had neighboring fill point changed to a boundary point. By permitting stretched arrows it is possible to correctly map and create an interpolant for this slightly changed system, despite the many bumps still along the edges. Some degenerate cases, even with the stretched arrows, may still not intersect a physical boundary. But, the fill points may still be mapped by using an alternate set of non-degenerate cases. A good example of this is shown in figure 4.23. Notice the corner problem when matching up with the S7 case as shown, that the last degenerate S7 mapping of figure 4.18 applies to left side of the 3×3 sub-stencil. The arrow extends to infinity, but after it exceeds 2 or 3 grid spacings this case would be rejected. However, notice that the same set of fill points may be solved by using two non-degenerate cases as shown in figure 4.23. The first 3×3 stencil number 1 in figure 4.23 is an S8 case in which all the Z's are interior points. This stencil solves the corner fill point that was previously mapped to infinity. The second 3×3 S7 stencil number 2 easily maps the remaining fill points. In this manner, it is possible to avoid degenerate mapping situations by replacing them with alternate non-degenerate stencil mappings.

One of the strengths of automation is that these many decisions can be performed systematically and quickly. In addition to avoiding degenerate cases, it is important to carefully select the order in which the fill points are solved as discussed in section 5.2.

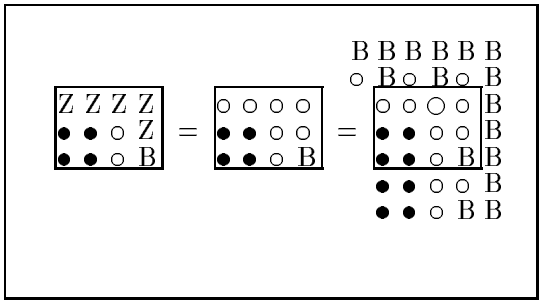

Figure 4.19: Degenerate Case: No S7 or S8 matching case

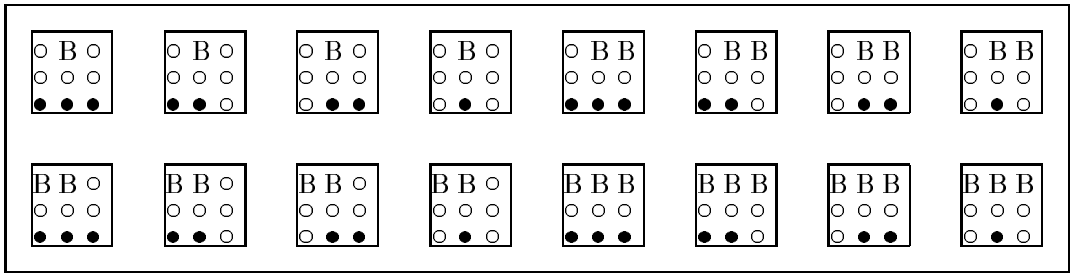

Figure 4.20: All Possible S7 Stencil Configurations with Fill at Center : 16 Cases

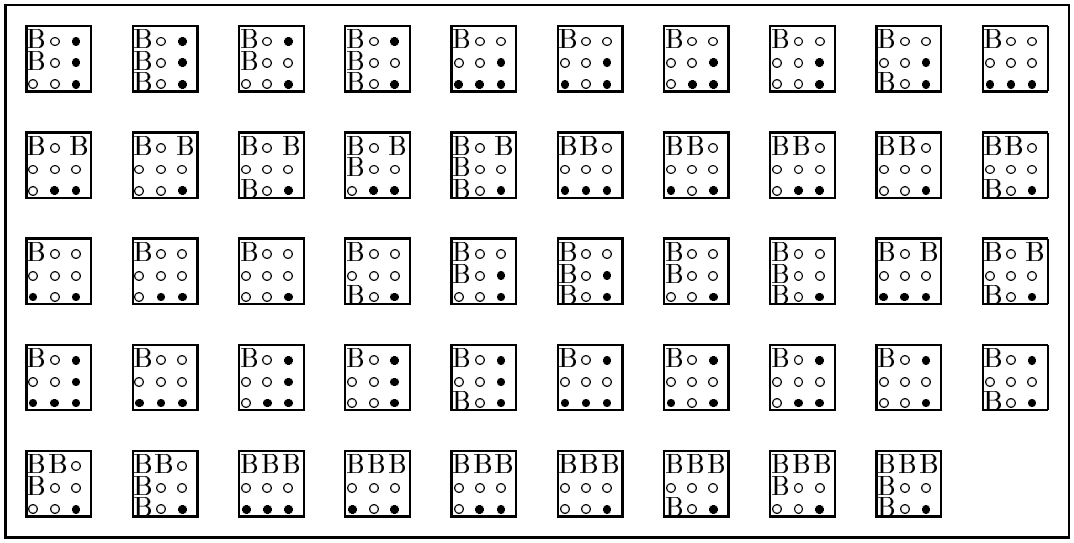

Figure 4.21: All Possible S8 Stencil Configurations with Fill at Center : 49 Cases

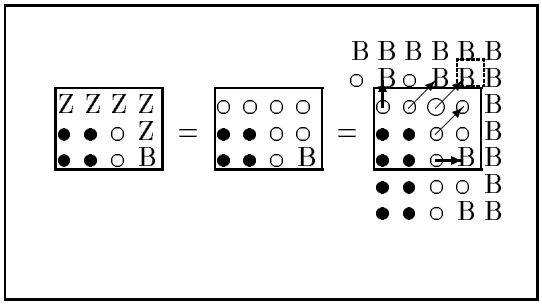

Figure 4.22: Degenerate Case: One S8 matching case

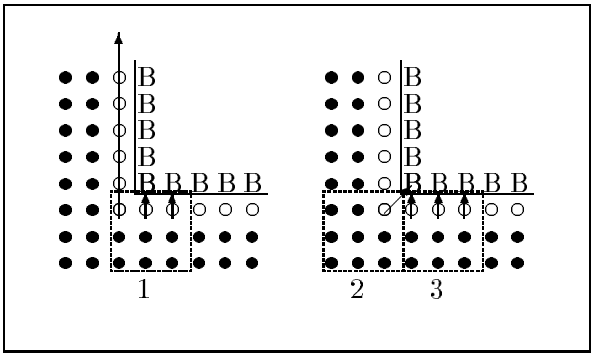

Figure 4.23: Substituting Non-Degenerate Cases 2 and 3 to Remove Degenerate Case 1

Chapter 5

Solving Near Boundary Grid Points

The objective of the previous chapter was to develop a unique mapping of the fill points to the solid wall boundaries. Each fill point on the Cartesian grid domain is contained within at least one 3×3 stencil domain. From within a particular 3×3 stencil domain, never more than 3 mapped fill and/or interior grid points are collinearly located in a line that is parallel to the coordinate axes. This chapter will take advantage of this property by creating a local spatial interpolant in each of the 3×3 stencil domains. The local spatial interpolant is then used at each time step to solve the fill grid points.

The domains of the local spatial interpolants that are defined on the 3×3 stencils for the rotated box problem are represented as shaded regions in figure 5.1. The solid circle grid points, open circle grid points and letter "B" grid points represent interior grid points, fill grid points, and boundary grid points, respectively. Each 3×3 stencil domain has a number in the center of it corresponding to the sequence in which its spatial interpolant is created and the sequence in which its fill points are evaluated. The arrows indicate where the fill points are mapped onto the wall boundaries. Some of the fill points are contained within two or more stencil domains. For example, stencil domains 1 and 2 share one fill point. This fill point may use either stencil domain's local spatial interpolant. The choice is not arbitrary however as the numerical stability can be affected as discussed later in section 5.2.

There are multiple ways of arranging the 3×3 stencil domains to cover all the fill points. In figure 5.3, a numerically stable scheme with twenty stencils are used for the 32 fill points as compared to the eight stencils used in figure 5.1. And in figure 5.2, eight stencils are used with the same fill point mapping and stencil domains as in figure 5.1, but this time the stencil domains are solved in a different sequence–resulting in an unstable scheme. In general, the numerical stability of the MESA schemes with wall boundaries depends upon the location of the stencil domains and the sequence of their evaluation as discussed in section 5.2.

5.1 Lagrangian Form VS. Multidimensional Taylor Series Form

A local polynomial spatial interpolation function is found for each of the 3×3 stencil domains (shaded regions in figure 5.1). The interpolator function is a polynomial that is consistent with the local boundary conditions at the mapped fill locations and it is simultaneously consistent with the interior grid points contained within the stencil domain that it is defined.

The local spatial interpolants are piecewise continuous across the regions of stencil domain overlap. The mathematical form of the local interpolating polynomial can reduce the computational effort of creating and evaluating it. For example, it is well known that using Horner's form [15] reduces the number of multiplies required to evaluate a polynomial. Similarly, it is possible to reduce the number of unknown parameters of the interpolating polynomial while maintaining the same accuracy by choosing a polynomial form that best fits the application. This section will demonstrate the benefits of using the Lagrangian forms of the spatial interpolants for computing fill grid point solutions.

5.1.1 Forming the Interpolant With Multidimensional Taylor Series

Each local polynomial spatial interpolation function can be formed in various ways once the fill points are mapped to the wall boundaries. The fill points are mapped in the manner of chapter 4, along a unit direction vector to a location on the physical wall boundary. The location is determined by finding the intersection of the line drawn in the direction of the unit vector and the parametric curve representing the boundary. If the interpolation function is

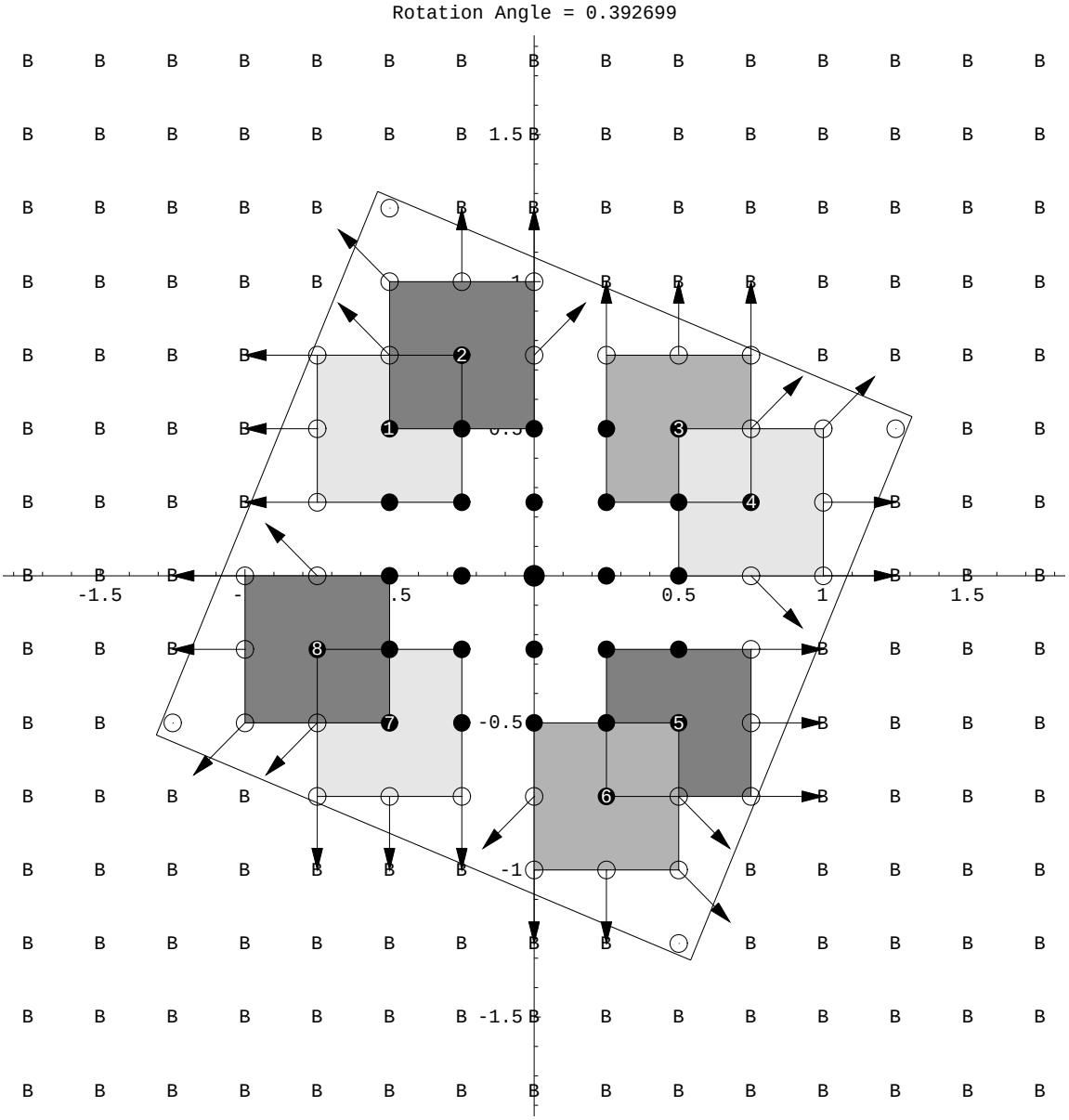

Figure 5.1: Stable Sample Mapping: Box Rotated $\frac{\pi}{8}$ Case

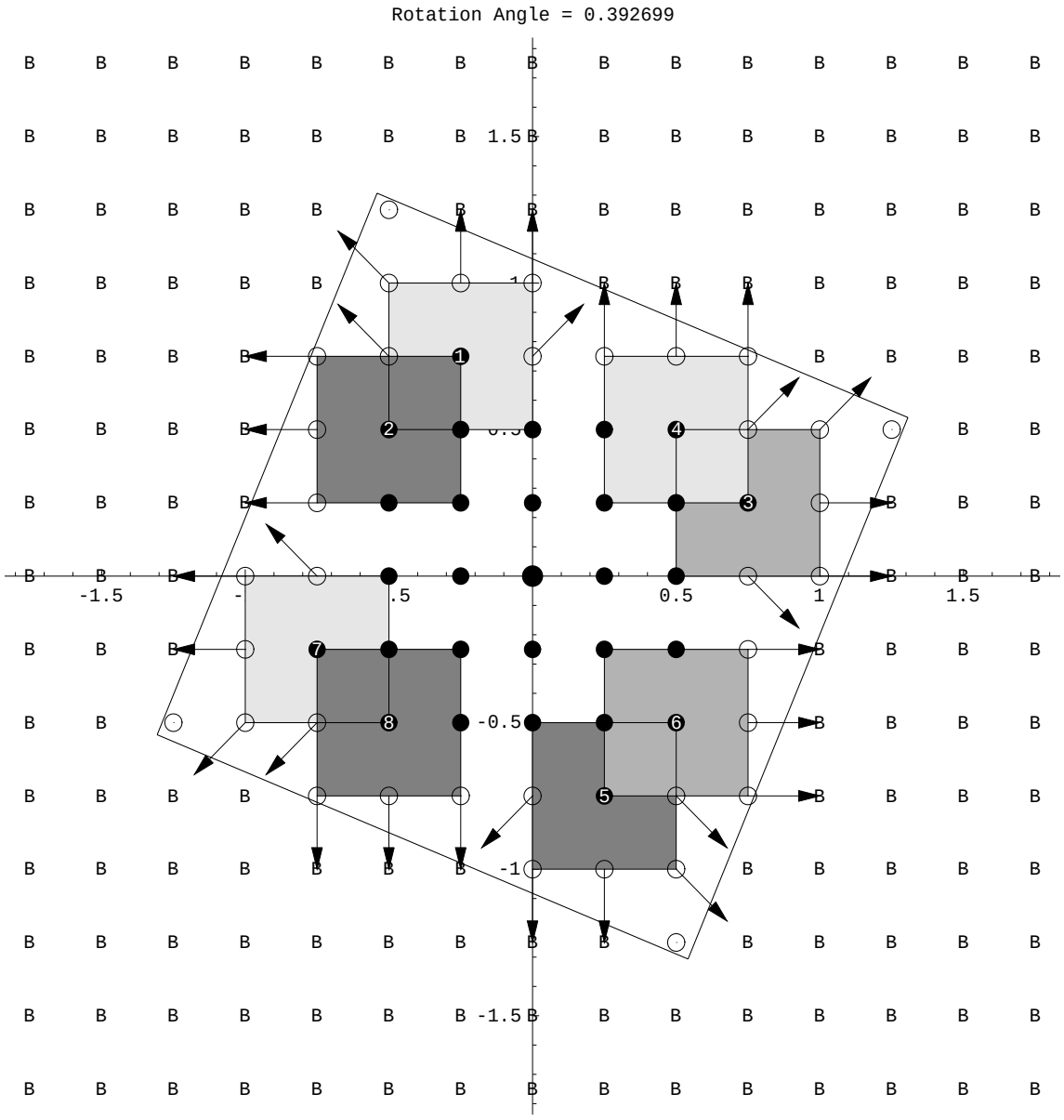

Figure 5.2: Unstable Sample Mapping: Box Rotated $\frac{\pi}{8}$ Case

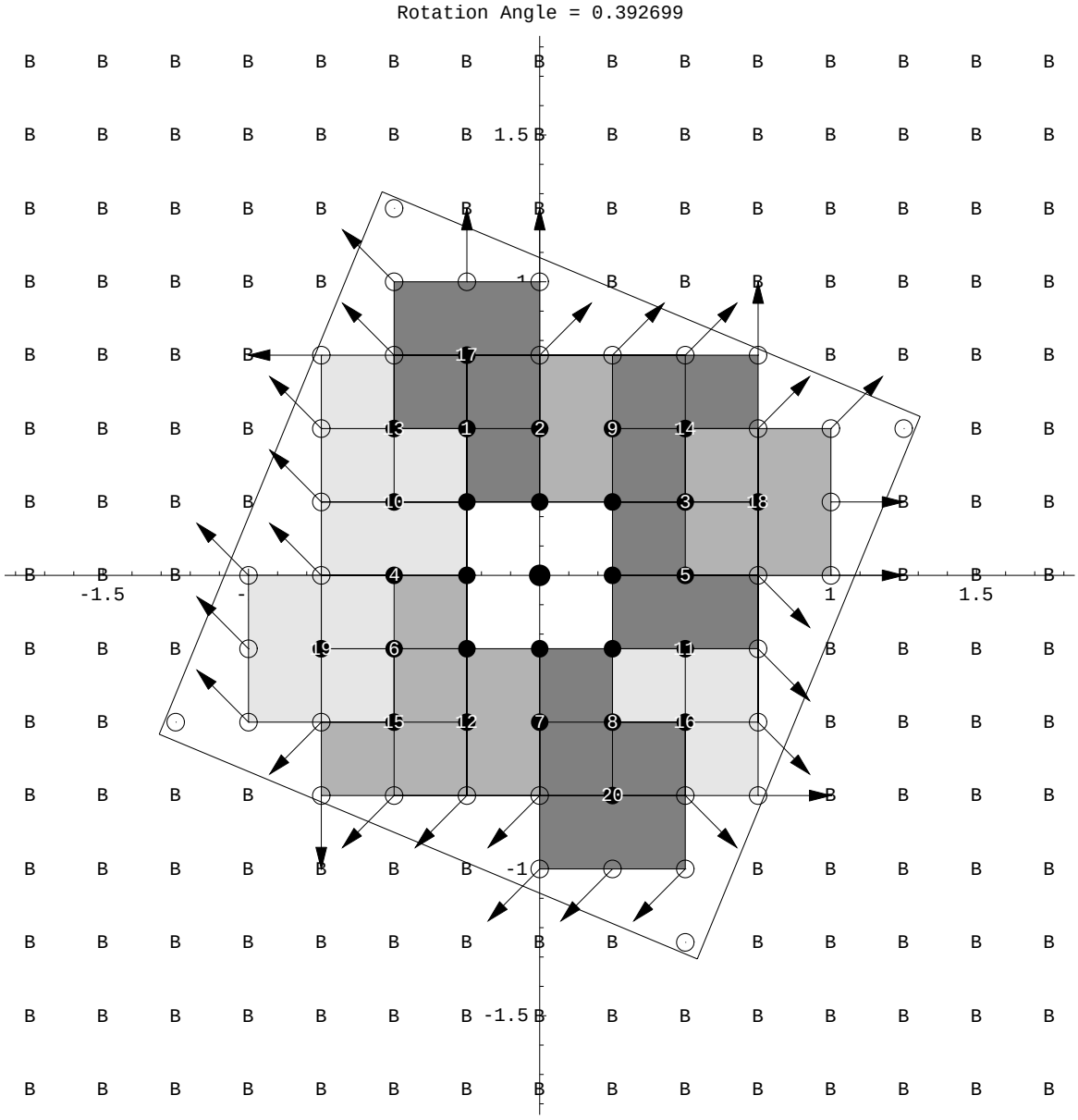

Figure 5.3: Stable Sample Mapping: Box Rotated $\frac{\pi}{8}$ Case

expressed in multidimensional Taylor series or Lagrangian form, it is then necessary to solve for the unknown parameters by forming a consistent set of linear equations. Each equation of the system is the spatial interpolant function evaluated at a known interior grid point which is set equal to the data at the interior grid point. Or, the spatial interpolant function is evaluated as part of a boundary condition at a wall boundary and set equal to zero. An example of this procedure is shown next.

Consider the c3o0 MESA scheme in which stencil number 1 (shaded box 1) of figure 5.1 needs an interpolation function to evaluate the 4 fill points within its domain at each time step. The assumed boundary conditions are:

$$\frac{\partial p}{\partial \eta} = 0 \tag{5.1}$$

$$V_\eta = 0 \tag{5.2}$$

$$\frac{\partial V_\tau}{\partial \eta} = 0 \tag{5.3}$$

where $V = (u, v)$ is the velocity vector with velocity components u and v in the Cartesian x and y-axis directions , p is the scalar pressure, η is the normal to the wall surface directions, τ is the tangent to the wall boundary direction, V_τ is the velocity tangential to the wall, and V_η is the velocity normal to the wall.

If the three $2nd$ order local interpolation polynomials are in multidimensional Taylor series form:

$$p(x, y) = \sum_{i=0}^{2} \sum_{j=0}^{2} cp_{i,j} x^i y^j \tag{5.4}$$

$$u(x, y) = \sum_{i=0}^{2} \sum_{j=0}^{2} cu_{i,j} x^i y^j \tag{5.5}$$

$$v(x, y) = \sum_{i=0}^{2} \sum_{j=0}^{2} cv_{i,j} x^i y^j \tag{5.6}$$

Then each function has 9 unknown coefficients that need solutions. Solving the coefficients ($cp_{i,j}$) is achieved by inverting a 9×9 matrix formed from the 4 boundary condition equations 5.1 evaluated at each mapped boundary point and the 5 interior equations 5.4 are evaluated at each interior point. The boundary conditions force the simultaneous solution of $cu_{i,j}$ and $cv_{i,j}$ since

they operate on the velocity vector, and are inherently multidimensionally coupled; this requires the inversion of an 18×18 matrix. This matrix is formed from the 8 boundary condition equations (4 of equation 5.2 and 4 of equation 5.3) evaluated at the same mapped boundary locations as the pressure and the 10 interior equations evaluated at the interior grid point locations (5 equations for $u(x,y)$ and 5 equations for $v(x,y)$).

Once the coefficients $cp_{i,j}$, $cu_{i,j}$, and $cv_{i,j}$ are solved, then the spatial interpolants $p(x,y)$, $u(x,y)$, and $v(x,y)$ are defined for all locations within the stencil domain (ie. box number 1 in figure 5.1). The 4 fill points are expressed as a linear combination of the 5 interior grid points by evaluating the spatial interpolants at each fill point location. The fill grid point shared by stencils 1 and 2 of figure 5.1 has the following stable pressure solution with local origin at the center of box 1 and h is the distance between grid points:

$$p(0,h) = 0.854193\,p(h,h) + 1.6013\,p(0,0) - 1.33704\,p(h,0) - 0.667329\,p(0,-h) + 0.548872\,p(h,-h) \tag{5.7}$$

And the same fill point has the following pressure solution with local origin at the center of box 1 when the stencil sequence is changed as in figure 5.2:

$$p(-h,0) = 0.95015\,p(0,0) + 0.23779\,p(-h,-h) + 0.198062\,p(0,-h) - 0.386002\,p(h,-h) \tag{5.8}$$

The latter solution uses fewer interior grid points (4) than the former (5) and results in a numerically unstable solution.

Once the 4 fill points are evaluated in box 1 of figure 5.1, only three of the five interior grid points have enough information in their 3×3 stencils to be time advanced. Two interior grid points require data from fill points in box number 2 and box number 8 of figure 5.1. In practice, all fill points are evaluated first (before the interior points), so that the interior grid points may be time advanced simultaneously for improved computational performance.

5.1.2 Forming the Interpolant with Lagrangian Polynomials

Regardless of which stencil domain box is being used, there will always be 9 unknown coefficients ($cf_{i,j} \forall i,j \in (0,1,2)$) per primitive variable function $f(x,y)$ when the multidimensional Taylor series form of the interpolants is used with the c3o0 MESA scheme in two-dimensions. The

Lagrangian form avoids this by taking advantage of the regular Cartesian grid structure used in this work.

If we assume the three local spatial interpolants $p(x,y)$,$u(x,y)$, and $v(x,y)$ are Lagrangian polynomials, then the unknown coefficients are simply the fill point values themselves. This results in no more than 7 unknown coefficients for each interpolation function since each stencil domain box will contain at least two interior grid points due to the S7 and S8 assumptions described in section 4.5.

For example, using the notation of figure 4.2, suppose we would like to time advance $U^n_{m,o}$ to $U^{n+1}_{m,o}$, where $U = p, u,$ or v. This corresponds to time advancing the center of box number 5 in figure 5.4.

Using the c3o0 MESA scheme requires using the six interior grid points $U^n_{m,p}$, $U_{m,o}$, $U^n_{m,m}$, $U^n_{l,p}$, $U^n_{l,o}$, $U^n_{l,m}$ and the fill grid points $U^n_{o,p}$, $U^n_{o,o}$, $U^n_{o,m}$ which are mapped to the locations $U^n_{b,p}$, $U^n_{b,o}$, $U^n_{b,m}$ respectively. The Lagrangian form of the spatial interpolants are:

$$fm(x) = \frac{x\,(-h+x)}{2\,h^2} \tag{5.9}$$

$$fo(x) = -\frac{(-h+x)\,(h+x)}{h^2} \tag{5.10}$$

$$fp(x) = \frac{x\,(h+x)}{2\,h^2} \tag{5.11}$$

$$gm(y) = \frac{y\,(-h+y)}{2\,h^2} \tag{5.12}$$

$$go(y) = -\frac{(-h+y)\,(h+y)}{h^2} \tag{5.13}$$

$$gp(y) = \frac{y\,(h+y)}{2\,h^2} \tag{5.14}$$

$$\begin{aligned} p(x,y) = {} & (p_{l,m}fm(x) + p_{m,m}fo(x) + p_{o,m}fp(x))gm(y) + \\ & (p_{l,o}fm(x) + p_{m,o}fo(x) + p_{o,o}fp(x))go(y) + \\ & (p_{l,p}fm(x) + p_{m,p}fo(x) + p_{o,p}fp(x))gp(y) \end{aligned} \tag{5.15}$$

$$\begin{aligned} u(x,y) = {} & (u_{l,m}fm(x) + u_{m,m}fo(x) + u_{o,m}fp(x))gm(y) + \\ & (u_{l,o}fm(x) + u_{m,o}fo(x) + u_{o,o}fp(x))go(y) + \\ & (u_{l,p}fm(x) + u_{m,p}fo(x) + u_{o,p}fp(x))gp(y) \end{aligned} \tag{5.16}$$

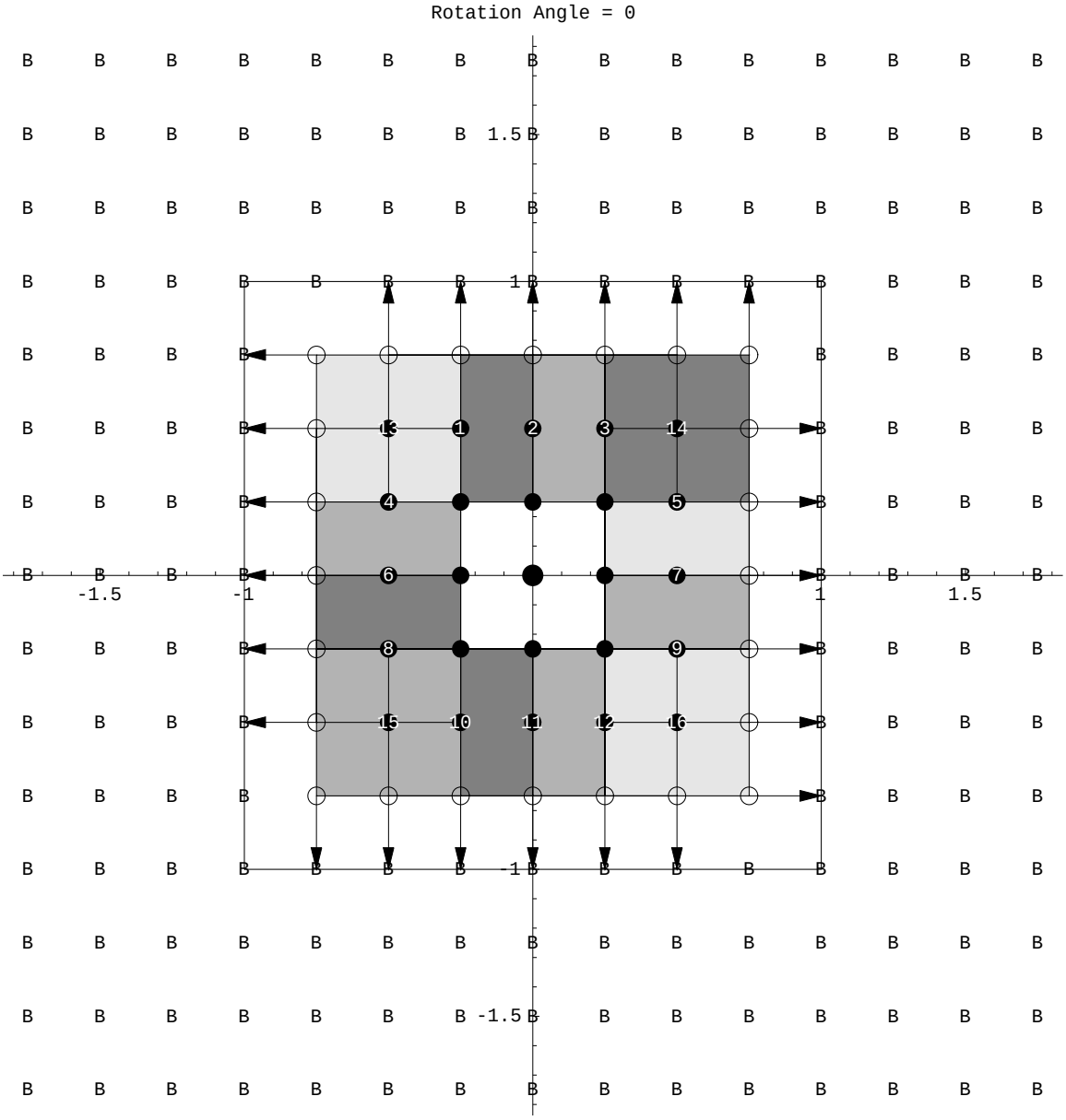

Figure 5.4: Stable Sample Mapping: Box Unrotated Case

$$
\begin{aligned}
v(x,y) \quad = \quad & (v_{l,m} fm(x) + v_{m,m} fo(x) + v_{o,m} fp(x)) gm(y) + \\
& (v_{l,o} fm(x) + v_{m,o} fo(x) + v_{o,o} fp(x)) go(y) + \\
& (v_{l,p} fm(x) + v_{m,p} fo(x) + v_{o,p} fp(x)) gp(y) \qquad (5.17)
\end{aligned}
$$

The coefficients $p_{a,b}$ are simply the pressure values at the corresponding grid points (a,b) : $a,b \in (m,o,p)$ in figure 4.2. In this case, six coefficients are known, corresponding to the six known interior grid points. The Lagrangian formula requires solving only 3 unknown coefficients of $p(x,y)$, ($p_{o,p}$, $p_{o,o}$, $p_{o,m}$), compared to the 9 unknown coefficients ($cp_{0,0}$, $cp_{1,0}$, $cp_{2,0}$, $cp_{0,1}$, $cp_{1,1}$, $cp_{2,1}$, $cp_{2,0}$, $cp_{0,2}$,$cp_{1,2}$, $cp_{2,2}$) required when using the multidimensional Taylor series form. Similarly, $u(x,y)$ and $v(x,y)$ have 3 unknown coefficients each. Solving the 3 unknowns in $p(x,y)$ requires only 3 equations or the inversion of a 3×3 matrix. The boundary condition 5.1 is applied at locations (b,p),(b,o),and (b,m) resulting in three equations which form a consistent linear system. In stencil domain 5 of figure 5.4, the boundary is exactly aligned with the grid so $a = 1$ in figure 4.2 and boundary locations (b,p),(b,o),and (b,m) are equivalent to grid locations (p,p),(p,o), and (p,m) respectively. The unknown coefficients, ($p_{o,p}$, $p_{o,o}$, $p_{o,m}$), will be expressed as a function of the six interior grid points contained within the stencil number 5 of figure 5.4. With its unknown coefficients solved, the primitive pressure function $p(x,y)$ in equation 5.15 is now known for all time as a function of interior grid values and is evaluated at each time step after the interior grid points are evolved in time.

Similarly, the coefficients of $u(x,y)$ and $v(x,y)$ are determined using the boundary conditions 5.2, 5.3 evaluated at the mapped wall locations. The coefficients $u_{a,b}$ and $v_{a,b}$ represent the grid values for $u(x,y)$ and $v(x,y)$ in the same way as the pressure coefficients $p_{a,b}$. The boundary conditions force the simultaneous solution of $u_{a,b}$ and $v_{a,b}$ since they operate on the velocity vector in an inherently multidimensional manner. There will be 6 equations required to solve the six fill points ($u_{o,p}$, $u_{o,o}$, $u_{o,m}$, $v_{o,p}$, $v_{o,o}$, $v_{o,m}$) compared to the 18 equations required when using the multidimensional Taylor series forms. Assuming the typically cubic cost, $O(N^3)$, to matrix inversion, the Lagrangian form's advantage grows with three-dimensional problems.

5.1.3 Forming the Interpolant with Hermitian Polynomials

The coefficients of the spatial interpolants used with the Hermitian MESA schemes may be solved using a Hermitian form of the interpolant to significantly reduce computational costs. The savings comes from minimizing the number of unknowns in the spatial interpolant by forming the spatial interpolant in a Hermitian form as shown in equation 5.23. Since only the aligned boundary case discussed in section 7.1.4 has been successfully solved, only a brief presentation of the techniques employed will be discussed here. The odd dimensioned stencils with Hermitian data are numerically unstable and so we will discuss the 2×2 stencil instead.

In the Hermitian form of the spatial interpolant, each unknown coefficient is defined by the data elements in the stencil. Since the interior grid points within a stencil are known, only the data elements at the fill points require a solution. Setting up the Hermitian form of the polynomial is accomplished using an extension of Newton's Interpolatory method [89]. The algorithm is based upon divided differences and the observation that each divided difference term is an approximation of the derivative of the function being interpolated. This procedure was developed and widely used before the advent of digital computers [15]. In this work, the procedure is executed in a completely symbolic manner so that the actual equation for the spatial interpolant is derived for a general set of data. Since the fill points are not known, it would not be possible to use this extension of Newton's Interpolatory method without the symbolic extension. In addition, the interior grid points vary in value at each time step and it would be necessary to run the algorithm at each time step if the symbolic form of the equation were not used instead as in equation 5.23.

Using the extension of Newton's Interpolatory method and executing it symbolically, a one-

dimensional Hermitian interpolant is found for the 5^{th} order c2o2 MESA scheme:

$$\begin{aligned} f(x) = & \frac{(x-x1)^3\ \left(6\,x^2+10\,x0^2-5\,x0\,x1+x1^2+3\,x\ (-5\,x0+x1)\right)\ fdata(0,x0)}{(x0-x1)^5} - \\ & \frac{(x-x0)^3\ \left(6\,x^2+x0^2+3\,x\ (x0-5\,x1)-5\,x0\,x1+10\,x1^2\right)\ fdata(0,x1)}{(x0-x1)^5} + \\ & \frac{(-x+x0)\ (x-x1)^3\ (3\,x-4\,x0+x1)\ fdata(1,x0)}{(x0-x1)^4} - \\ & \frac{(x-x0)^3\ (3\,x+x0-4\,x1)\ (x-x1)\ fdata(1,x1)}{(x0-x1)^4} + \\ & \frac{(x-x0)^2\,(x-x1)^3\,fdata(2,x0)}{2\,(x0-x1)^3} + \\ & \frac{(x-x0)^3\,(x-x1)^2\,fdata(2,x1)}{2\,(-x0+x1)^3} \end{aligned} \tag{5.18}$$

The two grid points in a row are labeled x0 and x1 in these equations and the data elements are labeled:

$$fdata(dx,x) = \frac{\partial^{dx} f(x)}{\partial x^{dx}} \tag{5.19}$$

Each coefficient of $fdata(dx, x_i)$ in equation 5.18 may be labeled HX_iD_{dx} and therefore the one-dimensional interpolant 5.18 may be written as:

$$\begin{aligned} f(x) = \quad & HX_0D_0\ fdata(0,x0) + HX_0D_1\ fdata(1,x0) + HX_0D_2\ fdata(2,x0) + \\ & HX_1D_0\ fdata(0,x1) + HX_1D_1\ fdata(1,x1) + HX_1D_2\ fdata(2,x1) \end{aligned} \tag{5.20}$$

Extending this to two-dimensions is a simple matter of creating the y-interpolant, f(y), by substituting y for x in equation 5.18 to get:

$$\begin{aligned} f(y) = \quad & HY_0D_0\ fdata(0,y0) + HY_0D_1\ fdata(1,y0) + HY_0D_2\ fdata(2,y0) + \\ & HY_1D_0\ fdata(0,y1) + HY_1D_1\ fdata(1,y1) + HY_1D_2\ fdata(2,y1) \end{aligned} \tag{5.21}$$

with

$$fdata(dy,y) = \frac{\partial^{dy} f(y)}{\partial y^{dy}} \tag{5.22}$$

The tensor product of these one-dimensional interpolants forms the following two-dimensional spatial interpolant in Hermitian form:

$$
\begin{aligned}
f(x,y) = \\
&(HX_0D_0\ fdata(0,0,x0,y0) + HX_0D_1\ fdata(1,0,x0,y0) + HX_0D_2\ fdata(2,0,x0,y0))HY_0D_0 + \\
&(HX_1D_0\ fdata(0,0,x1,y0) + HX_1D_1\ fdata(1,0,x1,y0) + HX_1D_2\ fdata(2,0,x1,y0))HY_0D_0 + \\
&(HX_0D_0\ fdata(0,0,x0,y1) + HX_0D_1\ fdata(1,0,x0,y1) + HX_0D_2\ fdata(2,0,x0,y1))HY_1D_0 + \\
&(HX_1D_0\ fdata(0,0,x1,y1) + HX_1D_1\ fdata(1,0,x1,y1) + HX_1D_2\ fdata(2,0,x1,y1))HY_1D_0 + \\
&(HX_0D_0\ fdata(0,1,x0,y0) + HX_0D_1\ fdata(1,1,x0,y0) + HX_0D_2\ fdata(2,1,x0,y0))HY_0D_1 + \\
&(HX_1D_0\ fdata(0,1,x1,y0) + HX_1D_1\ fdata(1,1,x1,y0) + HX_1D_2\ fdata(2,1,x1,y0))HY_0D_1 + \\
&(HX_0D_0\ fdata(0,1,x0,y1) + HX_0D_1\ fdata(1,1,x0,y1) + HX_0D_2\ fdata(2,1,x0,y1))HY_1D_1 + \\
&(HX_1D_0\ fdata(0,2,x1,y1) + HX_1D_1\ fdata(1,1,x1,y1) + HX_1D_2\ fdata(2,1,x1,y1))HY_1D_1 + \\
&(HX_0D_0\ fdata(0,2,x0,y0) + HX_0D_1\ fdata(1,2,x0,y0) + HX_0D_2\ fdata(2,2,x0,y0))HY_0D_2 + \\
&(HX_1D_0\ fdata(0,2,x1,y0) + HX_1D_1\ fdata(1,2,x1,y0) + HX_1D_2\ fdata(2,2,x1,y0))HY_0D_2 + \\
&(HX_0D_0\ fdata(0,2,x0,y1) + HX_0D_1\ fdata(1,2,x0,y1) + HX_0D_2\ fdata(2,2,x0,y1))HY_1D_2 + \\
&(HX_1D_0\ fdata(0,2,x1,y1) + HX_1D_1\ fdata(1,2,x1,y1) + HX_1D_2\ fdata(2,2,x1,y1))HY_1D_2
\end{aligned}
\tag{5.23}
$$

with

$$
fdata(dx,dy,x,y) = \frac{\partial^{dx+dy} f(x,y)}{\partial x^{dy} \partial y^{dy}} \tag{5.24}
$$

Notice that the spatial interpolant is now written in a form in which the unknowns are simply the data elements. If grid point (x0,y0) is the only fill point in a 2×2 stencil, then all data elements in equation 5.23 matching fdata(dx,dy,x0,y0) require a solution. The remaining elements, though left in algebraic form, are actually known at each time step and do no require a solution. The other data elements are calculated using the Tensor product form discussed in chapter 3.

Additional boundary conditions need to be developed for the Hermitian methods since they

have more information at each grid point. The boundary conditions are developed directly from the linearized Euler equations in which the convection velocity is zero.

The assumed boundary conditions for the rotated box problem of section 7.1.4 are:

$$\begin{aligned} \frac{\partial^{2n+1+t} p}{\partial \eta^{2n+1} \tau^t} &= 0 \\ \frac{\partial^{2n+t} V_\eta}{\partial \eta^{2n} \tau^t} &= 0, \quad \forall n,t : n,t \in (0,1,2,\ldots) \\ \frac{\partial^{2n+1+t} V_\tau}{\partial \eta^{2n+1} \tau^t} &= 0 \end{aligned} \tag{5.25}$$

where $V = (u, v)$ is the velocity vector with velocity components u and v in the Cartesian x and y-axis directions, p is the scalar pressure, η is the normal to the wall surface direction, τ is the tangent to the wall surface direction, V_τ is the velocity tangential to the wall, and V_η is the velocity normal to the wall. The maximum values of n and t depend upon the order of the MESA scheme, $O \geq (2n + 1 + t)$.

Construction of these boundary conditions is simplified using the following relationship [28]:

$$\frac{\partial^{n+t} f(x,y)}{\partial \eta^n \tau^t} = (\eta.\nabla)^n (\tau.\nabla)^t f(x,y) \tag{5.26}$$

For example, define the unit normal vector, $\eta = (normx, normy)$, then:

$$\frac{\partial^2 f(x,y)}{\partial \eta^1 \tau^1} = \begin{matrix} normx\, normy & f^{(0,2)}(x,y) + \\ (normx - normy)\ (normx + normy) & f^{(1,1)}(x,y) - \\ normx\, normy & f^{(2,0)}(x,y) \end{matrix} \tag{5.27}$$

.

The spatial interpolant function f(x,y) is defined in equation 5.23 and may represent the pressure ($p(x,y)$), or the velocity ($V(x,y) = (u,v)$) in two-dimensions. Using the coefficients of the function f(x,y) in equation 5.27 and the fact that data element $f^{(dx,dy)}(x_i, y_j)$ will always have the multiplier $(HX_i D_{dx})(HY_j D_{dy})$ in equation 5.23, the boundary conditions can be quickly constructed symbolically.

In particular, it is possible to represent the linear system as:

$$\mathcal{M}f = \mathcal{N}d \tag{5.28}$$

where $\mathcal{M}$ is the matrix of functions of normx, normy, HX_iD_{dx}, and HY_jD_{dy}; f is the vector of fill grid point data elements in the stencil; $\mathcal{N}$ is the matrix of functions of normx, normy, HX_iD_{dx}, and HY_jD_{dy}; d is the vector of interior grid point data elements.

The solution to the fill points is then:

$$f = \mathcal{M}^{-1}\mathcal{N}d \tag{5.29}$$

Finding the inverse of $\mathcal{M}$ is difficult with higher order MESA schemes due to poor conditioning of the matrix. It was possible to invert matrix $\mathcal{M}$ with schemes up to 11^{th} order accuracy. Higher accuracy has not yet been achieved though other approaches remain to be tried. First, the difficulty of a badly conditioned system may be avoided by solving the system 5.28 symbolically. Using current computer algebra packages it is not possible to quickly solve this system in a direct manner. However, an approach using interpolation is discussed in [72] that can produce the inverse of the matrix symbolically while avoiding the combinatorial explosion which occurs when using LU and Gaussian elimination symbolically. Second, the system may be solved using Modular Arithmetic in which only integers are manipulated [125]. Finally, the conditioning of the system changes by varying the set of boundary conditions used, changing the grid spacing, or multiplying the matrix system with pre-conditioners.

If the system 5.28 could be solved symbolically into form 5.29 then not only would the conditioning issue be avoided, but the solid wall geometry could be permitted to move (as in turbomachinery). Also, all possible stencil configurations could be pre-computed reducing the time required to develop spatial interpolants for each fill point. This will be pursued in later work.

5.1.4 Insuring Consistent Linear Systems

The mapping of the fill points to the wall boundary must never place 4 or more fill and/or interior grid points that are within the same stencil domain (shaded boxes) in a line that's

parallel to the coordinate axes. To do so would create an inconsistent linear system preventing the calculation of a local spatial interpolant for the stencil with this mapping. Fortunately, the mapping developed in chapter 4 insures that every possible 3×3 stencil will have a unique mapping that provides a consistent linear system (though the degenerate cases may require human assistance). A corollary to the Fundamental Theorem of Algebra [94] is that there is a unique polynomial curve of some degree that passes through a particular set of points. Therefore, the Lagrangian and Multidimensional Taylor series forms are equivalent and will produce the same results. It is possible then to study the properties of any polynomial of same degree that passes through the stencil points to understand the properties of the Lagrangian form. If more than 3 fill and/or interior grid points are on a line parallel to the coordinate axes, the only way to interpolate along this line is to use at least a third order interpolating polynomial in one dimension. For example, to create a spatial interpolant for the pressure in multidimensional Taylor series form that could interpolate across a stencil with four points along the y-axis, the following minimal degree polynomial is required:

$$p(0, y) = cp_{0,0} + cp_{0,1}y + cp_{0,2}y^2 + cp_{0,3}y^3 \tag{5.30}$$

But the c3o0 MESA scheme is only second order accurate and does not use the $cp_{0,3}$ term which can be considered a third order partial derivative term in the y direction [39]. Instead, it uses the terms of the pyramid mneumonic of chapter 3, ($cp_{0,0}$, $cp_{1,0}$, $cp_{2,0}$, $cp_{0,1}$, $cp_{1,1}$, $cp_{2,1}$, $cp_{0,2}$, $cp_{1,2}$, $cp_{2,2}$). It is desirable to use the same spatial interpolant forms when interpolating the fill points as the MESA schemes do in their spatial interpolation step to insure uniform treatment of the entire grid for stability, accuracy, and isotropy. One of the themes of the MESA approach is to treat all aspects of the problem in the same manner and to as closely as possible emulate the information flow of the actual physics. All acoustical physics interactions are nearest neighbor and the domain of dependence of the hyperbolic linearized Euler equations fits within a 3×3 stencil.

By using the Hermitian MESA schemes (c2oD, c4oD, c6oD, ...) it is possible to increase the accuracy and resolution of the numerical scheme without increasing the stencil size. Not only do small stencils simplify the mapping problem, but they more faithfully reflect the information flow of the physics itself. Typical CAA schemes use larger stencils to gain spatial accuracy

but most of the grid points used in these large stencils are not in the domain of dependence and therefore do not truly reflect the information flow of the actual physics. At the very least carrying the extra stencil information of a very large stencil can lead to significant inefficiencies and possibly introduces instabilities.

5.2 Systematic Stencil Selection

For typical applications many fill points will need to be mapped to the complex geometry solid wall boundaries. The mapping and stability of the scheme depends upon the sequence and location of the stencil domains shown by figures 5.1 and 5.2. A particular fill point may be solved using one of several possible interpolation functions depending upon which of the overlapping 3×3 stencils is used, unless of course only one stencil domain contains the fill point. A systematic way of selecting the location and sequence of the stencil domains has been developed in this work that provides accurate and stable solutions for the c3o0 MESA scheme. The Hermitian MESA schemes, however, need additional work to be stable in all cases.

5.2.1 Maximize Interior Information

A method for choosing the location and sequence of the stencil domains based upon maximizing the use of interior information was found that produces stable solutions to the c3o0 MESA scheme in all cases attempted. The method selects those 3×3 stencils that contain the most interior grid points. As was shown in equations 5.7 and 5.8, the number of interior grid points used to calculate a given fill point can vary. In figure 5.1 box number 1 has four fill and five interior grid points; whereas box number 2 has five fill and four interior grid points. The spatial interpolant in box number 1 should be solved before box number 2 to maximize the number of interior points influencing its fill points. But it is possible to do better than that as shown in figure 5.3. In this case, box number 1 has two fill points and seven interior points. Notice no fewer than two fill points can exist in a 3×3 stencil for this problem. A 3×3 stencil will always contain at least one fill point. Also, while box 1 in figure 5.3 has two fills, so do boxes 2, 3, 4, 5, 6, 7, and 8; Any of those boxes could have been solved first. However, all stencil domains containing 2 fill points should be solved before those contain 3 fills. Therefore, boxes 9, 10, 11, and 12 are solved next. This is followed by boxes with 4 fill points (boxes 13, 14, 15, and 16)

And finally, boxes 17, 18, 19, and 20 contain five fill points and are solved last.

This method of stencil selection also encourages multiple overlap of the stencils which was found to be necessary for numerically stable systems as well. However, overlapped stencils that minimize interior information are generally unstable as in figure 5.5.

An example of the lack of overlap using minimal interior ordering with small 2×2 stencils is shown in figure 5.6 and the same case but using maximal interior ordering is shown in figure 5.7.

The fill points shared by stencils in the overlapped region provide the coupling necessary to form local spatial interpolants that are piecewise continuous across the computational domain. But, this coupling can result in excessive error growth. Therefore, another advantage of ordering stencil domains by fill point count is the natural separation of the stencils which serves to quell the growth of certain propagation modes. This is seen in figure 5.4 in which boxes 1,2,3 are connected, but then box 4 is not connected to box 3.

It was also possible to create a stable MESA scheme with solid wall boundaries by alternately using two unstable schemes at every other time step. This essentially requires using two different sets of stencil selection algorithms but an automated method for doing this was not obvious nor apparently needed. But this does demonstrate some of the complex behavior that is occurring. Proving stability for all cases is not currently possible for complex geometries due to limits in the mathematical theory [45]. It is possible to prove stability for a specific case by examining the eigenvalues of the evolution matrix, but it is usually quicker and less expensive to simply test the algorithm on a real case. Since the stability of the general multidimensional case cannot be proven at this time, these numerical experiments offer guidance should future instabilities arise.

5.3 Isolated VS. Implicit VS. Recycled Fill Point Solution

Once the mapping of the fills and the selection of the shaded stencil domains is completed, the fill equations can be determined in several ways: isolated, implicit, or recycled. The wrong selection of stencil domains results in an unstable numerical scheme as mentioned. Slightly perturbing the mapping (adjusting the arrows) of the fills did not noticeably alter the accuracy or stability. The fill equations not only depend upon the mapping and location of the stencil domains, but also upon how the overlapped regions are dealt with. The Isolated method ignores the overlap,

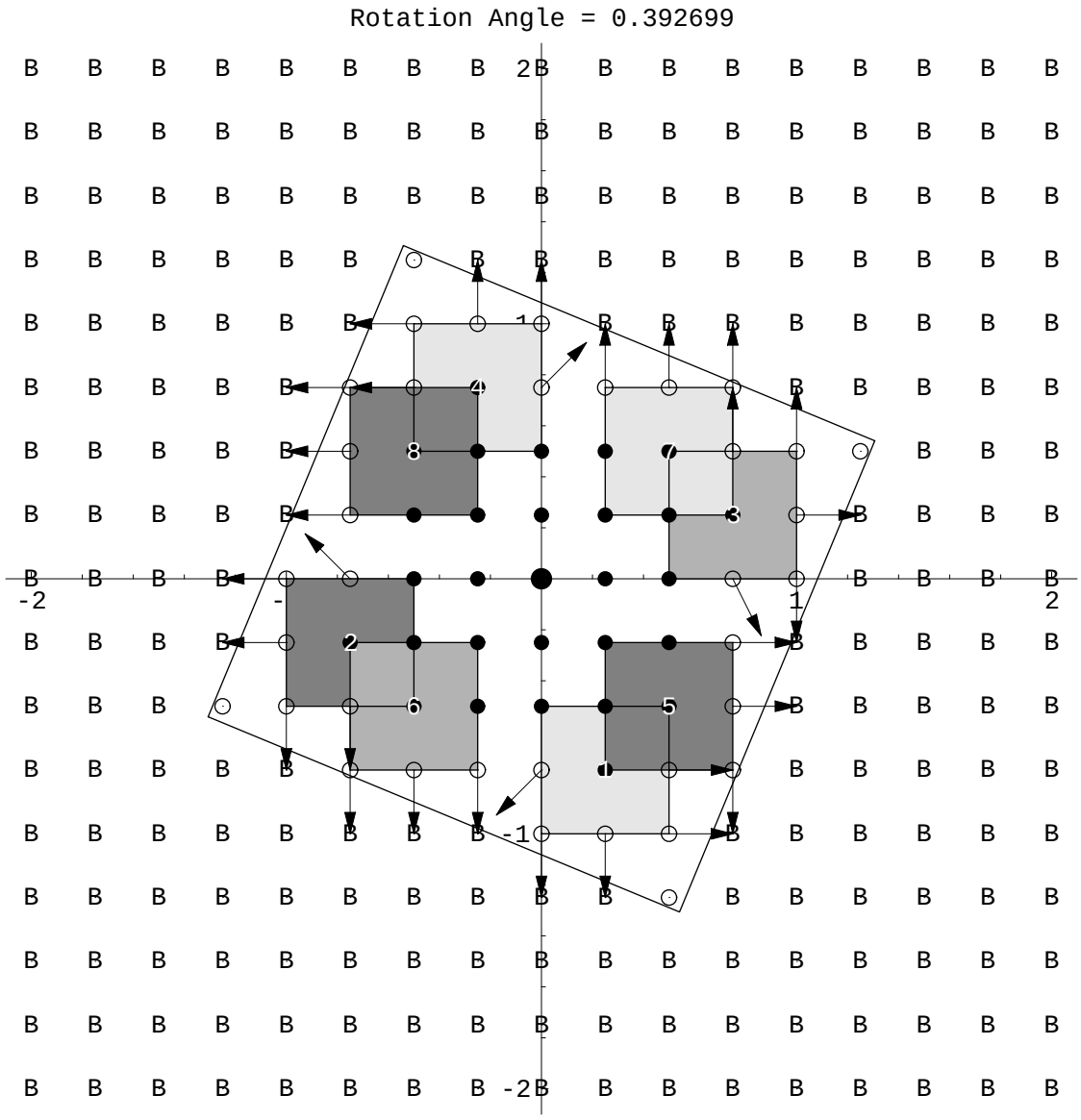

Figure 5.5: Sample Mapping Ordered by Minimal Interior Dependency: Box Rotated $\frac{\pi}{8}$ Case

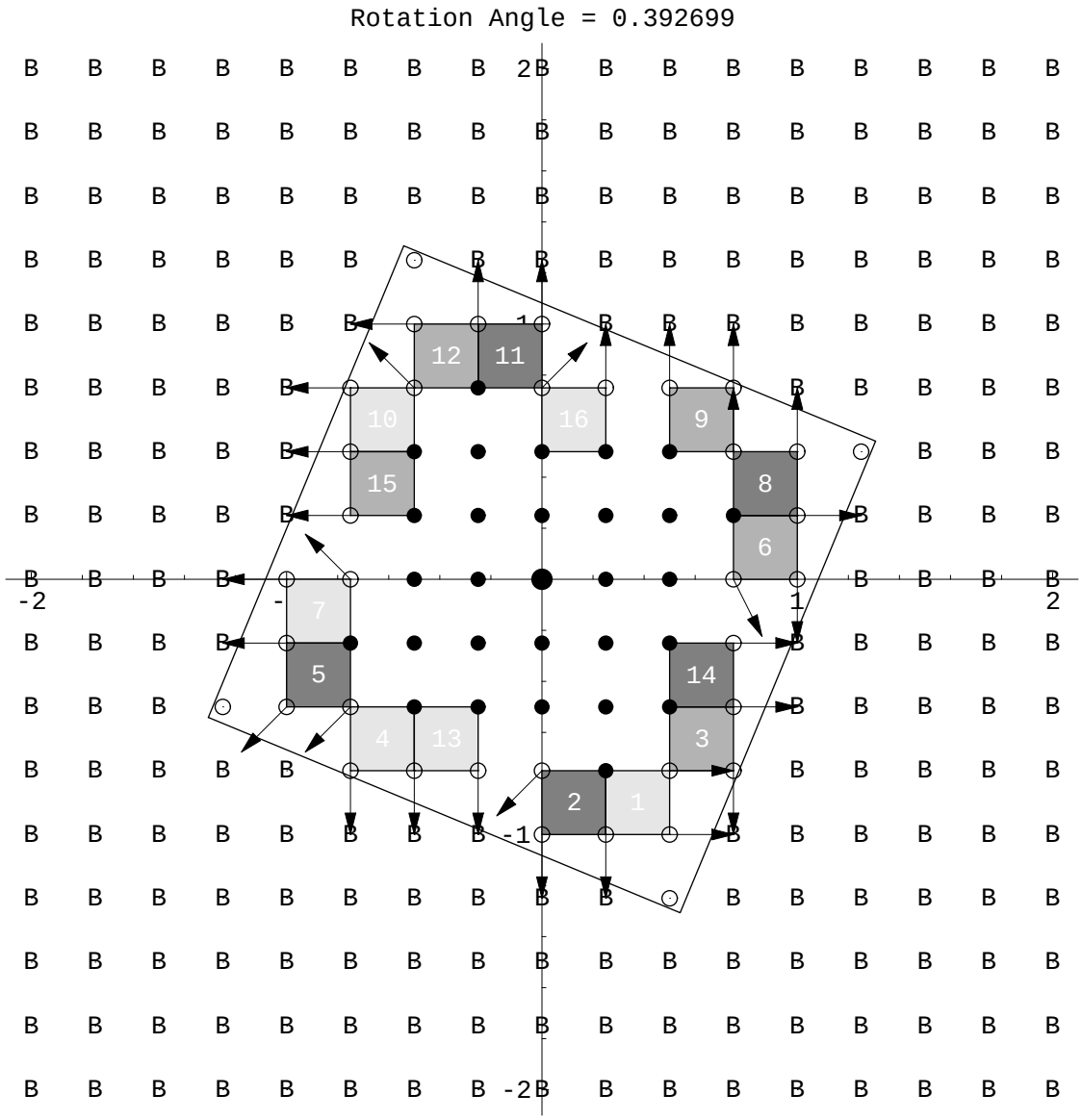

Figure 5.6: Sample Mapping Ordered by Minimal Interior Dependency – Small 2×2 Stencil: Box Rotated $\frac{\pi}{8}$ Case

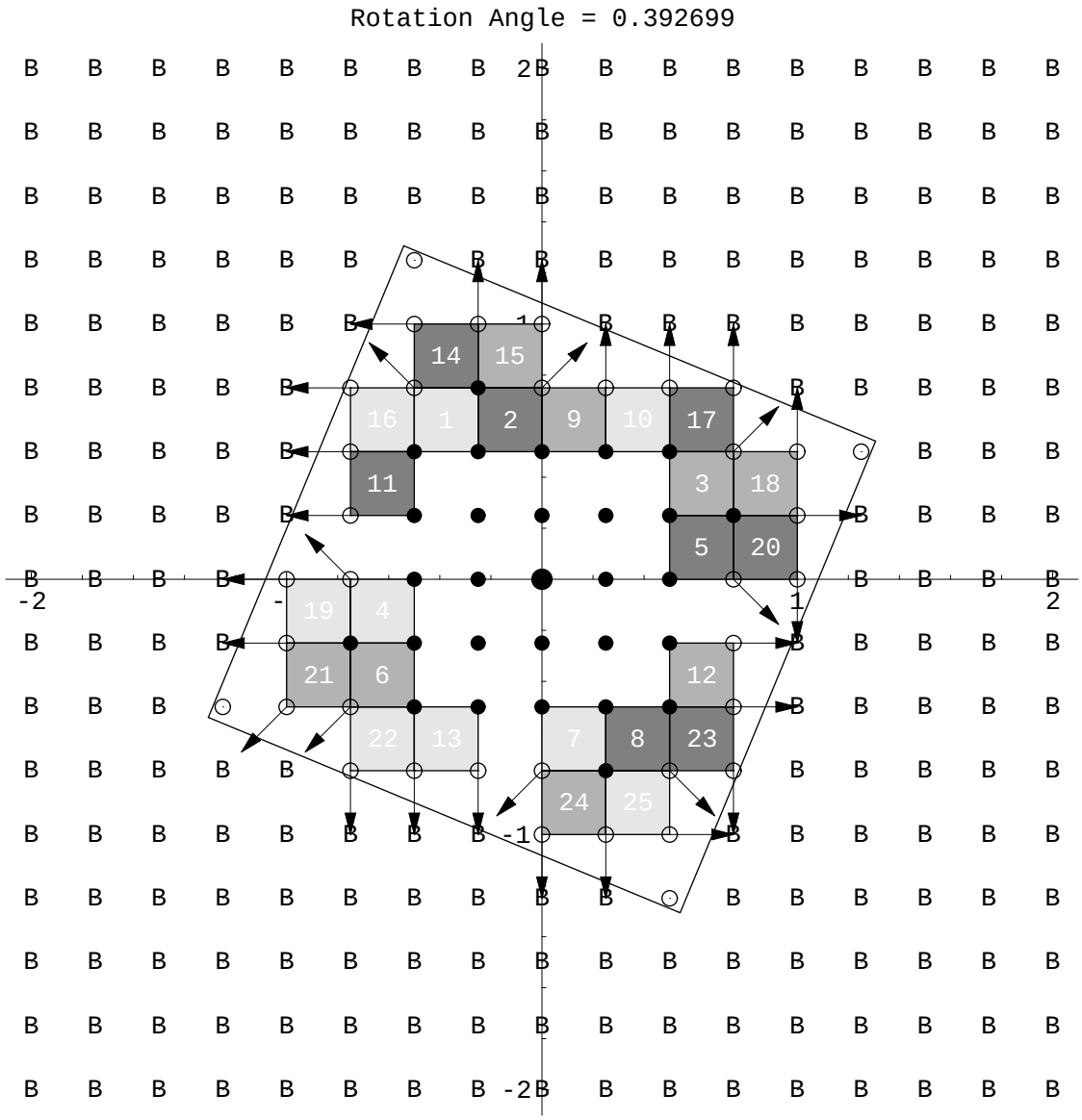

Figure 5.7: Sample Mapping Ordered by Maximal Interior Dependency – Small 2×2 Stencil: Box Rotated $\frac{\pi}{8}$ Case

and the Implicit and Recycled methods couple the overlapped stencils.

5.3.1 Isolated Method

The Isolated method is unstable but its basic principles help in the understanding of the other two methods, which are stable. The Isolated method treats each shaded region, as in figure 5.1, as an independent system. The box number 1 is solved first, creating solutions for the 4 fill points, including the one shared with box number 2 using the local spatial interpolant developed for box number 1. Then box number 2 is solved, creating solutions for its 5 fill points, except it does not assign an equation to the shared fill point since that point has been assigned already using the local spatial interpolant in box 1. This process is repeated for all shaded boxes. Since the boxes are solved independently, there is an inherent instability created at shared fill points since each stencil would in general have a different solution for its shared fill points. The arrow mapping could result in more than one arrow direction at a fill point creating another possible instability. This instability is again caused by the fact that, in general, each local spatial interpolant will generate differing solutions at a particular point that is shared by another local spatial interpolant–unless of course the shared point is an interior point since both spatial interpolants are by definition equal to the shared interior point.

5.3.2 Implicit Method

The Implicit method attempts to remove these instabilities by forming a piecewise continuous spatial approximation of the entire grid domain. This is achieved by simultaneously solving all the unknown spatial interpolant coefficients in every stencil domain that contains a fill point. Using the Lagrangian interpolation form results in one equation for each fill point, except when a fill point is shared between two or more stencil domains. Shared fill points can use one equation from any of the stencils it is contained within.

The Implicit method, in general, results in a large matrix to invert. For example, the rotated box in figure 5.1 has 32 fill points creating 32 equations which require solving 32 coefficients simultaneously. The advantage here is that the shared fill points are now truly shared since they will have the same values using any of the local spatial interpolants from stencils in which they are contained. This approach creates stable numerical methods, but is very expensive. Not

only is the matrix inversion process (which has cubic complexity) expensive, but the fill point solutions become a linear combination of many interior grid points. In figure 5.3, the continuous overlap of stencils implies that the last stencil number 20 depends upon information from the first stencil number 1. The fill point solutions in stencil number 20 could be a linear combination of the entire set of interior grid points in the computational domain, except the origin since it is not shaded! For real applications these costs would make this approach intractable.

5.3.3 Recycled Method

Fortunately, the Recycled method produces results identical to the Implicit method, but at a far reduced cost. The essential idea is to treat fill points that have been assigned by a previous stencil as an interior point in the next stencil domain that contains it. This way the overlapped stencils are still consistent with each other at all grid points in common. Rather than simultaneously solving the entire system as in the Implicit method, the smaller subsystems (each stencil) is solved in sequence. For the case in figure 5.3 the cost is $20O(3^3) = O(540)$ compared to the Implicit method cost of $O(32^3) = O(32768)$ for creating the equations for all the fills. And, the cost of evaluating the fill points is significantly lower as well. This savings occurs because treating the previously assigned shared fill points as an interior grid point in the subsequent stencil domains results in never having more than 8 grid points in a fill equation. Those pseudo interior grid points are actually fill points that must be evaluated in a previous step. Since some of the points in a stencil domain depend upon stencil data outside of the current stencil, the order in which the fill points are evaluated becomes important. This ordering couples the entire problem making it difficult to parallelize computations, though there are generally far fewer fill points than interior points in a typical engineering application. One advantage of the Implicit method is it expresses each fill point as a linear combination of interior grid points only and therefore, the sequence in which the fill equations are evaluated becomes irrelevant.

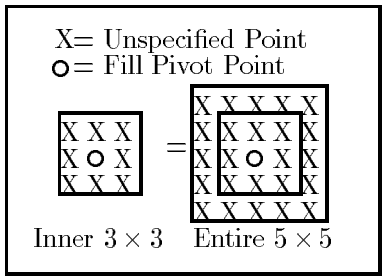

Figure 5.8: 5×5 stencil with fill point in center

5.4 Step-by-Step Demonstration of Mapping and Solving the Fill Points

It is important to systematize the process of mapping and solving the fill points so that arbitrarily complex geometries may readily be dealt with. An overview of one such process used in this work is presented next as a series of ten steps. These steps pertain to using 3×3 stencils as these were the most successful methods found in this work, but the same concepts may be applied to any size stencil.

Step 1 – Gather Needed Fills

Make a list (referred to as a "todo" queue) of all the fill points in the grid. Remove from consideration those fill points that do not have an adjacent interior grid point since they. are not part of the 3×3 stencil of an interior grid point and hence, are not needed. As an example, the fill points found in each corner (one in each corner) of figure 5.1 are not needed since the interior grid points will never require information from those grid points.

Step 2 – Test Stencil Configurations

Consider each 5×5 stencil in the entire computational domain that contains a fill point at its center (pivot point) as in figure 5.8.

Test the inner 3×3 stencil for correctness (9 tests per stencil) by using the stencil constraint tree with $N = 3$ as discussed in section 4.3. In two-dimensions, 216 possible cases under the no wrap assumptions (S1, S2, S3, S4, S5, S6, S7, and S8) are built into the tree as discussed in section 4.5 and require only 9 tests per stencil. If the inner stencil of figure 5.8 does not pass

this test, it is because the grid density needs to be increased, the CAD geometry file has an error, or a degenerate case has occurred that requires human assistance. Put these fill points at the end of the "todo" queue and deal with them later. These difficult fill points may actually be indirectly solved when a neighboring fill point that does not have these difficulties becomes the pivot point since all fill points around a pivot fill point are solved simultaneously.

Step 3 – Order the Fill Points

Order the fill points in the "todo" queue by the number of fill points contained in their 3×3 stencil from fewest to most. This has the effect of always using the most interior grid point information for each fill point and is used in the sequence shown in figure 5.3. If the order is reversed, the entire scheme is unstable despite providing piecewise continuous interpolation between stencil domains. The reversed order is equivalent to minimizing the number of interior grid points used for each fill point as shown in figure 5.2. Random sequences of stencil domains produce random results (some of which are stable despite not using the maximal number of interior points).

Step 4 – Rotate to Match S7/S8 Case

Those inner stencils that pass the test in step 2 will be of stencil type S1, S2, S3, S4, S5, S6, S7, or S8 as shown in figure 4.13. If it is not of type S7 or S8 then rotate it once about the fill pivot in figure 5.8 90 degrees and then retest it. Repeat this possibly two more times until the rotated stencil is of type S7 or S8 as shown in figure 4.14.

Step 5 – Map the Fill Points

Apply the correct mapping to the fill point(s) contained in the 3×3 S7 or S8 sub-stencil. The S7 sub-stencil has the fill pivot point in its top center position; While the S8 sub-stencil has the fill pivot point in its top left position. Use the mappings in figure 4.17 if the sub-stencil is type S8, or use the S7 mappings in figure 4.18 if its not. If the inner 3×3 stencil is of both S8 and S7 type, go with the S8 mapping since it is simpler to evaluate.

Step 6 – Unrotate the Stencil and Mapping

Unrotate the 3×3 inner stencil (if necessary) back to its original state, while at the same time rotating the arrows in the same direction about the fill pivot point by the same amount.

Step 7 – Determine Mapped Fill Point Locations on Wall Boundary

Starting at the fill points in the unrotated 3×3 S7 or S8 sub-stencil, extend a line in the direction of the assigned direction vector until it intersects the physical boundary. Repeat this for the other fill points (up to 7) in the sub-stencil. All the fill points within the sub-stencil are now uniquely mapped to a point on the physical boundary.

Step 8 – Compute the Local Spatial Interpolant

Generate the local spatial interpolant for the sub-stencil in the manner discussed in sections 5.1.2 and 5.3. The local origin of the spatial interpolant will be the center of the 3×3 sub-stencil, not the center (pivot point) of the inner stencil in figure 5.8. After this step, a local spatial interpolant is defined on the stencil's domain.

Step 9 – Assign Solutions to the Fill Points

Evaluate the local spatial interpolant at each fill point in the stencil, assigning its value to each fill point in the sub-stencil. Some of the fills that are not the fill pivot points are solved since the spatial interpolant may be evaluated at all grid points within the 3×3 sub-stencil. For example, consider the rotated box shown in figure 4.3. The topmost fill in the top corner of the box is not needed and is rejected in Step 1. The fill point directly beneath it is needed but it cannot be chosen as a fill pivot point since neither the S7 nor S8 assumption is satisfied when it is the center of a 5×5 stencil. It is placed at the end of the "todo" queue by Step 2. It is solved in this Step when either the fill to its left or to its right is a fill pivot point.

Step 10 – Remove Completed Fills From Queue

Remove those fill points from the "todo" queue that were assigned in Step 9. Pop the next fill point off the "todo" queue and process the next 5×5 stencil starting at Step 4. If the "todo" queue is empty then stop.

After completing all these steps, If the "todo" queue is not empty notify the human assistant.

5.5 Generating the FORTRAN Wall Boundary Input File

After developing the solutions for all the fill points in Mathematica, these solutions need to be conveyed to the FORTRAN code that is actually going to perform the calculations necessary for evolving the linearized Euler equations in time.

To accomplish this, Mathematica generates a file for the FORTRAN code to read that encodes the solutions for all the fill points. Since each fill point solution is essentially a linear combination of the interior grid point data, the file includes the coefficients and locations of all interior grid point data elements necessary for each fill point. In addition, this information is sequenced in the same order required from the Recycled method discussed in section 5.3. This file is simply stored as a one-dimensional array in FORTRAN and contains all the information necessary to correctly evaluate all the fill points in the computational domain at each time step.

The data file contains integer data types describing interior grid point locations and real data types representing the coefficients of each interior grid point. From FORTRAN's perspective all the data types are real, but the FORTRAN code converts the reals to integers when appropriate. Each fill point has a packet of information. The first two integers specify where the fill point is located. The next three integers specify the number of p, u, and v terms that form the linear combination solution for this fill point. Then the data contains sub-packets in groups of 5. The first two integers specify the location of an interior grid point, the next two specify its x and y derivatives, and the last number specifies its coefficient in the linear combination. The process repeats until all fill point solutions have been completely specified. In this manner, all fill point solutions are represented in a single ASCII file.

The FORTRAN code reads the fill point file once at the beginning and stores it in a one-dimensional array. Then the entire Cartesian grid domain is time advanced using the FORTRAN code generated in chapter 3. The fill and boundary grid points are time advanced as well using the interior MESA propagation scheme even though the data is garbage; This avoids the cost of determining which grid points are interior. Computing all the grid points at once also permits vectorized and parallelized execution of the MESA propagation scheme as discussed in the next chapter. After all the grid points in the computational domain are time advanced, the fill points

are evaluated using the information from the fill point file. Then this process repeats, all grid points are time advanced, then the fill points are solved. The fill points depend upon the interior grid points at the same time step, and therefore the interior points always need to be evaluated first. At the first time step, therefore, it is important to supply the fill points with the correct initial data. Afterward, the fill points are evaluated using the interior grid point data.

Chapter 6

Extension to Parallel Computational Domain

The algorithms presented thus far are capable of very high accuracy and resolution in space and time. In addition, since they are single-step explicit finite-difference methods that depend upon local data only, they can be easily implemented on a parallel computer. Since the algorithm development can be completely automated as demonstrated, it is desirable to also automate their parallel extensions. This chapter will discuss the procedures necessary to accomplish the automation that results in the generation of a load balanced, SPMD (Single-Program Multiple Data) model FORTRAN code which uses the MPI (Message-Passing Interface Standard). The ideas discussed apply to both two and three dimensional problems, but they were only implemented in two dimensions for this work.

6.1 Domain Decomposition

Developing a load balanced parallel FORTRAN code is greatly simplified by the use of Cartesian grids for the discretization of the physical domain. Recall that Cartesian grids also simplified the grid generation process and enabled automated treatment of boundary conditions in complex geometries. Regardless of the geometry's configuration, all problems may be approached in the same way.

As discussed, each grid point is defined to be a fill point, interior point, or boundary point. Rather than testing each grid point, it is more efficient to treat all points as an interior point and simply time advance the entire computational domain. After advancing all points, then the fill points are "filled" with the correct solution using the methods discussed. The boundary points are never used and so their values are irrelevant.

Therefore, a successful load balancing algorithm in this case will attempt to assign an equal number of grid points to each node. In most cases, it is not possible to achieve an equal number since the total number of grid lines in the computational domain may not be evenly divisible. The exact partitioning also depends upon the desired node topology. For example, it may be desirable to form a chain of 16 nodes in a line for certain duct problems or to form a 4×4 grid of nodes in a bi-periodic open domain problem as shown in the right side of figure 6.1.

In general, it is important to maximize the ratio of computation to communication, though this constraint may be lessened through the use of asynchronous communication as discussed later. For the bi-periodic open domain problem the computational domain should be decomposed into squares as this maximizes the area to perimeter ratio and thus minimizes communication delays. Each square will be assigned to a node and will be time advanced in parallel. At each time step, the perimeter of each square must be communicated to the node containing the neighboring square in the computational domain.

Each node uses the same FORTRAN code as is used in the serial (non-parallel) version. The nodes are assigned the proper initial data by defining the local grid origin in terms of a global grid origin. Error checking also uses this global coordinate information.

For the most part, each node is solving a rectangular open domain problem with the perimeter being communicated at each time step. In this way, no distinction is made of the type of data being communicated (repeated open domain data or neighboring interior grid data). Therefore, only the actual dimensions of each node in local coordinates is required. This assignment is achieved as follows:

1. Assign the minimum number of grid points in each dimension to each node

$$minh = \left\lfloor \frac{total\ grid\ points\ in\ horizontal\ direction}{nodes\ in\ horizontal\ direction} \right\rfloor \tag{6.1}$$

$$minv = \left\lfloor \frac{total\ grid\ points\ in\ vertical\ direction}{nodes\ in\ vertical\ direction} \right\rfloor \tag{6.2}$$

2. Assign the following extra points (if any) to the nodes in Round Robin fashion.

$$extraptsh = total\ horizontal\ grid\ points - (minh \times nodes\ in\ horizontal\ direction) \quad (6.3)$$

$$extraptsv = total\ vertical\ grid\ points - (minv \times nodes\ in\ vertical\ direction) \quad (6.4)$$

Starting with the left column of nodes, and proceeding a column at a time, the extra points are evenly assigned one per column. If there were extra points, then at least one column of nodes will contain one fewer column of grid points than the other columns of nodes. The same process is repeated for the extra vertical grid points, starting with the bottom row of nodes and proceeding one row at a time to the top row. When complete, the nodes will know how many grid points they have in each dimension.

3. Determine the local maximum coordinates for each node.

At this point, it is known how many grid points each node is assigned both vertically and horizontally. It is desired to have the origin of the local grid coordinates occurring in the center of each stencil so that the original serial code may be used without modification. This requires carefully assigning the maximum array dimensions of each node. If an even number of grid points is assigned to a node, then the maximum and minimum coordinates will differ by one; if it is an odd number these coordinates will be the same.

$$maxi = \frac{number\ of\ grid\ points\ assigned}{2} \quad (6.5)$$

$$mini = -maxi\ or\ if\ even\ mini = -maxi + 1 \quad (6.6)$$

4. Find which node contains the origin in global coordinates.

It is necessary for each node to know where its local origin is in relation to the physical problem's global coordinate system. This is achieved by starting with the absolute value of the bottom left node's (node 0) minimum local coordinate, mini. Subtracting from this the maximum global coordinate index, provides the global coordinate in the horizontal dimension of the center of node 0's local coordinate system.

$$horizontal\ center\ in\ global\ coordinates = -maximum\ global\ index + |mini| \quad (6.7)$$

The global coordinates of the center of the next node to the right of this node is found by adding the points per node found in step 2 to this value. In a similar manner, the remaining nodes are assigned their respective horizontal global coordinates at the center of their stencil. Next, the vertical global coordinates are determined at each center of each node. After this procedure, the node which contains global coordinates $(0, 0)$ contains the origin of the computational domain. All nodes can now quickly convert between global and local coordinate systems. The local coordinate systems simplify the extension of the serial code to the parallel code. The global coordinate systems permit the parallel assignment of initial conditions and error checking using the known analytical solution which are defined in terms of global coordinates.

All nodes will be aligned with their neighboring nodes after these procedures are completed as shown in figure 6.1. Alignment means that the boundary between any two nodes will contain the same number of grid points at both nodes. This alignment simplifies the communication among nodes.

6.2 Message Passing

The MESA schemes enjoy the advantage of requiring only local data. Some of the schemes in CAA, such as Compact Differences, require Spline interpolations across the entire computational domain which restrict their capacity for parallel computations and spread local errors across the entire computational domain. Also, the actual physics described by hyperbolic partial differential equations (such as the linearized Euler equations) only depend upon local data contained within the cone of characteristic curves (the domain of dependence). Indeed, the CFL constraint is based upon this fact and provides some basis into why the MESA methods perform well. Since only local data is required, communication between nodes is limited to nearest neighbor communication. Although, for the bi-periodic open domain problem, nodes on opposite sides of the computational domain must also communicate and so one would expect that typical engineering applications would actually perform better since they will have actual boundaries which reduces the need for communication.

Modern parallel computing systems include hardware for communicating and computing simultaneously. This capability can be fully utilized with the MESA schemes since the interior of each square may be time advanced independently of the perimeter communication. To simplify

the logic used in communicating, the nodes are assigned an (i,j) index with (0,0) being the bottom left node in the network topology. The engineer specifies in advance the desired number of nodes in each dimension. The computational topology will always be rectangular and is ideally suited for a mesh or toroidal network topology commonly found in today's parallel systems. If maxni and maxnj represent the maximum node count in the i and j directions respectively, then a particular node's ($node\ number\ =\ nodenumber$) coordinates, ($nodeindex_i, nodeindex_j$), are given by:

$$\begin{aligned} nodeindex_i &= mod(nodenumber, maxni) \\ nodeindex_j &= \frac{nodenumber}{maxni} \end{aligned} \tag{6.8}$$

With this indexing, it is straightforward to determine adjacent nodes. An example domain decomposition using 4 and 16 nodes with 8 grid points per unit in a mesh computational topology is shown in figure 6.1. The center of the physical domain is indicated by $\oplus$. The node number, ($nodenumber$) and its ($nodeindex_i, nodeindex_j$) coordinates are shown inside each node's domain.

The grid points identified as B's are not time advanced since a bi-periodic open domain boundary condition is assumed. Those grid points get their values from the opposite side of the grid. For example, in the four processor case, the left side of node 2 is identical to the right side of node 3; the bottom side of node 0 is identical to the top side of node 2; the top left corner of node 2 is identical to the bottom right corner of node 1. Since each node contains its own logical memory (MIMD), it does not know the values of the grid points contained within any other node. It is thus necessary for the nodes to communicate this information.

Sending information requires a node to know the node numbers of its neighbors. However, since the engineer may select any computational topology, the node topology is not known apriori. However, using the ($nodeindex_i, nodeindex_j$) information, it is possible to quickly identify the neighboring nodes. This is done by adding or subtracting one from the coordinates of its ($nodeindex_i, nodeindex_j$) pair. If $nodeindex_i$ equals 0 then the node is in the left column and its left node has $nodeindex_i = maxni - 1$. If $nodeindex_i$ equals $maxni - 1$ then the node is in the right column and its right node has $nodeindex_i = 0$.

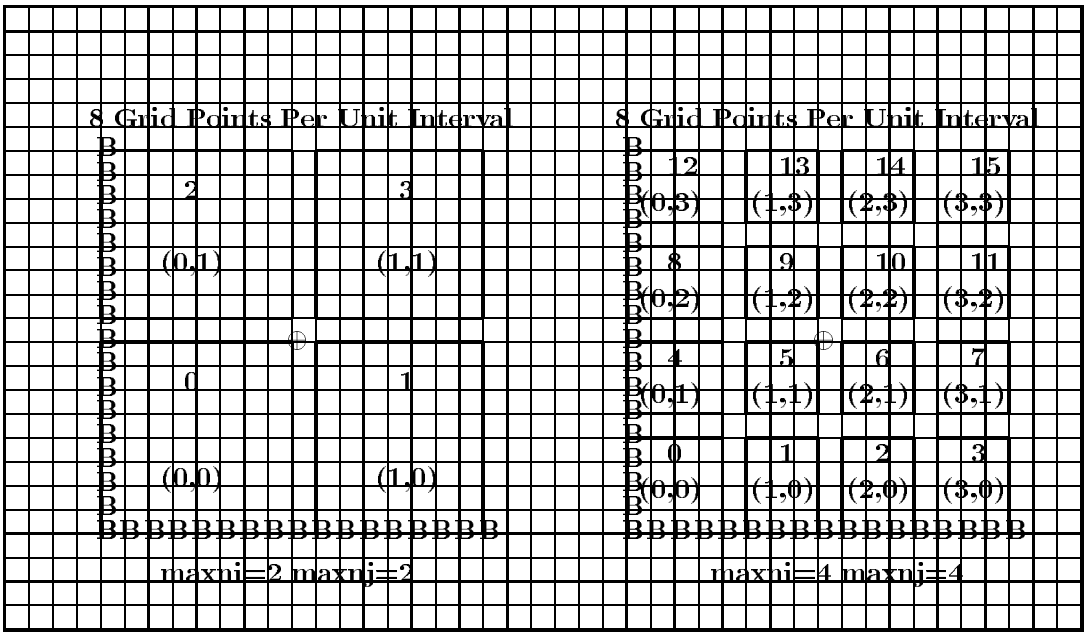

Figure 6.1: Solving bi-periodic open domain Linearized Euler Equations with MESA

These relations are true for the bi-periodic case, but not necessarily for actual engineering applications. The application may have radiation boundary conditions or solid wall conditions, both of which do not require information from neighboring nodes.

Once the new node index pair, $(nodeindex_i, nodeindex_j)$, is determined, it requires conversion to a node number, $nodenumber$, using the expression:

$$nodenumber = (nodeindex_j * maxni) + nodeindex_i \tag{6.9}$$

The message passing requirements of each node only depends upon its location in the mesh. If the node is not in the bottom row of figure 6.1, for example, then it will always need to communicate down and receive information from a node with index $(nodeindex_i, nodeindex_j - 1)$. Similarly, if a node is not in the right column of nodes (nodes 3, 7, 11, and 15 in figure 6.1) then it will always need to communicate right and receive information from a node with index $(nodeindex_i + 1, nodeindex_j)$.

6.3 Synchronous Communication

The periodic boundary information needs to be exchanged in the serial (non-parallel version) of the solver code. The exchange is accomplished by copying the data from one part of the data array to another (from within the same data storage area). MPI permits a single node to communicate to itself and so the parallel version running on a single node already has the information being communicated as in the serial version. For simplicity however, the solver uses the same communication scheme for any size computational mesh (including the 1×1 mesh). As soon as more than one node is used, then not only does periodic boundary information need to be communicated (the B's in figure 6.1), but in addition, the outer perimeter of interior grid points assigned to each node needs to be communicated to its neighboring nodes. The amount of information to be communicated depends upon the MESA scheme's stencil size and depth, which varies with the algorithm used. For example, a c2o2 MESA scheme will require more data to be communicated than a c2o1 MESA scheme because each grid point contains more information. Whereas the c9o0 MESA scheme requires the same communication as the c2o1 MESA scheme because c9o0 requires 4 rows of interior grid points and c2o1 requires one row of 4 data elements at a single grid point. Recall that some of the significant advantages of small stencils is their ability to be easily mapped to complex geometries and their high resolution capabilities.

In figure 6.2, the 5×5 stencil for the MESA c5o0 scheme has grid points labeled "X". The grid point designated "A" is to be time advanced and requires the grid point information labeled "X". Notice in this case that before "A" can be advanced in time, node 1 must communicate with nodes 0, 2, and 3 to get the necessary grid point information (due to the distributed memory model). A larger stencil would require even more information to be communicated. The MESA schemes that use derivative information at the grid points such as c2o3 or c4o1, may have a smaller stencil, but will require more information to be communicated since more information is stored at each grid point; The scalability studies in the next chapter examine the parallel performance of some of these schemes. In any event, before a time advance of an interior grid point near the edge of a node's domain may occur, the neighboring data from the neighboring nodes must be communicated.

It is often desirable to send fewer, but larger messages than it is to send many smaller

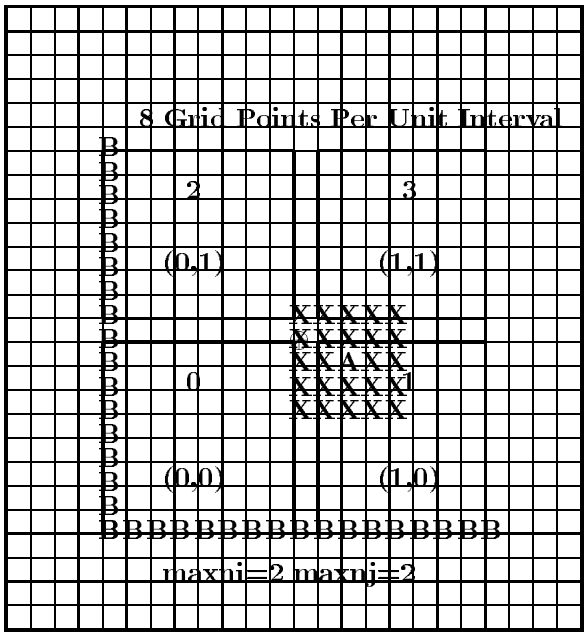

Figure 6.2: Nearest Neighbor Communication

messages in a parallel system since each message has an associated latency that can seriously degrade floating point performance. One way to send larger but fewer messages is to include the corner information in an implicit manner. Rather than explicitly communicating corner information as from node 2 to node 1 in the last example figure 6.2, it is possible to implicitly communicate the corner by having all nodes communicate to their left nodes simultaneously, then communicate to their right nodes, then communicate to their top and then finally to their bottom, in that order. In this case, the nodes send not just the interior grid points contained in their respective domains, but also a few extra columns of adjacent grid points that are not defined within their domain. This has the effect of communicating the corner information as shown in figure 6.3.

In figure 6.3 only the corner information from node 2 being implicitly communicated to node 1 is shown for simplicity. Notice that all nodes have an extra memory buffer on the perimeter of their defined nodes so that they can receive and store grid point data from neighboring nodes. The arrows indicate the direction of information flow and the numbers represent from which node the information came. The corner of node 2, "C1" is sent to node 3, where it is labeled "C2", but in addition, the grid point labeled "J1" in the buffer of node 2 is sent to node 3,

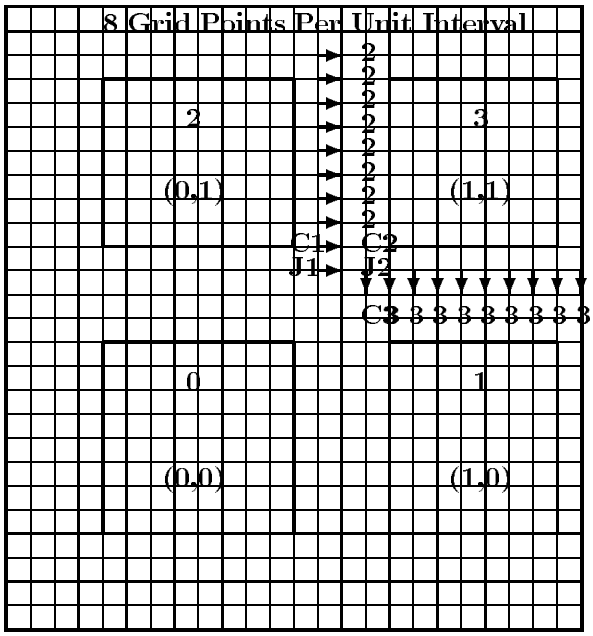

Figure 6.3: Synchronous Communication, Implicit Corner Exchange

where it is labeled "J2". At this point, grid point "J2" is junk information since it came from the undefined grid point "J1". Then during the up/down send operation the grid point labeled "C2" in node 3's buffer space, which is the corner of node 2 is sent to node 1 where it is labeled "C3". Even though it is not shown in the figure, after the send operations complete, the grid point labeled "J2" in node 3 actually gets the top-right interior grid point of node 0. The same occurs for all the corners in the computational domain.

Notice that the number of separate communications is cut in half using this procedure. Normally, it would be necessary to communicate left, right, up, down, down-right, down-left, up-right, and up-left. However, using the slightly larger communication buffers and communicating this information synchronously (not doing computations and communications simultaneously) in two steps (left/right and up/down) results in the corners be implicitly communicated. This eliminates the need to explicitly send corner information which would double the number of messages sent.

The periodic boundary data in the bi-periodic open domain problem interestingly can use the same communication scheme to avoid separate message passing of corners. This works despite the fact that the boundary data is not communicated to topologically neighboring nodes. In

this work it was possible therefore to treat all the nodes with the same communication logic.

6.4 Asynchronous Communication

While the left/right and up/down communication ordering can effectively exchange all necessary information, it is necessary to do that in two distinct steps. If left, right, up and down are done simultaneously then the corner information will not be communicated properly. It is, however, desirable to send all the information simultaneously on some parallel systems. In particular, the SGI ORIGIN2000 on which the parallel results are generated for this work have an additional communication processor that permits simultaneous computation and communication. Message passing libraries may then use asynchronous communication calls to get maximal floating point performance.

In the case of the MESA schemes this requires adding explicit communication calls for the corner information. To communicate asynchronously (simultaneously send all data and perform floating point operations), each node must send data left, right, up, down, right-up, left-up, right-down, and left-down simultaneously. Despite the doubling of messages compared to the synchronous case, significant improvement in performance can be realized with the MESA schemes. This is achieved by advancing the interior points of each node that do not depend upon information from neighboring nodes while at the same time communicating the outer perimeter of interior grid points. For a large enough interior, this effectively eliminates waiting for communication to complete. By the time the interior set of interior grid points has completed advancing, the information required for advancing the perimeter interior grid points is done transferring from the neighboring nodes.

Larger stencils reduce the number of interior grid points on a given node that may be computed without information from neighboring nodes. However, the Hermitian MESA schemes serve to shrink the stencil and increase the set of interior grid points on a given node that are independent of neighboring nodes. The Hermitian MESA schemes also increase the amount of work per grid point since the derivative data is advanced as well. Therefore, the Hermitian MESA schemes seem to be good choices for parallel applications. Some results on the efficiency of various Hermitian schemes are shown in section 7.3.

6.5 Generating the FORTRAN Parallel Propagation Code

One of the overall goals of this research has been to fully automate the code generation process. Therefore, the parallel FORTRAN code is also automatically generated and only requires specifying the dimensions of the mesh of nodes. The domain decomposition is easily automated because a Cartesian grid mesh is used. Load balancing the bi-periodic open domain problem is achieved by assigning all nodes approximately the same number of grid points. Each FORTRAN module has a corresponding Mathematica module and all modules communicate through a FORTRAN common block. The parallel extension relied extensively upon the automatic code generation Mathematica modules developed for the serial solver code. The extension to parallel essentially was an exercise in modifying local coordinate systems and array dimensions.

Specific tuning issues will vary depending upon the target architecture. For example, on systems without asynchronous communication hardware, it would be optimal to use the methods of section 6.3. Also, evaluating the fill grid points near walls requires an additional computational procedure and therefore the nodes with an excessively large set of fill points may be delayed. In addition, steep gradients may require adaptive grids or adaptive algorithms for their proper resolution, both of which may unbalance the computational load between nodes. It is likely some compromise solution that minimizes these effects will need to be developed later.

Nonetheless, the complete automation of the grid generation, algorithm generation, code generation, and parallel extension, is an advancement in CAA that may be used to advantage in the development of high accuracy and high resolution solver codes.

Chapter 7

Numerical Results

In this chapter, some example problems have been chosen which have analytical solutions that permit detailed examination of the accuracy and efficiency of the MESA algorithms. In addition, the linearized Euler equations describe a conservative physical system and therefore, in the case of no energy sources or sinks, the total energy of the system is constant in time and can provide an additional check on the numerical stability and dissipation of the schemes. The numerical stability of the algorithms can be determined analytically for open domain (no walls) problems, but the addition of complex wall geometry complicates the analysis. Techniques such as the Matrix Method or long running numerical experiments [100] are brute force methods of determining stability. The Matrix Method requires reducing the algorithms to a linear system of equations in which the eigenvalues of the matrix are examined. If all the eigenvalues are less than one, the method with that particular geometry is numerically stable. However, for typical problems the matrix is simply too large to do this efficiently, even in Mathematica. In this work, the MESA algorithms with wall boundaries are tested with long running experiments.

Both two and three-dimensional results will be discussed and will demonstrate the significant improvement in performance of the MESA schemes over traditional numerical methods. Finally, the two-dimensional MESA schemes are tested for their scalability on a typical MIMD parallel computer.

7.1 Two-Dimensional Problems

Most work in CAA is accomplished in two-dimensions today because of computer costs and the complicated grid generation required for complex geometries. And, perhaps most importantly, it is necessary to validate the new concepts in two-dimensions before extension to three-dimensions. For these and other reasons, the two-dimensional MESA schemes were developed and verified first.

The first two-dimensional case to be analyzed was the biperiodic open domain problem which tests the propagation, efficiency, and stability of the MESA schemes without geometry. Next, the rotated box cases test the stability and accuracy of the MESA schemes with straight wall boundaries. And finally, the circle case tests the stability and accuracy of the MESA schemes with curved wall boundaries. All cases were successful in demonstrating the accuracy and stability of the methods discussed in this dissertation. An overview of the geometry and fill point mapping for the test cases with wall boundaries at low grid resolution is shown in figure 7.1.

7.1.1 Bi-Periodic Open Domain up to 29^{th} order accuracy

The bi-periodic open domain problem is one in which the physical domain is a unit square ($[-1,1]\times[-1,1]\times[0,T]$). The solution of the linearized Euler equations in this case is assumed to be y-periodic (top and bottom of box repeat) and x-periodic (left and right sides of box repeat). Using separation of variables with periodic boundary conditions, on the linearized Euler equation system:

$$\begin{aligned} \frac{\partial u}{\partial t} + M_x\frac{\partial u}{\partial x} + M_y\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} &= 0, \\ \frac{\partial v}{\partial t} + M_x\frac{\partial v}{\partial x} + M_y\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} &= 0, \\ \frac{\partial p}{\partial t} + M_x\frac{\partial p}{\partial x} + M_y\frac{\partial p}{\partial y} + \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0, \end{aligned} \tag{7.1}$$

with boundary conditions :

$$\begin{aligned} p(1,y,t) &= p(-1,y,t) \\ u(1,y,t) &= u(-1,y,t) \end{aligned}$$

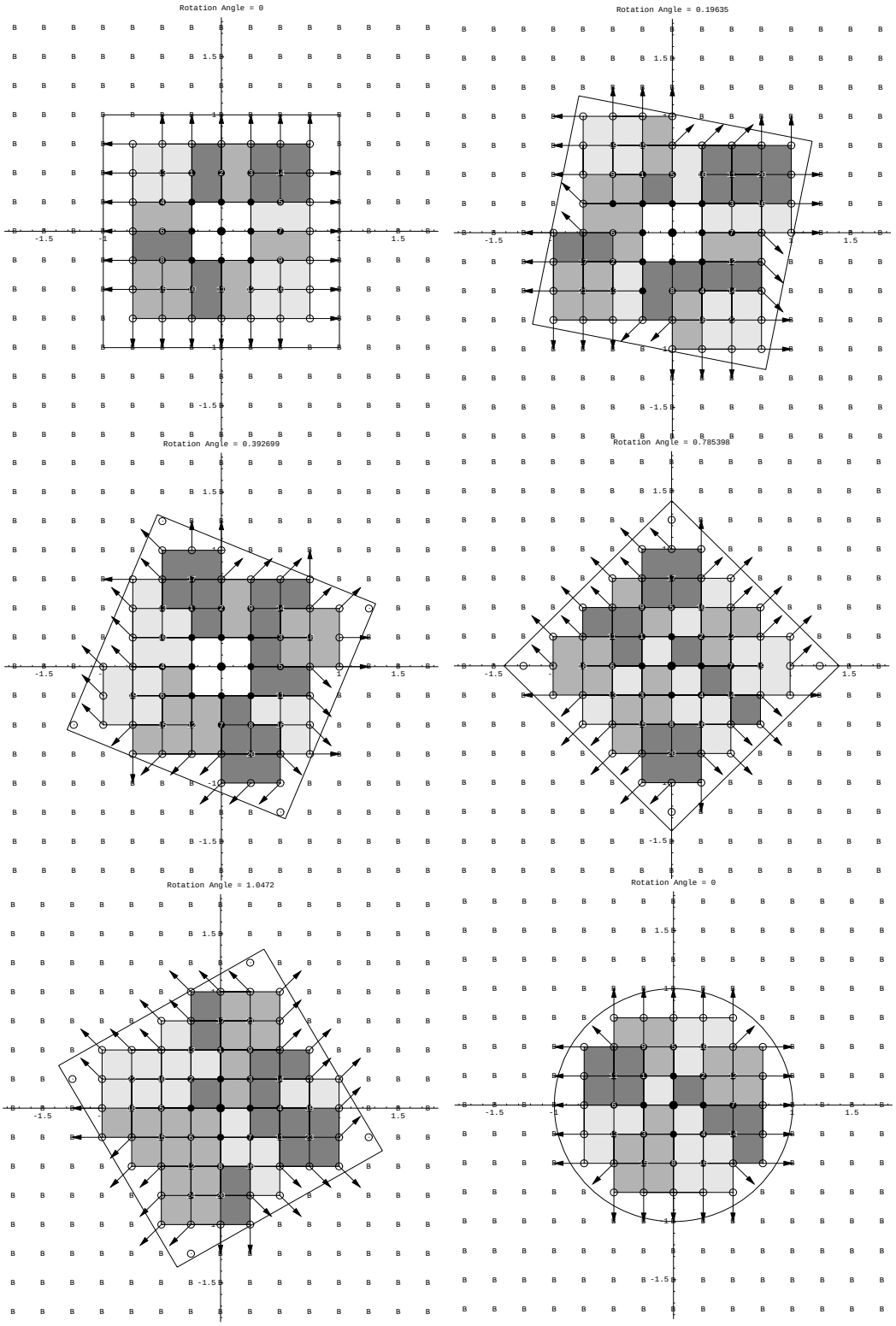

Figure 7.1: Rotated Boxes and Circle Low Resolution Cases

$$v(1,y,t) = v(-1,y,t)$$
$$p(x,1,t) = p(x,-1,t)$$
$$u(x,1,t) = u(x,-1,t)$$
$$v(x,1,t) = v(x,-1,t)$$

provides the following analytical solution:

$$p(x,y,t) = \cos(\pi t\sqrt{2})\,\sin(\pi\,(-\,(M_x t) + x))\,\sin(\pi\,(-\,(M_y t) + y)) \tag{7.2}$$

$$u(x,y,t) = -\frac{\cos(\pi\,(-\,(M_x t) + x))\sin(\pi t\sqrt{2})\sin(\pi\,(-\,(M_y t) + y))}{\sqrt{2}} \tag{7.3}$$

$$v(x,y,t) = -\frac{\cos(\pi\,(-\,(M_y t) + y))\sin(\pi t\sqrt{2})\sin(\pi\ \ (-\,(M_x t) + x))}{\sqrt{2}} \tag{7.4}$$

In figure 7.2, the slopes of the plots are determined by the accuracy of each algorithm. Next to each algorithm in the legend of figure 7.2 is the formal order of accuracy in space and time for the MESA scheme being graphed. The notation "c2d3" in the legend represents the c2o3 MESA scheme with a 2×2 stencil that has $(3+1)^2$ data values per grid point for each primitive variable. The actual data for these plots can be found in tables 7.1, 7.2, and 7.3. The first column in the tables represents the number of grid points per half-wavelength. The second column represents the number of time steps. And the remaining columns show the maximum error in the pressure for each MESA scheme. Notice that the 23^{rd} order c2o11 scheme is performing worse than the 19^{th} order c2o9 scheme due to the round off error from the high order spatial interpolation. This effect is known to occur in divided difference formulations [89] and while approaches for reducing this effect exist, it has not currently been pursued in this work. The need for higher precision floating point hardware is confirmed in figures 7.3 and 7.4 where the higher order schemes are rendered ineffective due to the round-off error. The higher order Hermitian methods are more computationally efficient than the lower order methods and require less memory to achieve a given maximum error. This efficiency is the result of the increased resolution of the higher order Hermitian MESA schemes which do not require as many grid points per wavelength.

Notice that the highest order scheme in table 7.3, the 29^{th} order c2o14, actually resolves well

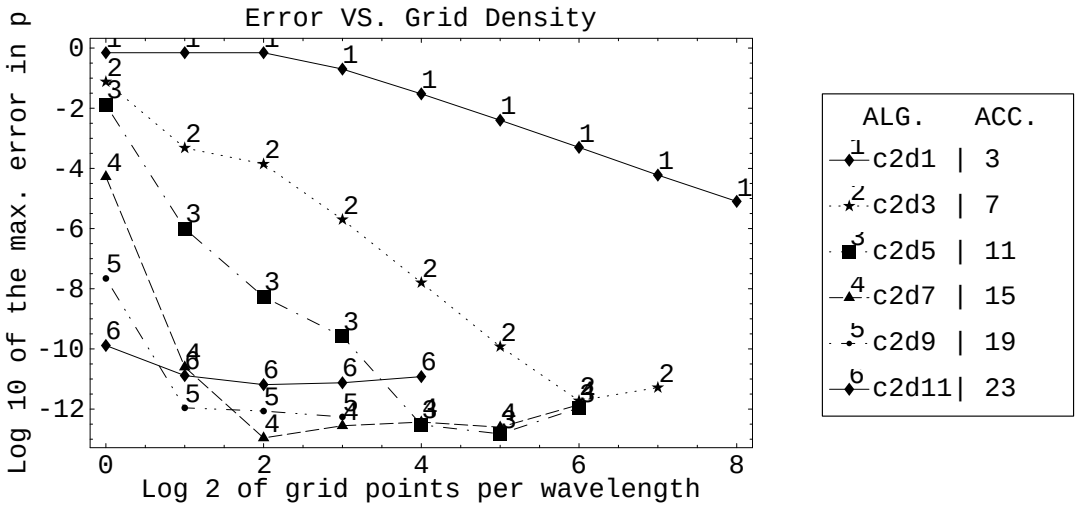

Figure 7.2: Maximum Absolute Error at time=10, with convection Mx=My=1

with two wavelengths per grid point! This is not a contradiction of the Nyquist criterion [85], however, since there are actually more than 4 data elements per wavelength; The c2o14 Hermitian MESA scheme has additional derivative information at each grid point as discussed in chapter 3.

7.1.2 Rotated Box at 2^{nd} order accuracy

As a test of the wall boundary formulation, a box was rotated at various angles relative to the Cartesian grid and grid resolution studies were performed on each rotated case. Their analytical solutions were derived using separation of variables in which no convection is present ($M_x = M_y = 0$).

For the unrotated case (box walls parallel to the Cartesian axes and positioned directly on the Cartesian grid points), boundary conditions on the left and right walls are:

$$p_x(x,y,t) = 0, x = -1, 1, -1 \geq y \leq 1, t \geq 0 \quad (7.5)$$

$$u(x,y,t) = 0, x = -1, 1, -1 \geq y \leq 1, t \geq 0 \quad (7.6)$$

$$v_x(x,y,t) = 0, x = -1, 1, -1 \geq y \leq 1, t \geq 0 \quad (7.7)$$

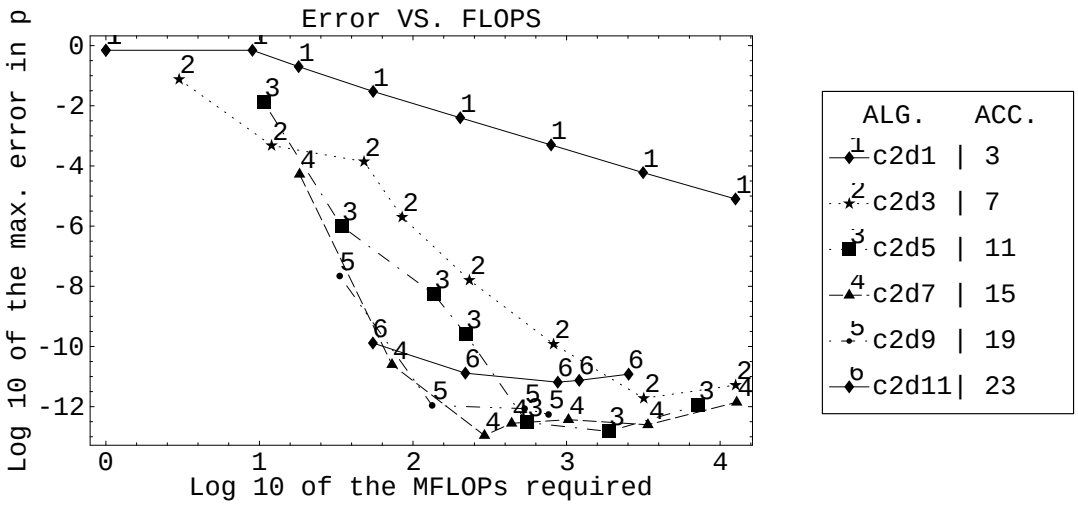

Figure 7.3: Maximum Absolute Error at time=10, with convection Mx=My=1

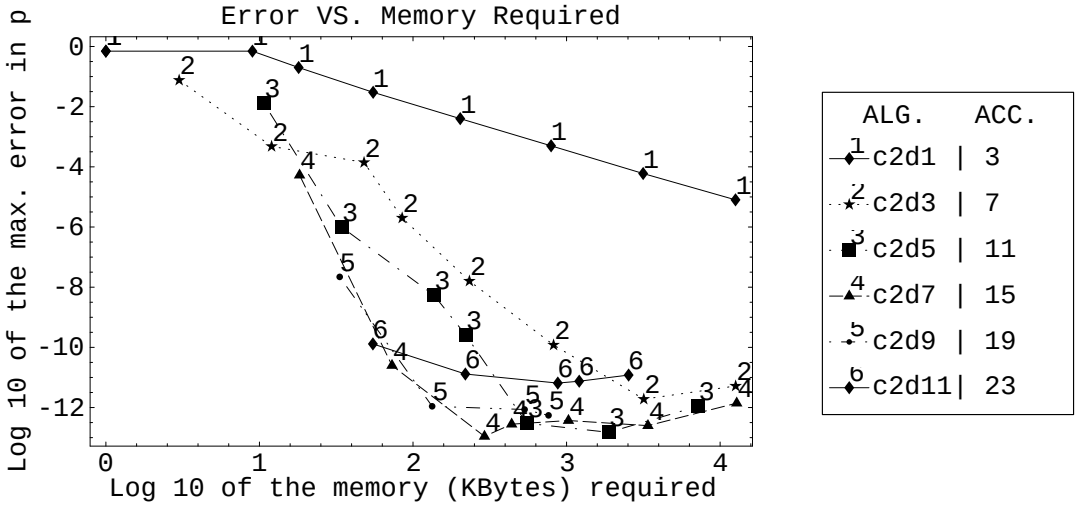

Figure 7.4: Maximum Absolute Error at time=10, with convection Mx=My=1

iun	n	c2o0	c2o1	c2o2	c2o3	c2o4
2/4	100	4.19313D-01	4.19313D-01	1.65510D-01	7.66908D-02	4.94527D-02
1	100	5.98758D-02	5.98758D-02	2.74867D-02	4.86309D-04	6.40222D-06
2	100	8.79878D-01	7.48778D-01	2.65588D-02	1.44219D-04	1.34963D-06
4	200	8.96398D-01	2.17517D-01	9.07266D-04	2.03451D-06	2.83584D-09
8	400	9.00479D-01	3.23784D-02	2.90656D-05	1.62115D-08	5.51315D-12
16	800	8.94956D-01	4.18709D-03	9.16133D-07	1.27500D-10	1.44773D-13
32	1600	8.34516D-01	5.29728D-04	2.87272D-08	1.90326D-12	1.18994D-12
2/4	1000	1.55423D-01	1.55423D-01	9.83802D-02	1.87211D-01	1.90080D-01
1	1000	8.40688D-02	8.40688D-02	8.40688D-02	3.49359D-02	1.64820D-03
2	1000	2.38576D-01	2.38576D-01	4.65859D-02	5.78433D-04	2.55880D-06
4	2000	2.43055D-01	2.13560D-01	2.08231D-03	5.52436D-06	9.29425D-09
8	4000	2.44184D-01	8.25183D-02	7.29347D-05	5.17616D-08	2.25119D-11
16	8000	2.44466D-01	1.31469D-02	2.39209D-06	3.68410D-10	6.28685D-11
32	16000	2.44537D-01	1.71994D-03	7.65553D-08	1.01852D-10	9.98046D-11
2/4	10000	6.55393D-01	6.55393D-01	4.01590D-01	3.12759D-01	3.09936D-01
1	10000	2.16718D-02	2.16718D-02	2.16718D-02	2.06896D-02	6.08435D-03
2	10000	7.64079D-01	7.64079D-01	7.44662D-01	2.02987D-02	1.14498D-04
4	20000	7.78425D-01	7.78425D-01	7.40483D-02	1.75117D-04	2.40618D-07
8	40000	7.82039D-01	7.55472D-01	2.50441D-03	1.40537D-06	2.96213D-09
16	80000	7.82944D-01	2.88551D-01	7.93319D-05	9.37781D-09	6.05023D-09
32	160000	7.83171D-01	4.49201D-02	2.48506D-06	1.06107D-08	

Table 7.1: Maximum Absolute Error of 2D Algorithms at time=10,100,1000, 1^{st} - 9^{th} order

And on the top and bottom walls the boundary conditions are:

$$p_y(x,y,t) = 0, -1 \geq x \leq 1, y = -1, 1, t \geq 0 \tag{7.8}$$

$$u_y(x,y,t) = 0, -1 \geq x \leq 1, y = -1, 1, t \geq 0 \tag{7.9}$$

$$v(x,y,t) = 0, -1 \geq x \leq 1, y = -1, 1, t \geq 0 \tag{7.10}$$

which are the direct result of the equations 5.1, 5.2 , and 5.3 when the box is not rotated. These boundary conditions are not multidimensional in the sense of coupling the u and v velocities.

The analytical solution to this problem is:

$$p(x,y,t) = -\Big(\cos(\sqrt{2}\,\pi\,t)\,\cos(\pi\,x)\,\cos(\pi\,y)\Big) \tag{7.11}$$

$$u(x,y,t) = -\frac{\cos(\pi\,y)\,\sin(\sqrt{2}\,\pi\,t)\,\sin(\pi\,x)}{\sqrt{2}} \tag{7.12}$$

$$v(x,y,t) = -\frac{\cos(\pi\,x)\,\sin(\sqrt{2}\,\pi\,t)\,\sin(\pi\,y)}{\sqrt{2}} \tag{7.13}$$

iun	n	c2o5	c2o6	c2o7	c2o8	c2o9
2/4	100	1.33692D-02	1.00134D-03	5.34796D-05	2.03112D-06	2.21581D-08
1	100	1.01308D-06	1.98289D-08	2.59540D-11	1.88191D-12	1.17385D-12
2	100	5.49155D-09	1.64495D-11	1.14436D-13	2.78083D-13	8.64669D-13
4	200	2.65099D-12	3.54161D-14	5.54556D-14	1.97731D-13	6.28071D-13
8	400	2.85993D-13	3.04090D-13	3.31901D-13	5.62439D-13	1.30090D-12
2/4	1000	1.27864D-01	1.16246D-02	6.08718D-04	2.46354D-05	4.04596D-07
1	1000	3.49923D-05	6.17069D-07	6.79935D-09	4.83705D-11	1.17557D-11
2	1000	1.30081D-08	4.54192D-11	5.56258D-12	5.83170D-12	7.37743D-12
4	2000	1.92279D-11	1.52294D-11	1.55838D-11	1.51723D-11	9.19279D-12
8	4000	2.41547D-11	2.41659D-11	2.39646D-11	2.47488D-11	2.67018D-11
2/4	10000	3.10413D-01	1.06928D-01	5.98842D-03	2.55310D-04	5.86179D-06
1	10000	2.64469D-04	4.82240D-06	2.86811D-08	3.17661D-10	2.45649D-10
2	10000	4.53566D-07	9.61101D-10	4.34892D-10	4.38126D-10	4.34286D-10
4	20000	1.45415D-09	1.39003D-09	1.39202D-09	1.39189D-09	4.34286D-10
8	40000	2.60486D-09	2.60449D-09	2.60512D-09	2.60578D-09	4.40577D-10

Table 7.2: Maximum Absolute Error of 2D Algorithms at time=10,100,1000, 11^{th} - 19^{th} order

The box is rotated about the origin of the Cartesian grid which is located at the center of the box. Its analytical solution is found by simply rotating the coordinates in equation 7.11 as well. The velocity boundary conditions are inherently multidimensional in the rotated case as discussed in chapter 5.

Table 7.4 shows the maximum error in the pressure variable from using the c3o0 MESA scheme applied to the box rotated to 5 different positions. The first column of the table is the number of grid points per half-wavelength.

Table 7.5 shows the change in total system energy as a fraction of the initial total system energy. The total system energy should not change within the rotated box; The ratio should always be 1. The total energy is the summation of $p(x,y)^2 + u(x,y)^2 + v(x,y)^2$ evaluated at all interior and fill grid points.

7.1.3 Circle at 2^{nd} order accuracy

A more complicated geometry in two-dimensions is the unit circle. Its orientation to the grid lines covers all angles resulting in many more unique boundary condition equations since the direction of the normal and tangent vectors on the circle's walls vary at all locations.

It too has an analytical solution which is developed as follows. First, the linearized Euler

iun	n	c2o10	c2o11	c2o12	c2o13	c2o14
2/8	100	2.29965D-03	2.47006D-04	2.05753D-05	1.79199D-06	1.67255D-06
2/4	100	4.48519D-10	1.31774D-10	4.10446D-10	1.56559D-09	5.34823D-09
1	100	5.99623D-12	1.37090D-11	6.43737D-11	7.08256D-11	3.81599D-10
2	100	2.19150D-12	6.59328D-12	4.29244D-11	7.90291D-11	2.36003D-10
2/8	1000	2.32224D-03	2.94998D-04	7.83313D-05	5.41336D-06	8.72029D-06
2/4	1000	7.56043D-09	6.57392D-10	5.30117D-09	7.33124D-09	2.22549D-08
1	1000	1.36526D-11	3.80372D-11	3.48912D-10	4.21926D-10	2.32803D-09
2	1000	9.60959D-12	2.50860D-11	2.05130D-10	1.68643D-10	9.40085D-10
2/8	10000	2.66386D-01	2.63985D-02	1.41920D-03	8.52095D-05	4.02444D-05
2/4	10000	1.09827D-07	8.79307D-09	8.15401D-08	4.70767D-08	9.65443D-08
1	10000	3.46658D-10	3.32700D-10	1.08017D-09	1.78962D-09	5.42603D-09
2	10000	4.49754D-10	5.24886D-10	1.90765D-09	1.12289D-09	3.16877D-09

Table 7.3: Maximum Absolute Error of 2D Algorithms at time=10,100,1000, 21^{st} - 29^{th} order

$\frac{1}{h}$	$\alpha = 0$	$\alpha = \frac{\pi}{16}$	$\alpha = \frac{\pi}{8}$	$\alpha = \frac{\pi}{4}$	$\alpha = \frac{\pi}{3}$
8	6.00340E-01	5.88561D-01	7.29709D-01	9.72177D-01	7.79207D-01
16	9.22801E-03	2.78385D-02	1.69012D-02	1.18011D-01	6.32180D-02
32	2.16148E-02	2.51592D-02	3.03067D-02	3.95514D-02	3.41046D-02
64	6.52761E-03	7.24661D-03	9.43153D-03	1.25927D-02	1.09241D-02
128	1.66121E-03	1.84735D-03	2.43511D-03	3.19933D-03	2.84123D-03
256	4.12058E-04	4.64566D-04	6.10075D-04	8.22738D-04	7.12248D-04

Table 7.4: Maximum Error in p at t=10, c3o0 scheme applied to box rotated by α

equations without convection are expressed as the wave equation,

$$\frac{\partial^2 p}{\partial t^2} = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} \tag{7.14}$$

Then, this is rewritten into polar coordinates [113],

$$\frac{\partial^2 p}{\partial t^2} = \frac{\partial^2 p}{\partial r} + \frac{1}{r}\frac{\partial p}{\partial r} + \frac{1}{r^2}\frac{\partial^2 p}{\partial \theta^2} \tag{7.15}$$

with $0 < r < 1, -\pi < \theta \leq \pi, t > 0$. And with the initial/boundary conditions,

$$p(1,\theta,t) = 0, -\pi < \theta \leq \pi, t > 0, \tag{7.16}$$

$$p_r(1,\theta,t) = 0, -\pi < \theta \leq \pi, t > 0, \tag{7.17}$$

$$p(r,\theta,0) = \frac{5J_0(\lambda r)}{\sqrt{\pi}J_0(\lambda)}, 0 < r < 1, -\pi < \theta \leq \pi \tag{7.18}$$

$\frac{1}{h}$	$\alpha = 0$	$\alpha = \frac{\pi}{16}$	$\alpha = \frac{\pi}{8}$	$\alpha = \frac{\pi}{4}$	$\alpha = \frac{\pi}{3}$
8	1.31769D-01	1.57350D-01	1.02348D-01	3.54850D-02	2.33908D-02
16	8.14386D-01	8.72280D-01	8.22221D-01	6.50197D-01	7.22285D-01
32	9.88311D-01	9.90026D-01	9.81165D-01	9.77070D-01	9.75765D-01
64	1.00151D+00	9.99237D-01	9.98688D-01	9.97840D-01	9.98150D-01
128	1.00134D+00	9.99869D-01	9.99895D-01	9.98897D-01	9.99860D-01
256	1.00072D+00	9.99931D-01	1.00000D+00	9.99518D-01	9.99997D-01

Table 7.5: Energy Ratio (should be 1) at t=10, c3o0 scheme applied to box rotated by α

$$p_t(r, \theta, 0) \quad = \quad 0, 0 < r < 1, -\pi < \theta \leq \pi \tag{7.19}$$

the analytical solution is:

$$p(r, \theta, t) = \frac{5 J_0(\lambda r) \cos(\lambda t)}{\sqrt{\pi} J_0(\lambda)} \tag{7.20}$$

with $\lambda = 3.83171$.

J_0 is the Bessel function of the first kind of order 0 and it is defined by:

$$J_0(x) = \sum_{n=0}^{\infty} \frac{-1^n}{(n!)^2} \left(\frac{x}{2}\right)^{2n} \tag{7.21}$$

The linearized Euler equations include the u and v variables which are not part of the wave equation 7.14. But the p variable will have solution 7.20 and can be used to test the accuracy of the algorithms if the additional initial conditions are assumed:

$$u(x, y, 0) = 0, -1 \leq x \leq 1, -1 \leq y \leq 1 \tag{7.22}$$

$$v(x, y, 0) = 0, -1 \leq x \leq 1, -1 \leq y \leq 1 \tag{7.23}$$

Table 7.6 shows the maximum error in the pressure variable from within the circle at 3 moments in time. The first column in the table represents the number of grid points per half-wavelength. Table 7.7 shows the change in total energy of the system within the circle. As in the rotated box case, the ratio should always be 1.

$\frac{1}{h}$	$t = 10$	$t = 100$	$t = 1000$
8	2.29533D+00	7.03234D+00	3.67081D+00
16	6.05969D-02	1.12127D+01	3.61856D+00
32	2.28263D-01	2.02830D+00	1.92185D+00
64	5.96043D-02	2.15448D-01	9.27343D+00
128	1.50125D-02	3.44447D-02	2.46248D+00
256	3.85009D-03	7.47578D-03	NA

Table 7.6: Maximum Error in p at t=10,100,1000, c3o0 scheme applied to circle

$\frac{1}{h}$	$t = 10$	$t = 100$	$t = 1000$
8	3.43444D-01	8.72832D-04	8.70954D-04
16	9.53462D-01	3.80642D-01	7.56545D-05
32	9.93788D-01	9.30932D-01	4.81021D-01
64	9.99809D-01	9.93336D-01	9.34796D-01
128	1.00021D+00	9.99340D-01	9.94067D-01
256	1.00004D+00	9.99928D-01	NA

Table 7.7: Energy Ratio (should be 1) at t=10,100,1000, c3o0 scheme applied to circle

7.1.4 Unrotated Box up to 11^{th} order accuracy

The wall boundary formulation for methods higher than 2^{nd} order have not provided stable solutions, so far, on Cartesian grids with unaligned wall boundaries. This is not unexpected since the highest accuracy wall boundary scheme reported in the literature on a Cartesian grid is 3^{rd} order and uses Lagrangian interpolation [45]. Most Cartesian grid based boundary conditions are 1^{st} order accurate and some new work using a finite volume formulation is 2^{nd} order accurate [27].

However, the case in which the geometry is aligned with the grid does provide stable solutions up to at least 11^{th} order accuracy. The results for the unrotated box problem are shown in figure 7.5. The legend in this figure shows the algorithm type and its accuracy. Notice that the high resolution of the Hermitian schemes with solid wall boundaries in this case is similar to the bi-periodic open domain results of section 7.1.1. These results suggest there may be a way to extend this approach to the unaligned wall boundary case which will maintain high resolution and efficiency.

In tables A.1, A.2, A.3, and A.4, the maximum error of the pressure in the unrotated box is shown at various times for various grid resolutions. The energy should not change with time

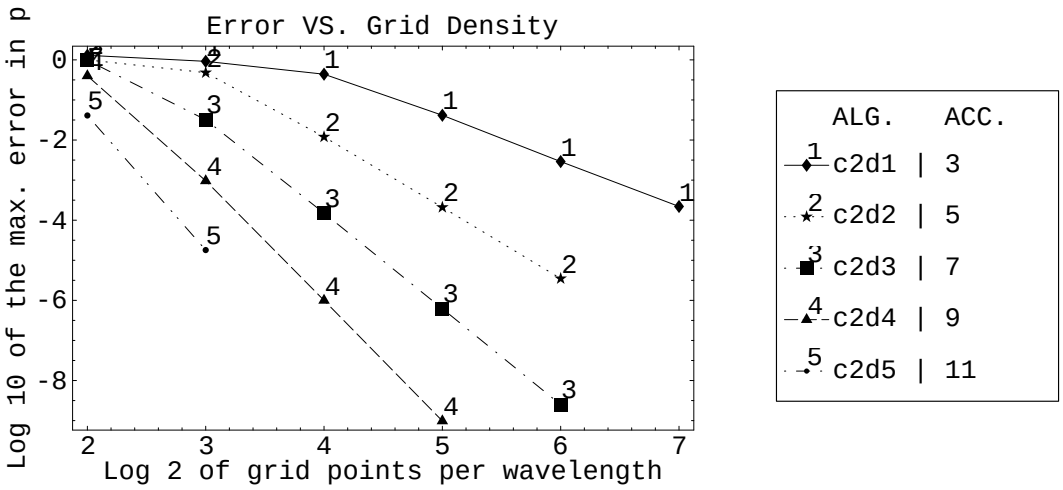

Figure 7.5: Unrotated box grid resolution studies, no convection, time=10

inside the box and therefore the energy ratio shown in the last column should always be one.

7.1.5 Complex Geometry Demonstration Mappings

These mappings are very preliminary and are included merely to demonstrate the potential flexibility of using the Cartesian grid mapping schemes developed in this dissertation.

In figure 7.6, an annular duct is described using 4 parametric curves, one per each half circle. The grid points inside the inner circle are labeled as boundary points and those grid points whose stencil intersects either circle are the fill points indicated by small open circles, and the dark circles are the interior grid points that can be time advanced using the standard MESA schemes with stencil width less than 4. The arrows indicate where the fill points are mapped to the wall using the automated methods discussed earlier in chapter 4. The shaded boxes indicate which data is used to interpolate the fill points in that box and the number indicates the order in which the fill points are solved as discussed in section 5.2. When a shaded box overlaps a neighboring box, the lower numbered box's fill points are treated as an interior grid point in the higher numbered box since those fill points will have been previously determined using the lower numbered box's spatial interpolant.

In figure 7.7, a grid for the three airfoil cascade is labeled with a grid spacing of 8 grid points per unit interval. In figure 7.8, the fill points are mapped to the wall boundaries. If the

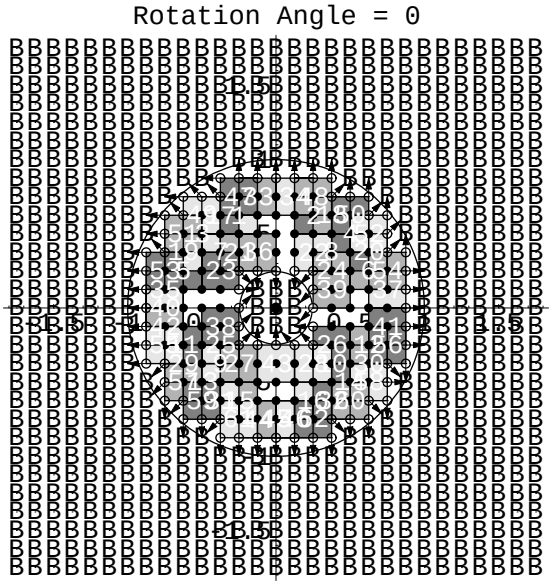

Figure 7.6: Annular Duct Fill Point Mapping in 2D, iun=8

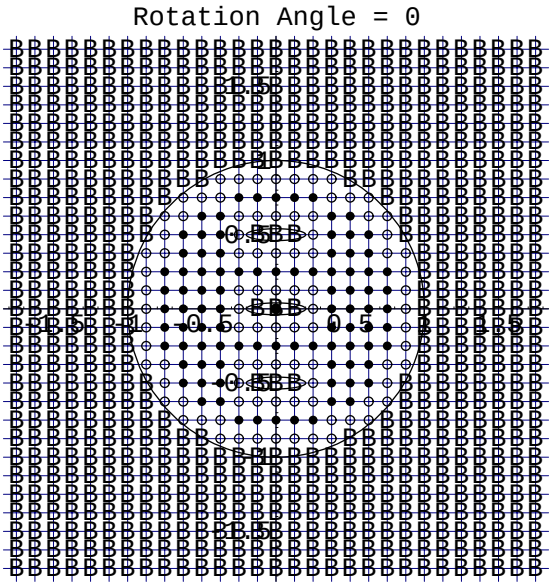

Figure 7.7: Airfoil Cascade Grid Point Labeling in 2D, iun=8

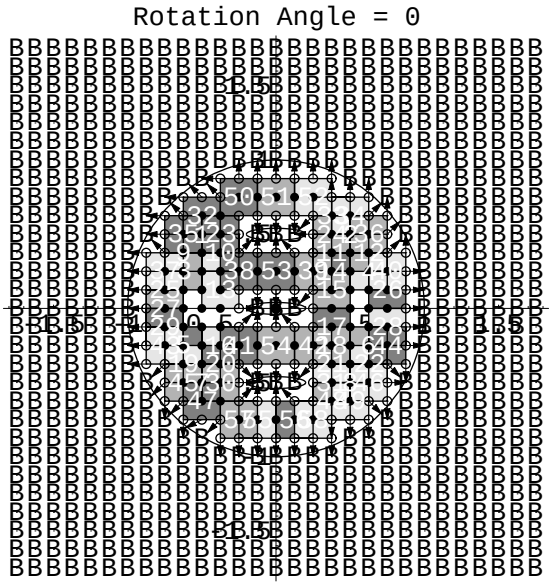

Figure 7.8: Airfoil Cascade Fill Point Mapping in 2D, iun=8

grid point resolution changes then the set of fill points will change–therefore a new mapping is required at each grid resolution.

7.2 Three-Dimensional Problems

Many interesting and challenging acoustical problems are inherently three dimensional, such as the generation of noise in the jet plume from a complex three-dimensional shear layer and the propagation of duct modes from the fan out the inlet. Some researchers are attempting 3D simulations through the use of 4^{th} order time accurate Compact Difference schemes on large parallel systems [81] and [86]. However, progress in these efforts requires improvement in the computer architectures and may require many years before the parallel systems can provide the floating point performance required for complete simulations. In this work, the MESA schemes were extended to 3D and applied to the tri-periodic open domain. The results, as expected, match the results from the 2D cases, and demonstrate the significant advantage of the MESA schemes in 3D - namely, the high order schemes with Hermitian data are very computationally efficient.

7.2.1 Tri-Periodic Domain up to 27^{th} order accuracy

The tri-periodic open domain problem is one in which the physical domain is a unit cube ($[-1,1]\times[-1,1]\times[-1,1]\times[0,T]$). The solution of the linearized Euler equations in this case is assumed to be x, y, and z-periodic. Using separation of variables with periodic boundary conditions, on the linearized Euler equation system:

$$\begin{aligned}
\frac{\partial p}{\partial t}+M_x\frac{\partial p}{\partial x}+M_y\frac{\partial p}{\partial y}+M_z\frac{\partial p}{\partial z}+\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}&=0,\\
\frac{\partial u}{\partial t}+M_x\frac{\partial u}{\partial x}+M_y\frac{\partial u}{\partial y}+M_z\frac{\partial u}{\partial z}+\frac{\partial p}{\partial x}&=0,\\
\frac{\partial v}{\partial t}+M_x\frac{\partial v}{\partial x}+M_y\frac{\partial v}{\partial y}+M_z\frac{\partial v}{\partial z}+\frac{\partial p}{\partial y}&=0,\\
\frac{\partial w}{\partial t}+M_x\frac{\partial w}{\partial x}+M_y\frac{\partial w}{\partial y}+M_z\frac{\partial w}{\partial z}+\frac{\partial p}{\partial z}&=0,
\end{aligned}\tag{7.24}$$

with boundary conditions :

$$\begin{aligned}
p(1,y,z,t)&=p(-1,y,z,t)\\
u(1,y,z,t)&=u(-1,y,z,t)\\
v(1,y,z,t)&=v(-1,y,z,t)\\
w(1,y,z,t)&=w(-1,y,z,t)\\
p(x,1,z,t)&=p(x,-1,z,t)\\
u(x,1,z,t)&=u(x,-1,z,t)\\
v(x,1,z,t)&=v(x,-1,z,t)\\
w(x,1,z,t)&=w(x,-1,z,t)\\
p(x,y,1,t)&=p(x,y,-1,t)\\
u(x,y,1,t)&=u(x,y,-1,t)\\
v(x,y,1,t)&=v(x,y,-1,t)\\
w(x,y,1,t)&=w(x,y,-1,t)
\end{aligned}\tag{7.25}$$

then the analytical solution is:

$$p(x,y,z,t) = \cos(\sqrt{3}\,\pi\,t)\,\sin(\pi\,(-(mx\,t)+x))\,\sin(\pi\,(-(my\,t)+y))\,\sin(\pi\,(-(mz\,t)+z)) \tag{7.26}$$

$$u(x,y,z,t) = -\frac{\cos(\pi\,(-(mx\,t)+x))\,\sin(\sqrt{3}\,\pi\,t)\,\sin(\pi\,(-(my\,t)+y))\,\sin(\pi\,(-(mz\,t)+z))}{\sqrt{3}} \tag{7.27}$$

$$v(x,y,z,t) = -\frac{\cos(\pi\,(-(my\,t)+y))\,\sin(\sqrt{3}\,\pi\,t)\,\sin(\pi\,(-(mx\,t)+x))\,\sin(\pi\,(-(mz\,t)+z))}{\sqrt{3}} \tag{7.28}$$

$$w(x,y,z,t) = -\frac{\cos(\pi\,(-(mz\,t)+z))\,\sin(\sqrt{3}\,\pi\,t)\,\sin(\pi\,(-(mx\,t)+x))\,\sin(\pi\,(-(my\,t)+y))}{\sqrt{3}} \tag{7.29}$$

Some of the grid resolution studies for the three-dimensional open domain problem are shown in figure 7.9. The actual numerical results of the maximum error in pressure is presented in tables 7.8, 7.9, and 7.10. The first column in the tables represents the number of grid points per half-wavelength. The second column represents the number of time steps. These results show the same high resolution performance as occurred in the two-dimensional case discussed in section 7.1.1. Recall from section 3.2.8 that the Recursive Tensor form of the MESA schemes is most efficient in three spatial dimensions and since the algorithms maintain their high fidelity characteristics in three dimensions (as shown in figure 7.9), the MESA schemes are ideally suited for simulating three-dimensional acoustics applications.

7.3 Parallel Scalability Studies

A test of the scalability of the MESA algorithms on a MIMD parallel computer was performed. The tests maintained the same work load on each processor by proportionately increasing the size of the problem domain. These tests were performed to determine the MESA scheme's suitably for large scale calculations.

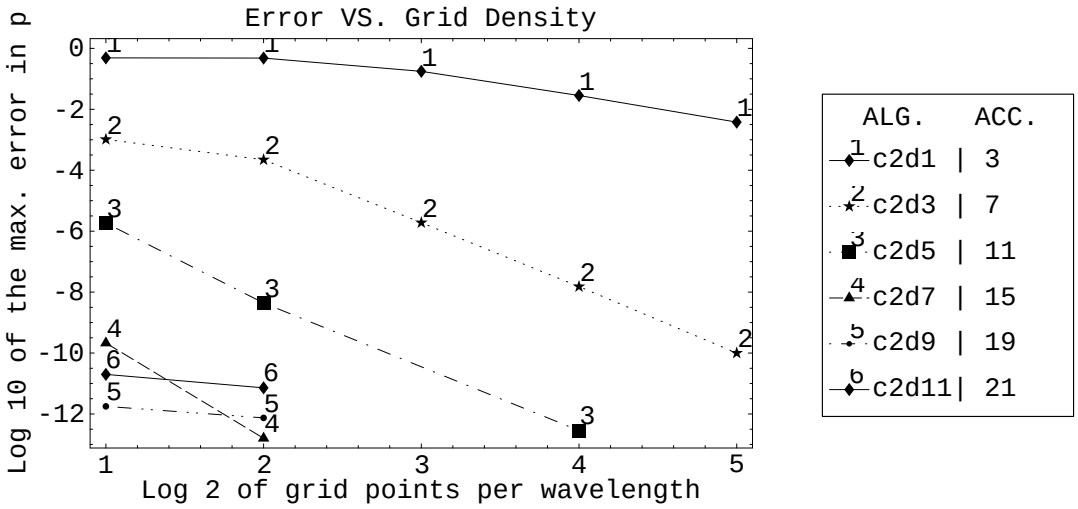

Figure 7.9: Maximum Absolute Error at time=10, with convection Mx=My=1 in 3D

iun	n	c2o0	c2o1	c2o2	c2o3	c2o4
2	100	5.14980D-01	4.80224D-01	2.19467D-02	2.21123D-04	1.19505D-06
4	200	5.29551D-01	1.76508D-01	7.92145D-04	1.90437D-06	2.59532D-09
8	400	5.33243D-01	2.83486D-02	2.58859D-05	1.51990D-08	4.95182D-12
16	800	5.33975D-01	3.79304D-03	8.23234D-07	NA	NA
2	1000	7.70364D-01	7.70364D-01	2.87374D-01	3.36613D-03	1.88335D-05
4	2000	7.92161D-01	7.79106D-01	1.21498D-02	2.86228D-05	3.98826D-08
8	4000	7.97684D-01	3.40102D-01	3.93693D-04	2.28286D-07	6.71208D-11
16	8000	7.99070D-01	5.47790D-02	1.24325D-05	NA	NA

Table 7.8: Maximum Absolute Error of 3D Algorithms at time=10, 100, 1^{st} - 9^{th} order

iun	n	c2o5	c2o6	c2o7	c2o8	c2o9
2/8	100	1.91686D-01	2.28186D-01	1.72699D-01	1.22931D-01	2.62557D-02
2/4	100	9.04854D-03	3.24260D-04	1.21824D-05	8.56003D-07	3.24471D-08
1	100	1.84961D-06	3.04300D-08	2.13233D-10	1.60284D-12	1.77729D-12
2	100	4.38982D-09	1.11129D-11	1.59497D-13	2.45484D-13	7.44085D-13
2/8	1000	5.16638D-01	3.62910D-01	3.64724D-01	3.81452D-01	2.70804D-01
2/4	1000	1.22614D-01	1.12744D-02	6.13017D-04	2.34683D-05	4.08609D-07
1	1000	9.65867D-06	1.55124D-07	2.92799D-10	7.56860D-12	8.00895D-12
2	1000	7.11438D-08	2.06639D-10	4.40782D-12	4.54260D-12	6.43510D-12

Table 7.9: Maximum Absolute Error of 3D Algorithms at time=10, 100, 11^{th} - 19^{th} order

iun	n	c2o10	c2o11	c2o12	c2o13
2/8	100	2.32539D-03	6.82271D-05	4.37158D-05	9.34356D-05
2/4	100	9.79661D-10	9.31879D-10	2.51697D-09	9.52672D-09
1	100	4.85311D-12	1.99288D-11	5.96948D-11	2.22166D-10
2	100	1.99729D-12	7.20823D-12	4.16325D-11	6.67628D-11
2/8	1000	3.64564D-02	2.28424D-03	2.03252D-04	6.65306D-04
2/4	1000	8.75725D-09	2.90691D-09	1.36760D-08	2.87604D-08
1	1000	1.73786D-11	5.33308D-11	2.32239D-10	5.28682D-10
2	1000	8.31935D-12	2.41510D-11	3.56261D-10	1.48749D-10

Table 7.10: Maximum Absolute Error of 3D Algorithms at time=10, 100, 21^{st} - 27^{th} order

7.3.1 Bi-Periodic Open Domain up to 21^{st} order accuracy

By simply repeating the bi-periodic unit interval open domain in both the x and y directions, it is possible to maintain the same work load per processor as the number of processors is increased.

In figure 7.10 a plot of the wall-clock execution time versus the number of parallel compute nodes is shown. The legend in the figure shows the MESA scheme used and its maximum error in pressure across all grid points. The table of numerical data used to create figure 7.10 may be found in section A.2.

This test was designed so that the wall-clock time should not change as the number of processors used is increased. Notice, however, that the lower order schemes are not as scalable as the higher order schemes as the number of processors increases. In particular, using more than 64 processors can result in rapidly decreasing scalability. This is likely due to the network's interconnect architecture on the SGI system being optimized for groups of 64 (the typical maximum size of ORIGIN 2000 systems). The 256 processor ORIGIN system from which these results were obtained is, in fact, the only one of its kind in the world at this time and therefore its interconnect is likely not optimized for 256 processors. Nonetheless, the scaling performance quickly improves as higher order MESA schemes are used.

Next, the grid resolution is doubled and the same set of MESA algorithms are tested. As shown in figure 7.11 the same trends observed in figure 7.10 can be seen, but they are less pronounced. And in figure 7.12 where there are 32 grid points per unit interval, even the c2o3 scheme shows good scalability. These graphs suggest good scalability is achieved by using higher order schemes and/or using increased grid resolution. Increasing the grid resolution or increasing the stencil depth have the effect of increasing the amount of interior work that a

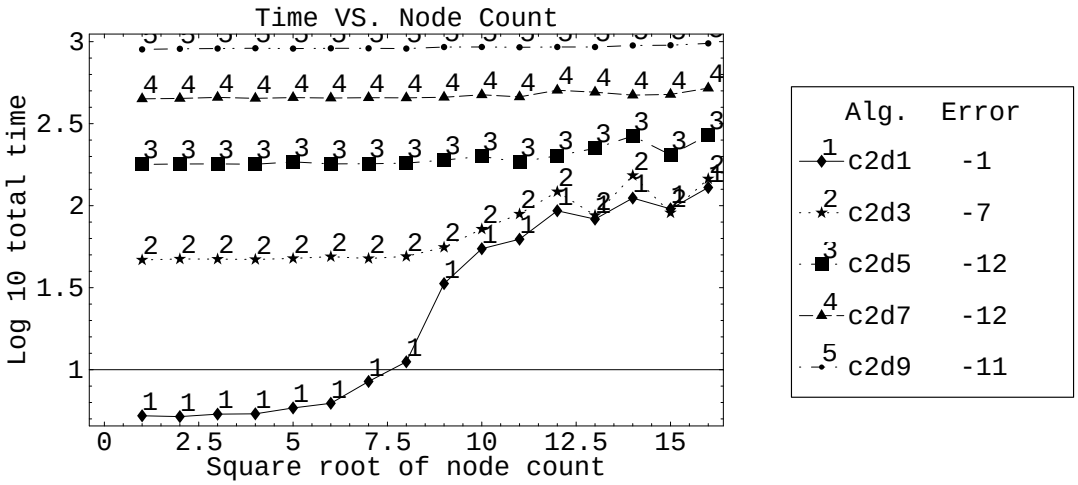

Figure 7.10: Scalability Performance to time=10, with convection Mx=My=1, iun=8

node may do before information must be communicated between nodes. However, increasing the grid resolution is less efficient on a single node than using a higher order MESA scheme as sections 7.1.1 and 7.2.1 have shown. Therefore, for best results, one should use the higher order MESA schemes to achieve not only high resolution, but single node efficiency and parallel node scalability.

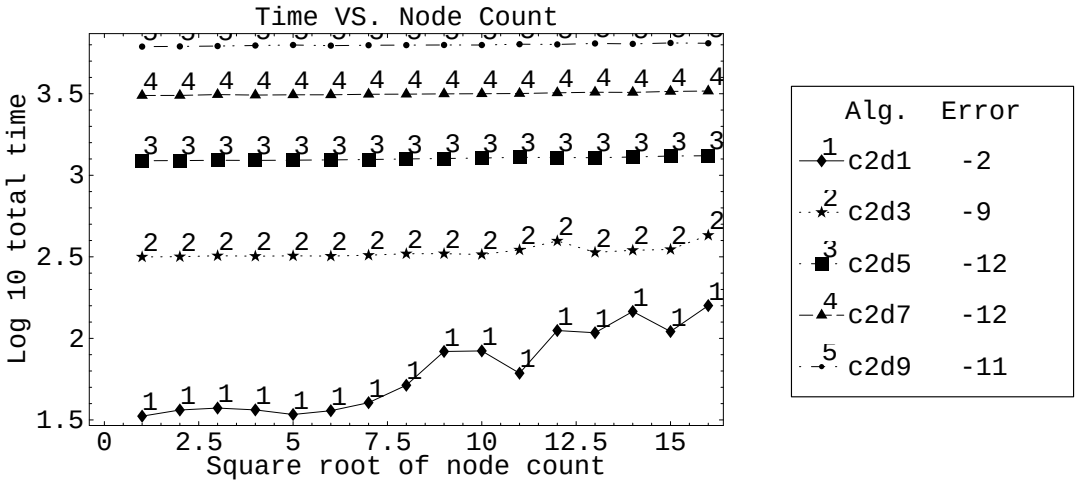

Figure 7.11: Scalability Performance to time=10, with convection Mx=My=1, iun=16

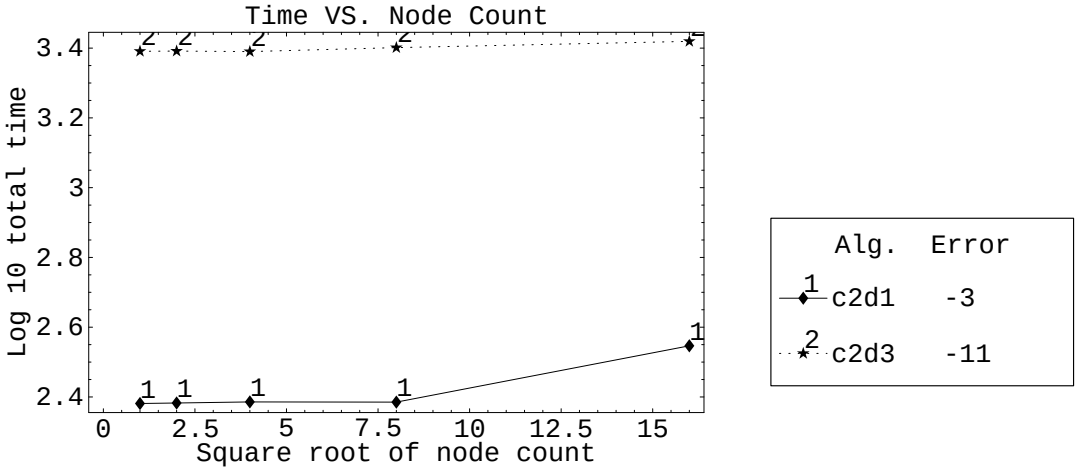

Figure 7.12: Scalability Performance to time=10, with convection Mx=My=1, iun=32

Chapter 8

Conclusions and Future Research

8.1 Summary

A new approach for solving computational aeroacoustics problems has been developed that eliminates the labor intensive tasks of grid generation, algorithm creation, code development and debugging commonly associated with these types of problems. This approach uses a higher-level language, Mathematica, to create a faster, lower level implementation in FORTRAN. Our new automated approach has the capacity to design sophisticated FORTRAN codes which are necessary for using the MESA schemes to solve acoustical problems with complex bodies on large-scale parallel computer systems. The MESA schemes [34], [35], [36] provide the basic numerical foundation, in this work, for solving the computationally demanding acoustics problems since they can be designed in an automated manner with arbitrarily high accuracy and resolution in space and time.

8.1.1 Scientific Developments in the Thesis

:

The following technical results were accomplished in this work:

- Automatic construction of the MESA methods in Mathematica as discussed in chapter 3. Both the spatial interpolation and time advancement processes of the MESA method were automatically created for both two and three spatial dimensions and were validated

by comparing them with the exact solution.

- Automatic generation of the FORTRAN code necessary to efficiently use the MESA methods as discussed in chapter 3. All three algebraically equivalent forms (Finite Difference, Spatial-Temporal, and Recursive Tensor) of the MESA methods were automatically written into a FORTRAN code. The FORTRAN code was validated by comparing its results with earlier hand-written codes and by checking them with the exact analytical solution available in the test problems.

- Developed a method for the reduction of all possible stencil configurations to a small set that could be efficiently mapped.

- Automatic mapping of all the near boundary grid points to the wall boundary in a way that insures local spatial interpolants can be generated near the walls as discussed in chapter 4. The locally defined spatial interpolants were then evaluated to solve for the values of the near boundary grid points. This mapping will work for any order Hermitian MESA scheme using a 2×2 staggered stencil.

- Automatic stencil selection to produce numerically stable 2^{nd} order wall boundary treatments as discussed in chapter 5. The near boundary grid points must be evaluated in a particular sequence that minimizes the use of wall boundary information and maximizes the overlap of the domain of dependence of the spatial interpolants.

- Automatic parallelization of the FORTRAN code using MPI as discussed in chapter 6. All domain decomposition, FORTRAN code generation, and message passing logic were accomplished without human assistance on a bi-periodic open domain problem.

8.1.2 Applications of the Scientific Developments

The following numerical experiments were completed in this thesis using the code generation tools:

- The two dimensional bi-periodic open domain wave propagation problem, which previously had been solved using only a 5^{th} order accurate MESA scheme [34], has now been solved using up to 29^{th} order accurate MESA methods.

- The MESA scheme was extended to three spatial dimensions and the tri-periodic open domain wave propagation problem was solved using from 2^{nd} to 29^{th} order MESA schemes.
- A unit square was embedded in a Cartesian mesh and the wave propagation on the interior of the square was simulated. It was possible to use from 2^{nd} to 11^{th} order accurate MESA schemes when the box was unrotated and aligned with the Cartesian grid.
- The same unit square was rotated about its center and only the 2^{nd} order method was numerically stable. Also, a unit circle was embedded in a Cartesian mesh and the wave propagation in its interior was simulated. As occurred with the rotated square test problem, only the 2^{nd} order MESA method was numerically stable.
- The bi-periodic open domain wave propagation problem was solved in parallel using up to 256 processors and using the 3^{rd} through 23^{rd} order accuracy MESA schemes. As expected, excellent parallel scalability was observed.

8.2 Conclusions

One significant advantage of the MESA schemes on Cartesian meshes is their stencils may be kept to a small 2×2 or $2 \times 2 \times 2$ foot-print. These small stencils, when used with high accuracy Hermitian MESA schemes may have a significant role to play in solving linear first-order hyerbolic systems of equations with irregular wall boundaries because of the following advantages. They:

- require less memory to achieve a particular error tolerance,
- are more efficient at achieving a particular error tolerance,
- obtain better resolution (up to 2 wavelengths per grid point with 29^{th} order method),
- maximize parallel efficiency (nearly perfect scalability),
- are most efficient when using the Recursive Tensor form,
- are easier to code in the Recursive Tensor form (the code reduces to a few lines),
- can exceed the accuracy of today's computer floating point hardware,

- maintains high accuracy for long time periods (ideal for time dependent problems),
- can be used to interpolate near boundary grid points without violating CFL condition,
- can achieve any level of accuracy in space and time,
- can be fully automated.

In short, the exceptional strengths of the Hermitian MESA schemes and the automatic code generation tools in this dissertation provide a productive framework from which to develop the more sophisticated FORTRAN codes required in the future to develop a turnkey approach to solving first-order linear hyperbolic partial differential equations in complex domains on large-scale parallel computers.

8.3 Future Work

The Hermitian boundary treatments (3^{rd} order or higher) are not numerically stable at this time for generalized, irregular wall boundaries. The issue is related to the difficulty of Birkhoff interpolation that must be done at each stencil [73]. It is in fact, a current topic of research in the mathematical research community dealing with approximation theory. But there is reason to believe this approach will be successful since the unrotated box case worked and because the MESA scheme itself uses the same spatial interpolation scheme–which is clearly successful.

Once a stable spatial interpolation process is found in two-dimensions, this will be extended to three-dimensions. Also, work is currently underway to extend the MESA schemes to variable coefficient and nonlinear systems. Their complexities will be minimized by using the same code generation approach previously discussed. Preliminary results suggest the high resolution Hermitian MESA schemes may be ideally suited to solve the viscous nonlinear Navier-Stokes equations.

With those tasks completed, it will then be possible to simulate the acoustics in a ducted airfoil cascade or to examine the effects of nozzle geometry on jet noise generation in detail.

Appendix A

Data from Numerical Experiments

This appendix provides the actual numerical results obtained from the many FORTRAN codes generated with the automation tools developed in this work. This data was collected from SGI systems with R10000 CPU's.

The first section shows the absolute error and energy ratios for an unrotated box at various times. This data is plotted at time, $t = 10$, in figure 7.5. The results are as expected and demonstrate the wall boundary treatment is at the same order of accuracy as the interior MESA scheme.

The second section shows the wall-clock execution time of the various MESA schemes with parallel extensions. This information is presented in graphical form in figures 7.10, 7.11, and 7.12. The results demonstrate the parallel scalability of the MESA algorithms up to 256 processors.

A.1 Unrotated Box Numerical Data

In this section, the Hermitian schemes on 2×2 stencils are applied to the unrotated box problem as discussed in section 7.1.4. All results are presented in the following tables for up to 11^{th} order accuracy. The first column shows the nondimensionalized time at which the data is gathered. The second column shows the number of grid points per half a wavelength, iun. The third

time	iun	error	energy
1	2	3.03409D-01	8.44204D-01
1	4	2.68635D-01	5.61817D-01
1	8	4.46000D-02	1.02293D+00
1	16	2.72473D-03	1.05281D+00
10	2	1.32696D+00	1.62567D+00
10	4	9.24036D-01	6.49811D-04
10	8	4.39834D-01	3.40264D-01
10	16	4.15104D-02	9.27166D-01
10	32	2.99916D-03	9.99615D-01
10	64	2.28536D-04	1.00247D+00
100	2	4.25006D-01	1.62567D+00
100	4	2.41269D-01	5.90033D-05
100	8	2.47026D-01	1.35196D-05
100	16	1.26843D-01	4.39830D-01
100	32	9.22886D-03	9.66763D-01
100	64	6.31680D-04	1.01000D+00
1000	2	1.20825D+00	1.62567D+00
1000	4	7.86538D-01	5.90033D-05
1000	8	7.83253D-01	2.01734D-10
1000	16	7.80100D-01	7.80100D-01
1000	32	2.21091D-01	5.35774D-01

Table A.1: Maximum Error in p at t=1, 10, 100, 1000, c2o1 scheme applied to unrotated box

column shows the maximum absolute error of the pressure at all grid points in the computational domain. The fourth column shows the energy ratio, $\frac{new\ energy}{old\ energy}$. The energy is computed as the sum of $p^2+u^2+v^2$ at each grid point. The energy ratio should be 1 for all time in this problem.

time	iun	error	energy
1	2	4.47733D-01	2.09415D+00
1	4	4.05765D-02	1.19465D+00
1	8	1.01121D-03	1.12995D+00
1	16	1.50919D-05	1.06189D+00
1	32	2.44167D-07	1.02997D+00
1	64	4.56731D-09	1.01475D+00
10	2	1.05676D+00	2.15754D-01
10	4	4.80494D-01	3.67935D-01
10	8	1.29059D-02	9.99843D-01
10	16	2.17390D-04	1.01197D+00
10	32	3.54538D-06	1.00601D+00
10	64	5.71791D-08	1.00296D+00
100	2	1.54806D-01	2.15685D-01
100	4	2.43898D-01	1.50925D-05
100	8	5.72942D-02	8.67869D-01
100	16	7.85736D-04	1.05779D+00
100	32	1.07485D-05	1.03025D+00
100	64	1.55296D-07	1.01492D+00
1000	2	9.38053D-01	2.15685D-01
1000	4	7.83264D-01	1.61926D-09
1000	8	6.39260D-01	7.42284D-02
1000	16	1.99893D-03	1.02184D+00
1000	32	2.99578D-04	1.01172D+00

Table A.2: Maximum Error in p at t=1, 10, 100, 1000, c2o2 scheme applied to unrotated box

time	iun	error	energy
1	2	1.20832D-01	1.57199D+00
1	4	2.19650D-03	1.30329D+00
1	8	1.08417D-05	1.13270D+00
1	16	4.21449D-08	1.06194D+00
1	32	1.85714D-10	1.02997D+00
10	2	9.81406D-01	1.53836D-02
10	4	3.22779D-02	1.00034D+00
10	8	1.51347D-04	1.02631D+00
10	16	6.25017D-07	1.01243D+00
10	32	2.51818D-09	1.00602D+00
10	64	1.17827D-11	1.00296D+00
100	2	2.40033D-01	1.84507D-04
100	4	1.50927D-01	7.01204D-01
100	8	6.41682D-04	1.13060D+00
100	16	2.01422D-06	1.06266D+00
100	32	7.06803D-09	1.03033D+00
100	64	2.59322D-10	1.01492D+00
1000	2	7.87774D-01	1.84508D-04
1000	4	7.95076D-01	2.21515D-03
1000	8	1.43518D-02	1.02138D+00
1000	16	5.55534D-05	1.02563D+00

Table A.3: Maximum Error in p at t=1, 10, 100, 1000, c2o3 scheme applied to unrotated box

time	iun	error	energy
1	2	1.57873D-02	1.86086D+00
1	4	6.45749D-05	1.30950D+00
1	8	7.33973D-08	1.13273D+00
1	16	6.57338D-11	1.06194D+00
10	2	4.01532D-01	6.38565D-01
10	4	9.69482D-04	1.06016D+00
10	8	1.04489D-06	1.02664D+00
10	16	9.82510D-10	1.01243D+00
100	2	2.47273D-01	3.83846D-04
100	4	4.84359D-03	1.28804D+00
100	8	4.05413D-06	1.13429D+00
100	16	2.32924D-09	1.06268D+00
1000	2	7.83302D-01	2.74898D-08
1000	4	8.84857D-02	9.32911D-01
1000	8	9.96391D-05	1.05497D+00
1000	16	8.03457D-08	1.02577D+00

Table A.4: Maximum Error in p at t=1, 10, 100, 1000, c2o4 scheme applied to unrotated box

time	iun	error	energy
1	2	1.24296D-03	1.92272D+00
1	4	1.21727D-06	1.30970D+00
10	2	4.19537D-02	1.11358D+00
10	4	1.85616D-05	1.06212D+00
100	2	2.12150D-01	9.04443D-01
100	4	8.31232D-05	1.31289D+00
1000	2	8.01028D-01	4.62971D-04
1000	4	1.76770D-03	1.12454D+00

Table A.5: Maximum Error in p at t=1, 10, 100, 1000, c2o5 scheme applied to unrotated box

A.2 Parallel Scalability Study Data

In this section, the numerical results of the two-dimensional MESA schemes applied to the bi-periodic open domain problem (discussed in section 7.3) are shown. The first column is the size of the stencil in one-dimension. The second column represents the number of x-derivative data elements per grid point (a MESA c2o5 scheme has size=2 and depth=5). The third column represents the number of grid points per a half-wavelength. The fourth column represents the number of parallel processing nodes in one-dimension of the mesh (this value squared is the total number of processors used). The fifth column represents the maximum error in the pressure across all grid points on all nodes. The sixth column represents the change in energy ratio and should be one. The last column shows the elapsed wall-time to run the simulation to a non-dimensional time, t=10. Ideally, the wall-time should not change as more processors are used since the problem size is proportionately increased as discussed in section 7.3.

The data for this section was obtained on the 256 processor ORIGIN 2000 SGI system at the Numerical Aerodynamic Facility at NASA Ames.

size	depth	iun	$\sqrt{nodes}$	error	energy	wall time (seconds)
2	1	8	1	3.23784D-02	9.28744D-0	5.237171999993734
2	1	8	2	3.23784D-02	9.28744D-01	5.175763199978974
2	1	8	3	3.23784D-02	9.28744D-01	5.357913600048050
2	1	8	4	3.23784D-02	9.28744D-01	5.382107199984603
2	1	8	5	3.23784D-02	9.28744D-01	5.844769600022119
2	1	8	6	3.23784D-02	9.28744D-01	6.238386400043964
2	1	8	7	3.23784D-02	9.28744D-01	8.475409599952400
2	1	8	8	3.23784D-02	9.28744D-01	11.17203040001914
2	1	8	9	3.23784D-02	9.28744D-01	33.48435039998731
2	1	8	10	3.23784D-02	9.28744D-01	54.62971999996807
2	1	8	11	3.23784D-02	9.28744D-01	62.45402240008116
2	1	8	12	3.23784D-02	9.28744D-01	93.10773200006224
2	1	8	13	3.23784D-02	9.28744D-01	82.79243520007003
2	1	8	14	3.23784D-02	9.28744D-01	111.2681471999967
2	1	8	15	3.23784D-02	9.28744D-01	95.57080720004160
2	1	8	16	3.23784D-02	9.28744D-01	129.0867583999643
2	3	8	1	1.62772D-08	1.00000D+00	46.70052399998531
2	3	8	2	1.62772D-08	1.00000D+00	47.32794719998492
2	3	8	3	1.62772D-08	1.00000D+00	47.22866720001912
2	3	8	4	1.62772D-08	1.00000D+00	47.01437280001119
2	3	8	5	1.62772D-08	1.00000D+00	47.67966800002614
2	3	8	6	1.62772D-08	1.00000D+00	48.90307600004598
2	3	8	7	1.62772D-08	1.00000D+00	47.63437759992667
2	3	8	8	1.62772D-08	1.00000D+00	49.07068880001316
2	3	8	9	1.62772D-08	1.00000D+00	55.78062559996033
2	3	8	10	1.62772D-08	1.00000D+00	72.05107759998646
2	3	8	11	1.62772D-08	1.00000D+00	89.01010640000459
2	3	8	12	1.62772D-08	1.00000D+00	121.7879936000099
2	3	8	13	1.62772D-08	1.00000D+00	87.26017040002625
2	3	8	14	1.62772D-08	1.00000D+00	152.9531184000662
2	3	8	15	1.62772D-08	1.00000D+00	90.48256879998371
2	3	8	16	1.62772D-08	1.00000D+00	145.8453919999301
2	5	8	1	2.93876D-13	9.99923D-01	178.4549544000183
2	5	8	2	2.93210D-13	9.99991D-01	179.5666744000046
2	5	8	3	2.97873D-13	9.99986D-01	179.7554208000074
2	5	8	4	3.00926D-13	1.00002D+00	178.9752455999842
2	5	8	5	2.98428D-13	1.00002D+00	184.9005968000274
2	5	8	6	2.98206D-13	1.00003D+00	179.8433984000003
2	5	8	7	3.02092D-13	1.00002D+00	180.1372600001050
2	5	8	8	3.02924D-13	1.00001D+00	181.7388320000027
2	5	8	9	3.00093D-13	1.00002D+00	189.7568999999785
2	5	8	10	2.99705D-13	1.00002D+00	199.6099071999779
2	5	8	11	2.98983D-13	1.00002D+00	184.5738287999993
2	5	8	12	3.00926D-13	1.00001D+00	201.6447256000247
2	5	8	13	3.00537D-13	1.00002D+00	223.6260383999906
2	5	8	14	3.05034D-13	1.00002D+00	265.9614831999643
2	5	8	15	3.01925D-13	1.00001D+00	203.8839583999943
2	5	8	16	3.01092D-13	1.00002D+00	269.0669479999924

Table A.6: Scalability of Even Stenciled 2D Algorithms, c2o1 - c2o5, iun=8

size	depth	iun	$\sqrt{nodes}$	error	energy	wall time (seconds)
2	7	8	1	3.56326D-13	1.32862D+11	448.1842528000125
2	7	8	2	3.46390D-13	1.14766D+11	450.6390879999963
2	7	8	3	3.82083D-13	1.13697D+11	457.2950864000013
2	7	8	4	3.87190D-13	1.13002D+11	451.2861640000483
2	7	8	5	4.18332D-13	1.10189D+11	456.0095087999944
2	7	8	6	3.94906D-13	1.15987D+11	453.2048775999574
2	7	8	7	4.23217D-13	1.14410D+11	455.4335280000232
2	7	8	8	4.15556D-13	1.13454D+11	454.3224384000059
2	7	8	9	4.46976D-13	1.18064D+11	458.1967968000099
2	7	8	10	4.24549D-13	1.13996D+11	474.2712151999585
2	7	8	11	4.15334D-13	1.15191D+11	460.6620823999401
2	7	8	12	4.13669D-13	1.17615D+11	505.5624687999953
2	7	8	13	4.13225D-13	1.15788D+11	491.8861192000331
2	7	8	14	4.07674D-13	1.14205D+11	472.0532631999813
2	7	8	15	4.13336D-13	1.14473D+11	476.7235392000293
2	7	8	16	4.32709D-13	1.13963D+11	520.7297504000599
2	9	8	1	1.19427D-12	2.31219D+28	897.2121983999969
2	9	8	2	1.36260D-12	2.41473D+28	904.5789711999823
2	9	8	3	1.69453D-12	2.50468D+28	908.1265935999691
2	9	8	4	1.69622D-12	2.29146D+28	911.9188191999565
2	9	8	5	1.73284D-12	2.35828D+28	907.4723824000102
2	9	8	6	1.93617D-12	2.33522D+28	911.1470751999877
2	9	8	7	1.79218D-12	2.29523D+28	910.9387560000177
2	9	8	8	1.94411D-12	2.34538D+28	907.4883976000128
2	9	8	9	2.00640D-12	2.37756D+28	927.2804231999908
2	9	8	10	1.84930D-12	2.33301D+28	928.2223328000400
2	9	8	11	2.06579D-12	2.33085D+28	924.2557263999479
2	9	8	12	1.90981D-12	2.30203D+28	928.1126103999559
2	9	8	13	2.01039D-12	2.36807D+28	927.6433960000286
2	9	8	14	2.10665D-12	2.33284D+28	948.0213632000377
2	9	8	15	1.98609D-12	2.35590D+28	951.3130775999743
2	9	8	16	2.18581D-12	2.35883D+28	976.0686616000021

Table A.7: Scalability of Even Stenciled 2D Algorithms, c2o7 - c2o9, iun=8

size	depth	iun	$\sqrt{nodes}$	error	energy	wall time (seconds)
4	0	8	1	2.38981D-01	5.43310D-01	6.056104799965397
4	0	8	2	2.38981D-01	5.43310D-01	6.095022400026210
4	0	8	4	2.38981D-01	5.43310D-01	6.619275200006086
4	0	8	8	2.38981D-01	5.43310D-01	14.98058799997671
4	0	8	16	2.38981D-01	5.43310D-01	127.1477040000027
4	1	8	1	2.37891D-06	9.99995D-01	51.41944319999311
4	1	8	2	2.37891D-06	9.99995D-01	51.64676319999853
4	1	8	4	2.37891D-06	9.99995D-01	52.10597440000856
4	1	8	8	2.37891D-06	9.99995D-01	54.93095439998433
4	1	8	16	2.37891D-06	9.99995D-01	139.8488951999461
4	2	8	1	1.47837D-12	1.00000D+00	175.6608000000124
4	2	8	2	1.47915D-12	1.00000D+00	176.3666416000342
4	2	8	4	1.48059D-12	1.00000D+00	176.2924056000193
4	2	8	8	1.48004D-12	1.00000D+00	176.5644056000165
4	2	8	16	1.48348D-12	1.00000D+00	217.2921216000104
4	3	8	1	2.86993D-13	1.00000D+00	424.8665967999841
4	3	8	2	2.89213D-13	1.00000D+00	424.2507880000048
4	3	8	4	2.90434D-13	1.00000D+00	428.3839087999659
4	3	8	8	2.91878D-13	1.00000D+00	425.8961191999842
4	3	8	16	2.93321D-13	1.00000D+00	460.5442184000276
4	4	8	1	3.00093D-13	1.00000D+00	867.3569967999938
4	4	8	2	2.96763D-13	1.00000D+00	863.3986783999717
4	4	8	4	3.02203D-13	1.00000D+00	867.5709088000003
4	4	8	8	3.03424D-13	1.00000D+00	870.2196703999653
4	4	8	16	3.06255D-13	1.00000D+00	887.8940992000280

Table A.8: Scalability of Even Stenciled 2D Algorithms, c4o0 - c4o4, iun=8

size	depth	iun	$\sqrt{nodes}$	error	energy	wall time (seconds)
3	0	8	1	3.42487D-01	7.68253D-01	1.087568799965084
3	0	8	2	3.42487D-01	7.68253D-01	1.159291199990548
3	0	8	4	3.42487D-01	7.68253D-01	1.276368800026830
3	0	8	8	3.42487D-01	7.68253D-01	18.02085120003903
3	0	8	16	3.42487D-01	7.68253D-01	107.3357264000224
7	0	8	1	3.09909D-04	9.99738D-01	16.79473039996810
7	0	8	2	3.09909D-04	9.99738D-01	15.83481199998641
7	0	8	4	3.09909D-04	9.99738D-01	15.91395760001615
7	0	8	8	3.09909D-04	9.99738D-01	17.38843120000092
7	0	8	16	3.09909D-04	9.99738D-01	106.7015184001066
11	0	8	1	3.42940D-07	1.00000D+00	62.61711920000380
11	0	8	2	3.42940D-07	1.00000D+00	62.68050879996736
11	0	8	4	3.42940D-07	1.00000D+00	62.92338079999899
11	0	8	8	3.42940D-07	1.00000D+00	63.69668079999974
11	0	8	16	3.42940D-07	1.00000D+00	109.4004799999529
15	0	8	1	4.16869D-10	1.00000D+00	160.0692976000137
15	0	8	2	4.16870D-10	1.00000D+00	160.0310920000193
15	0	8	4	4.16870D-10	1.00000D+00	160.2175512000103
15	0	8	8	4.16871D-10	1.00000D+00	160.7702752000187
15	0	8	16	4.16872D-10	1.00000D+00	190.8228232000256

Table A.9: Scalability of Odd Stenciled 2D Algorithms, c3o0 - c15o0, iun=8

size	depth	iun	$\sqrt{nodes}$	error	energy	wall time (seconds)
2	1	16	1	4.18709D-03	9.90684D-01	33.34540320001543
2	1	16	2	4.18709D-03	9.90684D-01	36.38025839999318
2	1	16	3	4.18709D-03	9.90684D-01	37.38562880002428
2	1	16	4	4.18709D-03	9.90684D-01	36.43411120004021
2	1	16	5	4.18709D-03	9.90684D-01	34.13949199998751
2	1	16	6	4.18709D-03	9.90684D-01	36.01643920002971
2	1	16	7	4.18709D-03	9.90684D-01	40.33804960001726
2	1	16	8	4.18709D-03	9.90684D-01	51.60536559997126
2	1	16	9	4.18709D-03	9.90684D-01	83.10855679993983
2	1	16	10	4.18709D-03	9.90684D-01	83.77599599992391
2	1	16	11	4.18709D-03	9.90684D-01	61.15366560000257
2	1	16	12	4.18709D-03	9.90684D-01	111.7266711999982
2	1	16	13	4.18709D-03	9.90684D-01	108.2574559999994
2	1	16	14	4.18709D-03	9.90684D-01	146.2095783999976
2	1	16	15	4.18709D-03	9.90684D-01	110.1791632000022
2	1	16	16	4.18709D-03	9.90684D-01	158.8355488000088
2	3	16	1	1.27779D-10	1.00000D+00	315.8375128000043
2	3	16	2	1.27780D-10	1.00000D+00	316.8705479999771
2	3	16	3	1.27781D-10	1.00000D+00	321.0610504000215
2	3	16	4	1.27783D-10	1.00000D+00	319.1082735999953
2	3	16	5	1.27781D-10	1.00000D+00	320.6734375999076
2	3	16	6	1.27782D-10	1.00000D+00	319.5226640000474
2	3	16	7	1.27781D-10	1.00000D+00	323.5534895999590
2	3	16	8	1.27782D-10	1.00000D+00	330.3301328000380
2	3	16	9	1.27782D-10	1.00000D+00	330.6389639999252
2	3	16	10	1.27783D-10	1.00000D+00	326.8845864000032
2	3	16	11	1.27783D-10	1.00000D+00	348.6675920000125
2	3	16	12	1.27783D-10	1.00000D+00	395.7727319999976
2	3	16	13	1.27783D-10	1.00000D+00	336.9418080000032
2	3	16	14	1.27784D-10	1.00000D+00	346.8016951999962
2	3	16	15	1.27783D-10	1.00000D+00	350.9623480000009
2	3	16	16	1.27784D-10	1.00000D+00	428.8525264000054
2	5	16	1	1.45328D-13	2.11148D+01	1228.121523199952
2	5	16	2	1.49769D-13	2.00565D+01	1230.585302400053
2	5	16	3	1.51879D-13	2.03366D+01	1235.663951199967
2	5	16	4	1.52212D-13	2.01497D+01	1233.671822399949
2	5	16	5	1.55653D-13	2.02570D+01	1237.722901600064
2	5	16	6	1.55098D-13	1.99053D+01	1242.435900800047
2	5	16	7	1.57763D-13	2.02560D+01	1249.013546400005
2	5	16	8	1.55320D-13	2.01438D+01	1259.287554400042
2	5	16	9	1.54876D-13	2.01459D+01	1266.828930400079
2	5	16	10	1.58318D-13	2.02172D+01	1275.836791199981
2	5	16	11	1.56541D-13	2.00356D+01	1288.187559200000
2	5	16	12	1.56763D-13	2.00856D+01	1284.793097600006
2	5	16	13	1.56430D-13	2.00036D+01	1281.903600799997
2	5	16	14	1.57319D-13	2.00260D+01	1294.268690400000
2	5	16	15	1.60205D-13	1.99225D+01	1312.503817600002
2	5	16	16	1.61204D-13	2.00325D+01	1317.534620799997

Table A.10: Scalability of Even Stenciled 2D Algorithms, c2o1 - c2o5, iun=16

size	depth	iun	$\sqrt{nodes}$	error	energy	wall time (seconds)
2	7	16	1	2.74475D-13	2.81362D+19	3085.163891999982
2	7	16	2	2.96874D-13	2.82098D+19	3087.646312800003
2	7	16	3	3.33733D-13	2.91273D+19	3124.434628799907
2	7	16	4	3.43892D-13	2.83502D+19	3103.314338400029
2	7	16	5	3.42393D-13	2.91375D+19	3115.198646400007
2	7	16	6	3.84248D-13	2.83731D+19	3115.035458400031
2	7	16	7	3.86136D-13	2.85317D+19	3138.926576000056
2	7	16	8	3.62266D-13	2.81157D+19	3143.350410399958
2	7	16	9	3.55160D-13	2.81470D+19	3156.271051199990
2	7	16	10	3.37785D-13	2.82885D+19	3162.656237600022
2	7	16	11	3.78475D-13	2.86809D+19	3171.837485600001
2	7	16	12	3.72147D-13	2.83225D+19	3203.977630399997
2	7	16	13	3.52107D-13	2.85238D+19	3229.511270399998
2	7	16	14	3.57603D-13	2.84159D+19	3222.951519199996
2	7	16	15	3.60489D-13	2.82847D+19	3267.425811199995
2	7	16	16	3.70592D-13	2.82556D+19	3284.770247199999
2	9	16	1	2.27607D-12	1.45926D+39	6150.135244000005
2	9	16	2	2.81408D-12	1.44775D+39	6169.245900800102
2	9	16	3	2.33982D-12	1.43026D+39	6199.318338399986
2	9	16	4	2.44680D-12	1.47093D+39	6247.247560799937
2	9	16	5	2.74331D-12	1.42069D+39	6292.342919999966
2	9	16	6	2.57855D-12	1.43481D+39	6234.565688800067
2	9	16	7	2.63278D-12	1.44191D+39	6272.793180799927
2	9	16	8	2.54285D-12	1.46191D+39	6282.411390399910
2	9	16	9	2.66986D-12	1.41666D+39	6293.421671200078
2	9	16	10	2.64755D-12	1.42460D+39	6290.047822399996
2	9	16	11	2.55329D-12	1.42340D+39	6363.927698400003
2	9	16	12	2.98259D-12	1.42804D+39	6343.667015200001
2	9	16	13	2.69806D-12	1.44301D+39	6431.174068000000
2	9	16	14	2.76029D-12	1.44698D+39	6391.318301600004
2	9	16	15	6.88456D-08	1.10447D+40	6477.088828000000
2	9	16	16	2.69351D-12	1.45205D+39	6448.539209600000

Table A.11: Scalability of Even Stenciled 2D Algorithms, c2o7 - c2o9, iun=16

size	depth	iun	$\sqrt{nodes}$	error	energy	wall time (seconds)
3	0	16	1	1.01725D-01	9.66443D-01	6.877860000007786
3	0	16	2	1.01725D-01	9.66443D-01	6.992657599970698
3	0	16	4	1.01725D-01	9.66443D-01	7.157536799961235
3	0	16	8	1.01725D-01	9.66443D-01	36.73547040001722
3	0	16	16	1.01725D-01	9.66443D-01	134.7845471999608
7	0	16	1	5.42150D-06	9.99998D-01	111.2408288000152
7	0	16	2	5.42150D-06	9.99998D-01	111.9104679999873
7	0	16	4	5.42150D-06	9.99998D-01	110.9043407999561
7	0	16	8	5.42150D-06	9.99998D-01	114.3528296000441
7	0	16	16	5.42150D-06	9.99998D-01	185.8904215999646
11	0	16	1	3.84612D-10	1.00000D+00	461.6958719999529
11	0	16	2	3.84615D-10	1.00000D+00	449.7394943999825
11	0	16	4	3.84615D-10	1.00000D+00	449.3641663999879
11	0	16	8	3.84616D-10	1.00000D+00	453.5119095999980
11	0	16	16	NA	NA	NA
15	0	16	1	3.42060D-13	1.00000D+00	1170.108012800047
15	0	16	2	3.45501D-13	1.00000D+00	1172.090083199961
15	0	16	4	3.46834D-13	1.00000D+00	1173.452475200000
15	0	16	8	3.46279D-13	1.00000D+00	1174.595946400019
15	0	16	16	NA	NA	NA

Table A.12: Scalability of Odd Stenciled 2D Algorithms, c3o0 - c15o0, iun=16

size	depth	iun	$\sqrt{nodes}$	error	energy	wall time (seconds)
4	0	16	1	3.47955D-02	9.24707D-01	39.92755999998190
4	0	16	2	3.47955D-02	9.24707D-01	40.30595120001817
4	0	16	4	3.47955D-02	9.24707D-01	40.66720399999758
4	0	16	8	3.47955D-02	9.24707D-01	57.08650400000624
4	0	16	16	3.47955D-02	9.24707D-01	151.7088871999877
4	1	16	1	1.92644D-08	1.00000D+00	350.7237432000111
4	1	16	2	1.92644D-08	1.00000D+00	352.1846247999929
4	1	16	4	1.92644D-08	1.00000D+00	353.2757240000064
4	1	16	8	1.92644D-08	1.00000D+00	354.4428503999952
4	1	16	16	1.92644D-08	1.00000D+00	418.2106800000183
4	2	16	1	1.36002D-13	1.00000D+00	1198.147397599998
4	2	16	2	1.37668D-13	1.00000D+00	1200.047472000006
4	2	16	4	1.38556D-13	1.00000D+00	1197.294346400013
4	2	16	8	1.39222D-13	1.00000D+00	1201.052784800006
4	2	16	16	1.42886D-13	1.00000D+00	1239.151003199979
4	3	16	1	1.38001D-13	1.00000D+00	2975.240424000018
4	3	16	2	1.40998D-13	1.00000D+00	2904.418075199996
4	3	16	4	1.40665D-13	1.00000D+00	2909.782975999988
4	3	16	8	1.42997D-13	1.00000D+00	2912.032973600028
4	3	16	16	1.45550D-13	1.00000D+00	2943.797028800007
4	4	16	1	1.37779D-13	9.99999D-01	6065.749917600013
4	4	16	2	1.41220D-13	9.99996D-01	6003.123391999980
4	4	16	4	1.44662D-13	9.99998D-01	5968.139892800013
4	4	16	8	1.47771D-13	1.00000D+00	5962.605800800025
4	4	16	16	1.50380D-13	1.00000D+00	6075.030088000000

Table A.13: Scalability of Even Stenciled 2D Algorithms, c4o0 - c4o4, iun=16

size	depth	iun	$\sqrt{nodes}$	error	energy	wall time (seconds)
2	1	32	1	5.29728D-04	9.98827D-01	240.5725144000025
2	1	32	2	5.29728D-04	9.98827D-01	241.3859056000365
2	1	32	4	5.29728D-04	9.98827D-01	243.1004591999808
2	1	32	8	5.29728D-04	9.98827D-01	242.7892943999614
2	1	32	16	5.29728D-04	9.98827D-01	352.0860415999778
2	3	32	1	1.90603D-12	1.00000D+00	2459.852386399987
2	3	32	2	1.90870D-12	1.00000D+00	2463.405324799998
2	3	32	4	1.90947D-12	1.00000D+00	2455.173050399986
2	3	32	8	1.91125D-12	1.00000D+00	2520.319598400034
2	3	32	16	1.91436D-12	1.00000D+00	2626.445582399960

Table A.14: Scalability of Even Stenciled 2D Algorithms, c2o1 - c2o3, iun=32

Appendix B

Mathematica Source Code For Acoustics Problems With Wall Boundaries

This appendix contains the entire Mathematica code developed in this dissertation for solving two-dimensional acoustical wave propagation problems that include irregularly shaped wall boundaties. To minimize the size of this document, it does not contain the code for three-dimensional wave propagation since it is a simple extension of the two-dimensional case as discussed in chapter 3 and it does not contain the code for the parallelized version of the two-dimensional bi-periodic open domain problem discussed in chapter 6. In this appendix, each section contains a separate Mathematica file for a total of five files. Each Mathematica file contains multiple modules that together perform a particular objective.

B.1 Code Generation System Overview

A flowchart overview of the inputs, Mathematica modules, FORTRAN subroutine, and the output is shown in figure B.1. Once the inputs are provided to the Mathematica modules, they produce all the FORTRAN files necessary for solving the linearized Euler equations on a Cartesian mesh which contains irregular boundaries. The following subsections will provide

an overview of the input parameters, Mathematica modules, FORTRAN subroutines, and the output.

B.1.1 Input Parameters

The input parameters are:

csize This parameter defines the size of the MESA stencil in one-dimension.

degree This parameter defines the depth of data at a grid point in the MESA.

maxmeminbytes This parameter corresponds to the maximum amount of memory available on a computer system and is used to limit the grid resolution to prevent core dumps.

maxiuntop This parameter is the maximum number of grid points per unit interval that will be used in the grid resolution studies.

miniun This parameter is the minimum number of grid points per unit interval that will be used in the grid resolution studies.

maxtime This parameter is the maximum time that the acoustical simulation will be run to during the grid resolution studies.

readgrid This parameter is a flag used for reading a grid definition file. The grid definition file is generated in FORTRAN for problems with many grid points. The file provides the labelling for all grid points on the Cartesian mesh. FORTRAN is faster than Mathematica on problems with simple geometry such as squares and circles. A more robust labelling algorithm may be written in C that could handle general geometries.

theta This parameter defines the angle of rotation about the center of the box relative to the fix Cartesian mesh.

listofcurves This variable is defined within the ma2d code but is also an input to this code. It defines the list of parametric curves that represent the geometry of the walls. In the future, this variable will be defined by a CAD definition file.

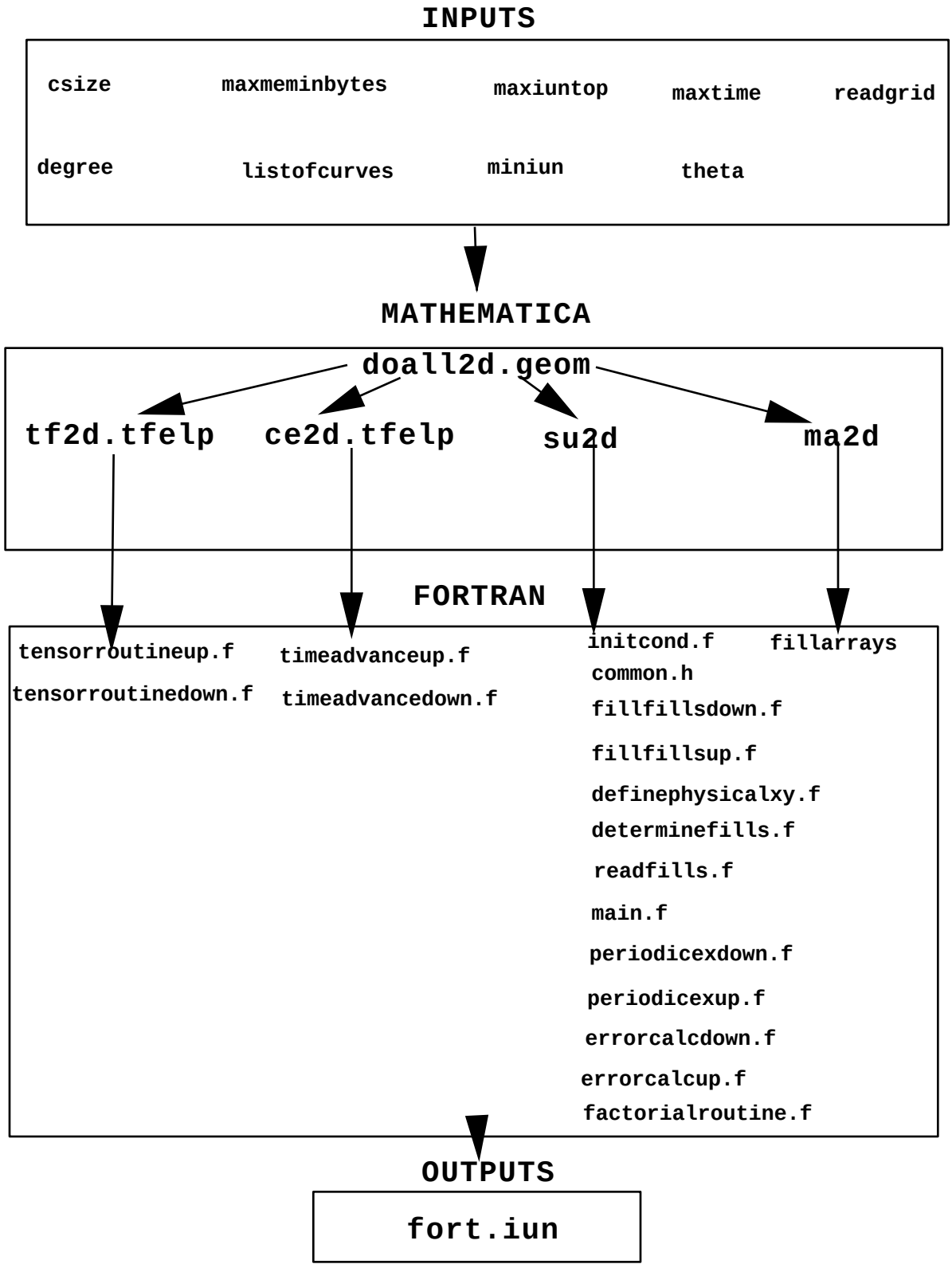

Figure B.1: Overview of Code Generation System

B.1.2 Mathematica Modules

doall2d.geom This code is the master file and should be installed first. It will read the input parameters, compile the FORTRAN code, and save the output results.

tf2d.tfelp This code writes the FORTRAN code that performs the spatial interpolation using the Tensor Product form discussed in section 3.1.4. It creates the FORTRAN files: tensorroutineup.f and tensorroutinedown.f.

ce2d.tfelp This code writes the FORTRAN code that performs the temporal evolution using the Recursive Tensor form discussed in section 3.2.3. It creates the FORTRAN files: timeadvanceup.f and timeadvancedown.f.

su2d.geom This code writes the FORTRAN code that performs the housekeeping functions such as reading and writing files, assigning initial conditions, and calculating the error at each time step. It creates the FORTRAN files: initcond.f, common.h, fillfillsdown.f, fillfillsup.f, definephysicalxy.f, determinefills.f, readfills.f, main.f, periodicexdown.f, periodicexup.f, errorcalcdown.f, errorcalcup.f, and factorialroutine.f.

ma2d This code performs all the analysis necessary to treat the wall boundary conditions in an automated manner. It creates a single file called "fillarrays". This array is a list of real numbers that provide all the information necessary to solve the near boundary grid points at each time step and is described in section 5.5. This file is read once by the FORTRAN code and stored in memory as a one-dimensional array.

B.1.3 FORTRAN Subroutines

It is important to minimize memory fetch strides to minimize cache misses. One approach is to use small arrays that can fit entirely with cache. Another approach is to minimize memory transfers. This is accomplished in the follow way. First, the primitive variables at time n, (p^n, u^n, v^n), are stored in a two-dimensional array A. They are then evolved to time n+1 and stored in array B. Rather than copy array B back to array A and repeat the time advance procedure, it is more efficient to time advance the variables in array B, $(p^{n+1}, u^{n+1}, v^{n+1})$, and directly store them in array A.

tensorroutineup.f This file performs the spatial interpolation using the Tensor Product form on the data in array B.

tensorroutinedown.f This file performs the spatial interpolation using the Tensor Product form on the data in array A.

timeadvanceup.f This file performs the Recursive-Tensor form of the timeadvance by advancing the data in array A to array B.

timeadvancedown.f This file performs the Recursive-Tensor form of the timeadvance by advancing the data in array B to array A.

initcond.f This file calculates the initial conditions for all the grid points in the computational domain.

common.h This file contains all the FORTRAN common block data used by all the modules. The common block data structure is used to minimize memory fetches and cache misses.

fillfillsdown.f This file, using the information from file, "fillarrays", calculates the values of all near boundary grid points in array A.

fillfillsup.f This file, using the information from file, "fillarrays", calculates the values of all near boundary grid points in array B.

definephysicalxy.f This file assigns a physical coordinate to each grid point.

determinefills.f This file determines which grid points are near boundary grid points.

readfills.f This file reads the file, "fillarrays", and stores its contents in memory for faster future retrieval.

main.f This file is the main FORTRAN code. It calls all the FORTRAN subroutines and reads the input and produces the output.

periodicexdown.f This file communicates the periodic boundary conditions on grid A. In the parallel implementation, it communicates between processors.

periodicexup.f This file communicates the periodic boundary conditions on grid B. In the parallel implementation, it communicates between processors.

errorcalcdown.f This file calculates the absolute error at each time step of the solution as compared to the exact analytical solution on grid A.

errorcalcup.f This file calculates the absolute error at each time step of the solution as compared to the exact analytical solution on grid B.

factorialroutine.f This file calculates all the factorials once at the beginning and stores them in memory. This avoids the need to recalculate the factorials at each grid point when time advancing using the Recursive-Tensor form.

fillarrays This file contains the list of real numbers that completely specifies how to evaluate all the near boundary grid points at each time step.

B.1.4 Outputs

The compiled FORTRAN code will produce output showing the error information at each time step.

fort.iun This file will show the maximum error in the pressure at each time step as the MESA schemes solve the linearized Euler equations. It will also show the ratio of energy change at each time step (which should be 1, no change). This file is the final product of the code generation system.

B.2 Master File – doall2d.geom

In this section is file, "doall2d.geom". It is the master file and it starts the other four files and queries for all input parameters. It also automatically creates directories, compiles the FORTRAN codes, and submits each executable to a batch queue on IRIS systems. *Note that this is a research code, it is constantly being modified and may indeed contain errors.*

```
(* ------------------------------------------- *)
(* AUTHOR: Rodger W. Dyson Jr.                 *)
(* DATE  : March, 1999                         *)
(* VERSION: Mathematica 3.0.2                  *)
(* COPYRIGHT 1999                              *)
(* EMPLOYER: NASA Glenn Research Center        *)
(*           21000 Brookpark Rd.               *)
(*           Cleveland, OH                     *)
(* ------------------------------------------------------------------------------*)
(* This routine will read all the MMA files and produce a complete FORTRAN code*)
(* ------------------------------------------------------------------------------*)
Clear["a*","b*","c*","d*","e*","f*","g*","h*","i*","j*","k*"];
Clear["l*","m*","n*","o*","p*","q*","r*","s*","t*","u*","v*"];
Clear["w*","x*","y*","z*"];
csize=Input["Enter the size of the stencil in one-dimension"];
degree=Input["Enter the maximum degree for 2D problem"];
maxmeminbytes=Input["Enter the maximum amount of memory available in Mbytes."];
maxmeminbytes=maxmeminbytes*1000000;
maxiuntop=Input["Enter the maximum iun you care to submit for batch runs."];
miniun=Input["Enter the minimum iun you care to submit for batch runs."];
maxtime=Input["Enter the maximum time you care to run the batch runs."];
readgrid=Input["Do you wish to read a grid definition file (1=Yes,0=No)"];
theta=Input["Enter the rotation angle in Radians for box "];

alpha=theta;



Print[" Generating Code for csize= ",csize," degree= ",degree];

maxiun = N[IntegerPart[Sqrt[maxmeminbytes]/Sqrt[768 + 1536 degree + 768
degree^2 ]]];
If[maxiun> maxiuntop, maxiun=maxiuntop];
(*
maxmemi=Input["Enter the maximum memory grid size in one dimension"];
*)

maxmemi=IntegerPart[(Sqrt[2]*maxiun)+3];
maxmemj=maxmemi;
Print["Using maximum grid dimensions of ",N[maxmemi]," by ",N[maxmemj]];
```

```
Print["And using maximum grid points per unit interval of ",N[maxiun]];

Print["Loading tf2d.tfelp"];
<< tf2d.tfelp;
Print["Generating tensorroutine.f (Computes cp,cu,cv coefficients)"];
starttf;

Print["Loading ce2d.tfelp"];
<< ce2d.tfelp;
Print["Generating factorialroutine.f (Solves factorial constants once)"];
Print["Generating timeadvanceroutine.f "];
startce;

Print["Loading su2d.geom"];
<< su2d.geom;

Print["Generating the common.h file used by all subroutines"];
Print["Generating initial condition routine initcond.f"];
Print["Generating Error Calculation subroutine errorcalcup and down.f"];
Print["Generating periodic boundary exchange subroutine periodicex.f"];
Print["Generating main FORTRAN program (Calls all other subroutines)"];
startsu;


Print["Loading ma2d"];
<< ma2d;

notcorrect=False;
Check[startma,notcorrect=True];

(* ------------------------------------------------------------ *)
(* The length of the fillarray is now known so make the common.h *)
(* ------------------------------------------------------------ *)
makecommonfile;

(* ----------------------------------- *)
(* Make New Directory and compile file  *)
(* ----------------------------------- *)
alg="c"<>ToString[csize]<>"o"<>ToString[degree];
rot="rot"<>ToString[N[theta]];
diun="d"<>ToString[iun];
dirname=alg<>"/"<>rot<>"/"<>diun;

command1="mkdir "<>ToString[alg];
Run[command1];
command4="cd "<>ToString[alg];
SetDirectory[alg];

command1="mkdir "<>ToString[rot];
Run[command1];
command4="cd "<>ToString[rot];
```

```
SetDirectory[rot];

command1="mkdir "<>ToString[diun];
Run[command1];
SetDirectory["../.."];


command2="mv *.f "<>dirname;
Run[command2];

command3="mv *.h "<>dirname;
Run[command3];

command35="mv fillarrays "<>dirname;
Run[command35];

command4="cd "<>ToString[dirname];
SetDirectory[dirname];

filename="MesaProp_c"<>ToString[csize]<>"o"<>ToString[degree]<>".geom";
Print["Compiling Code into executable ",filename];

(*
command5="f77 -O3 -r8 -col120 -Nn100000 -NC500 -u -o "<>filename<>" main.f
factorialroutine.f timeadvanceup.tfelp.f timeadvancedown.tfelp.f
tensorroutineup.f tensorroutinedown.f initcond.f errorcalcup.f  errorcalcdown.f
periodicexup.f periodicexdown.f";
*)
(*
command5="f77 -O3 -r8 -col120 -Nn100000 -NC500 -u -o "<>filename<>" main.f
factorialroutine.f timeadvanceup.tfelp.f timeadvancedown.tfelp.f
tensorroutineup.f tensorroutinedown.f initcond.f errorcalcup.f  errorcalcdown.f
readfills.f fillfillsup.f fillfillsdown.f determinefills.f rotate.f";
*)
command5="f77 -O3 -r8 -col120 -Nn100000 -NC500 -u -o "<>filename<>" main.f
factorialroutine.f timeadvanceup.tfelp.f timeadvancedown.tfelp.f
tensorroutineup.f tensorroutinedown.f initcond.f errorcalcup.f
errorcalcdown.f readfills.f fillfillsup.f fillfillsdown.f determinefills.f
definephysicalxy.f";
Run[command5];

(*
(* ------------------------------------------------------ *)
(* Create the Input Files and run the codes as batch jobs *)
(* ------------------------------------------------------ *)
Do[
(* Create Input File *)
ifilename="inputfile"<>ToString[2^iunct];
stmp=OpenWrite["inputfiletmp"];
Write[stmp,2^iunct];
If[EvenQ[csize],
```

```
lamtmp=0.2,lamtmp=0.4];
Write[stmp,lamtmp];
(*
cttmp=N[1000 * 2^iunct / lamtmp];
*)
cttmp=N[maxtime * 2^iunct / lamtmp];
Write[stmp,cttmp];
(*
ecttmp=N[cttmp/100];
*)
ecttmp=N[10 * 2^iunct / lamtmp];
Write[stmp,ecttmp];
wavenumberx=1;
wavenumbery=1;
Write[stmp,wavenumberx];
Write[stmp,wavenumbery];
Close[stmp];
Run["rm "<> ifilename];
Run["sed -e 's/\"//g' inputfiletmp >> "<>ifilename];
Run["rm inputfiletmp"];

(* Create the shell script *)

stmp=OpenWrite["shellscripttmp"];
Write[stmp,"#! /bin/csh"];
Write[stmp,"#BSUB -N -u xxdyson@gauss.lerc.nasa.gov"];
Write[stmp,"#BSUB -R irix"];
Write[stmp,"cd /u/xxdyson/golden2D/",dirname];
(*
Write[stmp,"cd ",ToString[Directory[]]];
*)
command2="./"<>filename<>" < ./inputfile"<>ToString[2^iunct];
Write[stmp,command2];
Close[stmp];
Run["rm shellscript"];
Run["sed -e 's/\"//g' shellscripttmp >> shellscript"];
Run["rm shellscripttmp"];


command3="bsub -q long -R irix -J"<>filename<>ToString[2^iunct]<>" -u
xxdyson@gauss.lerc.nasa.gov < shellscript";
Print[command3];
Run[command3];
,{iunct,N[IntegerPart[Log[miniun]/Log[2]]],N[IntegerPart[Log[maxiun]/Log[2]]]}];
SetDirectory[".."];
Clear["a*","b*","e*","f*","g*","h*","i*","j*","k*"];
Clear["l*","n*","o*","p*","q*","r*","s*","t*","u*","v*"];
Clear["w*","x*","y*","z*",cp,cu,cv,cfc,datalist];
(*
,{degree,0,1}]
,{csize,2,2}];
```

```
*)

(* Solve[{mem == 3 * (degree+1)^2 * (2*iuntmp)^2 * 64},iuntmp]  bytes *)

(*
command1="sed -e 's/\"//g' inputfiletmp >> inputfile";
command2=command1<>ToString[2^iunct];
*)
*)

If[Not[notcorrect],
Print["All systems seem non-singular"];
initializetimestepping;
dothetimestepping;
,
Print["*************************************************************************"];
Print["Something was wrong in startma!!!!"];
Print["*************************************************************************"];
Print["Message List=",$MessageList]
];
```

B.3 Tensor Form of Spatial Interpolation File – tf2d.tfelp

This Mathematica code will produce the FORTRAN code that calculates the spatial interpolant at each stencil using the Tensor Product form described in section 3.1.4.

```
<< LinearAlgebra`MatrixManipulation`;

(* ------------------------------------------------------------------------ *)
(* This procedure will compute the spatial coefficients using John's Tensor *)
(* form of evaluation.                                                      *)
(* ------------------------------------------------------------------------*)
starttf:=(


(* ---------------------------------------- *)
(* Initialize and Get Problem Specifications *)
(* ---------------------------------------- *)
initproc;

(* ------------------------------------------------------ *)
(* Determine the polynomial interpolant for the x-direction *)
(* ------------------------------------------------------ *)
getintformx;

(* ------------------------------------------------------ *)
(* Compute the S data which is located at x=0 on each strata *)
(* ------------------------------------------------------ *)
computes;

(* ------------------------------------------------------ *)
(* Determine the polynomial interpolant for the y-direction *)
(* ------------------------------------------------------ *)
getintformy;

(* ---------------------------------------- *)
(* Compute the spatial interpolant coefficients *)
(* ---------------------------------------- *)
computea;

(* ------------------------------------------------------------------------ *)
(* Generate the FORTRAN subroutine solving the cp,cu, and cv coefficients *)
(* ------------------------------------------------------------------------ *)
makecompleteroutine;


(* ------------------------------------------------- *)
(* Test the interpolant by evaluating at all data points *)
(* ------------------------------------------------- *)
(*
testall;
*)
```

```
);


initproc:=(
Clear[a,s,xc,yc,xf,yf];
Run["rm tensorroutine.f"];
Run["rm tensorroutinetmp.f"];


If[Not[ValueQ[csize]],
csize=Input["Enter the size of the stencil in one-dimension"];
];
If[Not[ValueQ[degree]],
degree=Input["Enter the maximum degree for 2D problem"];
];
(* ------------------------------------------------------------ *)
(* Make a list of all the data elements at a single grid point *)
(* ------------------------------------------------------------ *)
datalist={};
Do[
Do[
AppendTo[datalist,fc[dx,dy,i,j]]
,{dx,0,degree}]
,{dy,0,degree}];

(* ------------------------------------------------------------------------ *)
(* Calculate the spatial interpolant form to be used for all directions *)
(* ------------------------------------------------------------------------ *)

(* ------------------------------------------------------- *)
(* Determine the number of data elements in the x direction *)
(* ------------------------------------------------------- *)
Cases[datalist,fc[_,0,i,j]];

interpolantorder=csize* Count[datalist,fc[_,0,i,j]];
);
(* ------------------------------------------------------- *)
(* Determine one-dimensional interpolant form in x direction *)
(* ------------------------------------------------------- *)
getintformx:=(
xf[x_]:=Sum[xc[i] x^i,{i,0,interpolantorder-1}];
variablelist=CoefficientList[xf[x],x];


evenvariablelist={};
oddvariablelist={};
Do[AppendTo[evenvariablelist,xc[i]],{i,0,interpolantorder-1,2}];
Do[AppendTo[oddvariablelist, xc[i]],{i,1,interpolantorder-1,2}];


(* ------------------------------------------------- *)
```

```
(* Find the values of the coefficients c for c2 stencil *)
(* ------------------------------------------------- *)

If[EvenQ[csize],

(* ------------------------------------------------------------------------ *)
(* Solve the 1D interpolant by splitting into two independent sets of equation*)
(* ------------------------------------------------------------------------*)
evenequationlist={};
Do[
Do[
(*
AppendTo[equationlist,(D[xf[x],{x,dx}] /. {x->(ptct h) - h/2}) ==
fc[dx,dy,ptct,j]];
AppendTo[equationlist,(D[xf[x],{x,dx}] /. {x->-(ptct h) + h/2}) ==
fc[dx,dy,1-ptct,j]];
*)
AppendTo[evenequationlist,(D[xf[x],{x,dx}] /. {x->(ptct h) -
h/2})+(D[xf[x],{x,dx}] /. {x->-(ptct h) + h/2}) == fc[dx,dy,ptct,j]
+fc[dx,dy,1-ptct,j]];
If[dx+1<=degree,
AppendTo[evenequationlist,(D[xf[x],{x,dx+1}] /. {x->(ptct h) -
h/2})-(D[xf[x],{x,dx+1}] /. {x->-(ptct h) + h/2}) == fc[dx+1,dy,ptct,j]
-fc[dx+1,dy,1-ptct,j]]];
,{ptct,1,csize/2}]
,{dx,0,degree,2}];

oddequationlist={};
Do[
Do[
AppendTo[oddequationlist,(D[xf[x],{x,dx}] /. {x->(ptct h) -
h/2})-(D[xf[x],{x,dx}] /. {x->-(ptct h) + h/2}) == fc[dx,dy,ptct,j]
-fc[dx,dy,1-ptct,j]];
If[dx+1<=degree,
AppendTo[oddequationlist,(D[xf[x],{x,dx+1}] /. {x->(ptct h) -
h/2})+(D[xf[x],{x,dx+1}] /. {x->-(ptct h) + h/2}) == fc[dx+1,dy,ptct,j]
+fc[dx+1,dy,1-ptct,j]]];
,{ptct,1,csize/2}]
,{dx,0,degree,2}];

(* ------------------------- *)
(* Solve the even derivatives *)
(* ------------------------- *)
matrixrhs=LinearEquationsToMatrices[evenequationlist,evenvariablelist];
matrix=matrixrhs[[1]];
rhs=matrixrhs[[2]];
xvector=Collect[LinearSolve[matrix,rhs],fc[_,_,_,_]];
Clear[aside,bside];
makeequal[aside_,bside_]:= aside=bside;
pairs = {evenvariablelist, xvector};
Apply[makeequal,pairs];
```

```
(* ------------------------- *)
(* Solve the odd derivatives  *)
(* ------------------------- *)
matrixrhs=LinearEquationsToMatrices[oddequationlist,oddvariablelist];
matrix=matrixrhs[[1]];
rhs=matrixrhs[[2]];
xvector=Collect[LinearSolve[matrix,rhs],fc[_,_,_,_]];
Clear[aside,bside];
makeequal[aside_,bside_]:= aside=bside;
pairs = {oddvariablelist, xvector};
Apply[makeequal,pairs];

];

If[OddQ[csize],

(* ------------------------------------------------------------- *)
(* This system will never get too Big, I think! so do as one group *)
(* ------------------------------------------------------------- *)
equationlist={};
Do[
Do[
AppendTo[equationlist,(D[xf[x],{x,dx}] /. {x->(ptct) h}) == fc[dx,dy,ptct,j]];
,{ptct,-IntegerPart[csize/2],IntegerPart[csize/2]}]
,{dx,0,degree}];


matrixrhs=LinearEquationsToMatrices[equationlist,variablelist];
matrix=matrixrhs[[1]];
rhs=matrixrhs[[2]];
xvector=Collect[LinearSolve[matrix,rhs],fc[_,_,_,_]];
Clear[aside,bside];
makeequal[aside_,bside_]:= aside=bside;
pairs = {variablelist, xvector};
Apply[makeequal,pairs];
];
(*
fc[dx_,dy_,i_,j_]:=D[ff[x,y],{x,dx},{y,dy}]/.{x->i,y->j}
fc[dx_,dy_,i_,j_]:=D[f[x,y],{x,dx},{y,dy}]/.{x->i,y->j}

*)

);

(* ------------------------------------------------------------------------*)
(* Now have the form of the interpolant in one dimension                   *)
(* the c[i] coefficient represents the ith derivative of the function being *)
(* interpolated.                                                           *)
(* ------------------------------------------------------------------------*)
```

```
(* --------------------------- *)
(* Loop through all the y terms *)
(* --------------------------- *)
(* Compute all the S terms for x dimension *)
(* ------------------------------------- *)

computes:=(
numberofcterms=Length[variablelist]-1;
If[EvenQ[csize],
(* Works
Do[
Do[
Do[
s[dy,iindex,j]=Collect[xc[iindex],fc[_,_,_,_]];
,{iindex,0,numberofcterms}]
,{dy,0,degree}]
,{j,1-(csize/2),csize/2}];
*)

(* ----------------------- *)
(* Create FORTRAN Do loops *)
(* ----------------------- *)
(* ------------- *)
(* Define the fc *)
(* ------------- *)
(*
Do[
Do[Do[
xnot="";
Do[
xnot=xnot<>"x";
,{xct,1,dx}];
ynot="";
Do[
ynot=ynot<>"y";
,{yct,1,dy}];
xnot="";
xnot=ToString[dx]<>"x";
ynot="";
ynot=ToString[dy]<>"y";
dnotation=xnot<>ynot;


fcdnotation="fc"<>xnot<>ynot;
fc[dx,dy,i,j]=fcdnotation[gridi+i,gridj+j]
fcdnotation="fctmp";
fc[dx,dy,i,j]=fcdnotation[gridi+i,gridj+j,dx,dy]
,{dx,0,degree}]
,{dy,0,degree}]
,{i,1-(csize/2),csize/2}];
*)
```

```
fcdnotation="fctmp";
fc[dx_,dy_,i_,j_]:=fcdnotation[gridi+i,gridj+j,dx,dy];

Clear[dy,j];
stmp=OpenWrite["tensorroutinetmp.f",FormatType->FortranForm];
Write[stmp,"c****************** Doing the Sy terms ********************c"];
Write[stmp,"       do dy=0, degree"];
(*
Do[
*)
Write[stmp,"       do j=1-(",csize/2,"),",csize/2];
Do[
Write[stmp,"       ",s[dy,iindex,j],"=",xc[iindex] ];
,{iindex,0,numberofcterms}];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
(*
,{dy,0,degree}];
*)
Close[stmp];
Clear[fc];
Run["sed -e 's/\"//g' tensorroutinetmp.f >> tensorroutine.f"];
Run["rm tensorroutinetmp.f"];
];

If[OddQ[csize],
(* WORKS
Do[
Do[
Do[
s[dy,iindex,j]=xc[iindex]
,{iindex,0,numberofcterms}]
,{dy,0,degree}]
,{j,-IntegerPart[csize/2],IntegerPart[csize/2]}];
*)

(* ---------------------- *)
(* Create FORTRAN Do loops *)
(* ---------------------- *)
(* ------------- *)
(* Define the fc *)
(* ------------- *)
(*
Do[
Do[Do[
xnot="";
Do[
xnot=xnot<>"x";
,{xct,1,dx}];
ynot="";
```

```
Do[
ynot=ynot<>"y";
,{yct,1,dy}];
xnot="";
xnot=ToString[dx]<>"x";
ynot="";
ynot=ToString[dy]<>"y";
dnotation=xnot<>ynot;
fcdnotation="fc"<>dnotation;
fc[dx,dy,i,j]=fcdnotation[gridi+i,gridj+j]
,{dx,0,degree}]
,{dy,0,degree}]
,{i,-IntegerPart[csize/2],IntegerPart[csize/2]}];
*)

fcdnotation="fctmp";
fc[dx_,dy_,i_,j_]:=fcdnotation[gridi+i,gridj+j,dx,dy];

Clear[dy,j];
stmp=OpenWrite["tensorroutinetmp.f",FormatType->FortranForm];
(*
Do[
*)
Write[stmp,"c************************* Doing the Sy terms
*************************c"];
Write[stmp,"       do dy=0, degree"];
Write[stmp,"       do j=",-IntegerPart[csize/2],",",IntegerPart[csize/2]];
Do[
Write[stmp,"       ",s[dy,iindex,j],"=",xc[iindex] ];
,{iindex,0,numberofcterms}];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
(*
,{dy,0,degree}];
*)
Close[stmp];
Clear[fc];
Run["sed -e 's/\"//g' tensorroutinetmp.f >> tensorroutine.f"];
Run["rm tensorroutinetmp.f"];

];

);

(* ------------------------------------ *)
(* Now compute the spatial coefficients *)
(* ------------------------------------ *)

(* ------------------------------------------------------ *)
(* Compute the yc coefficients for use in the y direction *)
(* ------------------------------------------------------ *)
```

```
getintformy:=(
yf[y_]:=Sum[yc[i] y^i,{i,0,interpolantorder-1}];
variablelist=CoefficientList[yf[y],y];

evenvariablelist={};
oddvariablelist={};
Do[AppendTo[evenvariablelist,yc[i]],{i,0,interpolantorder-1,2}];
Do[AppendTo[oddvariablelist, yc[i]],{i,1,interpolantorder-1,2}];


(* ---------------------------------------------------- *)
(* Find the values of the coefficients c for c2 stencil *)
(* ---------------------------------------------------- *)

If[EvenQ[csize],
(*-------------------------------------------------------------------------------*)
(* Solve the 1D interpolant by splitting into two independent sets of equation*)
(*-------------------------------------------------------------------------------*)
evenequationlist={};
Do[
Do[
AppendTo[evenequationlist,(D[yf[y],{y,dy}] /. {y->(ptct h) -
h/2})+(D[yf[y],{y,dy}] /. {y->-(ptct h) + h/2}) == fc[dx,dy,i,ptct]
+fc[dx,dy,i,1-ptct]];
If[dy+1<=degree,
AppendTo[evenequationlist,(D[yf[y],{y,dy+1}] /. {y->(ptct h) -
h/2})-(D[yf[y],{y,dy+1}] /. {y->-(ptct h) + h/2}) == fc[dx,dy+1,i,ptct]
-fc[dx,dy+1,i,1-ptct]]];
,{ptct,1,csize/2}]
,{dy,0,degree,2}];


oddequationlist={};
Do[
Do[
AppendTo[oddequationlist,(D[yf[y],{y,dy}] /. {y->(ptct h) -
h/2})-(D[yf[y],{y,dy}] /. {y->-(ptct h) + h/2}) == fc[dx,dy,i,ptct]
-fc[dx,dy,i,1-ptct]];
If[dy+1<=degree,
AppendTo[oddequationlist,(D[yf[y],{y,dy+1}] /. {y->(ptct h) -
h/2})+(D[yf[y],{y,dy+1}] /. {y->-(ptct h) + h/2}) == fc[dx,dy+1,i,ptct]
+fc[dx,dy+1,i,1-ptct]]];
,{ptct,1,csize/2}]
,{dy,0,degree,2}];



(* -------------------------- *)
(* Solve the even derivatives *)
(* -------------------------- *)
```

```
matrixrhs=LinearEquationsToMatrices[evenequationlist,evenvariablelist];
matrix=matrixrhs[[1]];
rhs=matrixrhs[[2]];
xvector=Collect[LinearSolve[matrix,rhs],fc[_,_,_,_]];
Clear[aside,bside];
makeequal[aside_,bside_]:= aside=bside;
pairs = {evenvariablelist, xvector};
Apply[makeequal,pairs];

(* ------------------------- *)
(* Solve the odd derivatives  *)
(* ------------------------- *)
matrixrhs=LinearEquationsToMatrices[oddequationlist,oddvariablelist];
matrix=matrixrhs[[1]];
rhs=matrixrhs[[2]];
xvector=Collect[LinearSolve[matrix,rhs],fc[_,_,_,_]];
Clear[aside,bside];
makeequal[aside_,bside_]:= aside=bside;
pairs = {oddvariablelist, xvector};
Apply[makeequal,pairs];



];

If[OddQ[csize],
equationlist={};
Do[
Do[
AppendTo[equationlist,(D[yf[y],{y,dy}] /. {y->(ptct) h}) == fc[dx,dy,i,ptct]];
,{ptct,-IntegerPart[csize/2],IntegerPart[csize/2]}]
,{dy,0,degree}];

matrixrhs=LinearEquationsToMatrices[equationlist,variablelist];
matrix=matrixrhs[[1]];
rhs=matrixrhs[[2]];
xvector=Collect[LinearSolve[matrix,rhs],fc[_,_,_,_]];
Clear[aside,bside];
makeequal[aside_,bside_]:= aside=bside;
pairs = {variablelist, xvector};
Apply[makeequal,pairs];
];

);


(* ---------------------------------------------------------------- *)
(* Now the c coefficients represent the y interpolator spatial coeff. *)
(* Substitute the fc with the s data                                  *)
(* ---------------------------------------------------------------- *)
(* Compute Spatial Coefficients *)
```

```
(* -------------------------- *)

computea:=(
(* WORKS
Do[
Do[
Clear[newdx,newdy,newi,newj];
a[iindex,jindex]=yc[jindex] /. {fc[newdx:_,newdy:_,newi:_,newj:_] ->
                        s[newdy,newdx,newj] } /. {dx->iindex}
,{jindex,0,numberofcterms}]
,{iindex,0,numberofcterms}];
*)

(* ---------------------- *)
(* Create FORTRAN Do loops *)
(* ---------------------- *)
Clear[dy,j,s];
stmp=OpenWrite["tensorroutinetmp.f",FormatType->FortranForm];
Write[stmp,"c*************** Doing the fctmp coefficients *****************c"];
Write[stmp,"       do iindex=0,",numberofcterms];
Do[
Write[stmp,"       ",cffc[iindex,jindex,0],"=",yc[jindex] /.
{fc[newdx:_,newdy:_,newi:_,newj:_] -> s[newdy,newdx,newj] } /. {dx->iindex}];
,{jindex,0,numberofcterms}];
Write[stmp,"       end do"];
Close[stmp];
Run["sed -e 's/\"//g' tensorroutinetmp.f >> tensorroutine.f"];
Run["rm tensorroutinetmp.f"];

);

(* ------------------------------------------------------------------------ *)
(* This will read tensoroutine.f, and duplicate three times with p,u,v in place *)
(* of fc                                                                    *)
(* ------------------------------------------------------------------------ *)
makecompleteroutine:=(

(* -------------------- *)
(* Make the three copies *)
(* -------------------- *)

Run["rm tensorroutinetmpup.f"];
stmp=OpenAppend["tensorroutinetmpup.f",FormatType->FortranForm];
Write[stmp,"c*************************************c"];
Write[stmp,"c* Solving the p spatial coefficients *c"];
Write[stmp,"c*************************************c"];
Close[stmp];
Run["sed -e 's/ffc/p/g' -e 's/fctmp/p/g' tensorroutine.f >>
tensorroutinetmpup.f"];
stmp=OpenAppend["tensorroutinetmpup.f",FormatType->FortranForm];
Write[stmp,"c*************************************c"];
```

```
Write[stmp,"c* Solving the u spatial coefficients *c"];
Write[stmp,"c*************************************c"];
Close[stmp];
Run["sed -e 's/ffc/u/g' -e 's/fctmp/u/g' tensorroutine.f >>
tensorroutinetmpup.f"];
stmp=OpenAppend["tensorroutinetmpup.f",FormatType->FortranForm];
Write[stmp,"c*************************************c"];
Write[stmp,"c* Solving the v spatial coefficients *c"];
Write[stmp,"c*************************************c"];
Close[stmp];
Run["sed -e 's/ffc/v/g' -e 's/fctmp/v/g' tensorroutine.f >>
tensorroutinetmpup.f"];

(* --------------------- *)
(* Make the down version *)
(* --------------------- *)

Run["rm tensorroutinetmpdown.f"];
stmp=OpenAppend["tensorroutinetmpdown.f",FormatType->FortranForm];
Write[stmp,"c*************************************c"];
Write[stmp,"c* Solving the p spatial coefficients *c"];
Write[stmp,"c*************************************c"];
Close[stmp];
Run["sed -e 's/ffc/p/g' -e 's/fctmp/np/g' tensorroutine.f >>
tensorroutinetmpdown.f"];
stmp=OpenAppend["tensorroutinetmpdown.f",FormatType->FortranForm];
Write[stmp,"c*************************************c"];
Write[stmp,"c* Solving the u spatial coefficients *c"];
Write[stmp,"c*************************************c"];
Close[stmp];
Run["sed -e 's/ffc/u/g' -e 's/fctmp/nu/g' tensorroutine.f >>
tensorroutinetmpdown.f"];
stmp=OpenAppend["tensorroutinetmpdown.f",FormatType->FortranForm];
Write[stmp,"c*************************************c"];
Write[stmp,"c* Solving the v spatial coefficients *c"];
Write[stmp,"c*************************************c"];
Close[stmp];
Run["sed -e 's/ffc/v/g' -e 's/fctmp/nv/g' tensorroutine.f >>
tensorroutinetmpdown.f"];

Run["rm tensorroutine.f"];


stmp=OpenAppend["tensorroutinetmpup.f",FormatType->FortranForm];
Write[stmp,"       end"];
Close[stmp];
stmp=OpenAppend["tensorroutinetmpdown.f",FormatType->FortranForm];
Write[stmp,"       end"];
Close[stmp];

stmp=OpenWrite["tensorroutine1up.f",FormatType->FortranForm];
```

```
Write[stmp,"       subroutine tensorup"];
Write[stmp,"       include 'common.h'"];
Close[stmp];
stmp=OpenWrite["tensorroutine1down.f",FormatType->FortranForm];
Write[stmp,"       subroutine tensordown"];
Write[stmp,"       include 'common.h'"];
Close[stmp];
Run["sed -e 's/\"//g' tensorroutine1up.f >  tensorroutineup.f"];
Run["sed -e 's/\"//g' tensorroutinetmpup.f >> tensorroutineup.f"];
Run["sed -e 's/\"//g' tensorroutine1down.f >  tensorroutinedown.f"];
Run["sed -e 's/\"//g' tensorroutinetmpdown.f >> tensorroutinedown.f"];

Run["rm tensorroutine1up.f"];
Run["rm tensorroutine1down.f"];
Run["rm tensorroutinetmpup.f"];
Run["rm tensorroutinetmpdown.f"];

);
```

B.4 Temporal Evolution Using Recursive Tensor Form File – ce2d.tfelp

This code will create the FORTRAN code for computing the temporal evolution of each grid point at each time step using the method discussed in section 3.2.3.

```
(* --------------- *)
(* calculateelp:=( *)
(* --------------- *)

startce:=(
Run["rm timeadvanceroutine.f"];
Run["rm shiftmultiplyroutine.f"];

(* ------------------------------------------------------ *)
(* Get input data and setup interpolating polynomial forms *)
(* ------------------------------------------------------ *)
initproc;

(* ------------------------------------------- *)
(* Set linearized euler equations equal to zero *)
(* ------------------------------------------- *)
setupequations;

(* ------------------------------------------- *)
(* Express time coefficients as space coefficients *)
(* ------------------------------------------- *)
determineelp;

(* ----------------------- *)
(* Write Factorial Routine *)
(* ----------------------- *)
makefactorialfile;

(* ------------------------------------------------------------ *)
(* Rearrange exact propagator time advance solutions as a linear *)
(* combination of cp,cu,cv spatial coefficients                  *)
(* ------------------------------------------------------------ *)
(*
collectterms;
*)

(* ------------------------------------------------------------------- *)
(* Find the constant equation for each cp,cu,and cv, and write to FORTRAN *)
(* ------------------------------------------------------------------- *)
(*
getkvalues;
*)

(* ------------------------------------------------------------------- *)
(* Generate the time advance shift multiply routine and write to FORTRAN *)
```

```
(* ------------------------------------------------------------------ *)
(*
shiftmultiply;
*)

(* ------------------------------------------------------- *)
(* To avoid copying memory, make an up and down time advance *)
(* ------------------------------------------------------- *)
makedownanduptimeadvance;

);

initproc:=(
Clear[cp,cu,cv,pp,uu,vv,p,u,v];
If[Not[ValueQ[csize]],
csize = Input["Enter the stencil size of the problem: "];
];
If[Not[ValueQ[degree]],
degree = Input["Enter the maximum degree of the stencil data: "] ;
];
maxind = (csize*degree) + (csize-1);

Collect[D[p[x,y,t],x]/.{x->0,y->0},{cp[_,_,_],cu[_,_,_],cv[_,_,_]}];


(* ------------------------------------------------------------ *)
(* Form the interpolating polynomial with unknowns cp,cu,cv[i,j] *)
(* ------------------------------------------------------------ *)
p[x_,y_,t_]:=Sum[cp[i,j,k] x^i y^j t^k, {j,0,maxind},{i,0,maxind},{k,0,(csize
degree)+ (csize(degree+2)-2)-(i+j)}];

u[x_,y_,t_]:=Sum[cu[i,j,k] x^i y^j t^k, {j,0,maxind},{i,0,maxind},{k,0,(csize
degree) + (csize(degree+2)-2)-(i+j)}];

v[x_,y_,t_]:=Sum[cv[i,j,k] x^i y^j t^k, {j,0,maxind},{i,0,maxind},{k,0,(csize
degree) + (csize(degree+2)-2)-(i+j)}];

);

setupequations:=(
(* ------------------------------------------------ *)
(* Linearized Euler Equations                       *)
(* ------------------------------------------------ *)

equation1= Expand[D[p[x,y,t],t] + mx D[p[x,y,t],x] + my D[p[x,y,t],y] +
                          D[u[x,y,t],x] +    D[v[x,y,t],y] ];
(*
equation345= Expand[D[D[p[x,y,t],t] + mx D[p[x,y,t],x] + my D[p[x,y,t],y] +
                          D[u[x,y,t],x] +    D[v[x,y,t],y],{x,3},{y,3},{t,5}]
];
```

```
*)
equation2= Expand[D[u[x,y,t],t] + mx D[u[x,y,t],x] + my D[u[x,y,t],y] +
                          D[p[x,y,t],x] ];

equation3= Expand[D[v[x,y,t],t] + mx D[v[x,y,t],x] + my D[v[x,y,t],y] +
                                              D[p[x,y,t],y] ];

);

determineelp:=(
Clear[a,b,c];
(* ------------------------------------------------------------------------- *)
(* These conditions are derived directly from LEE are required for exact     *)
(* propagators                                                               *)
(* ------------------------------------------------------------------------- *)
ncp[a_,b_,c_]:=(-(b+1)(my cp[a,b+1,c-1] + cv[a,b+1,c-1]) -(a+1) (mx
cp[a+1,b,c-1] + cu[a+1,b,c-1]))/c;
ncu[a_,b_,c_]:=(-(b+1) my cu[a,b+1,c-1] - (a+1) ( mx cu[a+1,b,c-1] +
cp[a+1,b,c-1] ))/c;
ncv[a_,b_,c_]:=(-(b+1) ( my cv[a,b+1,c-1] + cp[a,b+1,c-1] ) -(a+1) mx
cv[a+1,b,c-1])/c;

Run["rm timeadvance.tfelp.f"];
stmp=OpenWrite["timeadvance.tfelptmp.f",FormatType->FortranForm];
Write[stmp,"       subroutine timeadvance"];
Write[stmp,"       include 'common.h'"];
Write[stmp,"c**********************************************************************c"];
Write[stmp,"c* Calculate the cp[_,_,_>0],cu[_,_,_>0],cv[_,_,_>0] terms now *c"];
Write[stmp,"c**********************************************************************c"];
Write[stmp,"       do kindex=1, 2*maxind "];
Write[stmp,"       do jindex=0, maxind "];
(*
Write[stmp,"       do iindex=0, 2*maxind-kindex-jindex "];
*)
Write[stmp,"       do iindex=0, min(maxind,2*maxind-kindex-jindex) "];
Write[stmp,"       cp(iindex,jindex,kindex)=",ncp[iindex,jindex,kindex]];
Write[stmp,"       cu(iindex,jindex,kindex)=",ncu[iindex,jindex,kindex]];
Write[stmp,"       cv(iindex,jindex,kindex)=",ncv[iindex,jindex,kindex]];
Write[stmp,"       end do "];
Write[stmp,"       end do "];
Write[stmp,"       end do "];

(*
Do[Do[
xnot="";
xnot=ToString[dx]<>"x";
ynot="";
ynot=ToString[dy]<>"y";
dnotation=xnot<>ynot;
*)
(*
```

```
Write[stmp,"c************** Begin time advance using Horner Form
************c"];
Write[stmp,"       do dy=0,degree"];
Write[stmp,"       do dx=0,degree"];
Write[stmp,"       psum=0.0"];
Write[stmp,"       usum=0.0"];
Write[stmp,"       vsum=0.0"];

Write[stmp,"       do kindex=0,",2*maxind];
Write[stmp,"       psum=psum+fac(dx) * fac(dy) *
physicaltstep**kindex*cp(dx,dy,kindex)"];
Write[stmp,"       enddo "];

Write[stmp,"       do kindex=0,",2*maxind];
Write[stmp,"       usum=usum+fac(dx) * fac(dy) *
physicaltstep**kindex*cu(dx,dy,kindex)"];
Write[stmp,"       enddo "];

Write[stmp,"       do kindex=0,",2*maxind];
Write[stmp,"       vsum=vsum+fac(dx) * fac(dy) *
physicaltstep**kindex*cv(dx,dy,kindex)"];
Write[stmp,"       enddo "];

Write[stmp,"       np(gridi+stagger,gridj+stagger,dx,dy)=psum"];
Write[stmp,"       nu(gridi+stagger,gridj+stagger,dx,dy)=usum"];
Write[stmp,"       nv(gridi+stagger,gridj+stagger,dx,dy)=vsum"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
Write[stmp,"c****************** Done advancing time  **************c"];
*)
Write[stmp,"c************** Begin time advance using Horner Form
************c"];
Write[stmp,"       do dy=0,degree"];
Write[stmp,"       do dx=0,degree"];
Write[stmp,"       factterm=fac(dx)*fac(dy) "];
Write[stmp,"       psum=0.0"];
Write[stmp,"       usum=0.0"];
Write[stmp,"       vsum=0.0"];

Write[stmp,"       do kindex=",2*maxind,",1,-1"];
Write[stmp,"       psum=physicaltstep*((factterm * cp(dx,dy,kindex))+psum)"];
Write[stmp,"       enddo "];

Write[stmp,"       do kindex=",2*maxind,",1,-1"];
Write[stmp,"       usum=physicaltstep*((factterm * cu(dx,dy,kindex))+usum)"];
Write[stmp,"       enddo "];

Write[stmp,"       do kindex=",2*maxind,",1,-1"];
Write[stmp,"       vsum=physicaltstep*((factterm * cv(dx,dy,kindex))+vsum)"];
Write[stmp,"       enddo "];
```

```
Write[stmp,"
np(gridi+stagger,gridj+stagger,dx,dy)=psum+(cp(dx,dy,0)*factterm)"];
Write[stmp,"
nu(gridi+stagger,gridj+stagger,dx,dy)=usum+(cu(dx,dy,0)*factterm)"];
Write[stmp,"
nv(gridi+stagger,gridj+stagger,dx,dy)=vsum+(cv(dx,dy,0)*factterm)"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
Write[stmp,"c****************** Done advancing time  **************c"];
(*
,{dx,0,degree}]
,{dy,0,degree}];
*)
Write[stmp,"       end"];
Close[stmp];
Run["sed -e 's/\"//g' timeadvance.tfelptmp.f >> timeadvance.tfelp.f"];
Run["rm timeadvance.tfelptmp.f"];

);

(* ------------------------------------------------------- *)
(* Create Factorial FORTRAN subroutine to reduce multiplies *)
(* ------------------------------------------------------- *)
makefactorialfile:=(
Run["rm factorialroutine.f"];
stmp=OpenWrite["factorialroutinetmp.f",FormatType->FortranForm];
Write[stmp,"       subroutine computefactorials"];
Write[stmp,"       include 'common.h'"];
Do[
Write[stmp,"       fac(",ct,")=",ct!];
,{ct,0,degree}];
Write[stmp,"       end"];
Close[stmp];
Run["sed -e 's/\"//g' factorialroutinetmp.f >> factorialroutine.f"];
Run["rm factorialroutinetmp.f"];
);

shiftmultiply:=(
(* ------------------------------------------------------------------ *)
(* Create the shift multiply for time advancing with minimal multiplies *)
(* and data storage, keep memory together using multiple loops          *)
(* ------------------------------------------------------------------ *)
stmp=OpenWrite["shiftmultiplyroutinetmp.f",FormatType->FortranForm];

Write[stmp,"       subroutine shiftmultiply"];
Write[stmp,"       include 'common.h'"];

(*
Write[stmp,"       do dy=0,degree"];
Write[stmp,"       do dx=0,degree"];
*)
```

```
Do[Do[

Write[stmp,"c********************** Advancing ",dx,"x",dy,"y"," terms
**************c"];
Write[stmp,"       psum=0"];
Write[stmp,"       usum=0"];
Write[stmp,"       vsum=0"];

Write[stmp,"       do jindex=0,maxind-",dy];
Write[stmp,"       do iindex=0,maxind-",dx];
Write[stmp,"
mfac=(fac(iindex+",dx,")*fac(jindex+",dy,"))/(fac(iindex)*fac(jindex))"];
Write[stmp,"       mfacp=cp(iindex+",dx,",","jindex+",dy,") * mfac"];
Write[stmp,"       psum=psum+(mfacp*ppkp(iindex,jindex))"];
Write[stmp,"       usum=usum+(mfacp*uukp(iindex,jindex))"];
Write[stmp,"       vsum=vsum+(mfacp*vvkp(iindex,jindex))"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];

Write[stmp,"       do jindex=0,maxind-",dy];
Write[stmp,"       do iindex=0,maxind-",dx];
Write[stmp,"
mfac=(fac(iindex+",dx,")*fac(jindex+",dy,"))/(fac(iindex)*fac(jindex))"];
Write[stmp,"       mfacu=cu(iindex+",dx,",","jindex+",dy,") * mfac"];
Write[stmp,"       psum=psum+(mfacu*ppku(iindex,jindex))"];
Write[stmp,"       usum=usum+(mfacu*uuku(iindex,jindex))"];
Write[stmp,"       vsum=vsum+(mfacu*vvku(iindex,jindex))"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];

Write[stmp,"       do jindex=0,maxind-",dy];
Write[stmp,"       do iindex=0,maxind-",dx];
Write[stmp,"
mfac=(fac(iindex+",dx,")*fac(jindex+",dy,"))/(fac(iindex)*fac(jindex))"];
Write[stmp,"       mfacv=cv(iindex+",dx,",","jindex+",dy,") * mfac"];
Write[stmp,"       psum=psum+(mfacv*ppkv(iindex,jindex))"];
Write[stmp,"       usum=usum+(mfacv*uukv(iindex,jindex))"];
Write[stmp,"       vsum=vsum+(mfacv*vvkv(iindex,jindex))"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];

xnot="";
xnot=ToString[dx]<>"x";
ynot="";
ynot=ToString[dy]<>"y";
dnotation=xnot<>ynot;
Write[stmp,"       np",dnotation,"(gridi+stagger,gridj+stagger)=psum"];
Write[stmp,"       nu",dnotation,"(gridi+stagger,gridj+stagger)=usum"];
Write[stmp,"       nv",dnotation,"(gridi+stagger,gridj+stagger)=vsum"];
Write[stmp,"c****************** Done advancing ",dnotation," terms
```

```
**********************c"];

(*
Write[stmp,"       end do"];
Write[stmp,"       end do"];
*)
,{dx,0,degree}]
,{dy,0,degree}];

Write[stmp,"       end"];
Close[stmp];

Run["sed -e 's/\"//g' shiftmultiplyroutinetmp.f >> shiftmultiplyroutine.f"];
Run["rm shiftmultiplyroutinetmp.f"];


Do[Do[

psum=0;
usum=0;
vsum=0;
Do[
Do[

mfac=(fac[iindex+dx]*fac[jindex+dy])/(fac[iindex]*fac[jindex]);

psum = psum+(mfac*ppkp[iindex,jindex] * cp[iindex+dx,jindex+dy]) ;
psum = psum+(mfac*ppku[iindex,jindex] * cu[iindex+dx,jindex+dy]) ;
psum = psum+(mfac*ppkv[iindex,jindex] * cv[iindex+dx,jindex+dy]) ;

usum = usum+(mfac*uukp[iindex,jindex] * cp[iindex+dx,jindex+dy]) ;
usum = usum+(mfac*uuku[iindex,jindex] * cu[iindex+dx,jindex+dy]) ;
usum = usum+(mfac*uukv[iindex,jindex] * cv[iindex+dx,jindex+dy]) ;

vsum = vsum+(mfac*vvkp[iindex,jindex] * cp[iindex+dx,jindex+dy]) ;
vsum = vsum+(mfac*vvku[iindex,jindex] * cu[iindex+dx,jindex+dy]) ;
vsum = vsum+(mfac*vvkv[iindex,jindex] * cv[iindex+dx,jindex+dy]) ;
,{iindex,0,maxind-dx}]
,{jindex,0,maxind-dy}];
p[dx,dy]=psum;
u[dx,dy]=usum;
v[dx,dy]=vsum;
,{dx,0,degree}];
,{dy,0,degree}];

);

fac[x_]:=x!;

makeequal:=(
Do[Do[
```

```
cp[iindex,jindex,0]=cp[iindex,jindex];
cu[iindex,jindex,0]=cu[iindex,jindex];
cv[iindex,jindex,0]=cv[iindex,jindex];
,{iindex,0,maxind}]
,{jindex,0,maxind}];
);


makedownanduptimeadvance:=(
Run["rm timeadvanceup.tfelp.f"];
Run["sed -e 's/timeadvance/timeadvanceup/g' timeadvance.tfelp.f >>
timeadvanceup.tfelp.f"];
Run["rm timeadvancedown.tfelp.f"];
Run["sed -e 's/timeadvance/timeadvancedown/g' -e 's/np/p/g' -e 's/nu/u/g' -e
's/nv/v/g' timeadvance.tfelp.f >> timeadvancedown.tfelp.f"];
Run["rm timeadvance.tfelp.f"];

);
```

B.5 Create All The Administrative Files – su2d.geom

This code will create the FORTRAN code that computes the initial conditions, checks the errors at each time step, reads the fill file described in section 5.5, and constructs the FORTRAN common data area.

```
startsu:=(

(* This is done separately after ma2d so that length of fillarrays is known
makecommonfile;
*)
makeinitcondfile;
makeerrorcalcupfile;
makeerrorcalcdownfile;
makemainfile;
(*
makeperiodicexfile;
*)
makefillfillsfile;
makereadfillsfile;
(* Put it directly in code
makerotatefile;
*)
makedeterminefillsfile;
makedefinephysicalxyfile;
);

(* ------------------------------------------------------------------ *)
(* This procedure creates the common file used by all subroutines     *)
(* Instead of using parameters, this permits easier inlining later    *)
(* and minimizes memory moves to maxmimize floating point performance *)
(* ------------------------------------------------------------------ *)
makecommonfile:=(
Print["Removing common.h"];
Run["rm common.h"];
Print["Generating common.h"];
stmp=OpenWrite["commontmp.h",FormatType->FortranForm];
Write[stmp,"       integer maxind,maxi,maxj,memcost"];
Write[stmp,"       real pi,alpha"];
Write[stmp,"       parameter (maxind=",maxind,")"];
Write[stmp,"       common maxi,maxj "];
Write[stmp,"       integer maxmemi,maxmemj,maxlength "];
Write[stmp,"       parameter (maxmemi=",maxmemi,")"];
Write[stmp,"       parameter (maxmemj=",maxmemj,")"];
Write[stmp,"       parameter (maxlength=",maxlength,")"];
Write[stmp,"       parameter (alpha=",N[alpha],")"];

(*
Write[stmp,"       parameter (maxi=",maxi,")"];
Write[stmp,"       parameter (maxj=",maxj,")"];
*)
Write[stmp,"       parameter (pi=",N[Pi,30],")"];
```

```
Write[stmp,"       integer stepn, stagger "];
Write[stmp,"       common /intstep/ stepn, stagger "];
Write[stmp,"       integer idx,fillct,ipct,iuct,ivct "];
Write[stmp,"       real h,lam,mx,my,physicalt,wx,wy,physicaltstep,iun"];
Write[stmp,"       common /stepinfo/ h,lam,mx,my,physicalt,wx,wy "];
Write[stmp,"       common /stepinfo/ physicaltstep, iun "];
Write[stmp,"       real maxp,minp,l1err,maxperr,perr,initialenergy"];
Write[stmp,"       real currentenergy, physicalnx,physicalny,rotatex,rotatey "];
Write[stmp,"       common /errorinfo/ maxp,minp,l1err,maxperr,perr "];
Write[stmp,"       common /errorinfo/ initialenergy, currentenergy "];
Write[stmp,"       real cp(0:maxind+2,0:maxind+2,0:2*maxind)"];
Write[stmp,"       real cu(0:maxind+2,0:maxind+2,0:2*maxind)"];
Write[stmp,"       real cv(0:maxind+2,0:maxind+2,0:2*maxind)"];
Write[stmp,"       common /coef/ cp,cu,cv"];
Write[stmp,"       real factterm "];
Write[stmp,"       real fac(0:",degree,")"];
Write[stmp,"       common fac "];
Write[stmp,"       integer dx,dy,degree,iindex,jindex,kindex,csize"];
Write[stmp,"       parameter (degree=",degree,")"];
Write[stmp,"       parameter (csize=",csize,")"];
Write[stmp,"       real psum,usum,vsum,mfac,mfacp,mfacu,mfacv"];
Write[stmp,"       common psum,usum,vsum,mfac,mfacp,mfacu,mfacv "];
Write[stmp,"       integer gridi,gridj"];
Write[stmp,"       common /gridcoords/ gridi,gridj"];
Write[stmp,"       integer lgridi,lgridj "];
Write[stmp,"       real
physicalx(-maxmemi:maxmemi),physicaly(-maxmemj:maxmemj)"];
Write[stmp,"       common /physicalxy/ physicalx,physicaly "];
Write[stmp,"       integer errorsteps, numberofnsteps "];
Write[stmp,"       integer ioffset,joffset,i,j "];
Write[stmp,"       real totalperr"];
Write[stmp,"       real
s(0:",degree,",","0:",numberofcterms,",",-IntegerPart[csize/2],":",IntegerPart[c
size/2],")"];
Write[stmp,"       common errorsteps,
numberofnsteps,ioffset,joffset,i,j,totalperr,s"];
Write[stmp,"       integer lengthoffillarrays,numberofp,numberofu,numberofv "];
Write[stmp,"       integer filli,fillj,filldx,filldy,lci,lcj,lcdx,lcdy"];
Write[stmp,"       integer numberoffillpts "];
Write[stmp,"       real   lccoef,onelongarray(maxlength) "];
Write[stmp,"       common /fillstuff/
lengthoffillarrays,numberoffillpts,onelongarray"];

Write[stmp,"       real
p(-maxmemi:maxmemi,-maxmemj:maxmemj,0:degree,0:degree)"];
Write[stmp,"       real
u(-maxmemi:maxmemi,-maxmemj:maxmemj,0:degree,0:degree)"];
Write[stmp,"       real
v(-maxmemi:maxmemi,-maxmemj:maxmemj,0:degree,0:degree)"];
Write[stmp,"       common /dataongrid/ p,u,v"];
Write[stmp,"       real
```

```
np(-maxmemi:maxmemi,-maxmemj:maxmemj,0:degree,0:degree)"];
Write[stmp,"       real
nu(-maxmemi:maxmemi,-maxmemj:maxmemj,0:degree,0:degree)"];
Write[stmp,"       real
nv(-maxmemi:maxmemi,-maxmemj:maxmemj,0:degree,0:degree)"];
Write[stmp,"       common /dataongrid/ np,nu,nv"];
Write[stmp,"       integer interior(-maxmemi:maxmemi,-maxmemj:maxmemj) "];
Write[stmp,"       common interior "];


Close[stmp];
Run["sed -e 's/\"//g'  -e 's/\\\\//g' commontmp.h >> common.h"];
Run["rm commontmp.h"];


);
makeinitcondfile:=(
Print["Removing initcond.f"];
Run["rm initcond.f"];
Print["Generating initial condition routine initcond.f"];
Clear[p,u,v,x,y,t];
(*
(* ------------------------------------------- *)
(* Analytical solution to bi-periodic LEE problem *)
(* ------------------------------------------- *)
p[x_,y_,t_]:=Cos[Sqrt[wx^2+wy^2] pi t]*Sin[wx pi (x - mx t)]*Sin[wy pi (y - my
t)];
u[x_,y_,t_]:=-(wx/Sqrt[wx^2+wy^2])*Sin[Sqrt[wx^2+wy^2] pi t]*Cos[wx pi (x - mx
t)]*Sin[wy pi (y - my t)];
v[x_,y_,t_]:=-(wy/Sqrt[wx^2+wy^2])*Sin[Sqrt[wx^2+wy^2] pi t]*Sin[wx pi (x - mx
t)]*Cos[wy pi (y - my t)];
*)
(* ------------------------------------------- *)
(* Analytical solution to rotated box LEE problem *)
(* ------------------------------------------- *)
p[x_,y_,t_]:=-Cos[Sqrt[wx^2+wy^2] Pi t] Cos[wx Pi (x- mx t)] Cos[wy Pi (y-my
t)];

u[x_,y_,t_]:=-(wx/Sqrt[wx^2+wy^2])*Sin[Sqrt[wx^2+wy^2] Pi t]*Sin[wx Pi (x - mx
t)]*Cos[wy Pi (y - my t)];
v[x_,y_,t_]:=-(wy/Sqrt[wx^2+wy^2])*Sin[Sqrt[wx^2+wy^2] Pi t]*Cos[wx Pi (x - mx
t)]*Sin[wy Pi (y - my t)];

If[Not[ValueQ[degree]],
degree=Input["Enter the degree"];
];

If[Not[ValueQ[maxmemi]],
maxmemi=Input["Enter the maximum x coordinate"];
maxmemj=maxmemi;
];
```

```
stmp=OpenWrite["initcondtmp.f",FormatType->FortranForm];
Write[stmp,"       subroutine initcond"];
Write[stmp,"       include 'common.h'"];

Write[stmp,"c************************************c"];
Write[stmp,"c Defining physical coordinates       c"];
Write[stmp,"c************************************c"];

Write[stmp,"       write(iun,*) 'Rotated Box LEE Analytical Solution :'"];
mx:=0;
my:=0;
Write[stmp,"       write(iun,*) 'p[x,y,t]= ",p[x,y,t],"'"];
Write[stmp,"       write(iun,*) 'u[x,y,t]= ",u[x,y,t],"'"];
Write[stmp,"       write(iun,*) 'v[x,y,t]= ",v[x,y,t],"'"];

Write[stmp,"c       do lgridi=-",maxmemi,",",maxmemi];
Write[stmp,"c       physicalx(lgridi)= lgridi * h"];
Write[stmp,"c       end do"];
Write[stmp,"c       do lgridj=-",maxmemj,",",maxmemj];
Write[stmp,"c       physicaly(lgridj)= lgridj * h"];
Write[stmp,"c       end do"];
Write[stmp,"c*******************************************************c"];
Write[stmp,"c* For the tensor form of time advance, do this once *c"];
Write[stmp,"c*******************************************************"];
Write[stmp,"       do iindex=0,",maxind+2];
Write[stmp,"       do jindex=0,",maxind+2];
Write[stmp,"       do kindex=0,",maxind*2];
Write[stmp,"       cp(iindex,jindex,kindex)=0.0 "];
Write[stmp,"       cu(iindex,jindex,kindex)=0.0 "];
Write[stmp,"       cv(iindex,jindex,kindex)=0.0 "];
Write[stmp,"       end do "];
Write[stmp,"       end do "];
Write[stmp,"       end do "];

Write[stmp,"       initialenergy=0. "];
Do[Do[
dnotation=ToString[dx]<>"x"<>ToString[dy]<>"y";
Write[stmp,"c****** Define the initial conditions for the ",dnotation," terms
****c"];
Write[stmp,"       do lgridj=-maxmemj,maxmemj"];
Write[stmp,"       do lgridi=-maxmemi,maxmemj"];
Write[stmp,"               if
((interior(lgridi,lgridj).eq.1).or.(interior(lgridi,lgridj).eq.2)) then"];
Write[stmp,"c
physicalnx=rotatex(physicalx(lgridi),physicaly(lgridj),-alpha)"];
Write[stmp,"c
physicalny=rotatey(physicalx(lgridi),physicaly(lgridj),-alpha)"];
Write[stmp,"     physicalnx=(cos(-alpha) * physicalx(lgridi)) + (sin(-alpha)
* physicaly(lgridj))"];
Write[stmp,"     physicalny=(-sin(-alpha) * physicalx(lgridi)) +
(cos(-alpha) * physicaly(lgridj))"];
```

```
Write[stmp,"
p(lgridi,lgridj,",dx,",",dy,")=",D[p[x,y,t],{x,dx},{y,dy}]/.{x->physicalnx,y->ph
ysicalny,t->0}];
Write[stmp,"
u(lgridi,lgridj,",dx,",",dy,")=",D[u[x,y,t],{x,dx},{y,dy}]/.{x->physicalnx,y->ph
ysicalny,t->0}];
Write[stmp,"
v(lgridi,lgridj,",dx,",",dy,")=",D[v[x,y,t],{x,dx},{y,dy}]/.{x->physicalnx,y->ph
ysicalny,t->0}];
Write[stmp,"        else"];
(*
Write[stmp,"        p(lgridi,lgridj,dx,dy)=0.0"];
Write[stmp,"        u(lgridi,lgridj,dx,dy)=0.0"];
Write[stmp,"        v(lgridi,lgridj,dx,dy)=0.0"];
*)
Write[stmp,"        p(lgridi,lgridj,",dx,",",dy,")=0.0"];
Write[stmp,"        u(lgridi,lgridj,",dx,",",dy,")=0.0"];
Write[stmp,"        v(lgridi,lgridj,",dx,",",dy,")=0.0"];
Write[stmp,"        endif"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
,{dx,0,degree}]
,{dy,0,degree}];

(* ---------------------------------------------- *)
(* Calculate initial energy using generalized form *)
(* ---------------------------------------------- *)
(*
Write[stmp,"       do dy=0,degree"];
Write[stmp,"       do dx=0,degree"];
*)
Write[stmp,"       do dy=0,0"];
Write[stmp,"       do dx=0,0"];
Write[stmp,"       do lgridj=-maxmemj,maxmemj"];
Write[stmp,"       do lgridi=-maxmemi,maxmemi"];
Write[stmp,"       if
((interior(lgridi,lgridj).eq.1).or.(interior(lgridi,lgridj).eq.2)) then"];

Write[stmp,"       initialenergy=initialenergy+(p(lgridi,lgridj,dx,dy)**2 "];
Write[stmp,"     -+u(lgridi,lgridj,dx,dy)**2"];
Write[stmp,"     -+v(lgridi,lgridj,dx,dy)**2)"];
Write[stmp,"       endif"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];

Write[stmp,"       end do"];
Write[stmp,"       end do"];
Write[stmp,"       initialenergy=initialenergy*h*h "];
Write[stmp,"       end"];
Close[stmp];
Run["sed -e 's/\"//g' -e 's/\\\\//g' initcondtmp.f >> initcond.f"];
```

```
Run["rm initcondtmp.f"];


);

makeerrorcalcdownfile:=(
Print["Removing errorcalcdown.f"];
Run["rm errorcalcdown.f"];
Print["Generating error calculation routine errorcalcdown.f"];
Clear[p,u,v];
(*
(* ---------------------------------------------- *)
(* Analytical solution to bi-periodic LEE problem *)
(* ---------------------------------------------- *)
p[x_,y_,t_]:=Cos[Sqrt[wx^2+wy^2] pi t]*Sin[wx pi (x - mx t)]*Sin[wy pi (y - my
t)];
u[x_,y_,t_]:=-(wx/Sqrt[wx^2+wy^2])*Sin[Sqrt[wx^2+wy^2] pi t]*Cos[wx pi (x - mx
t)]*Sin[wy pi (y - my t)];
v[x_,y_,t_]:=-(wy/Sqrt[wx^2+wy^2])*Sin[Sqrt[wx^2+wy^2] pi t]*Sin[wx pi (x - mx
t)]*Cos[wy pi (y - my t)];
*)
(* ---------------------------------------------- *)
(* Analytical solution to rotated box LEE problem *)
(* ---------------------------------------------- *)
p[x_,y_,t_]:=-Cos[Sqrt[wx^2+wy^2] Pi t] Cos[wx Pi (x- mx t)] Cos[wy Pi (y-my
t)];

u[x_,y_,t_]:=-(wx/Sqrt[wx^2+wy^2])*Sin[Sqrt[wx^2+wy^2] Pi t]*Sin[wx Pi (x - mx
t)]*Cos[wy Pi (y - my t)];
v[x_,y_,t_]:=-(wy/Sqrt[wx^2+wy^2])*Sin[Sqrt[wx^2+wy^2] Pi t]*Cos[wx Pi (x - mx
t)]*Sin[wy Pi (y - my t)];



stmp=OpenWrite["errorcalctmp.f",FormatType->FortranForm];
Write[stmp,"       subroutine errorcalcdown"];
Write[stmp,"       include 'common.h'"];
(*
Write[stmp,"       real rotatex"];
Write[stmp,"       real rotatey"];
*)

Write[stmp,"       integer bigi,bigj "];
Write[stmp,"       real eratio "];
Write[stmp,"       totalperr=0.0 "];
Write[stmp,"       maxperr=0.0 "];
Write[stmp,"       maxp=0.0 "];
Write[stmp,"       minp=0.0 "];
Write[stmp,"       currentenergy=0.0"];
Write[stmp,"       do lgridj=-maxmemj,maxmemj"];
Write[stmp,"       do lgridi=-maxmemi,maxmemi"];
```

```
Write[stmp,"       if
((interior(lgridi,lgridj).eq.1).or.(interior(lgridi,lgridj).eq.2)) then"];
Write[stmp,"c
physicalnx=rotatex(physicalx(lgridi),physicaly(lgridj),-alpha)  "];
Write[stmp,"c
physicalny=rotatey(physicalx(lgridi),physicaly(lgridj),-alpha)  "];
Write[stmp,"        physicalnx=(cos(-alpha) * physicalx(lgridi)) + (sin(-alpha)
*
physicaly(lgridj))"];
Write[stmp,"        physicalny=(-sin(-alpha) * physicalx(lgridi)) +
(cos(-alpha) * physicaly(lgridj))"];



Write[stmp,"       if (p(lgridi,lgridj,0,0).gt.maxp)
maxp=p(lgridi,lgridj,0,0)"];
Write[stmp,"       if (p(lgridi,lgridj,0,0).lt.minp)
minp=p(lgridi,lgridj,0,0)"];

Write[stmp,"
perr=abs(p(lgridi,lgridj,0,0)-",p[x,y,t]/.{x->physicalnx,y->physicalny,t->physic
alt},")"];
Write[stmp,"       if (perr.gt.maxperr) then "];
Write[stmp,"       bigi=lgridi "];
Write[stmp,"       bigj=lgridj "];
Write[stmp,"       maxperr=perr "];
Write[stmp,"       endif "];
Write[stmp,"       totalperr=totalperr+perr"];
(*
Write[stmp,"       do dy=0,degree "];
Write[stmp,"       do dx=0,degree "];
*)
Write[stmp,"       do dy=0,0 "];
Write[stmp,"       do dx=0,0 "];
Write[stmp,"       currentenergy=currentenergy+(p(lgridi,lgridj,dx,dy)**2 "];
Write[stmp,"     -+u(lgridi,lgridj,dx,dy)**2"];
Write[stmp,"     -+v(lgridi,lgridj,dx,dy)**2)"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];

Write[stmp,"       endif"];

Write[stmp,"       end do"];
Write[stmp,"       end do"];

Write[stmp,"       l1err=totalperr*h*h "];
Write[stmp,"       eratio = currentenergy*h*h/initialenergy "];
Write[stmp,
"       write(*,900) stepn+1, physicalt, maxperr, l1err, maxp, minp,eratio"];
Write[stmp,"       write(6,*) bigi,bigj "];
Write[stmp,
```

```
"       write(iun,900) stepn+1, physicalt, maxperr, l1err, maxp, minp,eratio"];
Write[stmp,"  900 format(1x,i5,1x,1p,4d12.5,1x,2d12.5)"];

Write[stmp,"       end"];
Close[stmp];
Run["sed -e 's/\"//g'  -e 's/\\\\//g' errorcalctmp.f >> errorcalcdown.f"];
Run["rm errorcalctmp.f"];


);
makeerrorcalcupfile:=(
Print["Removing errorcalcup.f"];
Run["rm errorcalcup.f"];
Print["Generating error calculation routine errorcalcup.f"];
Clear[p,u,v];
(*
(* ------------------------------------------- *)
(* Analytical solution to bi-periodic LEE problem *)
(* ------------------------------------------- *)
p[x_,y_,t_]:=Cos[Sqrt[wx^2+wy^2] pi t]*Sin[wx pi (x - mx t)]*Sin[wy pi (y - my
t)];
u[x_,y_,t_]:=-(wx/Sqrt[wx^2+wy^2])*Sin[Sqrt[wx^2+wy^2] pi t]*Cos[wx pi (x - mx
t)]*Sin[wy pi (y - my t)];
v[x_,y_,t_]:=-(wy/Sqrt[wx^2+wy^2])*Sin[Sqrt[wx^2+wy^2] pi t]*Sin[wx pi (x - mx
t)]*Cos[wy pi (y - my t)];
*)
(* ------------------------------------------- *)
(* Analytical solution to rotated box LEE problem *)
(* ------------------------------------------- *)
p[x_,y_,t_]:=-Cos[Sqrt[wx^2+wy^2] Pi t] Cos[wx Pi (x- mx t)] Cos[wy Pi (y-my
t)];

u[x_,y_,t_]:=-(wx/Sqrt[wx^2+wy^2])*Sin[Sqrt[wx^2+wy^2] Pi t]*Sin[wx Pi (x - mx
t)]*Cos[wy Pi (y - my t)];
v[x_,y_,t_]:=-(wy/Sqrt[wx^2+wy^2])*Sin[Sqrt[wx^2+wy^2] Pi t]*Cos[wx Pi (x - mx
t)]*Sin[wy Pi (y - my t)];



stmp=OpenWrite["errorcalctmp.f",FormatType->FortranForm];
Write[stmp,"       subroutine errorcalcup"];
Write[stmp,"       include 'common.h'"];
(*
Write[stmp,"       real rotatex"];
Write[stmp,"       real rotatey"];
*)

Write[stmp,"       integer bigi,bigj "];
Write[stmp,"       real eratio "];
Write[stmp,"       totalperr=0.0 "];
Write[stmp,"       maxperr=0.0 "];
```

```
Write[stmp,"       maxp=0.0 "];
Write[stmp,"       minp=0.0 "];
Write[stmp,"       currentenergy=0.0"];
Write[stmp,"       do lgridj=-maxmemj+stagger,maxmemj+stagger"];
Write[stmp,"       do lgridi=-maxmemi+stagger,maxmemi+stagger"];
Write[stmp,"       if
((interior(lgridi,lgridj).eq.1).or.(interior(lgridi,lgridj).eq.2)) then"];
Write[stmp,"c
physicalnx=rotatex(physicalx(lgridi),physicaly(lgridj),-alpha)  "];
Write[stmp,"c
physicalny=rotatey(physicalx(lgridi),physicaly(lgridj),-alpha)  "];
Write[stmp,"        physicalnx=(cos(-alpha) * physicalx(lgridi)) + (sin(-alpha)
*
physicaly(lgridj))"];
Write[stmp,"        physicalny=(-sin(-alpha) * physicalx(lgridi)) +
(cos(-alpha) * physicaly(lgridj))"];


Write[stmp,"       if (np(lgridi,lgridj,0,0).gt.maxp)
maxp=np(lgridi,lgridj,0,0)"];
Write[stmp,"       if (np(lgridi,lgridj,0,0).lt.minp)
minp=np(lgridi,lgridj,0,0)"];
Write[stmp,"
perr=abs(np(lgridi,lgridj,0,0)-",p[x,y,t]/.{x->physicalnx,y->physicalny,t->physi
calt},")"];
Write[stmp,"       if (perr.gt.maxperr) then "];
Write[stmp,"       bigi=lgridi "];
Write[stmp,"       bigj=lgridj "];
Write[stmp,"       maxperr=perr "];
Write[stmp,"       endif "];
Write[stmp,"       totalperr=totalperr+perr"];
(*
Write[stmp,"       do dy=0,degree"];
Write[stmp,"       do dx=0,degree"];
*)
Write[stmp,"       do dy=0,0"];
Write[stmp,"       do dx=0,0"];
Write[stmp,"       currentenergy=currentenergy+(np(lgridi,lgridj,dx,dy)**2 "];
Write[stmp,"     -+nu(lgridi,lgridj,dx,dy)**2"];
Write[stmp,"     -+nv(lgridi,lgridj,dx,dy)**2)"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];

Write[stmp,"       endif"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
Write[stmp,"       l1err=totalperr*h*h "];
Write[stmp,"       eratio = currentenergy*h*h/initialenergy "];
Write[stmp,
"       write(*,900) stepn, physicalt, maxperr, l1err, maxp, minp,eratio"];
Write[stmp,"       write(6,*) bigi,bigj "];
```

```
Write[stmp,
"       write(iun,900) stepn, physicalt, maxperr, l1err, maxp, minp,eratio"];
Write[stmp,"  900 format(1x,i5,1x,1p,4d12.5,1x,2d12.5)"];

Write[stmp,"       end"];
Close[stmp];
Run["sed -e 's/\"//g'  -e 's/\\\\//g'  errorcalctmp.f >> errorcalcup.f"];
Run["rm errorcalctmp.f"];


);

makemainfile:=(
Print["Removing main.f"];
Run["rm main.f"];
Print["Generating main execution routine main.f"];
If[Not[ValueQ[maxmemi]],
maxmemi=Input["Enter the maximum memory grid size in one dimension"];
maxmemj=maxmemi];

stmp=OpenWrite["maintmp.f",FormatType->FortranForm];

Write[stmp,"       program main "];
Write[stmp,"       include 'common.h'"];
Write[stmp,"c**** Enter problem parameters *****c"];
Write[stmp,"       write(6,*) 'Enter the number of i grid cells per unit
interval' "];
Write[stmp,"       read(5,*) iun"];
Write[stmp,"       if (iun+",csize,".gt.maxmemi) then "];
Write[stmp,"       write(6,*) 'Need to allocate more memory ' "];
Write[stmp,"       stop "];
Write[stmp,"       endif"];
Write[stmp,"c       maxi=maxmemi-2"];
Write[stmp,"c       maxj=maxmemj-2"];
Write[stmp,"       h=1.0/dble(iun)"];
Write[stmp,"       write(6,*) 'Enter lambda' "];
Write[stmp,"       read(5,*) lam"];

Write[stmp,"       physicaltstep=lam/dble(iun) "];

Write[stmp,"       write(6,*) 'Enter # of time steps ' "];
Write[stmp,"       read(5,*) numberofnsteps "];

Write[stmp,"       write(6,*) 'Enter # of time steps between error output ' "];
Write[stmp,"       read(5,*) errorsteps "];

Write[stmp,"       if (mod(csize,2).eq.0) then "];
Write[stmp,"       if (mod(errorsteps+1,2).eq.0) then "];
Write[stmp,"       write(*,*) 'Staggered grid requires even errorsteps' "];
Write[stmp,"       errorsteps=errorsteps+1 "];
Write[stmp,"       write(*,*) 'Changed it to :',errorsteps "];
```

```
Write[stmp,"       endif "];
Write[stmp,"       endif "];

Write[stmp,"       write(6,*) 'Enter wave number for wx :'"];
Write[stmp,"       read(5,*) wx "];
Write[stmp,"       write(6,*) 'Enter wave number for wy :'"];
Write[stmp,"       read(5,*) wy "];

Write[stmp,"       stagger=0"];
Write[stmp,"       physicalt=0.0"];
Write[stmp,"       mx=0.0"];
Write[stmp,"       my=0.0"];

realsizeinbytes=8;
memcost=  (4*iun + 24*(1 + degree)^2*iun^2 +
   csize*(1 + degree)*(1 + numberofcterms) +
   3*(2 + csize + csize*degree)^2*(-1 + 2*csize*(1 + degree)))*realsizeinbytes;
(*
tffloatcost= (iun+iun+1)^2 3 *  (-1 + csize + degree*csize)^2*(-1 + 2*(1 +
degree)*csize);
m=csize(degree+1)-1;
cefloatcost= (iun+iun+1)^2 3 * 10*(1 + degree)^2*csize^2*(-1 + csize +
degree*csize)+(3*2 m (degree+1)^2);
flocost=cefloatcost+tffloatcost;
*)
flocostperstep=(iun+iun+1)^2*(3*(-1 + csize + degree*csize)^2*(-1 + 2*(1 +
degree)*csize) + (1 + degree)^2*(5*csize^2*(-1 + csize + degree*csize) + 3*(-3
+ 4*(1 + degree)*csize)));
totalflocost=flocostperstep*numberofnsteps;

Write[stmp,"      write(*,*) '2D LEE Constant Coefficient Rotated Box Problem
'"];
Write[stmp,"      write(*,*) 'Algorithm :c",csize,"d",degree," ' "];
Write[stmp,"      write(*,*) 'Grid Points Per Wavelength = ',2*iun "];
Write[stmp,"      write(*,*) 'Mx Convection Velocity (Nondimensionalized) :
',mx"];
Write[stmp,"      write(*,*) 'My Convection Velocity (Nondimensionalized) :
',my"];
Write[stmp,"      write(*,*) 'Wx Wave Number : ',wx"];
Write[stmp,"      write(*,*) 'Wy Wave Number : ',wy"];
Write[stmp,"      write(*,*) 'Lambda dt/dx : ',lam"];
Write[stmp,"      write(*,*) 'Number of time steps : ',numberofnsteps"];
Write[stmp,"      write(*,*) 'Memory Cost is (in Bytes):',",memcost];
Write[stmp,"      write(*,*) 'Floating Point Cost Per Time Step is
:',",flocostperstep];
Write[stmp,"      write(*,*) 'Total Floating Point Cost Counting all steps is
:',",totalflocost];

Write[stmp,"      write(iun,*) '2D LEE Constant Coefficient Biperiodic B.C.
Problem '"];
Write[stmp,"      write(iun,*) 'Algorithm :c",csize,"d",degree," ' "];
```

```
Write[stmp,"      write(iun,*) 'Grid Points Per Wavelength = ',2*iun "];
Write[stmp,"      write(iun,*) 'Mx Convection Velocity (Nondimensionalized) :
',mx"];
Write[stmp,"      write(iun,*) 'My Convection Velocity (Nondimensionalized) :
',my"];
Write[stmp,"      write(iun,*) 'Wx Wave Number : ',wx"];
Write[stmp,"      write(iun,*) 'Wy Wave Number : ',wy"];
Write[stmp,"      write(iun,*) 'Lambda dt/dx : ',lam"];
Write[stmp,"      write(iun,*) 'Number of time steps : ',numberofnsteps"];
Write[stmp,"      write(iun,*) 'Memory Cost is (Bytes):',",memcost];
Write[stmp,"      write(iun,*) 'Floating Point Cost Per Time Step is
:',",flocostperstep];
Write[stmp,"      write(iun,*) 'Total Floating Point Cost Counting all steps is
:',",totalflocost];
Write[stmp,"       call readfills "];
Write[stmp,"       call definephysicalxy "];
Write[stmp,"       call determinefills "];
Write[stmp,"       call initcond "];
Write[stmp,"       call computefactorials "];
Write[stmp,"      write(*,*) ' ' "];
Write[stmp,"      write(*,*) ",
"'    n  t          maxperr     l1perr     phmax",
"       phmin        energy'"];
Write[stmp,"      write(*,*) ' '"];
Write[stmp,"      write(iun,*) ' '"];
Write[stmp,"      write(iun,*) ",
"'    n  t          maxperr     l1perr     phmax",
"       phmin        energy'"];
Write[stmp,"      write(iun,*) ' ' "];

Write[stmp,"c       call periodicexdown "];
Write[stmp,"       stepn=-1"];
Write[stmp,"        call errorcalcdown "];

Write[stmp,"       do stepn=1,numberofnsteps,2 "];
Write[stmp,"       physicalt=physicalt+physicaltstep "];
If[EvenQ[csize],
Write[stmp,"       stagger=1 "];
];

Write[stmp,"       do gridj=-maxmemj+2,maxmemj-2"];
Write[stmp,"       do gridi=-maxmemi+2,maxmemi-2"];


Write[stmp,"       call tensorup "];
Write[stmp,"       call timeadvanceup "];

Write[stmp,"       end do"];
Write[stmp,"       end do"];

Write[stmp,"c       call periodicexup "];
```

```
(* ------------------------------------------------------------ *)
(* Odd stencils need fill done on each time step, top and bottom *)
(* ------------------------------------------------------------ *)
If[OddQ[csize],
Write[stmp,"       call fillfillsup"]
];
Write[stmp,"       if (mod(stepn,errorsteps).eq.0) call errorcalcup"];
If[EvenQ[csize],
Write[stmp,"       stagger=0 "];
];


Write[stmp,"       physicalt=physicalt+physicaltstep "];

Write[stmp,"       do gridj=-maxmemj+2,maxmemj-2"];
Write[stmp,"       do gridi=-maxmemi+2,maxmemi-2"];

Write[stmp,"       call tensordown "];
Write[stmp,"       call timeadvancedown "];


Write[stmp,"       end do"];
Write[stmp,"       end do"];

Write[stmp,"c       call periodicexdown "];
Write[stmp,"       call fillfillsdown "];
Write[stmp,"       if (mod(stepn+1,errorsteps).eq.0) call errorcalcdown"];
Write[stmp,"       end do"];

Write[stmp,"       end"];

Close[stmp];

Run["sed -e 's/\"//g' -e 's/\\\\//g' maintmp.f >> main.f"];
Run["rm maintmp.f"];


);

makeperiodicexfile:=(
Print["Removing periodicex.f"];
Run["rm periodicexdown.f"];
Run["rm periodicexup.f"];
Print["Generating periodic boundary exchange routine periodicex.f"];

If[Not[ValueQ[csize]],
csize=Input["Enter a stencil size"];
];

If[Not[ValueQ[degree]],
degree=Input["Enter the degree"];
```

```
];

stmp=OpenWrite["periodicextmp.f",FormatType->FortranForm];

Write[stmp,"       subroutine periodicex"];
Write[stmp,"       include 'common.h'"];

Write[stmp,"c***** Exchange Right and Left *****************"];
varlist={"p","u","v"};
Do[
thevar=varlist[[ct]];
(*
Do[Do[
*)
dnotation=ToString[dx]<>"x"<>ToString[dy]<>"y";
Write[stmp,"       do dy=0,degree"];
Write[stmp,"       do dx=0,degree"];
Write[stmp,"       do ioffset=0,",IntegerPart[csize/2]];
Write[stmp,"       do lgridj=-maxj+1,maxj"];
Write[stmp,"          ",thevar,"(-maxi-ioffset,lgridj,dx,dy)="];
Write[stmp,"     *    ",thevar,"( maxi-ioffset,lgridj,dx,dy)  "];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
Write[stmp,"       do ioffset=0,",IntegerPart[csize/2]];
Write[stmp,"       do lgridj=-maxj+1,maxj"];
Write[stmp,"          ",thevar,"(maxi+1+ioffset,lgridj,dx,dy)="];
Write[stmp,"     *    ",thevar,"(1-maxi+ioffset,lgridj,dx,dy)"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
(*
,{dx,0,degree}]
,{dy,0,degree}];
*)
,{ct,1,3}];

Write[stmp,"c***** Exchange Top and Bottom *****************"];
Do[
thevar=varlist[[ct]];
(*
Do[Do[
*)
dnotation=ToString[dx]<>"x"<>ToString[dy]<>"y";
Write[stmp,"       do dy=0,degree"];
Write[stmp,"       do dx=0,degree"];
Write[stmp,"       do joffset=0,",IntegerPart[csize/2]];
Write[stmp,"       do lgridi=-maxi+1,maxi"];
Write[stmp,"          ",thevar,"(lgridi,-maxj-joffset,dx,dy)="];
Write[stmp,"     *    ",thevar,"(lgridi, maxj-joffset,dx,dy)  "];
Write[stmp,"       end do"];
```

```
Write[stmp,"       end do"];
Write[stmp,"       do joffset=0,",IntegerPart[csize/2]];
Write[stmp,"       do lgridi=-maxi+1,maxi"];
Write[stmp,"          ",thevar,"(lgridi,maxj+1+joffset,dx,dy)="];
Write[stmp,"     *    ",thevar,"(lgridi,1-maxj+joffset,dx,dy)"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
(*
,{dx,0,degree}]
,{dy,0,degree}];
*)
Write[stmp,"       end do"];
Write[stmp,"       end do"];
,{ct,1,3}];

Write[stmp,"c***** Exchange Corners *****************"];
Do[
thevar=varlist[[ct]];
(*
Do[Do[
*)
dnotation=ToString[dx]<>"x"<>ToString[dy]<>"y";
Write[stmp,"       do dy=0,degree"];
Write[stmp,"       do dx=0,degree"];
Write[stmp,"c****** Top Right ***********c "];
Write[stmp,"       do joffset=0,",IntegerPart[csize/2]];
Write[stmp,"       do ioffset=0,",IntegerPart[csize/2]];
Write[stmp,"          ",thevar,"(maxi+1+ioffset,maxj+1+joffset,dx,dy)="];
Write[stmp,"     *    ",thevar,"(1-maxi+ioffset,1-maxj+joffset,dx,dy)  "];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
Write[stmp,"c****** Top Left ***********c "];
Write[stmp,"       do joffset=0,",IntegerPart[csize/2]];
Write[stmp,"       do ioffset=0,",IntegerPart[csize/2]];
Write[stmp,"          ",thevar,"(-maxi-ioffset,maxj+1+joffset,dx,dy)="];
Write[stmp,"     *    ",thevar,"( maxi-ioffset,1-maxj+joffset,dx,dy)"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
Write[stmp,"c****** Bottom Right ***********c "];
Write[stmp,"       do joffset=0,",IntegerPart[csize/2]];
Write[stmp,"       do ioffset=0,",IntegerPart[csize/2]];
Write[stmp,"          ",thevar,"(maxi+1+ioffset,-maxj-joffset,dx,dy)="];
Write[stmp,"     *    ",thevar,"(1-maxi+ioffset, maxj-joffset,dx,dy)"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];
Write[stmp,"c****** Bottom Left ***********c "];
Write[stmp,"       do joffset=0,",IntegerPart[csize/2]];
Write[stmp,"       do ioffset=0,",IntegerPart[csize/2]];
Write[stmp,"          ",thevar,"(-maxi-ioffset,-maxj-joffset,dx,dy)="];
Write[stmp,"     *    ",thevar,"( maxi-ioffset, maxj-joffset,dx,dy)"];
Write[stmp,"       end do"];
```

```
Write[stmp,"       end do"];
(*
,{dx,0,degree}]
,{dy,0,degree}];
*)
Write[stmp,"       end do"];
Write[stmp,"       end do"];
,{ct,1,3}];

Write[stmp,"       end"];

Close[stmp];

Run["sed -e 's/\"//g'  -e 's/\\\\//g' -e 's/periodicex/periodicexdown/g'
periodicextmp.f >> periodicexdown.f"];
Run["sed -e 's/\"//g'  -e 's/\\\\//g' -e 's/periodicex/periodicexup/g' -e 's/
p/  np/g' -e 's/  u/  nu/g' -e 's/  v/  nv/g' periodicextmp.f >>
periodicexup.f"];
Run["rm periodicextmp.f"];

);

(* ---------------------------------------------------------------- *)
(* This makes the routine that fills in the fills at each time step.     *)
(* Note that Hermitian schemes only fill on the bottom as the staggered  *)
(* step will have the information it needs                               *)
(* ---------------------------------------------------------------- *)
makefillfillsfile:=(
Print["Removing fillfills.f"];
Run["rm fillfillsdown.f"];
Run["rm fillfillsup.f"];
Print["Generating fill the fill points routine fillfills.f"];


stmp=OpenWrite["fillfillstmp.f",FormatType->FortranForm];

Write[stmp,"       subroutine fillfills"];
Write[stmp,"       include 'common.h'"];

varlist={"pp","uu","vv"};
Write[stmp,"c*************************************************************c"];
Write[stmp,"c* Loop through all fill locations in correct order       *c"];
Write[stmp,"c*************************************************************c"];
Write[stmp,"       idx=1 "];
Write[stmp,"       do fillct=1, numberoffillpts "];
Write[stmp,"       filli = onelongarray(idx)"];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       fillj = onelongarray(idx)"];
Write[stmp,"       idx=idx+1"];
Do[
Write[stmp,"c*************************************************************c"];
```

```
Write[stmp,"c* Loop through all the derivatives of ",varlist[[ct]],"      *c"];
Write[stmp,"c* At a single location                                       *c"];
Write[stmp,"c*************************************************************c"];
Write[stmp,"       do filldy=0, degree "];
Write[stmp,"       do filldx=0, degree "];
Write[stmp,"       numberofp=onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       numberofu=onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       numberofv=onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       psum=0.0"];
Write[stmp,"       do ipct=1,numberofp "];
Write[stmp,"       lci = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       lcj = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       lcdx = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       lcdy = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       lccoef = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       psum=psum+(lccoef*pp(lci,lcj,lcdx,lcdy))"];
Write[stmp,"       if ((filli.eq.4).and.(fillj.eq.-2)) then "];
Write[stmp,"       write(6,*)
'np',lccoef,lci,lcj,lcdx,lcdy,np(lci,lcj,lcdx,lcdy)"];
Write[stmp,"       endif"];
Write[stmp,"       end do "];
Write[stmp,"       usum=0.0"];
Write[stmp,"       do iuct=1,numberofu "];
Write[stmp,"       lci = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       lcj = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       lcdx = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       lcdy = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       lccoef = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       usum=usum+(lccoef*uu(lci,lcj,lcdx,lcdy))"];
Write[stmp,"       if ((filli.eq.4).and.(fillj.eq.-2)) then "];
Write[stmp,"       write(6,*)
'nu',lccoef,lci,lcj,lcdx,lcdy,nu(lci,lcj,lcdx,lcdy)"];
Write[stmp,"       endif"];
Write[stmp,"       end do "];
Write[stmp,"       vsum=0.0"];
Write[stmp,"       do ivct=1,numberofv "];
Write[stmp,"       lci = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
```

```
Write[stmp,"       lcj = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       lcdx = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       lcdy = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       lccoef = onelongarray(idx) "];
Write[stmp,"       idx=idx+1"];
Write[stmp,"       vsum=vsum+(lccoef*vv(lci,lcj,lcdx,lcdy))"];
Write[stmp,"       if ((filli.eq.4).and.(fillj.eq.-2)) then "];
Write[stmp,"       write(6,*)
'nv',lccoef,lci,lcj,lcdx,lcdy,nv(lci,lcj,lcdx,lcdy)"];
Write[stmp,"       endif"];
Write[stmp,"       end do "];
Write[stmp,"
",varlist[[ct]],"(filli,fillj,filldx,filldy)=psum+usum+vsum"];
Write[stmp,"       if ((filli.eq.4).and.(fillj.eq.-2)) then "];
Write[stmp,"       write(6,*) 'assigning
',filli,fillj,filldx,filldy,psum,usum,vsum"];
Write[stmp,"       endif"];
Write[stmp,"       end do "];
Write[stmp,"       end do "];
,{ct,1,Length[varlist]}];
Write[stmp,"       end do "];
Write[stmp,"       end"];

Close[stmp];

Run["sed -e 's/\"//g'  -e 's/\\\\//g' -e 's/fillfills/fillfillsdown/g' -e
's/pp/ p/g' -e 's/uu/ u/g' -e 's/vv/ v/g' fillfillstmp.f >> fillfillsdown.f"];
Run["sed -e 's/\"//g'  -e 's/\\\\//g' -e 's/fillfills/fillfillsup/g' -e 's/pp/
np/g' -e 's/uu/ nu/g' -e 's/vv/ nv/g' fillfillstmp.f >> fillfillsup.f"];
Run["rm fillfillstmp.f"];

);

makereadfillsfile:=(
(*
If[Not[ValueQ[maxlength]],
maxlength=Input["Enter the maximum length of fillarray: "];
];
*)
Print["Removing readfills.f"];
Run["rm readfills.f"];
Print["Generating read the fill points definition array routine readfills.f"];

stmp=OpenWrite["readfillstmp.f",FormatType->FortranForm];

Write[stmp,"       subroutine readfills"];
Write[stmp,"       include 'common.h'"];
```

```
Write[stmp,"       open(file='fillarrays',unit=3)"];
Write[stmp,"c*********************************************************************
******"];
Write[stmp,"c* Read in the fill solutions from the mathematica output of file"];
Write[stmp,"c* computefillarrays"];
Write[stmp,"c*********************************************************************
******"];

Write[stmp,"c*************************************************************c"];
Write[stmp,"c* The first element of fillarrays is the # of elements  *c"];
Write[stmp,"c* The second is the # of fill point locations           *c"];
Write[stmp,"c*************************************************************c"];
Write[stmp,"       read(3,*) lengthoffillarrays "];
Write[stmp,"       read(3,*) numberoffillpts "];
Write[stmp,"       write(6,*) 'Number of fill points is ',numberoffillpts "];
Write[stmp,"       if (lengthoffillarrays.lt.maxlength) then "];
Write[stmp,"c*********************************************************************
c"];
Write[stmp,"c* The minus one is because the second number in fillarrays
*c"];
Write[stmp,"c* is the number of fill points
*c"];
Write[stmp,"c*********************************************************************
c"];
Write[stmp,"       do i=1,lengthoffillarrays-1"];
Write[stmp,"       read(3,*) onelongarray(i) "];
Write[stmp,"       end do "];
Write[stmp,"       else "];
Write[stmp,"       write(6,*) 'The fillarrays file is too large ' "];
Write[stmp,"       write(6,*) 'Increase maxlength from ',maxlength,' to
',lengthoffillarrays+1 "];
Write[stmp,"       stop "];
Write[stmp,"       endif"];
Write[stmp,"       close(3)"];
Write[stmp,"       end"];

Close[stmp];

Run["sed -e 's/\"//g'  -e 's/\\\\//g'  readfillstmp.f >> readfills.f"];
Run["rm readfillstmp.f"];

);

makedeterminefillsfile:=(
Print["Removing determinefills.f"];
Run["rm determinefills.f"];
Print["Generating determine fill points routine determinefills.f"];


stmp=OpenWrite["determinefillstmp.f",FormatType->FortranForm];
```

```
Write[stmp,"       subroutine determinefills"];
Write[stmp,"       include 'common.h'"];

Write[stmp,"c****"];
Write[stmp,"c*Determine which points are interior, fillin, and nothing"];
Write[stmp,"c* Note that boundary points are nothing, not used since assumed
0"];
Write[stmp,"c* 0 = boundary  , 1 = interior,  2 = fillin needed,  3 = fillin
not needed"];
Write[stmp,"c****"];

Write[stmp,"       do lgridi=-maxmemi,maxmemi"];
Write[stmp,"       do lgridj=-maxmemj,maxmemj"];
Write[stmp,"c
physicalnx=rotatex(physicalx(lgridi),physicaly(lgridj),-alpha) "];
Write[stmp,"c
physicalny=rotatey(physicalx(lgridi),physicaly(lgridj),-alpha) "];
Write[stmp,"        physicalnx=(cos(-alpha) * physicalx(lgridi)) + (sin(-alpha)
*
physicaly(lgridj))"];
Write[stmp,"        physicalny=(-sin(-alpha) * physicalx(lgridi)) +
(cos(-alpha) * physicaly(lgridj))"];


Write[stmp,"c* interior (not fillin though)"];
Write[stmp,"        if
((physicalnx.gt.-1).and.(physicalnx.lt.1).and.(physicalny.gt.-1).and.(physicalny
.lt.1)) then"];
Write[stmp,"        interior(lgridi,lgridj)=1 "];
Write[stmp,"        else"];
Write[stmp,"c* outside of rotated box or on boundary"];
Write[stmp,"        interior(lgridi,lgridj)=0"];
Write[stmp,"        endif"];

Write[stmp,"        end do"];
Write[stmp,"        end do"];

Write[stmp,"c****"];
Write[stmp,"c* now determine the interior points which need filled in"];
Write[stmp,"c* Note that this will work for the hermitain c3ons2 algorithms
too."];
Write[stmp,"c****"];
Write[stmp,"c* Do not need to compute outer most square of points, they are 0"];
Write[stmp,"        do lgridi=-maxmemi+1,maxmemi-1"];
Write[stmp,"        do lgridj=-maxmemj+1,maxmemj-1"];
Write[stmp,"        if (interior(lgridi,lgridj).eq.1) then "];
Write[stmp,"        interior(lgridi,lgridj)=2"];
Write[stmp,"        if ((interior(lgridi-1,  lgridj).ne.0).and."];
Write[stmp,"     -      (interior(lgridi+1,  lgridj).ne.0).and."];
Write[stmp,"     -      (interior(lgridi-1,lgridj+1).ne.0).and."];
Write[stmp,"     -      (interior(lgridi+1,lgridj+1).ne.0).and."];
```

```
Write[stmp,"     -      (interior(lgridi-1,lgridj-1).ne.0).and."];
Write[stmp,"     -      (interior(lgridi+1,lgridj-1).ne.0).and."];
Write[stmp,"     -      (interior(lgridi  ,lgridj+1).ne.0).and."];
Write[stmp,"     -      (interior(lgridi  ,lgridj-1).ne.0)) then"];
Write[stmp,"        interior(lgridi,lgridj)=1 "];
Write[stmp,"        endif"];
Write[stmp,"        endif"];
Write[stmp,"       end do"];
Write[stmp,"       end do"];

Write[stmp,"c* Now determine which fill ins are not used or needed"];
Write[stmp,"c* they are not next to an interior point"];
Write[stmp," "];
Write[stmp,"        do lgridi=-maxmemi+1,maxmemi-1"];
Write[stmp,"        do lgridj=-maxmemj+1,maxmemj-1"];
Write[stmp,"        if ((interior(lgridi,lgridj).eq.2).and."];
Write[stmp,"     -   (interior(lgridi-1,  lgridj).ne.1).and."];
Write[stmp,"     -   (interior(lgridi+1,  lgridj).ne.1).and."];
Write[stmp,"     -   (interior(lgridi-1,lgridj+1).ne.1).and."];
Write[stmp,"     -   (interior(lgridi+1,lgridj+1).ne.1).and."];
Write[stmp,"     -   (interior(lgridi-1,lgridj-1).ne.1).and."];
Write[stmp,"     -   (interior(lgridi+1,lgridj-1).ne.1).and."];
Write[stmp,"     -   (interior(lgridi  ,lgridj+1).ne.1).and."];
Write[stmp,"     -   (interior(lgridi,  lgridj-1).ne.1)) then"];
Write[stmp,"        interior(lgridi,lgridj)=3"];
Write[stmp,"        endif"];

Write[stmp,"        end do"];
Write[stmp,"        end do"];

Write[stmp,"c*************************************************************c"];
Write[stmp,"c* output the grid definition for mathematica to a file *c"];
Write[stmp,"c*************************************************************c"];

Write[stmp,"        write(4,*) maxmemi "];
Write[stmp,"        write(4,*) iun "];
Write[stmp,"        do lgridi=-maxmemi,maxmemi "];
Write[stmp,"        do lgridj= maxmemj,-maxmemj,-1"];
Write[stmp,"        write(4,*) interior(lgridi,lgridj)"];
Write[stmp,"        end do"];
Write[stmp,"        end do"];



Write[stmp,"       end"];

Close[stmp];

Run["sed -e 's/\"//g'  -e 's/\\\\//g'  determinefillstmp.f >>
determinefills.f"];
Run["rm determinefillstmp.f"];
```

```
);

makerotatefile:=(
Print["Removing rotate.f"];
Run["rm rotate.f"];
Print["Generating rotate routine rotate.f"];


stmp=OpenWrite["rotatetmp.f",FormatType->FortranForm];

Write[stmp,"        real function rotatex(xcoord,ycoord,lalpha)"];
Write[stmp,"        real xcoord,ycoord,lalpha,newalpha"];
Write[stmp,""];
Write[stmp,"c* Rotation in MMA is backwards"];
Write[stmp,"        newalpha=lalpha"];
Write[stmp,"        rotatex=(cos(newalpha) * xcoord) + (sin(newalpha) *
ycoord)"];
Write[stmp,""];
Write[stmp,"        end"];
Write[stmp,""];

Write[stmp,"        real function rotatey(xcoord,ycoord,lalpha)"];
Write[stmp,"        real xcoord,ycoord,lalpha,newalpha"];
Write[stmp,""];
Write[stmp,"c* Rotation in MMA is backwards"];
Write[stmp,"        newalpha=lalpha"];
Write[stmp,"        rotatey=(-sin(newalpha) * xcoord) + (cos(newalpha) *
ycoord)"];
Write[stmp,""];


Write[stmp,"       end"];

Close[stmp];

Run["sed -e 's/\"//g'  -e 's/\\\\//g'  rotatetmp.f >> rotate.f"];
Run["rm rotatetmp.f"];

);


makedefinephysicalxyfile:=(
Print["Removing rotate.f"];
Run["rm definephysicalxy.f"];
Print["Generating rotate routine rotate.f"];


stmp=OpenWrite["definephysicalxytmp.f",FormatType->FortranForm];

Write[stmp,"        subroutine definephysicalxy"];
```

```
Write[stmp,"        include 'common.h'"];
Write[stmp,""];
Write[stmp,"c* Define the physical coordinates in terms of integer indeces"];

Write[stmp,"       do lgridi=-",maxmemi,",",maxmemi];
Write[stmp,"       physicalx(lgridi)= lgridi * h"];
Write[stmp,"       end do"];
Write[stmp,"       do lgridj=-",maxmemj,",",maxmemj];
Write[stmp,"       physicaly(lgridj)= lgridj * h"];
Write[stmp,"       end do"];
Write[stmp,"       end"];

Close[stmp];

Run["sed -e 's/\"//g'  -e 's/\\\\//g'  definephysicalxytmp.f >>
definephysicalxy.f"];
Run["rm definephysicalxytmp.f"];

);
```

B.6 Wall Boundary Calculation File – ma2d

This code will produce the fill file described in section 5.5.

```
(* ---------------------- *)
(* Do the rotated box case *)
(* ---------------------- *)
(* ------------------------------------------- *)
(* Use memory constrain later for larger objects *)
(* ------------------------------------------- *)
(*
<< Utilities`MemoryConserve`
*)
<< Geometry`Rotations`
<< LinearAlgebra`MatrixManipulation`
<< NumericalMath`Horner`
<< Graphics`PlotField`
<< Utilities`BinaryFiles`
startma:= (
Clear[makeequal];
If[Not[ValueQ[csize]],
csize=Input["Enter the stencil size: "];
];

If[Not[ValueQ[degree]],
degree=Input["Enter the degree: "];
];

correctfillordering={};
johnlist7={};
johnlist8={};
topleftlist={};
If[Not[ValueQ[readgrid]],
readgrid=Input["Do you wish to read a grid definition file (1=Yes,0=No)"];
];

(* ------------------------------------------------------------ *)
(* Either read grid definition file or create one in mathematica *)
(* ------------------------------------------------------------ *)
If[readgrid==0,
buildcurves;
buildgrid,
buildcurves;
Print["Reading Grid File"];
readgridproc;
Print["Drawing Grid"];
drawgrid
];

Print["Calculating the 2D Hermitian Polynomial based Boundary Conditions"];
buildequationsforafillpoint;
```

```
Print["Defining Arrows"];
definearrows2;

Clear[x,y];

Print["Draw Entire Graph"];
drawentiregraph;

Print["Making fill arrays"];
makefillarrays;

Clear[x,y,i,j,xc,yc,xf,yf];
initproc;
getintformx;
getintformy;
Clear[a,b,c];
ncp[a_,b_,c_]:=(-(b+1)(my cp[a,b+1,c-1] + cv[a,b+1,c-1]) -(a+1) (mx
cp[a+1,b,c-1] + cu[a+1,b,c-1]))/c;
ncu[a_,b_,c_]:=(-(b+1) my cu[a,b+1,c-1] - (a+1) ( mx cu[a+1,b,c-1] +
cp[a+1,b,c-1] ))/c;
ncv[a_,b_,c_]:=(-(b+1) ( my cv[a,b+1,c-1] + cp[a,b+1,c-1] ) -(a+1) mx
cv[a+1,b,c-1])/c;

(*
initializetimestepping;
dothetimestepping;
*)

);


initializetimestepping:=(
physicaltime=0.0;
stepnumber=0;
mx=0;
my=0;
stagger=0;
(*
assigninitialdata;
*)
assigninitialdata2;
Print["Starting time advance "];
Print["N,   t,   maxperr    l1perr     phmax     phmin     eratio"];
showerror;
lam=.2;
timestep=lam*deltax;
);

(* ------------------------------------------------ *)
(* This loop performs the timestepping of the solution *)
```

```
(* ------------------------------------------------ *)

dothetimestepping:=(
While[physicaltime<=1.0,
(*
While[stepnumber<=1,
*)
(* Up Step *)
If[EvenQ[csize],
stagger=1];
dotimeadvance;
(* Do not do fill in on staggered grid for even stencils *)
If[OddQ[csize],
assignfillins;
];
physicaltime=physicaltime+timestep;
stepnumber=stepnumber+1;
(* ------------------------------------------------------------------- *)
(* The stagger step should not be displayed, but could be later if desired *)
(* ------------------------------------------------------------------- *)
If[OddQ[csize],
showerror;
];
(* Down Step *)
If[EvenQ[csize],
stagger=0];
dotimeadvance;
assignfillins;
physicaltime=physicaltime+timestep;
stepnumber=stepnumber+1;
showerror;
];
);


(* ------------------------------------------------------------------- *)
(* This procedure will create a list of each fill data type            *)
(*                                        (p,u,v,px,pxx,pxy,ux,... etc.)   *)
(* Each list will contain information to be read by the FORTRAN code to   *)
(* compute its fill pts                                                *)
(* using interior known grid points only.                              *)
(* ------------------------------------------------------------------- *)

makefillarrays := (

(* ------------------------------------------------------------------- *)
(* get the list of fill positions - those not needed in terms of matrix   *)
(* coordinates                                                         *)
(* ------------------------------------------------------------------- *)

fillposlist=correctfillordering;
```

```
Print["Total number of needed fill points is ",Length[fillposlist]];
(* ------------------------------------------------ *)
(* Convert the matrix coordinates to grid coordinates *)
(* ------------------------------------------------ *)
Clear[matrixi,matrixj];
fillgridposlist=fillposlist /. {{matrixi:_,matrixj:_} -> {matrixj - im -
1,-matrixi + im + 1}};
(*
fillgridposlistFORTRAN=Flatten[fillgridposlist];
*)

(* ------------------------------------------------------------------------ *)
(* Create an array for each data type using the same ordering as fillposlist *)
(* ------------------------------------------------------------------------ *)

(*
dataPerGridPoint=3;
*)
numberofprimtypes=3;
onelongarray={};
alldatalists={};
(* ------------------------------------------------ *)
(* Loop through all needed fills , creating a list  *)
(* ------------------------------------------------ *)
Do[
gridi=fillgridposlist[[fillptct]][[1]];
gridj=fillgridposlist[[fillptct]][[2]];
packetlist={gridi,gridj};
Print["starting with packet at ",gridi,gridj];
packetlistFORTRAN={};
(* ------------------------------------------------------------------------ *)
(* loop through all the data types at a single fill/grid point,             *)
(* a list for each                                                          *)
(* ------------------------------------------------------------------------ *)
Do[Do[ Do[

matrixi=fillposlist[[fillptct]][[1]];
matrixj=fillposlist[[fillptct]][[2]];
variablelist=Variables[fillsolutiongrid[[matrixi,matrixj,primtypect,dx+1,dy+1]]]
;
(* ----------------------------------------------------------*)
(* Determine # of p data, u data, and v data for onelongarray *)
(* ---------------------------------------------------------- *)
datacases={};
AppendTo[datacases,Cases[variablelist, pressure[_,_,_,_]]];
AppendTo[datacases,Cases[variablelist,uvelocity[_,_,_,_]]];
AppendTo[datacases,Cases[variablelist,vvelocity[_,_,_,_]]];

(* ---------------- *)
(* Start the packet *)
```

```
(* ---------------- *)
(* ------------ *)
(*
packetlist={Length[datacases]};
packetlist={};
*)
Do[
(* --------------------------- *)
(* Insert # of p, u and v data  *)
(* --------------------------- *)
AppendTo[packetlist,Length[datacases[[casect]]]];
,{casect,1,Length[datacases]}];

(* ---------------------------- *)
(* Add the data points to packet *)
(* ---------------------------- *)
(* Loop though all variable types *)
(* ------------------------------ *)
Do[
(* ------------------------------------ *)
(* Loop through all variables of type ct *)
(* ------------------------------------ *)
Do[
stencilpt=datacases[[casect]][[ct]];
Clear[mi,mj,mdx,mdy];
(* ---------------------------------------------- *)
(* Get the matrix coordinates of the data element *)
(* ---------------------------------------------- *)
matrixitemp=stencilpt /.{ _[mi:_,_,_,_] -> mi };
matrixjtemp=stencilpt /.{ _[_,mj:_,_,_] -> mj };

(* ------------------------------------ *)
(* Get the Derivatives of the data element *)
(* ------------------------------------ *)
ldx=stencilpt /.{ _[_,_,mdx:_,_] -> mdx };
ldy=stencilpt /.{ _[_,_,_,mdy:_] -> mdy };

(* ---------------------------- *)
(* Convert it to grid coordinates *)
(* ---------------------------- *)
gridi= matrixjtemp - im - 1;
gridj=-matrixitemp + im + 1;

(* ------------------------- *)
(* For now use no derivatives *)
(* ------------------------- *)
coef=FortranForm[Coefficient[fillsolutiongrid[[matrixi,matrixj,primtypect,dx+1,d
y+1]],stencilpt]];
(* ---------------------------------- *)
(* Use ptdata={gridi,gridj,ldx,ldy,coef}; later *)
(* ---------------------------------- *)
```

```
ptdata={gridi,gridj,ldx,ldy,coef};
(*
ptdata={gridi,gridj,coef};
*)
AppendTo[packetlist,ptdata];
(* ------------------------------------------------ *)
(* # of p or u or v data elements for this fill point *)
(* ------------------------------------------------ *)
(* Add loop for dx, and dy data *)
,{ct,1,Length[datacases[[casect]]]}];
(* ------------------------------------------------ *)
(* Will be 3 for p,u,v in 2D and 4 for p,u,v,w in 3D *)
(* ------------------------------------------------ *)
,{casect,1,Length[datacases]}];

(* ------------------------------------------------ *)
(* Now have a complete packet for current fill point *)
(* ------------------------------------------------ *)
(* Add the packet to the list of other packets for this data type list *)
(* ---------------------------------------------------------------- *)

AppendTo[packetlistFORTRAN,Flatten[packetlist]];

(* packetlistFORTRAN has  *)
(*{fillptlocx,fillptlocy,#ofp,#ofu,#ofv,pi,pj,pdx,pdy,pcoef,...,
                                       ui,uj,udx,udy,ucoef,...,
                                       vi,vj,vdx,vdy,vcoef,...,
                          #ofp,#ofu,#ofv,pi,pj,pdx,pdy,pcoef,...,
                                       ui,uj,udx,udy,ucoef,...,
                                       vi,vj,vdx,vdy,vcoef,...,
                     ... Until all data elements at fill point are
done...
   fillptlocx,fillptlocy,#ofp,#ofu,#ofv,pi,pj,pdx,pdy,pcoef,...,
                              ... etc.
                    until all fill points are defined ...
*)

(* ---------------------------------------------------------------------- *)
(* packetlistFORTRAN contains a list of packets.  Each packet corresponds to *)
(* a fill point.  All packets are for a single data element type however.    *)
(* This process is repeated for all the data elements that are at grid point *)
(* Each packet provides a linear combination of the data required to compute *)
(* single data element at a single fill point location                      *)
(* ---------------------------------------------------------------------- *)

,{dx,0,degree}]
,{dy,0,degree}]
,{primtypect,1,numberofprimtypes}];

AppendTo[onelongarray,packetlist];
```

```
,{fillptct,1,Length[fillposlist]}];

onelongarray=Flatten[onelongarray];

PrependTo[onelongarray,Length[fillposlist]];
PrependTo[onelongarray,Length[onelongarray]];
maxlength=Length[onelongarray];

(*
stmp=OpenWriteBinary["fillarrays.bin",FormatType->FortranForm];
*)
stmp=OpenWrite["fillarrays"];
Do[
(*
Write[stmp,onelongarray];
*)
Write[stmp,onelongarray[[ct]]];
,{ct,1,Length[onelongarray]}];
Close[stmp];

);

(* ------------------------------------------------------------------------ *)
(* This procedure will use the values of the interior points to determine  *)
(* the value of the  fill in points.                                        *)
(* ------------------------------------------------------------------------ *)
assignfillins := (
(* ------------------------------------------------------------------------------ *)
(* Adjust the grid indexing with 0,0 in center to matrix indexing with 1,1 in *)
(* top left                                                                       *)
(* ------------------------------------------------------------------------------ *)
correctedi[thisj_]:=im-thisj+1;
correctedj[thisi_]:=im+thisi+1;
Do[
(*
matrixicoord=correctedi[jct];
matrixjcoord=correctedj[ict];
*)
matrixicoord=correctfillordering[[fillptct]][[1]];
matrixjcoord=correctfillordering[[fillptct]][[2]];
(* ------------------------------------------------------------------------ *)
(* If this location is a fillin and its needed, then fill it in with data *)
(* ------------------------------------------------------------------------ *)
If[thegrid[[matrixicoord,matrixjcoord]]==2,
If[Not[MemberQ[ignorelist,{matrixicoord,matrixjcoord}]],
(* ------------------------- *)
(* For now use no derivatives *)
(* ------------------------- *)
(*
ldx=0; ldy=0;
*)
```

```
Do[Do[
prelation=fillsolutiongrid[[matrixicoord,matrixjcoord,1,ldx+1,ldy+1]];
urelation=fillsolutiongrid[[matrixicoord,matrixjcoord,2,ldx+1,ldy+1]];
vrelation=fillsolutiongrid[[matrixicoord,matrixjcoord,3,ldx+1,ldy+1]];
(* ------------------------------------------------------------- *)
(* Assign the fill in using the equation stored in fillsolutiongrid *)
(* ------------------------------------------------------------- *)
Clear[a1,b1,c1,d1,e1,f1,g1,i1,j1,k1];
Clear[aa1,bb1,cc1,dd1,ee1,ff1,gg1,ii1,jj1,kk1];
rhs1=(prelation /. {
pressure[a1:_Integer,b1:_Integer,aa1:_Integer,bb1:_Integer] ->
pressuregrid[[a1,b1,aa1+1,bb1+1]]});
Clear[a1,b1,c1,d1,e1,f1,g1,i1,j1,k1];
Clear[aa1,bb1,cc1,dd1,ee1,ff1,gg1,ii1,jj1,kk1];
rhs2=(urelation /.
{uvelocity[c1:_Integer,d1:_Integer,cc1:_Integer,dd1:_Integer] ->
uvelocitygrid[[c1,d1,cc1+1,dd1+1]],
vvelocity[e1:_Integer,f1:_Integer,ee1:_Integer,ff1:_Integer] ->
vvelocitygrid[[e1,f1,ee1+1,ff1+1]]});
Clear[a1,b1,c1,d1,e1,f1,g1,i1,j1,k1];
Clear[aa1,bb1,cc1,dd1,ee1,ff1,gg1,ii1,jj1,kk1];
rhs3=(vrelation /.
{vvelocity[g1:_Integer,i1:_Integer,gg1:_Integer,ii1:_Integer] ->
vvelocitygrid[[g1,i1,gg1+1,ii1+1]],
uvelocity[j1:_Integer,k1:_Integer,jj1:_Integer,kk1:_Integer] ->
uvelocitygrid[[j1,k1,jj1+1,kk1+1]]});
(*
Print["Fill assigning ",matrixicoord," , ",matrixjcoord," ",rhs1," ",rhs2,"
",rhs3];
*)
 pressuregrid[[matrixicoord,matrixjcoord,ldx+1,ldy+1]]=rhs1;
uvelocitygrid[[matrixicoord,matrixjcoord,ldx+1,ldy+1]]=rhs2;
vvelocitygrid[[matrixicoord,matrixjcoord,ldx+1,ldy+1]]=rhs3;
,{ldx,0,degree}]
,{ldy,0,degree}]
]]
,{fillptct,1,Length[correctfillordering]}];
(*
,{ict,-im,im}]
,{jct,-im,im}];
*)
);



(* ------------------------------------------------------------------------ *)
(* This procedure will advance the primitive variables p,u,v to the next time *)
(*  step                                                                    *)
(* ------------------------------------------------------------------------ *)
dotimeadvance := (
h=deltax;
```

```
nextpressuregrid
=Table[Table[Table[Table[0,{dy,0,degree}],{dx,0,degree}],{j,-im,im}],{i,-im,im}]
;
nextuvelocitygrid
=Table[Table[Table[Table[0,{dy,0,degree}],{dx,0,degree}],{j,-im,im}],{i,-im,im}]
;
nextvvelocitygrid
=Table[Table[Table[Table[0,{dy,0,degree}],{dx,0,degree}],{j,-im,im}],{i,-im,im}]
;

(* ------------------------------------------------------------------------ *)
(* Adjust the grid indexing with 0,0 in center to matrix indexing with     *)
(* 1,1 in top left                                                          *)
(* ------------------------------------------------------------------------ *)
correctedi[thisj_]:=im-thisj+1;
correctedj[thisi_]:=im+thisi+1;

Do[Do[
matrixicoord=correctedi[jct];
matrixjcoord=correctedj[ict];

If[Not[ValueQ[interpolantorder]],
interpolantorder=csize* (degree+1);
];
numberofcterms=interpolantorder-1;
(* --------- *)
(* Compute s *)
(* --------- *)
If[EvenQ[csize],
Do[
Do[
Do[
s[dy,iindex,j]=Collect[xc[iindex],fc[_,_,_,_]];
,{iindex,0,numberofcterms}]
,{dy,0,degree}]
,{j,1-(csize/2),csize/2}];
,
Do[
Do[
Do[
s[dy,iindex,j]=Collect[xc[iindex],fc[_,_,_,_]];
,{iindex,0,numberofcterms}]
,{dy,0,degree}]
,{j,-IntegerPart[csize/2],IntegerPart[csize/2]}];
];

(* ---------------------------------------- *)
(* Compute a, spatial interpolants cp,cu,cv *)
(* ---------------------------------------- *)
Clear[iindex,jindex,kindex];
Do[Do[Do[
```

```
(*
If[(iindex>maxind || jindex >maxind),
cp[iindex,jindex,kindex]=0;
cu[iindex,jindex,kindex]=0;
cv[iindex,jindex,kindex]=0;
]
*)
cp[iindex,jindex,kindex]=0;
cu[iindex,jindex,kindex]=0;
cv[iindex,jindex,kindex]=0
,{iindex,0,maxind+2}]
,{jindex,0,maxind+2}]
,{kindex,0,2 maxind}];


Do[
Do[
Clear[newdx,newdy,newi,newj];
Clear[newdx2,newdy2,newi2,newj2];
(* CP SPATIAL COEFFICIENTS *)
cp[iindex,jindex,0]=yc[jindex] /. {fc[newdx:_,newdy:_,newi:_,newj:_] ->
                        s[newdy,newdx,newj] } /. {dx->iindex} /.
{ fc[newdx2:_,newdy2:_,newi2:_,newj2:_] ->
pressuregrid[[matrixicoord-newj2,matrixjcoord+newi2,newdx2+1,newdy2+1]]}; ;

(*
Clear[newdx,newdy,newi,newj];
Clear[newdx2,newdy2,newi2,newj2];
cp[iindex,jindex,0]=cp[iindex,jindex,0] /. {
fc[newdx2:_,newdy2:_,newi2:_,newj2:_] ->
newfc[matrixicoord-newj2,matrixjcoord+newi2,newdx2+1,newdy2+1] };
Clear[newdx2,newdy2,newi2,newj2];
cp[iindex,jindex,0]=cp[iindex,jindex,0] /. {
fc[newdx2:_,newdy2:_,newi2:_,newj2:_] ->
pressuregrid[[newdx2,newdy2,newi2,newj2]]};
*)
(*
pressuregrid[[matrixicoord-newj2,matrixjcoord+newi2,newdx2+1,newdy2+1]]};
*)

(* CU SPATIAL COEFFICIENTS *)
Clear[newdx,newdy,newi,newj];
Clear[newdx2,newdy2,newi2,newj2];
cu[iindex,jindex,0]=yc[jindex] /. {fc[newdx:_,newdy:_,newi:_,newj:_] ->
                        s[newdy,newdx,newj] } /. {dx->iindex} /. {
fc[newdx2:_,newdy2:_,newi2:_,newj2:_]->
uvelocitygrid[[matrixicoord-newj2,matrixjcoord+newi2,newdx2+1,newdy2+1]]};
(* CV SPATIAL COEFFICIENTS *)
Clear[newdx,newdy,newi,newj];
Clear[newdx2,newdy2,newi2,newj2];
cv[iindex,jindex,0]=yc[jindex] /. {fc[newdx:_,newdy:_,newi:_,newj:_] ->
```

```
                        s[newdy,newdx,newj] } /. {dx->iindex} /. {
fc[newdx2:_,newdy2:_,newi2:_,newj2:_]->
vvelocitygrid[[matrixicoord-newj2,matrixjcoord+newi2,newdx2+1,newdy2+1]]};
,{jindex,0,numberofcterms}]
,{iindex,0,numberofcterms}];

(* ------------------------------------------------------ *)
(* Compute cp,cu,cv, time-space interpolant coefficients *)
(* ------------------------------------------------------ *)

Do[Do[Do[
cp[iindex,jindex,kindex]=ncp[iindex,jindex,kindex];
cu[iindex,jindex,kindex]=ncu[iindex,jindex,kindex];
cv[iindex,jindex,kindex]=ncv[iindex,jindex,kindex];
,{iindex,0,Min[maxind,2*maxind-kindex-jindex]}]
,{jindex,0,maxind}]
,{kindex,1,2 maxind}];


(* Time advance NOT using Horner form *)
Do[Do[
psum=0.0;usum=0.0; vsum=0.0;
Do[
(*
psum=psum+dx! * dy! * physicaltstep**kindex*cp[dx,dy,kindex];
usum=usum+dx! * dy! * physicaltstep**kindex*cu[dx,dy,kindex];
vsum=vsum+dx! * dy! * physicaltstep**kindex*cv[dx,dy,kindex];
*)
psum=psum+(dx! * dy!) * (timestep^kindex)*cp[dx,dy,kindex];
usum=usum+(dx! * dy!) * (timestep^kindex)*cu[dx,dy,kindex];
vsum=vsum+(dx! * dy!) * (timestep^kindex)*cv[dx,dy,kindex];
,{kindex,0,2 maxind}];
(* ------------------------------------------ *)
(* Use -stagger since using matrix indeces    *)
(* dx+1,dy+1 not dx,dy since in matrix notation *)
(* ------------------------------------------ *)
(*
Print["Assigning ",matrixicoord-stagger," , ",matrixjcoord+stagger," , ",psum,"
, ",usum," , ",vsum];
*)
 nextpressuregrid[[matrixicoord-stagger,matrixjcoord+stagger,dx+1,dy+1]]=psum;
nextuvelocitygrid[[matrixicoord-stagger,matrixjcoord+stagger,dx+1,dy+1]]=usum;
nextvvelocitygrid[[matrixicoord-stagger,matrixjcoord+stagger,dx+1,dy+1]]=vsum;
,{dx,0,degree}]
,{dy,0,degree}];

,{ict,-im+2,im-2}]
,{jct,-im+2,im-2}];

(* ---------------------------------------------------- *)
(* Put the time advanced level back onto the first time level *)
```

```
(* ------------------------------------------------------- *)
pressuregrid=nextpressuregrid;
uvelocitygrid=nextuvelocitygrid;
vvelocitygrid=nextvvelocitygrid;

);

showerror := (
(*
stmp4=OpenWrite["mmaerrorfile3"];
*)
(* ------------------------------------------------------------------------ *)
(* Adjust the grid indexing with 0,0 in center to matrix indexing with 1,1 in   *)
(* top left                                                                 *)
(* ------------------------------------------------------------------------ *)
correctedi[thisj_]:=im-thisj+1;
correctedj[thisi_]:=im+thisi+1;
maximumerrorfound=0.0;
l1error=0.0;
maxpressurefound=0.0;
minpressurefound=0.0;
currentenergy=0.0;
(* ------------------------------------------------------------------------ *)
(* Loop through all fill's and int's computing pressure error at this timestep   *)
(* number                                                                   *)
(* ------------------------------------------------------------------------ *)
Do[Do[
matrixicoord=correctedi[jct];
matrixjcoord=correctedj[ict];
oldphysicalpositionvector={ict*deltax,jct*deltax};
newphysicalpositionvector=Rotate2D[oldphysicalpositionvector,N[-theta],{0,0}];
newphysicalicoord=newphysicalpositionvector[[1]];
newphysicaljcoord=newphysicalpositionvector[[2]];

If[thegrid[[matrixicoord,matrixjcoord]]==1 ||
thegrid[[matrixicoord,matrixjcoord]]==2,
If[Not[MemberQ[ignorelist,{matrixicoord,matrixjcoord}]],
(* ---------------------------------------- *)
(* Add up the current global energy in system *)
(* ---------------------------------------- *)
(*
Write[stmp4,pressuregrid[[matrixicoord,matrixjcoord,1,1]],"
",uvelocitygrid[[matrixicoord,matrixjcoord,1,1]],"
",vvelocitygrid[[matrixicoord,matrixjcoord,1,1]]," ",currentenergy," ",ict,"
",jct];
*)
currentenergy=currentenergy+(pressuregrid[[matrixicoord,matrixjcoord,1,1]]^2
+uvelocitygrid[[matrixicoord,matrixjcoord,1,1]]^2+vvelocitygrid[[matrixicoord,ma
trixjcoord,1,1]]^2);
(* ------------------------------- *)
(* The plus comes from a minus minus *)
```

```
(* -------------------------------- *)
absoluteperror =
Abs[N[pressuregrid[[matrixicoord,matrixjcoord,1,1]]+(Cos[Sqrt[2] Pi
physicaltime] Cos[Pi newphysicalicoord] Cos[Pi newphysicaljcoord])]];
(* ------------------------------------------------ *)
(* Convert x and y coordinates to polar coordinates *)
(* ------------------------------------------------ *)
(*
physicaltheta = ArcTan[newphysicalicoord,newphysicaljcoord];
*)
physicalr = Sqrt[newphysicalicoord^2+newphysicaljcoord^2];
(* ------------------------------------------------------------ *)
(* Can use any of the eigenvalues from BesselJPrimeZeros[0,n] *)
(* ------------------------------------------------------------ *)
bessellam:=3.83171;
R[r_] := Sqrt[2] BesselJ[0,bessellam r]/BesselJ[0,bessellam];
(* Can use any constant *)
besseld=5;


(*
correctanswer=besseld R[physicalr] Cos[bessellam physicaltime] / Sqrt[2 Pi];
Print["Correct Answer at matrixi,matrixjj is
",matrixicoord,",",matrixjcoord,",",correctanswer,"
",pressuregrid[[matrixicoord,matrixjcoord,1,1]]];
*)
(*
absoluteperror =
Abs[N[pressuregrid[[matrixicoord,matrixjcoord,1,1]]-correctanswer]];
*)
If[absoluteperror > maximumerrorfound,
   maximumerrorfound=absoluteperror;
   matrixilocationoferror=matrixicoord;
   matrixjlocationoferror=matrixjcoord;
];
  l1error = l1error + absoluteperror;
(*
Print["p=",matrixicoord,",",matrixjcoord,",",pressuregrid[[matrixicoord,matrixjc
oord,1,1]]];
Print["u=",matrixicoord,",",matrixjcoord,",",uvelocitygrid[[matrixicoord,matrixj
coord,1,1]]];
Print["v=",matrixicoord,",",matrixjcoord,",",vvelocitygrid[[matrixicoord,matrixj
coord,1,1]]];
*)
If[pressuregrid[[matrixicoord,matrixjcoord,1,1]] > maxpressurefound,
maxpressurefound = pressuregrid[[matrixicoord,matrixjcoord,1,1]]];
If[pressuregrid[[matrixicoord,matrixjcoord,1,1]] < minpressurefound,
minpressurefound = pressuregrid[[matrixicoord,matrixjcoord,1,1]]];
]]
,{ict,-im,im}]
,{jct,-im,im}];
```

```
(*
Close[stmp4];
*)
energyratio=currentenergy/initialenergy;
l1error=l1error*deltax^2;
Print[stepnumber,".  ", physicaltime,"   ", maximumerrorfound,"   ", l1error,"
 ", maxpressurefound, "   ",minpressurefound, "   ",energyratio];
);

(* ------------------------------------------------------------------------ *)
(* This procecure will assign all primitive variables their initial data *)
(* ------------------------------------------------------------------------ *)

assigninitialdata := (
pressuregrid
=Table[Table[Table[Table[0,{dy,0,degree}],{dx,0,degree}],{j,-im,im}],{i,-im,im}]
;
uvelocitygrid
=Table[Table[Table[Table[0,{dy,0,degree}],{dx,0,degree}],{j,-im,im}],{i,-im,im}]
;
vvelocitygrid
=Table[Table[Table[Table[0,{dy,0,degree}],{dx,0,degree}],{j,-im,im}],{i,-im,im}]
;
(*
uvelocitygrid=Table[Table[0,{j,-im,im}],{i,-im,im}];
vvelocitygrid=Table[Table[0,{j,-im,im}],{i,-im,im}];
*)
(* ------------------------------------------------------------------------ *)
(* Adjust the grid indexing with 0,0 in center to matrix indexing with 1,1   *)
(* in top left                                                              *)
(* ------------------------------------------------------------------------ *)
correctedi[thisj_]:=im-thisj+1;
correctedj[thisi_]:=im+thisi+1;
(* ------------------------------------------------------------ *)
(* Loop through all grid points, assigning rotated initial data *)
(* ------------------------------------------------------------ *)
initialenergy=0.0;
Do[Do[
matrixicoord=correctedi[jct];
matrixjcoord=correctedj[ict];
oldphysicalpositionvector={ict*deltax,jct*deltax};
newphysicalpositionvector=Rotate2D[oldphysicalpositionvector,N[-theta],{0,0}];
newphysicalicoord=newphysicalpositionvector[[1]];
newphysicaljcoord=newphysicalpositionvector[[2]];
(* ------------------------------------------------------------------------ *)
(* if the point is an interior or fill, then assign it an initial condition *)
(* ------------------------------------------------------------------------ *)
(* ---------------------------------------- *)
(* Assign Bessel Function for circle problem *)
(* ---------------------------------------- *)
If[thegrid[[matrixicoord,matrixjcoord]]==1 ||
```

```
thegrid[[matrixicoord,matrixjcoord]]==2,
If[Not[MemberQ[ignorelist,{matrixicoord,matrixjcoord}]],
pressuregrid[[matrixicoord,matrixjcoord,1,1]]=N[ -(Cos[Pi newphysicalicoord] *
Cos[Pi newphysicaljcoord])];
uvelocitygrid[[matrixicoord,matrixjcoord,1,1]]= 0.0;
vvelocitygrid[[matrixicoord,matrixjcoord,1,1]]= 0.0;
(* ---------------------- *)
(* Add up initial energy  *)
(* ---------------------- *)
initialenergy=initialenergy+(pressuregrid[[matrixicoord,matrixjcoord,1,1]]^2
+uvelocitygrid[[matrixicoord,matrixjcoord,1,1]]^2+vvelocitygrid[[matrixicoord,ma
trixjcoord,1,1]]^2);
]
]
,{ict,-im,im}]
,{jct,-im,im}]
);

assigninitialdata2 := (
pressuregrid
=Table[Table[Table[Table[0,{dy,0,degree}],{dx,0,degree}],{j,-im,im}],{i,-im,im}]
;
uvelocitygrid
=Table[Table[Table[Table[0,{dy,0,degree}],{dx,0,degree}],{j,-im,im}],{i,-im,im}]
;
vvelocitygrid
=Table[Table[Table[Table[0,{dy,0,degree}],{dx,0,degree}],{j,-im,im}],{i,-im,im}]
;
(*
uvelocitygrid=Table[Table[0,{j,-im,im}],{i,-im,im}];
vvelocitygrid=Table[Table[0,{j,-im,im}],{i,-im,im}];
*)
(* ------------------------------------------------------------------------ *)
(* Adjust the grid indexing with 0,0 in center to matrix indexing with 1,1   *)
(* in top left                                                              *)
(* ------------------------------------------------------------------------ *)
correctedi[thisj_]:=im-thisj+1;
correctedj[thisi_]:=im+thisi+1;
(* ------------------------------------------------------------ *)
(* Loop through all grid points, assigning rotated initial data *)
(* ------------------------------------------------------------ *)
initialenergy=0.0;
Do[Do[
matrixicoord=correctedi[jct];
matrixjcoord=correctedj[ict];
oldphysicalpositionvector={ict*deltax,jct*deltax};
newphysicalpositionvector=Rotate2D[oldphysicalpositionvector,N[-theta],{0,0}];
newphysicalicoord=newphysicalpositionvector[[1]];
newphysicaljcoord=newphysicalpositionvector[[2]];
(* ------------------------------------------------------------------------ *)
(* if the point is an interior or fill, then assign it an initial condition *)
```

```
(* ------------------------------------------------------------------------ *)
(* ---------------------------------------- *)
(* Assign Bessel Function for circle problem *)
(* ---------------------------------------- *)
If[thegrid[[matrixicoord,matrixjcoord]]==1 ||
thegrid[[matrixicoord,matrixjcoord]]==2,
If[Not[MemberQ[ignorelist,{matrixicoord,matrixjcoord}]],
Do[Do[
correctp=D[-Cos[Pi x] Cos[Pi y],{x,dx},{y,dy}];
pressuregrid[[matrixicoord,matrixjcoord,dx+1,dy+1]]=N[ correctp /. {
x->newphysicalicoord, y->newphysicaljcoord}];
(*
Print["p=",matrixicoord,matrixjcoord,dx+1,dy+1,newphysicalicoord,newphysicaljcoo
rd,correctp];
*)
uvelocitygrid[[matrixicoord,matrixjcoord,dx+1,dy+1]]= 0.0;
vvelocitygrid[[matrixicoord,matrixjcoord,dx+1,dy+1]]= 0.0;
,{dx,0,degree}]
,{dy,0,degree}];
(* ---------------------- *)
(* Add up initial energy  *)
(* ---------------------- *)
initialenergy=initialenergy+(pressuregrid[[matrixicoord,matrixjcoord,1,1]]^2
+uvelocitygrid[[matrixicoord,matrixjcoord,1,1]]^2+vvelocitygrid[[matrixicoord,ma
trixjcoord,1,1]]^2);
]
]
,{ict,-im,im}]
,{jct,-im,im}]
);

(* ------------------------------------------------------------------------ *)
(* This procedure will generate a matrix of the entire grid, marked with *)
(* 1 for Interior, 2 for Fill, and 0 for boundary                         *)
(* It needs to be given the list of curves, and a single point for each  *)
(* object signifying the inside of the solid object                       *)
(* ------------------------------------------------------------------------ *)

buildgrid := (


(* ------------------------------------------------------------------------ *)
(* Expand depth of recursion, uses 10,000 bytes per depth, use memory         *)
(* constrained                                                              *)
(* ------------------------------------------------------------------------ *)

$RecursionLimit=(2 im + 1 ) ^ 2;

bigcount=0;
(* ---------------------------------- *)
(* Set entire grid to 0, all boundary *)
```

```
(* ---------------------------------- *)
thegrid=Table[Table[0,{i,-im,im}],{j,-im,im}];
arrowgrid=Table[Table[{{0,0}},{i,-im,im}],{j,-im,im}];
(*
udxdylist=Table[Table["U",{dx,0,degree}],{dy,0,degree}];
(* ---------------------------------- *)
(* Make a big list combining p,u, and v *)
(* ---------------------------------- *)
bigulist={udxdylist,udxdylist,udxdylist};
fillsolutiongrid=Table[Table[bigulist,{i,-im,im}],{j,-im,im}];
*)
fillsolutiongrid=Table[Table[Table[Table[Table["U",{dy,0,degree}],{dx,0,degree}]
,{ict,1,3}],{j,-im,im}],{i,-im,im}];

(*
fillsolutiongrid=Table[Table[{"U","U","U"},{i,-im,im}],{j,-im,im}];
*)
gridp=Table[Table[Table[Table[pressure[i+im+1,j+im+1,dx,dy],{dy,0,degree}],{dx,0
,degree}],{j,-im,im}],{i,-im,im}];
gridu=Table[Table[Table[Table[uvelocity[i+im+1,j+im+1,dx,dy],{dy,0,degree}],{dx,
0,degree}],{j,-im,im}],{i,-im,im}];
gridv=Table[Table[Table[Table[vvelocity[i+im+1,j+im+1,dx,dy],{dy,0,degree}],{dx,
0,degree}],{j,-im,im}],{i,-im,im}];
(*
gridu=Table[Table[uvelocity[i+im+1,j+im+1],{j,-im,im}],{i,-im,im}];
gridv=Table[Table[vvelocity[i+im+1,j+im+1],{j,-im,im}],{i,-im,im}];
*)

(* ---------------------------------------------------------- *)
(* Fill all areas with interior and fill labels where justified *)
(* By starting at each point defined to be inside, recursively. *)
(* ---------------------------------------------------------- *)
Do[
gridi=listofcenterpoints[[centerct]][[1]];
gridj=listofcenterpoints[[centerct]][[2]];
recursivelabel[gridi,gridj]
,{centerct,1,Length[listofcenterpoints]}];

$RecursionLimit=256;

(* ------------------------------------------------ *)
(* Draw a picture of fills, ints, B's, grid and curves *)
(* ------------------------------------------------ *)

correctedi[thisj_]:=im-thisj+1;
correctedj[thisi_]:=im+thisi+1;
alpha=theta;
picturelist={};
Do[Do[
thevalue=thegrid[[correctedi[j]]][[correctedj[i]]];
physicali=i*deltax;
```

```
physicalj=j*deltay;
If[thevalue==1,theobject=Disk[{physicali,physicalj},.03]];
If[thevalue==2,theobject=Circle[{physicali,physicalj},.03]];
If[thevalue==0,theobject=Text["B",{physicali,physicalj},{0,0}]];
picturelist=Append[picturelist,theobject]
,{i,-im,im}]
,{j,-im,im}];


titlestring="Rotation Angle = "<>ToString[N[alpha]];
filestring="boxat"<>ToString[N[alpha]];
SetOptions[Display,ImageSize-> 72 * 8, ImageRotated->True];

(*
Display[filestring,
Show[Graphics[{PointSize[0.05],picturelist}],PlotLabel->titlestring],"EPS"];
Show[Graphics[{PointSize[0.05*deltax],picturelist}],PlotLabel->titlestring,Aspec
tRatio->Automatic];
*)

(*
Show[Graphics[{PointSize[0.05],picturelist,Line[line[1]],Line[line[2]],Line[line
[3]],Line[line[4]],{PointSize[.02],Point[{0,0}]}}],PlotLabel->titlestring,Axes->
True,AspectRatio->Automatic,GridLines->{ticklist,ticklist}];
*)
Show[Graphics[{PointSize[0.05],picturelist,{PointSize[.02],Point[{0,0}]}}],g2,Di
splayFunction->$DisplayFunction,PlotLabel->titlestring,Axes->True,AspectRatio->A
utomatic,GridLines->{ticklist,ticklist}];

);

(* ------------------------------------------------------------- *)
(* This procedure will label a Cartesian grid with 1's and 2's     *)
(* if one of its edges cuts a boundary, this is a fill point = 2   *)
(* Will recursively call its other neighbors that do not have a    *)
(* boundary in between.                                            *)
(*             X   X   X           *)
(*               \ | /             *)
(*             X - 0 - X           *)
(*               / | \             *)
(*             X   X   X           *)
(* ------------------------------ *)

recursivelabel[myi_,myj_] := (

Print["myi = ",myi," myj = ",myj," bigcount = ",++bigcount," Memory In Use =
",MemoryInUse[]];

(* ---------------------------------------------------------------------- *)
(* Need to change coordinate systems between matrix notation and curve with *)
(* (x,y) = (0,0) at center of matrix at thegrid[[im]][[im]]                 *)
```

```
(* ------------------------------------------------------------------- *)

correctedi[thisj_]:=im-thisj+1;
correctedj[thisi_]:=im+thisi+1;

(* ------------------------------------------------------------------- *)
(* Define this grid point as an interior, perhaps change to fill later *)
(* ------------------------------------------------------------------- *)
thegrid[[correctedi[myj],correctedj[myi]]]=1;
(*
Print["Grid pt ",correctedi[myj]," ",correctedj[myi]];
*)

(*************************************************************************)
(* ----- *)
(* Right *)
(* ----- *)
(* x=t, y=myj,  myi<=t<=myi+1 *)
(* ------------------------- *)

theline={t,myj*deltay,myi*deltax,(myi+1)*deltax};

If[intersection[theline],
(* ---------------------------------------------- *)
(* This point is a fill since boundary intersects *)
(* ---------------------------------------------- *)
thegrid[[correctedi[myj],correctedj[myi]]]=2,
(* ------------------------------------------------------------ *)
(* If not intersects, and next point is not defined then call it *)
(* ------------------------------------------------------------ *)
If[thegrid[[correctedi[myj]]][[correctedj[myi+1]]]==0,
recursivelabel[myi+1,myj]]];

(*************************************************************************)

(* -------------- *)
(* Right, Bottom  *)
(* -------------- *)

theline={t, -t +(myi*deltax +myj*deltay),myi*deltax,(myi+1)*deltax};
If[intersection[theline],
(* ---------------------------------------------- *)
(* This point is a fill since boundary intersects *)
(* ---------------------------------------------- *)
thegrid[[correctedi[myj],correctedj[myi]]]=2,
(* ------------------------------------------------------------ *)
(* If not intersects, and next point is not defined then call it *)
(* ------------------------------------------------------------ *)
If[thegrid[[correctedi[myj-1]]][[correctedj[myi+1]]]==0,
recursivelabel[myi+1,myj-1]]];
```

```
(*************************************************************************)

(* ------ *)
(* Bottom *)
(* ------ *)

theline={myi*deltax,t,(myj-1)*deltay,myj*deltay};
If[intersection[theline],
(* ---------------------------------------------- *)
(* This point is a fill since boundary intersects *)
(* ---------------------------------------------- *)
thegrid[[correctedi[myj],correctedj[myi]]]=2,
(* ------------------------------------------------------------- *)
(* If not intersects, and next point is not defined then call it *)
(* ------------------------------------------------------------- *)
If[thegrid[[correctedi[myj-1]]][[correctedj[myi]]]==0,
recursivelabel[myi,myj-1]]];

(*************************************************************************)

(* ------------ *)
(* Left, Bottom *)
(* ------------ *)

theline={t,t-(myi*deltax)+(myj*deltay),(myi-1)*deltax,myi*deltax};
If[intersection[theline],
(* ---------------------------------------------- *)
(* This point is a fill since boundary intersects *)
(* ---------------------------------------------- *)
thegrid[[correctedi[myj],correctedj[myi]]]=2,
(* ------------------------------------------------------------- *)
(* If not intersects, and next point is not defined then call it *)
(* ------------------------------------------------------------- *)
If[thegrid[[correctedi[myj-1]]][[correctedj[myi-1]]]==0,
recursivelabel[myi-1,myj-1]]];

(*************************************************************************)

(* ---- *)
(* Left *)
(* ---- *)

theline={t,myj*deltay,(myi-1)*deltax,myi*deltax};
If[intersection[theline],
(* ---------------------------------------------- *)
(* This point is a fill since boundary intersects *)
(* ---------------------------------------------- *)
thegrid[[correctedi[myj],correctedj[myi]]]=2,
(* ------------------------------------------------------------- *)
(* If not intersects, and next point is not defined then call it *)
(* ------------------------------------------------------------- *)
```

```
If[thegrid[[correctedi[myj]]][[correctedj[myi-1]]]==0,
recursivelabel[myi-1,myj]]];

(*************************************************************************)

(* --------- *)
(* Left, Top *)
(* --------- *)

theline={t,-t+(myi*deltax)+(myj*deltay),(myi-1)*deltax,(myi*deltax)};
If[intersection[theline],
(* ------------------------------------------- *)
(* This point is a fill since boundary intersects *)
(* ------------------------------------------- *)
thegrid[[correctedi[myj],correctedj[myi]]]=2,
(* ------------------------------------------------------------ *)
(* If not intersects, and next point is not defined then call it *)
(* ------------------------------------------------------------ *)
If[thegrid[[correctedi[myj+1]]][[correctedj[myi-1]]]==0,
recursivelabel[myi-1,myj+1]]];

(*************************************************************************)


(* --- *)
(* Top *)
(* --- *)

theline={(myi*deltax),t,(myj*deltay),(myj+1)*deltay};
If[intersection[theline],
(* ------------------------------------------- *)
(* This point is a fill since boundary intersects *)
(* ------------------------------------------- *)
thegrid[[correctedi[myj],correctedj[myi]]]=2,
(* ------------------------------------------------------------ *)
(* If not intersects, and next point is not defined then call it *)
(* ------------------------------------------------------------ *)
If[thegrid[[correctedi[myj+1]]][[correctedj[myi]]]==0,
recursivelabel[myi,myj+1]]];

(*************************************************************************)

(* ---------- *)
(* Right, Top *)
(* ---------- *)

theline={t,t-(myi*deltax)+(myj*deltay),myi*deltax,(myi+1)*deltax};
If[intersection[theline],
(* ------------------------------------------- *)
(* This point is a fill since boundary intersects *)
(* ------------------------------------------- *)
```

```
thegrid[[correctedi[myj],correctedj[myi]]]=2,
(* ------------------------------------------------------------ *)
(* If not intersects, and next point is not defined then call it *)
(* ------------------------------------------------------------ *)
If[thegrid[[correctedi[myj+1]]][[correctedj[myi+1]]]==0,
recursivelabel[myi+1,myj+1]]]


);

(* ------------------------------------------------------------------------ *)
(* This procedure will test if theline intersects any of the specific curves *)
(* ------------------------------------------------------------------------ *)
intersection[theline_]:= (

intersect=False;
Do[
(* ------------------------------------ *)
(* the x equation for parametrized curve *)
(* x=f(t), y=g(t), tstart <= t <= tend   *)
(* ------------------------------------ *)
curvexequation=   listofcurves[[curvenumber]][[1]];
curveyequation=   listofcurves[[curvenumber]][[2]];
curvetstart=      listofcurves[[curvenumber]][[3]];
curvetend=        listofcurves[[curvenumber]][[4]];

curvetmin=Min[curvetstart,curvetend];
curvetmax=Max[curvetstart,curvetend];

x2equation=theline[[1]];
y2equation=theline[[2]];
t2start=   theline[[3]];
t2end=     theline[[4]];

t2min=Min[t2start,t2end];
t2max=Max[t2start,t2end];

allts=Solve[{curvexequation==x2equation,curveyequation==y2equation},{t,curvet}];

Do[
thecurvet=curvet/. Flatten[allts[[tct]]];
thelinet=t/. Flatten[allts[[tct]]];
If[ (( thecurvet >= curvetmin ) && ( thecurvet <= curvetmax ) &&
     ( thelinet  >= t2min )     && (  thelinet <= t2max)) ,
intersect=True]
,{tct,1,Length[allts]}]
,{curvenumber,1,Length[listofcurves]}];

intersect
);
```

```
(* ------------------------------------------------------------------------*)
(* This section will, when given a 3 by 3 stencil of only interior and fill   *)
(* points generate the set of polynomials that need solved, solve them and    *)
(* provide a function for each fill point in the stencil                      *)
(* It uses the arrow information provided from the bigger 5 by 5 stencil      *)
(* to put in the correct equations.                                           *)
(* Work on this wording                                                       *)
(* ------------------------------------------------------------------------ *)

(* ---------------------------------------------------------------- *)
(* The three interpolation functions that need cp, cu, and cv defined *)
(* ---------------------------------------------------------------- *)

(* ---------------------------------------------------------- *)
(* maxind = # of grid points in one direction of stencil -1 *)
(* including derivative data for Hermitian data             *)
(* ---------------------------------------------------------- *)
(*
p[x_,y_]:=Horner[Sum[cp[i3,j3] x^i3 y^j3 , {j3,0,maxind},{i3,0,maxind}]];
u[x_,y_]:=Horner[Sum[cu[i3,j3] x^i3 y^j3 , {j3,0,maxind},{i3,0,maxind}]];
v[x_,y_]:=Horner[Sum[cv[i3,j3] x^i3 y^j3 , {j3,0,maxind},{i3,0,maxind}]];
*)

(* ------------------------------------------------------------------------ *)
(* This procedure will compute the normals for a boundary curve    specified in *)
(* listofcurves                                                               *)
(* ------------------------------------------------------------------------ *)
(* Returns the unit normal vector on curve # curvect at position              *)
(* t=thevalueofcurvet                                                         *)
(* ------------------------------------------------------------------------ *)

computenormals[curvect_,thevalueofcurvet_] := (
(* ---------------------------------------------------------------- *)
(* Get x and y functions of parameter curvet for a particular curve *)
(* ---------------------------------------------------------------- *)
curvexequation=listofcurves[[curvect]][[1]];
curveyequation=listofcurves[[curvect]][[2]];

(* ------------------------------------------------------------------------*)
(* Compute the derivatives of the x and y functions with respect to the      *)
(* parameter                                                                  *)
(* ------------------------------------------------------------------------ *)
yderiv=D[curveyequation,curvet] /. {curvet -> N[thevalueofcurvet]};
xderiv=D[curvexequation,curvet] /. {curvet -> N[thevalueofcurvet]};
If[(xderiv==0 && yderiv==0), Print["Error in normal vector slope"],
If[xderiv!=0 && yderiv!=0 &&
   Not[xderiv === ComplexInfinity] &&
```

```
   Not[yderiv === ComplexInfinity] ,
(* ------------------------- *)
(* Tangential slope direction *)
(* ------------------------- *)
tslope=yderiv/xderiv;
(* ---------------------- *)
(* Normal slope direction *)
(* ---------------------- *)
nslope=-xderiv/yderiv;
un={ 1/(1+nslope^2), nslope/(1+nslope^2) };
(*
ut[curvect]={ 1/(1+tslope^2), tslope/(1+tslope^2) };
*)
,
(* ---------------- *)
(* Vertical Tangent *)
(* ---------------- *)
If[xderiv==0 || yderiv === ComplexInfinity,
un = {1,0};
(*
ut[curvect] := {0,1};
*)
];
(* ----------------- *)
(* Horizontal Tangent *)
(* ----------------- *)
If[yderiv==0 || xderiv === ComplexInfinity,
un = {0,1};
(*
ut[curvect] := {1,0};
*)
];
]];
(* ------------------------------------------------------------------------ *)
(* Return the unit normal vector of the curve at the correct location on the  *)
(* curve                                                                      *)
(* ------------------------------------------------------------------------ *)
un=un/Sqrt[un[[1]]^2+un[[2]]^2];
un /. { curvet -> thevalueofcurvet }
);

(* ------------------------------------------------------------------------ *)
(* This procedure will compute the tangents for a boundary curve specified in  *)
(* listofcurves                                                               *)
(* ------------------------------------------------------------------------ *)
(* Returns the unit tangent vector on curve # curvect at position             *)
(* t=thevalueofcurvet                                                         *)
(* ------------------------------------------------------------------------ *)

computetangents[curvect_,thevalueofcurvet_] := (
(* ------------------------------------------------------------- *)
```

```
(* Get x and y functions of parameter curvet for a particular curve *)
(* ---------------------------------------------------------------- *)
curvexequation=listofcurves[[curvect]][[1]];
curveyequation=listofcurves[[curvect]][[2]];

(* ---------------------------------------------------------------------------- *)
(* Compute the derivatives of the x and y functions with respect to the      *)
(* parameter                                                                  *)
(* ---------------------------------------------------------------------------- *)
yderiv=D[curveyequation,curvet] /. {curvet -> N[thevalueofcurvet]};
xderiv=D[curvexequation,curvet] /. {curvet -> N[thevalueofcurvet]};
If[(xderiv==0 && yderiv==0), Print["Error in normal vector slope"],
If[xderiv!=0 && yderiv!=0 &&
   Not[xderiv === ComplexInfinity] &&
   Not[yderiv === ComplexInfinity] ,
(* ------------------------- *)
(* Tangential slope direction *)
(* ------------------------- *)
tslope=yderiv/xderiv;
(* ---------------------- *)
(* Normal slope direction *)
(* ---------------------- *)
nslope=-xderiv/yderiv;
un={ 1/(1+nslope^2), nslope/(1+nslope^2) };
ut={ 1/(1+tslope^2), tslope/(1+tslope^2) };
,
(* ---------------- *)
(* Vertical Tangent *)
(* ---------------- *)
If[xderiv==0 || yderiv === ComplexInfinity,
un = {1,0};
ut = {0,1};
];
(* ------------------ *)
(* Horizontal Tangent *)
(* ------------------ *)
If[yderiv==0 || xderiv === ComplexInfinity,
un = {0,1};
ut = {1,0};
];
]];
(* ---------------------------------------------------------------------------- *)
(* Return the unit tangent vector of the curve at the correct location on the  *)
(* curve                                                                        *)
(* ---------------------------------------------------------------------------- *)
ut=ut/Sqrt[ut[[1]]^2+ut[[2]]^2];
ut /. { curvet -> thevalueofcurvet }
);
```

```
(* ------------------------------------------------------------------------ *)
(* This procedure creates a single equation for each fill in a given        *)
(* 2 by 2 stencil.  These equations are evaluated then at each time step    *)
(* to solve for the fills.                                                  *)
(* ------------------------------------------------------------------------ *)
(* It needs the mapped location for each fill point, and the surface normal *)
(* and tangent at that point                                                *)
(* ------------------------------------------------------------------------ *)

computesolutionsforall[standardmatrixpositionsin2by2grid_,
                standardphysicalpositionsin2by2grid_,
                        physicalpositionsin2by2grid_,
                                normalsin2by2grid_,
                                tangentsin2by2grid_]:=(
Print["In computesolutionsforall
",standardmatrixpositionsin2by2grid,standardphysicalpositionsin2by2grid,physical
positionsin2by2grid,normalsin2by2grid];
Clear[cp,p,dpn,x,y,lhs,rhs];
(* ------------------------------------------------------------------------ *)
(* Get the global physical coordinates of the center of the standard        *)
(* 2 by 2 stencil                                                           *)
(* Will use it to convert global physical coordinates to local physical     *)
(* coordinates                                                              *)
(* ------------------------------------------------------------------------ *)
centerphysicalx=standardphysicalpositionsin2by2grid[[1,1]][[1]]+(deltax/2);
centerphysicaly=standardphysicalpositionsin2by2grid[[1,1]][[2]]-(deltay/2);
maxind=csize(degree+1)-1;
i=standardmatrixicoord;
j=standardmatrixjcoord;
(* -------------------------- *)
(* Mathmetica Matrix Indexing  *)
(* -------------------------- *)
(* -------------------------- *)
(* (i  ,j  ) (i  ,j+1)        *)
(* (i+1,j  ) (i+1,j+1)        *)
(* -------------------------- *)

(* ---------------------------------- *)
(* Osculatory Formulation of Interpolant *)
(* ---------------------------------- *)
(* ------------------------------------------------------------------ *)
(* The array indeces for FORTRAN or the matrix indeces for Mathematica *)
(* of the particular interior points that are required                *)
(* ------------------------------------------------------------------ *)
(* Find which points are known and unknown, the unknown points become the  *)
(* unknown interpolation coefficients/variables to be solved later.        *)
(* ------------------------------------------------------------------------*)
(* This can be either a variable or a data element from the grid *)
(* ------------------------------------------------------------ *)
Clear["pvari*","uvari*","vvari*"];
```

```
(* ------------------------------------------------------------------------ *)
(* Assign a dummy variable to each Hermitian grid point element that will  *)
(* remain this dummy variable if its undefined or will be reassigned if it *)
(* has a definition or is known                                             *)
(* ------------------------------------------------------------------------ *)
(*
pvarordata=Table[Table[Table[Table[Symbol["pvari"<>ToString[i]<>"j"<>ToString[j]
<>"dx"<>ToString[dx]<>"dy"<>ToString[dy]],{dy,0,degree}],{dx,0,degree}],{j,1,2}]
,{i,1,2}];
uvarordata=Table[Table[Table[Table[Symbol["uvari"<>ToString[i]<>"j"<>ToString[j]
<>"dx"<>ToString[dx]<>"dy"<>ToString[dy]],{dy,0,degree}],{dx,0,degree}],{j,1,2}]
,{i,1,2}];
vvarordata=Table[Table[Table[Table[Symbol["vvari"<>ToString[i]<>"j"<>ToString[j]
<>"dx"<>ToString[dx]<>"dy"<>ToString[dy]],{dy,0,degree}],{dx,0,degree}],{j,1,2}]
,{i,1,2}];
*)
Do[Do[Do[Do[
pvarordata[i,j,dx+1,dy+1]=Symbol["pvari"<>ToString[i]<>"j"<>ToString[j]<>"dx"<>T
oString[dx]<>"dy"<>ToString[dy]];
uvarordata[i,j,dx+1,dy+1]=Symbol["uvari"<>ToString[i]<>"j"<>ToString[j]<>"dx"<>T
oString[dx]<>"dy"<>ToString[dy]];
vvarordata[i,j,dx+1,dy+1]=Symbol["vvari"<>ToString[i]<>"j"<>ToString[j]<>"dx"<>T
oString[dx]<>"dy"<>ToString[dy]]
,{dy,0,degree}],{dx,0,degree}],{j,1,2}],{i,1,2}];

(* ------------------------------------------------------------------------ *)
(* This will contain the list of unknown Hermitian data elements that needs *)
(* solved.                                                                  *)
(* ------------------------------------------------------------------------ *)
variablelistp={};
variablelistu={};
variablelistv={};


(* ------------------------------------------------------------------------ *)
(* Loop through all four grid points in this stencil to determine interior  *)
(* or fill.  If interior, then all of its Hermitian data is already known   *)
(* and simply needs assigned.  Or if a fill that has been recycled or filled *)
(* already then again simply assign this information now                    *)
(* ------------------------------------------------------------------------ *)
Do[Do[
standardmatrixicoord=standardmatrixpositionsin2by2grid[[matrixict,matrixjct]][[1
]];
standardmatrixjcoord=standardmatrixpositionsin2by2grid[[matrixict,matrixjct]][[2
]];
(* ------------------------------------------------------------------------ *)
(* If this stencil point is an interior, then insert it into Lagrangian     *)
(* Formulation                                                              *)
(* ------------------------------------------------------------------------ *)
(* Or if a fill point is being considered as an interior point (ie. recycled) *)
(* ------------------------------------------------------------------------ *)
```

```
If[thegrid[[standardmatrixicoord,standardmatrixjcoord]]==1 ||
Length[arrowgrid[[standardmatrixicoord,standardmatrixjcoord]]]>2,
(* ------------------------------------------------------------------ *)
(* The data is known so loop through all Hermitian data and assign it *)
(* ------------------------------------------------------------------ *)
Do[Do[

pvarordata[matrixict,matrixjct,ldx+1,ldy+1]=gridp[[standardmatrixicoord,standard
matrixjcoord,ldx+1,ldy+1]];

uvarordata[matrixict,matrixjct,ldx+1,ldy+1]=gridu[[standardmatrixicoord,standard
matrixjcoord,ldx+1,ldy+1]];

vvarordata[matrixict,matrixjct,ldx+1,ldy+1]=gridv[[standardmatrixicoord,standard
matrixjcoord,ldx+1,ldy+1]]
,{ldx,0,degree}]
,{ldy,0,degree}]
,
(* ------------------------------------------------------------------ *)
(* If this stencil point is a fill point, then assign a variable to it *)
(* This variable symbol will be used in the Solve step later           *)
(* ------------------------------------------------------------------ *)
Do[Do[
AppendTo[variablelistp,pvarordata[matrixict,matrixjct,ldx+1,ldy+1]];
AppendTo[variablelistu,uvarordata[matrixict,matrixjct,ldx+1,ldy+1]];
AppendTo[variablelistv,vvarordata[matrixict,matrixjct,ldx+1,ldy+1]]
,{ldx,0,degree}]
,{ldy,0,degree}]
]
,{matrixict,1,2}]
,{matrixjct,1,2}];

(*
Print[variablelistp,variablelistu,variablelistv];

Print["p[x,y] = ",InputForm[p[x,y]]];
wait=Input["Press Enter"];
Print["u[x,y] = ",InputForm[u[x,y]]];
wait=Input["Press Enter"];
Print["v[x,y] = ",InputForm[v[x,y]]];
wait=Input["Press Enter"];
*)

(* ------------------------------------------------------------------------ *)
(* Now need to substitute in the values for each fill point into the        *)
(* boundary condition equations already formed.                             *)
(* Supply the normx, normy, x, y values for the mapped location of fill     *)
(* ------------------------------------------------------------------------ *)

(* ---------------------------------------- *)
(* Loop through all four points in 2 by 2 stencil *)
```

```
(* ------------------------------------------ *)

pequationlist={};
uandvequationlist={};
Do[Do[

(* -------------------------------- *)
(* Pressure Wall Boundary Condition *)
(* -------------------------------- *)

(* ------------------------------------------------------------------------ *)
(* The actual physical coordinates for either mapped fills or unmapped      *)
(* interiors                                                                *)
(* ------------------------------------------------------------------------ *)
xcoord=physicalpositionsin2by2grid[[matrixict,matrixjct]][[1]];
ycoord=physicalpositionsin2by2grid[[matrixict,matrixjct]][[2]];


(* ---------------------------------------------------------------- *)
(* The actual physical coordinates converted into local coordinates *)
(* ----------------------------------------------------------------*)
xdiff=Abs[N[(xcoord-centerphysicalx)]];
ydiff=Abs[N[(ycoord-centerphysicaly)]];
If[centerphysicalx > xcoord, localxcoord=-xdiff, localxcoord=xdiff];
If[centerphysicaly > ycoord, localycoord=-ydiff, localycoord=ydiff];


(* ---------------------------------------------------------------- *)
(* The array indeces for FORTRAN or the matrix indeces for Mathematica *)
(* of the particular interior points that are required              *)
(* ---------------------------------------------------------------- *)
standardmatrixicoord=standardmatrixpositionsin2by2grid[[matrixict,matrixjct]][[1
]];
standardmatrixjcoord=standardmatrixpositionsin2by2grid[[matrixict,matrixjct]][[2
]];

(*
Print["normalsin2by2grid
",matrixict,matrixjct,normalsin2by2grid[[matrixict,matrixjct]]];
*)
(* ---------------------------------------------------------------- *)
(* Using Osculatory Formulation, a set of equations for each fill,  *)
(* ---------------------------------------------------------------- *)
If[thegrid[[standardmatrixicoord,standardmatrixjcoord]]==2 &&
Length[arrowgrid[[standardmatrixicoord,standardmatrixjcoord]]]==2,
(*
Clear[normx,normy];
normx=normalsin2by2grid[[matrixict,matrixjct]][[1]];
normy=normalsin2by2grid[[matrixict,matrixjct]][[2]];
*)
(*
```

```
normx=normxtemp[matrixict,matrixjct];
normy=normytemp[matrixict,matrixjct];
*)

(*
tangx=-normy;
tangy=normx;
*)

(*
tangx=tangentsin2by2grid[[matrixict,matrixjct]][[1]];
tangy=tangentsin2by2grid[[matrixict,matrixjct]][[2]];
*)

(*
x=localxcoord;
y=localycoord;
*)

(*
x=xcoord2[matrixict,matrixjct];
y=ycoord2[matrixict,matrixjct];
*)

(* ------------------------------------------------------------------------ *)
(* The normx,normy,x,y are now defined so form the set of equations required *)
(* for this fill point                                                      *)
(* ------------------------------------------------------------------------ *)
newderpntaulist=derpntaulist /. {x->localxcoord,y->localycoord,
          normx->normalsin2by2grid[[matrixict,matrixjct]][[1]],
          normy->normalsin2by2grid[[matrixict,matrixjct]][[2]],
          tangx->-normalsin2by2grid[[matrixict,matrixjct]][[2]],
          tangy->normalsin2by2grid[[matrixict,matrixjct]][[1]]};
newderuntaulist=deruntaulist /. {x->localxcoord,y->localycoord,
          normx->normalsin2by2grid[[matrixict,matrixjct]][[1]],
          normy->normalsin2by2grid[[matrixict,matrixjct]][[2]],
          tangx->-normalsin2by2grid[[matrixict,matrixjct]][[2]],
          tangy->normalsin2by2grid[[matrixict,matrixjct]][[1]]};
newdervntaulist=dervntaulist /. {x->localxcoord,y->localycoord,
          normx->normalsin2by2grid[[matrixict,matrixjct]][[1]],
          normy->normalsin2by2grid[[matrixict,matrixjct]][[2]],
          tangx->-normalsin2by2grid[[matrixict,matrixjct]][[2]],
          tangy->normalsin2by2grid[[matrixict,matrixjct]][[1]]};

Print["Using fill equations at localxcoord and localycoord :",localxcoord," and
", localycoord," with normal (",normx,normy,")"];
AppendTo[pequationlist,newderpntaulist];
AppendTo[uandvequationlist,newderuntaulist];
AppendTo[uandvequationlist,newdervntaulist];
]
,{matrixict,1,2}]
```

```
,{matrixjct,1,2}];

(**************************)
(* p7 p8 p9 *)
(* p4 p5 p6 *) (* Point location definition *)
(* p1 p2 p3 *)
(************)
Clear[h,i,j,cp,cu,cv];

maxind=csize(degree+1)-1;
pvariables=Flatten[variablelistp];
uandvvariables=Flatten[{variablelistu,variablelistv}];

pequationlist=Flatten[pequationlist];
uandvequationlist=Flatten[uandvequationlist];

(* ------------------------------------------------------ *)
(* Solve for all of the spatial interpolant coefficients *)
(* ------------------------------------------------------ *)

pmatvec=LinearEquationsToMatrices[pequationlist, pvariables];
pmatrix=pmatvec[[1]];
prhs=pmatvec[[2]];
uandvmatvec=LinearEquationsToMatrices[uandvequationlist, uandvvariables];
uandvmatrix=uandvmatvec[[1]];
uandvrhs=uandvmatvec[[2]];

(*
wait=Input["Wait1"];
*)
(*
psings=Flatten[SingularValues[pmatrix]];
uandvsings=Flatten[SingularValues[uandvmatrix]];
pconditionnumber=Max[psings]/Min[psings];
uandvconditionnumber=Max[uandvsings]/Min[uandvsings];
pdet=Det[pmatrix];
uandvdet=Det[uandvmatrix];
Print["P Matrix Condition #:",pconditionnumber," and determinant :",pdet];
Print["UandV Matrix Condition #:",uandvconditionnumber," and determinant
:",uandvdet];
*)
(* -------------------------- *)
(* The vector of coefficients *)
(* -------------------------- *)
(*
pcoef=LinearSolve[pmatrix,prhs];
uandvcoef=LinearSolve[uandvmatrix,uandvrhs];
*)
(* ------------------------------------------------------------------------ *)
(* Find inverse of p matrix, handles poorly conditioned systems better than  *)
(* linearsolve                                                               *)
```

```
(* ------------------------------------------------------------------------ *)
(*
wait=Input["Wait2"];
*)
(*
pinvmatrix=Inverse[pmatrix];
pcoef=Collect[pinvmatrix . prhs,pressure[_,_,_,_]];
*)
(*
pcoef=LinearSolve[pmatrix,prhs];
*)
pcoef=solvesystemincnofc[pmatrix,prhs];

(* ---------------------------- *)
(* Find Inverse of uandv matrix *)
(* ---------------------------- *)
(*
uandvinvmatrix=Inverse[uandvmatrix];
uandvcoef=Collect[uandvinvmatrix .
uandvrhs,{uvelocity[_,_,_,_],vvelocity[_,_,_,_]}];
*)

(*
uandvcoef=LinearSolve[uandvmatrix,uandvrhs];
*)
uandvcoef=solvesystemincnofc[uandvmatrix,uandvrhs];

Clear[aside,bside];
makeequal[aside_,bside_]:= aside=bside;
(* ---------------------------------- *)
(* Set the p coefficients with solution *)
(* ---------------------------------- *)
ppairs = {pvariables, pcoef};
Apply[makeequal,ppairs];
(* ----------------------------------------- *)
(* Set the u and v coefficients with solution *)
(* ----------------------------------------- *)
uandvpairs = {uandvvariables, uandvcoef};
Apply[makeequal,uandvpairs];

(* ------------------------------------------------------------------------ *)
(*Produce equation for each fill point, so that it may be filled at each     *)
(* time step                                                                *)
(* ------------------------------------------------------------------------ *)
(* the equations will be absolute and in terms of matrix coordinates        *)
(*------------------------------------------------------------------------- *)

(*
solutionmatrix=Table[{0,0,0},{ii,1,2},{jj,1,2}];
*)
solutionmatrix=Table[Table[Table[Table[Table[0,{dy,0,degree}],{dx,0,degree}],{va
```

```
rct,1,3}],{ii,1,2}],{jj,1,2}];
Do[Do[
Do[Do[
(* ---------------------------------------------------------------- *)
(* The unmapped standard physical location of a typical 2 by 2 stencil *)
(* ---------------------------------------------------------------- *)
standardxcoord=standardphysicalpositionsin2by2grid[[matrixi,matrixj]][[1]];
standardycoord=standardphysicalpositionsin2by2grid[[matrixi,matrixj]][[2]];
localphysicalx=(standardxcoord-centerphysicalx);
localphysicaly=(standardycoord-centerphysicaly);

(* ---------------------------------- *)
(* The p solutions for all derivatives *)
(* ---------------------------------- *)
newsymbolp=Symbol["pvari"<>ToString[matrixi]<>"j"<>ToString[matrixj]<>"dx"<>ToSt
ring[ldx]<>"dy"<>ToString[ldy]];
solutionmatrix[[matrixi,matrixj,1,ldx+1,ldy+1]]=
pvarordata[matrixi,matrixj,ldx+1,ldy+1];
(*
newsymbolp;
Simplify[D[p[x,y],{x,ldx},{y,ldy}] /.
{x-> localphysicalx, y-> localphysicaly}];
*)

(* ---------------------------------- *)
(* The u solutions for all derivatives *)
(* ---------------------------------- *)
newsymbolu=Symbol["uvari"<>ToString[matrixi]<>"j"<>ToString[matrixj]<>"dx"<>ToSt
ring[ldx]<>"dy"<>ToString[ldy]];
solutionmatrix[[matrixi,matrixj,2,ldx+1,ldy+1]]=
uvarordata[matrixi,matrixj,ldx+1,ldy+1];
(*
newsymbolu;
Simplify[D[u[x,y],{x,ldx},{y,ldy}] /.
{x-> localphysicalx, y-> localphysicaly}];
*)

(* ---------------------------------- *)
(* The v solutions for all derivatives *)
(* ---------------------------------- *)
newsymbolv=Symbol["vvari"<>ToString[matrixi]<>"j"<>ToString[matrixj]<>"dx"<>ToSt
ring[ldx]<>"dy"<>ToString[ldy]];
solutionmatrix[[matrixi,matrixj,3,ldx+1,ldy+1]]=
vvarordata[matrixi,matrixj,ldx+1,ldy+1];
(*
newsymbolv;
Simplify[D[v[x,y],{x,ldx},{y,ldy}] /.
{x-> localphysicalx, y-> localphysicaly}];
*)
,{ldx,0,degree}]
,{ldy,0,degree}]
```

```
,{matrixi,1,2}]
,{matrixj,1,2}];
Clear[x,y];
solutionmatrix
);


(* ------------------------------------------------------------------------ *)
(* This procedure creates a single equation for each fill in a given 2 by 2  *)
(* stencil                                                                  *)
(* These equations are evaluated then at each time step to solve for the    *)
(* fills.                                                                   *)
(* ------------------------------------------------------------------------ *)
(* It needs the mapped location for each fill point, and the surface normal  *)
(* at that point                                                            *)
(* And the surface tangent at that point                                    *)
(* ------------------------------------------------------------------------ *)


(* ------------------------------------------------------ *)
(* This procedure will scan entire grid looking for fills *)
(* And assigning arrows to them                           *)
(* ------------------------------------------------------ *)

definearrows2:= (


(* ------------------------------------------------------------------------ *)
(* Get list of fills locations sorted according to number of fills in its *)
(* spatial interpolation stencil.                                           *)
(* Will contain only the corners of the stencils                            *)
(* ------------------------------------------------------------------------ *)
fillposlist=minimizeboundary2;

(* ------------------------------------------------------------------ *)
(* Loop through all the fill points, in the correct order of course *)
(* ------------------------------------------------------------------ *)
While[Length[fillposlist]>0,

(* ------------------------------------------------------------ *)
(* This is the location of the fill point in matrix coordinates *)
(* ------------------------------------------------------------ *)
matrixi=fillposlist[[1]][[1]];
matrixj=fillposlist[[1]][[2]];
quadtouse=fillposlist[[1]][[3]];
Print["Working on ",matrixi," ",matrixj," ",quadtouse];


(* ------------------------------------------------- *)
(* Get the 7 by 7 stencil with the fill in the center *)
(* ------------------------------------------------- *)
```

```
stencil7by7=SubMatrix[thegrid,{matrixi-3,matrixj-3},{7,7}];


(* ------------------------------------------------------------------ *)
(* Will use 4 different mapping algorithms, depends upon quadrant in use *)
(* Need to insure stencil expands for CFL stability                    *)
(* ------------------------------------------------------------------ *)
themapping=getmapping[stencil7by7,quadtouse];

If[Not[mappingsuccessful],
Print["No mapping found at ",matrixi," ",matrixj];
Print["Later let code try to use another neighboring stencil for mapping"];
Quit;
];

(* ------------------------------------------------------------------ *)
(* Get list of fill points that are solved by this 2 by 2 system *)
(* ------------------------------------------------------------------ *)
listofdonefills=computedonefills[matrixi,matrixj,stencil7by7,quadtouse];

(* ------------------------------------------------------------------ *)
(* Those fills not already in correctfillordering but should be now *)
(* ------------------------------------------------------------------ *)
undonelistofdonefills =
IntegerPart[Complement[listofdonefills,correctfillordering]];
AppendTo[correctfillordering,undonelistofdonefills];
correctfillordering=IntegerPart[Partition[Flatten[correctfillordering],2]];

(* ------------------------------------------------------------------------ *)
(* Need to remove from dolist those fills that are solved with this 2 by 2    *)
(* spat. int.                                                               *)
(* ------------------------------------------------------------------------ *)
Do[
fillposlist=DeleteCases[fillposlist,{IntegerPart[listofdonefills[[ct]][[1]]],
                        IntegerPart[listofdonefills[[ct]][[2]]],m_}];
,{ct,1,Length[listofdonefills]}];

(* ------------------------------ *)
(* assign the mapping to arrowgrid *)
(* ------------------------------ *)
If[quadtouse==1,toplefti=matrixi;   topleftj=matrixj];
If[quadtouse==2,toplefti=matrixi-1;   topleftj=matrixj];
If[quadtouse==3,toplefti=matrixi-1;   topleftj=matrixj-1];
If[quadtouse==4,toplefti=matrixi;   topleftj=matrixj-1];

(* ------------------------------------------------------------------------ *)
(* If no arrow assigned, then assign it, otherwise use current arrow instead *)
(* And recycle the fill data by changing its arrow to {0,0}                 *)
(* To remove instability                                                    *)
(* ------------------------------------------------------------------------ *)
```

```
Do[Do[
If[Length[arrowgrid[[toplefti+iii,topleftj+jjj]]]==1,
         AppendTo[arrowgrid[[toplefti+iii,topleftj+jjj]],
themapping[[iii+1,jjj+1]]],
(* ------------------------------------------------------------------------ *)
(* If already has an arrow, consider it an interior point for this stencil   *)
(* Add a {0,0} in third position to indicate it is now a recycled fill point *)
(* Can count the number of {0,0} in this list to see how many times it is    *)
(* recycled                                                                  *)
(* ------------------------------------------------------------------------*)
         AppendTo[arrowgrid[[toplefti+iii,topleftj+jjj]], {0,0}];
];
,{iii,0,1}]
,{jjj,0,1}];

(*
Print["toplefti = ",toplefti,topleftj,themapping];
*)
AppendTo[topleftlist,{toplefti,topleftj}];


(* ------------------------------------------------------------------------ *)
(* Now assign fill solutions at these points,                               *)
(* need to get boundary intersection and normals                            *)
(* first.  Will use the global grid coordinates just determined,            *)
(* and the matrix of arrow                                                  *)
(* directions in correctlyplacedarrows, if the grid point is an interior,   *)
(* do not assign it                                                         *)
(* if the grid point is already an assigned fill point, do not reassign it  *)
(* ------------------------------------------------------------------------*)


(*
the2by2solutionsmatrix=get2by2solutionmatrix[toplefti,topleftj,correctlyplacedar
rows];
*)

the2by2solutionsmatrix=get2by2solutionmatrix[toplefti,topleftj,themapping];

(*
Print["the2by2solutionsmatrix=",the2by2solutionsmatrix];
*)
(*
te=Input["Press Enter"];
*)
(* ------------------------------------------------------------------------ *)
(* Only place a solution for a fill if it is undefined,                      *)
(* this way the first stencil solution                                       *)
(* that applies to a fill is used only.                                      *)
(* ------------------------------------------------------------------------ *)
Do[Do[
```

```
(*
ldx=0;ldy=0;
*)
Do[Do[
If[fillsolutiongrid[[toplefti+iii-1,topleftj+jjj-1,1,ldx+1,ldy+1]]=="U",
(* ------------------------------ *)
(* For now use no derivative data *)
(* ------------------------------ *)
(* ------ *)
(* P DATA *)
(* ------ *)
fillsolutiongrid[[toplefti+iii-1,topleftj+jjj-1,1,ldx+1,ldy+1]]=Chop[the2by2solu
tionsmatrix[[iii,jjj,1,ldx+1,ldy+1]]];
(* ------ *)
(* U DATA *)
(* ------ *)
fillsolutiongrid[[toplefti+iii-1,topleftj+jjj-1,2,ldx+1,ldy+1]]=Chop[the2by2solu
tionsmatrix[[iii,jjj,2,ldx+1,ldy+1]]];
(* ------ *)
(* V DATA *)
(* ------ *)
fillsolutiongrid[[toplefti+iii-1,topleftj+jjj-1,3,ldx+1,ldy+1]]=Chop[the2by2solu
tionsmatrix[[iii,jjj,3,ldx+1,ldy+1]]]]
,{ldx,0,degree}]
,{ldy,0,degree}]
,{iii,1,2}]
,{jjj,1,2}];

];

(* ------------------------------------------------------------------------ *)
(* Create the list of fills that are not needed and should not be included *)
(* in error calculations.                                                   *)
(* ------------------------------------------------------------------------ *)
ignorelist=Complement[Position[thegrid,2],correctfillordering]

);

(* ------------------------------------------------------------------------- *)
(* This procedure will return a set of 4 vectors showing the arrow or        *)
(* direction  for each of the 4 grid points to be mapped to.                 *)
(* If they are an interior point                                             *)
(* they do not get mapped, so assigned a none or {0,0} vector                *)
(* ------------------------------------------------------------------------- *)
getmapping[the7by7mat_,lquadtouse_]:=(
l7by7mat=the7by7mat;

mappingsuccessful=False;
p1notfound=False;
p2notfound=False;
p3notfound=False;
```

```
p4notfound=False;

(* ------------------------------------------------------------------- *)
(* Generate all of the possible locations to map to in local coordinates *)
(* centered at the grid point being mapped                             *)
(* ------------------------------------------------------------------- *)
directionlist={};
Do[Do[
AppendTo[directionlist,{N[Sqrt[ict^2+jct^2]],ict,jct}]
,{ict,-2,2}]
,{jct,-2,2}];

directionlist=Complement[directionlist,{{0,0,0}}];

(* ------------------------------------------------------------------- *)
(* Each of the 4 grid points is only allowed to be mapped in a way *)
(* that expands the stencil for CFL stability                      *)
(* ------------------------------------------------------------------- *)
p1directionlist=Sort[DeleteCases[directionlist,{m_,x_,y_} /; x<=0 && y>=0]];
p2directionlist=Sort[DeleteCases[directionlist,{m_,x_,y_} /; x<=0 && y<=0]];
p3directionlist=Sort[DeleteCases[directionlist,{m_,x_,y_} /; x>=0 && y<=0]];
p4directionlist=Sort[DeleteCases[directionlist,{m_,x_,y_} /; x>=0 && y>=0]];

(* ------------------------------------------------------------------- *)
(* Each of the 4 grid points needs to find the first B it can using its *)
(* directionlist which is presorted to minimize distance               *)
(* ------------------------------------------------------------------- *)

(* ------------------------------------------------------------------- *)
(* Find where the topleft of the 2 by 2 stencil is in relation to the   *)
(* 7 by 7 stencil.                                                     *)
(* ------------------------------------------------------------------- *)

If[lquadtouse==1, matrixtoplefti=4; matrixtopleftj=4];
If[lquadtouse==2, matrixtoplefti=3; matrixtopleftj=4];
If[lquadtouse==3, matrixtoplefti=3; matrixtopleftj=3];
If[lquadtouse==4, matrixtoplefti=4; matrixtopleftj=3];

(* ------------------------------------------------ *)
(* Change the 1 to a 5 if inside the 2 by 2 stencil *)
(* ------------------------------------------------ *)
Do[Do[
If[l7by7mat[[matrixtoplefti+iict,matrixtopleftj+jjct]]==1,
l7by7mat[[matrixtoplefti+iict,matrixtopleftj+jjct]]=5];
,{iict,0,1}]
,{jjct,0,1}];

(* ------ *)
(* p3 p2  *)
(* p4 p1  *)
(*=====================================================================*)
```

```
(* MAP P1 *)
(* ------------------------------------------------------- *)
(* Find the first B to map to using the ordered direction list *)
(* ------------------------------------------------------- *)
p1dirct=1;
p1={0,0};
p1loci=matrixtoplefti+1;
p1locj=matrixtopleftj+1;
If[l7by7mat[[p1loci,p1locj]]==2,
p1notfound=True;
(* ------------------------------------------------------------------- *)
(* Loop through all permissible directions until a suitable map is found *)
(* ------------------------------------------------------------------- *)
While[p1notfound && p1dirct<=Length[p1directionlist],
mag    =p1directionlist[[p1dirct]][[1]];
xoffset=p1directionlist[[p1dirct]][[2]];
yoffset=p1directionlist[[p1dirct]][[3]];
row=Length[Cases[l7by7mat[[p1loci-yoffset]],5]];
column=Length[Cases[Map[#[[p1locj+xoffset]]&,l7by7mat],5]];
(* ------------------------------------------------------------------------ *)
(* Is the offseted grid point a boundary point and                          *)
(* Test to be sure that no more than one other grid point is assigned to the *)
(* row or column                                                            *)
(* ------------------------------------------------------------------------ *)
If[l7by7mat[[p1loci-yoffset,p1locj+xoffset]]==0 && row<=1 && column<=1,
p1notfound=False;
(* ---------------------------------- *)
(* Define normalized arrow direction *)
(* ---------------------------------- *)
p1={xoffset,yoffset}/mag;
(* ------------------------------------------------ *)
(* Assign a 5 indicating a mapping to this location *)
(* ------------------------------------------------ *)
l7by7mat[[p1loci-yoffset,p1locj+xoffset]]=5;
];
p1dirct++;
];
];
(*=====================================================================*)
(* MAP P2 *)
(* ------------------------------------------------------- *)
(* Find the first B to map to using the ordered direction list *)
(* ------------------------------------------------------- *)
p2dirct=1;
p2={0,0};
p2loci=matrixtoplefti;
p2locj=matrixtopleftj+1;
If[l7by7mat[[p2loci,p2locj]]==2,
p2notfound=True;
(* ------------------------------------------------------------------- *)
(* Loop through all permissible directions until a suitable map is found *)
```

```
(* ------------------------------------------------------------------- *)
While[p2notfound && p2dirct<=Length[p2directionlist],
mag    =p2directionlist[[p2dirct]][[1]];
xoffset=p2directionlist[[p2dirct]][[2]];
yoffset=p2directionlist[[p2dirct]][[3]];
row=Length[Cases[l7by7mat[[p2loci-yoffset]],5]];
column=Length[Cases[Map[#[[p2locj+xoffset]]&,l7by7mat],5]];
(* ------------------------------------------------------------------- *)
(* Is the offseted grid point a boundary point and                     *)
(* Test to be sure that no more than one other grid point is assigned to the *)
(* row or column                                                       *)
(* ------------------------------------------------------------------- *)
If[l7by7mat[[p2loci-yoffset,p2locj+xoffset]]==0 && row<=1 && column<=1,
p2notfound=False;
(* --------------------------------- *)
(* Define normalized arrow direction *)
(* --------------------------------- *)
p2={xoffset,yoffset}/mag;
(* ------------------------------------------------ *)
(* Assign a 5 indicating a mapping to this location *)
(* ------------------------------------------------ *)
l7by7mat[[p2loci-yoffset,p2locj+xoffset]]=5;
];
p2dirct++;
];
];

(*====================================================================*)
(* MAP P3 *)
(* ---------------------------------------------------------- *)
(* Find the first B to map to using the ordered direction list *)
(* ---------------------------------------------------------- *)
p3dirct=1;
p3={0,0};
p3loci=matrixtoplefti;
p3locj=matrixtopleftj;
If[l7by7mat[[p3loci,p3locj]]==2,
p3notfound=True;
(* ------------------------------------------------------------------- *)
(* Loop through all permissible directions until a suitable map is found *)
(* ------------------------------------------------------------------- *)
While[p3notfound && p3dirct<=Length[p3directionlist],
mag    =p3directionlist[[p3dirct]][[1]];
xoffset=p3directionlist[[p3dirct]][[2]];
yoffset=p3directionlist[[p3dirct]][[3]];
row=Length[Cases[l7by7mat[[p3loci-yoffset]],5]];
column=Length[Cases[Map[#[[p3locj+xoffset]]&,l7by7mat],5]];
(* ------------------------------------------------------------------- *)
(* Is the offseted grid point a boundary point and                     *)
(* Test to be sure that no more than one other grid point is assigned to the *)
(* row or column                                                       *)
```

```
(* ------------------------------------------------------------------------ *)
If[17by7mat[[p3loci-yoffset,p3locj+xoffset]]==0 && row<=1 && column<=1,
p3notfound=False;
(* --------------------------------- *)
(* Define normalized arrow direction *)
(* --------------------------------- *)
p3={xoffset,yoffset}/mag;
(* ------------------------------------------------ *)
(* Assign a 5 indicating a mapping to this location *)
(* ------------------------------------------------ *)
17by7mat[[p3loci-yoffset,p3locj+xoffset]]=5;
];
p3dirct++;
];
];

(*=========================================================================*)
(* MAP P4 *)
(* ------------------------------------------------------------ *)
(* Find the first B to map to using the ordered direction list *)
(* ------------------------------------------------------------ *)
p4dirct=1;
p4={0,0};
p4loci=matrixtoplefti+1;
p4locj=matrixtopleftj;
If[17by7mat[[p4loci,p4locj]]==2,
p4notfound=True;
(* ------------------------------------------------------------------------ *)
(* Loop through all permissible directions until a suitable map is found *)
(* ------------------------------------------------------------------------ *)
While[p4notfound && p4dirct<=Length[p4directionlist],
mag    =p4directionlist[[p4dirct]][[1]];
xoffset=p4directionlist[[p4dirct]][[2]];
yoffset=p4directionlist[[p4dirct]][[3]];
row=Length[Cases[17by7mat[[p4loci-yoffset]],5]];
column=Length[Cases[Map[#[[p4locj+xoffset]]&,17by7mat],5]];
(* ------------------------------------------------------------------------ *)
(* Is the offseted grid point a boundary point and                          *)
(* Test to be sure that no more than one other grid point is assigned to the *)
(* row or column                                                            *)
(* ------------------------------------------------------------------------ *)
If[17by7mat[[p4loci-yoffset,p4locj+xoffset]]==0 && row<=1 && column<=1,
p4notfound=False;
(* --------------------------------- *)
(* Define normalized arrow direction *)
(* --------------------------------- *)
p4={xoffset,yoffset}/mag;
(* ------------------------------------------------ *)
(* Assign a 5 indicating a mapping to this location *)
(* ------------------------------------------------ *)
17by7mat[[p4loci-yoffset,p4locj+xoffset]]=5;
```

```
];
p4dirct++;
];
];

notfoundlist={p1notfound,p2notfound,p3notfound,p4notfound};
If[Length[Cases[notfoundlist,True]]==0,mappingsuccessful=True];
If[p1notfound,Print["Did not find a mapping for p1"]];
If[p2notfound,Print["Did not find a mapping for p2"]];
If[p3notfound,Print["Did not find a mapping for p3"]];
If[p4notfound,Print["Did not find a mapping for p4"]];

dirmat={{p3,p2},{p4,p1}};
dirmat
);




(* ------------------------------------------------------------------------- *)
(* This procedure will return a set of 9 vectors showing the arrow or direction  *)
(* for each of the 9 grid points to be mapped to.  If they are an interior point *)
(* they do not get mapped, so assigned a none or {0,0} vector                    *)
(* ------------------------------------------------------------------------- *)
s8mapping[s8matrix_]:=(
Print["Calling s8mapping"];
(* p7 p8 p9 *)
(* p4 p5 p6 *)
(* p1 p2 p3 *)

(* ------------------------------------ *)
(* Set of all normalized vector directions *)
(* ------------------------------------ *)
none = {0,0};
up = {0,1};
upright = {1/Sqrt[2],1/Sqrt[2]};
right = {1,0};
downright = {1/Sqrt[2],-1/Sqrt[2]};
down = {0,-1};
downleft = {-1/Sqrt[2],-1/Sqrt[2]};
left = {-1,0};
upleft = {-1/Sqrt[2],1/Sqrt[2]};

(* Try a 22.5 instead of 45 slope *)
(*
none = {0,0};
up = {0,1};
upright = {2/Sqrt[5],1/Sqrt[5]};
right = {1,0};
downright = {2/Sqrt[5],-1/Sqrt[5]};
```

```
down = {0,-1};
downleft = {-2/Sqrt[5],-1/Sqrt[5]};
left = {-1,0};
upleft = {-2/Sqrt[5],1/Sqrt[5]};
*)
(* ---------------------------------- *)
(* By s8 assumption, these do not vary *)
(* ---------------------------------- *)
(* If not matching cases it is important to check for undefined *)
(* Unless a prescreening with stencil constraint tree was done already *)
p1=undefined;
p2=undefined;
p3=none;
p4=undefined;
p5=none;
p6=undefined;
p7=upleft;
p8=undefined;
p9=undefined;
(* ------------------------------------------------------------------------ *)
(* Non-Double Fill Cases with fill using 5 by 5 info. only w/s8 modified    *)
(* assumption                                                               *)
(* ------------------------------------------------------------------------ *)
(* o . o *)
(*     o *)

If[s8matrix[[3,4]]==1 && s8matrix[[3,5]]==2 && s8matrix[[4,5]]==2,
p6=upright; p8=none; p9=upright ];

(* o     *)
(* .     *)
(* o o   *)

If[s8matrix[[4,3]]==1 && s8matrix[[5,3]]==2 && s8matrix[[5,4]]==2,
p1=downleft; p2=downleft; p4=none];

(* ------------------------------------------------------------------------ *)
(* Non-Double Fill Cases with interior using 5 by 5 info. only w/s8 modified *)
(* assumption                                                               *)
(* ------------------------------------------------------------------------ *)
(* o . o *)
(*     . *)
If[s8matrix[[3,4]]==1 && s8matrix[[3,5]]==2 && s8matrix[[4,5]]==1,
p6=none; p8=none; p9=upright ];

(* o o . *)
(*     . *)
If[s8matrix[[3,4]]==2 && s8matrix[[3,5]]==1 && s8matrix[[4,5]]==1,
p6=none; p8=upleft; p9=none];

(* o . . *)
```

```
(*     . *)
If[s8matrix[[3,4]]==1 && s8matrix[[3,5]]==1 && s8matrix[[4,5]]==1,
p6=none; p8=none; p9=none];

(* o     *)
(* .     *)
(* o .   *)

If[s8matrix[[4,3]]==1 && s8matrix[[5,3]]==2 && s8matrix[[5,4]]==1,
p1=downleft; p2=none; p4=none];

(* o     *)
(* o     *)
(* . .   *)
If[s8matrix[[4,3]]==2 && s8matrix[[5,3]]==1 && s8matrix[[5,4]]==1,
p1=none; p2=none; p4=upleft];

(* o     *)
(* .     *)
(* . .   *)

If[s8matrix[[4,3]]==1 && s8matrix[[5,3]]==1 && s8matrix[[5,4]]==1,
p1=none; p2=none; p4=none];

(* ------------------------------------------------------------------------ *)
(* Double Fill with interior and fill cases using 5by5 info. only with modified*)
(* s8                                                                       *)
(* Here the 2nd row of 5 by 5 affects mapping.                              *)
(* ------------------------------------------------------------------------ *)

(* ? ? ? *)
(* o o o *)
(*     Z *)
If[s8matrix[[3,4]]==2 && s8matrix[[3,5]]==2 ,

(* X X B *)

If[s8matrix[[2,3]]!=0 && s8matrix[[2,4]]!=0 && s8matrix[[2,5]]==0,
p8=up; p9=up];

(* B X B *)
If[s8matrix[[2,3]]==0 && s8matrix[[2,4]]!=0 && s8matrix[[2,5]]==0,
p8=upleft; p9=up];

(* B X X *)
If[s8matrix[[2,3]]==0 && s8matrix[[2,4]]!=0 && s8matrix[[2,5]]!=0,
p8=upleft; p9=upright];

(* X B X *)
If[s8matrix[[2,3]]!=0 && s8matrix[[2,4]]==0 && s8matrix[[2,5]]!=0,
p8=upleft; p9=upleft];
```

```
(* X B B *)
If[s8matrix[[2,3]]!=0 && s8matrix[[2,4]]==0 && s8matrix[[2,5]]==0,
p8=up; p9=up];

(* B B X *)
If[s8matrix[[2,3]]==0 && s8matrix[[2,4]]==0 && s8matrix[[2,5]]!=0,
p8=upleft; p9=upleft];

(* B B B *)
If[s8matrix[[2,3]]==0 && s8matrix[[2,4]]==0 && s8matrix[[2,5]]==0,
p7=up; p8=up; p9=up];

(* Fill or Int p6 *)
If[s8matrix[[4,5]]==1,p6=none,p6=upright];
];

(* ------------ *)
(* Do left side *)
(* ------------ *)
(* ? o   *)
(* ? o   *)
(* ? o . *)

If[s8matrix[[4,3]]==2 && s8matrix[[5,3]]==2 ,

(* X     *)
(* X     *)
(* B     *)

If[s8matrix[[3,2]]!=0 && s8matrix[[4,2]]!=0 && s8matrix[[5,2]]==0,
p1=left; p4=left];

(* B     *)
(* X     *)
(* B     *)

If[s8matrix[[3,2]]==0 && s8matrix[[4,2]]!=0 && s8matrix[[5,2]]==0,
p1=left; p4=upleft];
(* B     *)
(* X     *)
(* X     *)

If[s8matrix[[3,2]]==0 && s8matrix[[4,2]]!=0 && s8matrix[[5,2]]!=0,
p1=downleft; p4=upleft];

(* X     *)
(* B     *)
(* X     *)

If[s8matrix[[3,2]]!=0 && s8matrix[[4,2]]==0 && s8matrix[[5,2]]!=0,
```

```
p1=upleft; p4=upleft];

(* X     *)
(* B     *)
(* B     *)

If[s8matrix[[3,2]]!=0 && s8matrix[[4,2]]==0 && s8matrix[[5,2]]==0,
p1=left; p4=left];

(* B     *)
(* B     *)
(* X     *)

If[s8matrix[[3,2]]==0 && s8matrix[[4,2]]==0 && s8matrix[[5,2]]!=0,
p1=upleft; p4=upleft];

(* B     *)
(* B     *)
(* B     *)

If[s8matrix[[3,2]]==0 && s8matrix[[4,2]]==0 && s8matrix[[5,2]]==0,
p1=left; p4=left; p7=left];

(* Fill or Int p6 *)
If[s8matrix[[5,4]]==1,p2=none,p2=downleft];
];

dirmat={{p7,p8,p9},{p4,p5,p6},{p1,p2,p3}};
dirmat
);


(* ------------------------------------------------------------------------ *)
(* This procedure will compute the set of fills that are solved             *)
(* simultaneously using this 3 by 3 s8 stencil,                             *)
(* needs back rotated to fit into global grid coordinates                   *)
(* ------------------------------------------------------------------------*)

computedonefills[centerfilli_,centerfillj_,the7by7_,lquadtouse_]:=(
(* ------------------------------------------------------------------------ *)
(* the center of 7 by 7 is by definition a fill, does not need tested,      *)
(* add to list                                                              *)
(* ------------------------------------------------------------------------ *)

donelist={{centerfilli,centerfillj}};
(* Test quad1 for fills *)
If[lquadtouse==1,
If[the7by7[[5,4]]==2,AppendTo[donelist,{centerfilli+1,centerfillj}]];
If[the7by7[[5,5]]==2,AppendTo[donelist,{centerfilli+1,centerfillj+1}]];
If[the7by7[[4,5]]==2,AppendTo[donelist,{centerfilli,centerfillj+1}]];
];
```

```
(* Test quad2 for fills *)
If[lquadtouse==2,
If[the7by7[[3,4]]==2,AppendTo[donelist,{centerfilli-1,centerfillj}]];
If[the7by7[[3,5]]==2,AppendTo[donelist,{centerfilli-1,centerfillj+1}]];
If[the7by7[[4,5]]==2,AppendTo[donelist,{centerfilli,centerfillj+1}]];
];
(* Test quad1 for fills *)
If[lquadtouse==3,
If[the7by7[[3,3]]==2,AppendTo[donelist,{centerfilli-1,centerfillj-1}]];
If[the7by7[[4,3]]==2,AppendTo[donelist,{centerfilli  ,centerfillj-1}]];
If[the7by7[[3,4]]==2,AppendTo[donelist,{centerfilli-1,centerfillj  }]];
];
(* Test quad1 for fills *)
If[lquadtouse==4,
If[the7by7[[4,3]]==2,AppendTo[donelist,{centerfilli  ,centerfillj-1}]];
If[the7by7[[5,3]]==2,AppendTo[donelist,{centerfilli+1,centerfillj-1}]];
If[the7by7[[5,4]]==2,AppendTo[donelist,{centerfilli+1,centerfillj  }]];
];
IntegerPart[donelist]
);


(* ------------------------------------------------------------------- *)
(* This procedure will return a matrix of solutions for the fills with   *)
(* Nulls for interior grid points.  It will convert the global matrix    *)
(* coordinates to global physical coordinates                            *)
(* And then will find the intersection with a boundary, then compute the  *)
(* normal vector to the                                                  *)
(* surface, Then call the spatial interpolator with all the mapping      *)
(* points and normals to                                                 *)
(* find the solution for all the fill points in this particular 2 by 2   *)
(* stencil                                                               *)
(* ------------------------------------------------------------------- *)

get2by2solutionmatrix[topleftmatrixi_,topleftmatrixj_,correctlyplacedarrows_]:=
(
(*
Print["called get2by2solutionmatrix"];
*)
(* ------------------------------------------------- *)
(* Convert the topleftmatrixi,j into  grid coordinates *)
(* ------------------------------------------------- *)

topleftgridi = topleftmatrixj - im - 1;
topleftgridj =-topleftmatrixi + im + 1;

(*------------------------------------------------------------------- *)
(* Define the 2 by 2 matrices corresponding to the tangents,normals      *)
(* and actual physical location on the boundary that a fill is mapped to *)
(* ------------------------------------------------------------------- *)
tangentsin2by2grid=Table[Null,{iiii,1,2},{jjjj,1,2}];
```

```
normalsin2by2grid=Table[Null,{iiii,1,2},{jjjj,1,2}];
physicalpositionsin2by2grid=Table[Null,{iiii,1,2},{jjjj,1,2}];
(* ------------------------------------------------------------------- *)
(* Define the 2 by 2 matrices corresponding to the unmapped location in *)
(* global grid coordinates and global matrix coordinates               *)
(* ------------------------------------------------------------------- *)
standardphysicalpositionsin2by2grid=Table[Null,{iiii,1,2},{jjjj,1,2}];
standardmatrixpositionsin2by2grid=Table[Null,{iiii,1,2},{jjjj,1,2}];
(* ---------------------------------------- *)
(* Loop through all 4 points in 2 by 2 stencil *)
(* ---------------------------------------- *)
Do[Do[
(* ------------------------------------------------------------------ *)
(* Find global physical location of this grid point in 2 by 2 stencil *)
(* ------------------------------------------------------------------ *)
physicali=(topleftgridi+ict-1)*deltax;
physicalj=(topleftgridj-jct+1)*deltay;
(* ---------------------------------------------------------- *)
(* This is the matrix coordinate location of this stencil point *)
(* ---------------------------------------------------------- *)
standardmatrixi    =(topleftmatrixi+ict-1);
standardmatrixj    =(topleftmatrixj+jct-1);
(* ------------------------------------------------ *)
(* Get the direction of the arrow for this grid point *)
(* ------------------------------------------------ *)
thearrowdirection=correctlyplacedarrows[[jct,ict]];
(* ------------------------------------------------------------------------ *)
(* Compute the intersection of this arrow with the defined geometry curves *)
(* If arrow is {0,0} it is an interior point, do not find intersection     *)
(* ------------------------------------------------------------------------ *)
If[thearrowdirection!={0,0},
xf= physicali+ (t thearrowdirection[[1]]);
yf= physicalj+ (t thearrowdirection[[2]]);
t1=0;
t2=deltax Sqrt[2];
theline={xf,yf,t1,t2};
{{xcoord,ycoord},{xnorm,ynorm},{xtang,ytang}}=findintersectionandnormalandtangen
t[theline];
Print["Grid point ",physicali,",",physicalj," is mapped to
pt,normal,tang=",{{xcoord,ycoord},{xnorm,ynorm},{xtang,ytang}} ];
normalsin2by2grid[[jct,ict]]={xnorm,ynorm};
tangentsin2by2grid[[jct,ict]]={xtang,ytang};
standardmatrixpositionsin2by2grid[[ict,jct]]={standardmatrixi,standardmatrixj};
physicalpositionsin2by2grid[[jct,ict]]={xcoord,ycoord};
standardphysicalpositionsin2by2grid[[jct,ict]]={physicali,physicalj};
,
(* ------------------------------------------------------------------------ *)
(* If its an interior point then return its physical coordinates as its     *)
(* solution                                                                 *)
(* ------------------------------------------------------------------------ *)
normalsin2by2grid[[jct,ict]]=Null;
```

```
tangentsin2by2grid[[jct,ict]]=Null;
standardmatrixpositionsin2by2grid[[ict,jct]]={standardmatrixi,standardmatrixj};
physicalpositionsin2by2grid[[jct,ict]]={physicali,physicalj};
standardphysicalpositionsin2by2grid[[jct,ict]]={physicali,physicalj};
]
,{ict,1,2}]
,{jct,1,2}];
(* ------------------------------------------------------------------------ *)
(* Now have position information and normal information to surface required  *)
(* to form a 2 by 2 spatial interpolant.  And the tangent information.       *)
(* ------------------------------------------------------------------------ *)

allsolutionsin2by2grid=computesolutionsforall[standardmatrixpositionsin2by2grid,
standardphysicalpositionsin2by2grid,physicalpositionsin2by2grid,normalsin2by2gri
d,tangentsin2by2grid];
allsolutionsin2by2grid
);

(* ------------------------------------------------------------------------ *)
(* This procedure will test if theline intersects any of the specific curves *)
(* And return the curve coordinates and normal at the point of intersection  *)
(* ------------------------------------------------------------------------ *)
findintersectionandnormalandtangent[theline_]:= (
(*
Print["In findintersectionandnormalandtangent"];
*)
intersect=False;
Do[
(* ------------------------------------ *)
(* the x equation for parametrized curve *)
(* x=f(t), y=g(t), tstart <= t <= tend   *)
(* ------------------------------------ *)
curvexequation=   listofcurves[[curvenumber]][[1]];
curveyequation=   listofcurves[[curvenumber]][[2]];
curvetstart=      listofcurves[[curvenumber]][[3]];
curvetend=        listofcurves[[curvenumber]][[4]];

curvetmin=Min[curvetstart,curvetend];
curvetmax=Max[curvetstart,curvetend];

x2equation=theline[[1]];
y2equation=theline[[2]];
t2start=   theline[[3]];
t2end=     theline[[4]];

t2min=Min[t2start,t2end];
t2max=Max[t2start,t2end];

allts=Solve[{curvexequation==x2equation,curveyequation==y2equation},{t,curvet}];

Do[
```

```
thecurvet=curvet/. Flatten[allts[[tct]]];
thelinet=t/. Flatten[allts[[tct]]];
If[ (( thecurvet >= curvetmin ) && ( thecurvet <= curvetmax ) &&
     ( thelinet  >= t2min )     && (  thelinet <= t2max)) ,
intersect=True;
xcoord=curvexequation /. {curvet->thecurvet};
ycoord=curveyequation /. {curvet->thecurvet};
{xnorm,ynorm} = computenormals[curvenumber,thecurvet];
{xtang,ytang} = computetangents[curvenumber,thecurvet];
]
,{tct,1,Length[allts]}]
,{curvenumber,1,Length[listofcurves]}];

intersectionpoint={xcoord,ycoord};
normalvector     ={xnorm, ynorm };
tangentvector    ={xtang, ytang };
{intersectionpoint, normalvector, tangentvector}
);

(* ------------------------------------------------------------------------ *)
(* This procedure will read a grid definition file written by a FORTRAN or    *)
(* C code                                                                   *)
(* the grid file consists of 0 = boundary, 1 = interior, 2 = needed fill,     *)
(* 3 = unneeded fill                                                        *)
(* ------------------------------------------------------------------------
*)
(* File is stored in column major ordering *)
(* ------------------------------------- *)

readgridproc := (
stmp=OpenRead["fort.4"];
(* ------------------------------------------------------------ *)
(* First item in file is size of Cartesian grid in one dimension *)
(* Is a square in 2D *)
(* ----------------- *)
im=Read[stmp];
(* ---------------------------------------------------- *)
(* Second item in file is grid density per unit interval *)
(* ---------------------------------------------------- *)
iun=Read[stmp];
(* ------------ *)
(* grid spacing *)
(* -------------*)
deltax=1/iun;
deltay=deltax;

(* ---------------------------------- *)
(* Set entire grid to 0, all boundary *)
(* ---------------------------------- *)
thegrid=Table[Table[0,{i,-im,im}],{j,-im,im}];
arrowgrid=Table[Table[{{0,0}},{i,-im,im}],{j,-im,im}];
```

```
(*
fillsolutiongrid=Table[Table[{"U","U","U"},{i,-im,im}],{j,-im,im}];
*)
(*
pdataPerGridPt=(degree+1)^2;
ulist={};
Do[AppendTo[ulist,"U"],{ct,1,pdataPerGridPt}];
*)
(*
udxdylist=Table[Table["U",{dx,0,degree}],{dy,0,degree}];
(* ------------------------------------ *)
(* Make a big list combining p,u, and v *)
(* ------------------------------------ *)
bigulist={udxdylist,udxdylist,udxdylist};
fillsolutiongrid=Table[Table[bigulist,{i,-im,im}],{j,-im,im}];
*)
fillsolutiongrid=Table[Table[Table[Table[Table["U",{dy,0,degree}],{dx,0,degree}]
,{ict,1,3}],{j,-im,im}],{i,-im,im}];

gridp=Table[Table[Table[Table[pressure[i+im+1,j+im+1,dx,dy],{dy,0,degree}],{dx,0
,degree}]{j,-im,im}],{i,-im,im}];
gridu=Table[Table[Table[Table[uvelocity[i+im+1,j+im+1,dx,dy],{dy,0,degree}],{dx,
0,degree}]{j,-im,im}],{i,-im,im}];
gridv=Table[Table[Table[Table[vvelocity[i+im+1,j+im+1,dx,dy],{dy,0,degree}],{dx,
0,degree}]{j,-im,im}],{i,-im,im}];
(*
gridu=Table[Table[uvelocity[i+im+1,j+im+1],{j,-im,im}],{i,-im,im}];
gridv=Table[Table[vvelocity[i+im+1,j+im+1],{j,-im,im}],{i,-im,im}];
*)

(* ------------------------------------------------------ *)
(* Read entire grid definition into thegrid in matrix form *)
(* ------------------------------------------------------ *)
Do[Do[
thegrid[[matrixict,matrixjct]]=Read[stmp]
,{matrixict,1,im+im+1}]
,{matrixjct,1,im+im+1}];

);

(* ------------------------------------------------- *)
(* Draw a picture of fills, ints, B's, grid and curves *)
(* ------------------------------------------------- *)
drawgrid := (

correctedi[thisj_]:=im-thisj+1;
correctedj[thisi_]:=im+thisi+1;
alpha=theta;
picturelist={};
Do[Do[
thevalue=thegrid[[correctedi[j]]][[correctedj[i]]];
```

```
physicali=i*deltax;
physicalj=j*deltay;
If[thevalue==1,theobject=Disk[{physicali,physicalj},.03]];
If[thevalue==2,theobject=Circle[{physicali,physicalj},.03]];
If[thevalue==3,theobject=Circle[{physicali,physicalj},.01]];
If[thevalue==0,theobject=Text["B",{physicali,physicalj},{0,0}]];
picturelist=Append[picturelist,theobject]
,{i,-im,im}]
,{j,-im,im}];


titlestring="Rotation Angle = "<>ToString[N[alpha]];
filestring="boxat"<>ToString[N[alpha]];
SetOptions[Display,ImageSize-> 72 * 8, ImageRotated->True];

Show[Graphics[{PointSize[0.05],picturelist,{PointSize[.02],Point[{0,0}]}}],g2,Di
splayFunction->$DisplayFunction,PlotLabel->titlestring,Axes->True,AspectRatio->A
utomatic,GridLines->{ticklist,ticklist}];

);


buildcurves := (
If[Not[ValueQ[maxmemi]],
im=Input["Enter the maximum field size "];
,
im=maxmemi;
Print["Will use a global domain of size -",im,":",im,"^2"];
];
If[Not[ValueQ[maxiun]],
iun=Input["Enter the number of grid points per unit length"];
,
iun=maxiun;
];
If[Not[ValueQ[theta]],
theta=Input["Enter the rotation angle in Radians for box "];
];
(*
im=7;
iun=4;
*)
(* ------------ *)
(* grid spacing *)
(* -------------*)
deltax=1/iun;
deltay=deltax;
correctedi[thisj_]:=im-thisj+1;
correctedj[thisi_]:=im+thisi+1;
(* ------------------------- *)
(* x=t, y=t^2,  -1 <= t <= 1 *)
(* ------------------------- *)
```

```
line[1]={Rotate2D[{-1,-1},N[theta],{0,0}],Rotate2D[{1,-1},N[theta],{0,0}]};
line[2]={Rotate2D[{ 1,-1},N[theta],{0,0}],Rotate2D[{1, 1},N[theta],{0,0}]};
line[3]={Rotate2D[{ 1, 1},N[theta],{0,0}],Rotate2D[{-1,1},N[theta],{0,0}]};
line[4]={Rotate2D[{-1, 1},N[theta],{0,0}],Rotate2D[{-1,-1},N[theta],{0,0}]};
ticklist=Table[i,{i,-im*deltax,im*deltax,deltax}];

(*
Show[Graphics[{Line[line[1]],Line[line[2]],Line[line[3]],Line[line[4]],{PointSiz
e[.02],Point[{0,0}]}}],Axes->True,AspectRatio->Automatic,GridLines->{ticklist,ti
cklist}];
*)

(* ---------------------------------- *)
(* Convert lines to parametric curves *)
(* ---------------------------------- *)
listofcurves={};
displaylist={};
Do[
xpt1=line[linect][[1]][[1]];
ypt1=line[linect][[1]][[2]];
xpt2=line[linect][[2]][[1]];
ypt2=line[linect][[2]][[2]];
(* ------------------------------------------- *)
(* If line is not vertical, use x=t else use y=t *)
(* ------------------------------------------- *)
If[(xpt2-xpt1)!=0,
slope=(ypt2-ypt1)/(xpt2-xpt1);
xf=curvet;
yf=ypt1 + slope ( curvet - xpt1 );
t1=xpt1;
t2=xpt2,
(* ----------- *)
(* Is Vertical *)
(* ----------- *)
xf=xpt1;
yf=curvet;
t1=ypt1;
t2=ypt2];
thecurve={xf,yf,t1,t2};
AppendTo[listofcurves,thecurve];
AppendTo[displaylist,ParametricPlot[{xf,yf},{curvet,t1,t2},DisplayFunction->Iden
tity]]
,{linect,1,4}];

(* Do circle *)
(*
listofcurves={};
displaylist={};
xf=curvet;
yf=Sqrt[1-curvet^2];
t1=-1;
```

```
t2=1;
thecurve={xf,yf,t1,t2};
AppendTo[listofcurves,thecurve];
AppendTo[displaylist,ParametricPlot[{xf,yf},{curvet,t1,t2},DisplayFunction->Iden
tity]];
xf=curvet;
yf=-Sqrt[1-curvet^2];
t1=-1;
t2=1;
thecurve={xf,yf,t1,t2};
AppendTo[listofcurves,thecurve];
AppendTo[displaylist,ParametricPlot[{xf,yf},{curvet,t1,t2},DisplayFunction->Iden
tity]];
*)


g2=Show[displaylist,AspectRatio->automatic,DisplayFunction->Identity];
Print["List of Parametric Curves ",listofcurves];

listofcenterpoints={{correctedi[0],correctedj[0]}};
listofcenterpoints={{0,0}};

);
drawentiregraph := (
(* ---------------------------------- *)
(* Draw entire graph with arrows added *)
(* ---------------------------------- *)
(* Put together the list of arrows into a picture *)
(* ------------------------------------------- *)
plotarrowgrid={};
Do[Do[
Do[
pt={i*deltax,j*deltay};
vect=arrowgrid[[correctedi[j],correctedj[i]]][[arrowct]];
arrowvec={pt,vect};
AppendTo[plotarrowgrid,arrowvec];
,{arrowct,1,Length[arrowgrid[[correctedi[j],correctedj[i]]]]}]
,{i,-im,im}]
,{j,-im,im}];

(* ---------------------------------- *)
(* Store arrow picture in a variable *)
(* ---------------------------------- *)
g1=ListPlotVectorField[plotarrowgrid,DisplayFunction->Identity,ScaleFactor->delt
ax];

(* ------------------------------------------- *)
(* Put together the list of boxes into a picture *)
(* ------------------------------------------- *)
colorboxlist={};
boxlist={};
```

```
grayboxlist={};
grayrectlist={};
stencilorderlist={};
(* -------------------- *)
(* Start with a red box *)
(* -------------------- *)
red=0;
green=0;
blue=1;
Do[
matrixi=topleftlist[[ct]][[1]];
matrixj=topleftlist[[ct]][[2]];
i= matrixj-im-1;
j=-matrixi+im+1;
pt1={i*deltax,j*deltay};
pt2={(i+1)*deltax,j*deltay};
pt3={(i+1)*deltax,(j-1)*deltay};
pt4={i*deltax,(j-1)*deltay};
pt5=pt1;
midpt={(i+1/2)*deltax,(j-1/2)*deltax};
If[  red==1,red=0;green=1;blue=0;ltype={1};gray=0.5,
If[green==1,red=0;green=0;blue=1;ltype={0.02,0.02};gray=.7,
If[ blue==1,red=1;green=0;blue=0;ltype={0.01,0.02,0.02,0.02};gray=.9]]];
AppendTo[colorboxlist,{RGBColor[red,green,blue],Line[{pt1,pt2,pt3,pt4,pt5}]}];
ltype={1};
AppendTo[boxlist,{Dashing[ltype],Line[{pt1,pt2,pt3,pt4,pt5}]}];
AppendTo[grayboxlist,{GrayLevel[gray],Line[{pt1,pt2,pt3,pt4,pt5}]}];
AppendTo[grayrectlist,{GrayLevel[gray],Rectangle[pt4,pt2]}];
AppendTo[stencilorderlist,{GrayLevel[1],Text[ct,midpt]}];
,{ct,1,Length[topleftlist]}];

g3=Show[Graphics[grayrectlist],DisplayFunction->Identity];
g4=Show[Graphics[boxlist],DisplayFunction->Identity];
g5=Show[Graphics[stencilorderlist],DisplayFunction->Identity];

Show[g3,g4,
Graphics[{PointSize[0.05],picturelist, {PointSize[.02],Point[{0,0}]}}],
g1,g2,g5,DisplayFunction->$DisplayFunction,PlotLabel->titlestring,Axes->True,
AspectRatio->Automatic];


(*
Show[Graphics[{PointSize[0.05],picturelist,{PointSize[.02],Point[{0,0}]}}],
g1,g2,g3,DisplayFunction->$DisplayFunction,PlotLabel->titlestring,Axes->True,
AspectRatio->Automatic,GridLines->{ticklist,ticklist}];
*)
);

makebesselfile:=(
stmp=OpenWrite["besselfile"];
deltax=N[1.0/iun,30];
```

```
(* ------------------------------------------------------- *)
(* Can use any of the eigenvalues from BesselJPrimeZeros[0,n] *)
(* ------------------------------------------------------- *)
bessellam:=3.831705970207512315614435886308l;
R[r_] := N[Sqrt[2.0] BesselJ[0,bessellam r]/BesselJ[0,bessellam],50];
Do[
Do[
physicalicoord=N[i*deltax,50];
physicaljcoord=N[j*deltax,50];
physicalr = N[Sqrt[physicalicoord^2+physicaljcoord^2],50];
besseld=5.0;
Write[stmp,N[besseld R[physicalr]/Sqrt[2 * Pi],50]];
(*
pressuregrid[[matrixicoord,matrixjcoord]]=N[d R[physicalr] / Sqrt[2 Pi]];
*)
,{i,-im,im}]
,{j,-im,im}];
Close[stmp];
);
(* ------------------------------------------------------------------------- *)
(* This procedure will return a set of 9 vectors showing the arrow or direction  *)
(* for each of the 9 grid points to be mapped to.  If they are an interior point *)
(* they do not get mapped, so assigned a none or {0,0} vector                    *)
(* ------------------------------------------------------------------------- *)
s7mapping[s7matrix_]:=(
Print["Calling s7mapping"];
(* p7 p8 p9 *)
(* p4 p5 p6 *)
(* p1 p2 p3 *)

(* ------------------------------------ *)
(* Set of all normalized vector directions *)
(* ------------------------------------ *)
none = {0,0};
up = {0,1};
upright = {1/Sqrt[2],1/Sqrt[2]};
right = {1,0};
downright = {1/Sqrt[2],-1/Sqrt[2]};
down = {0,-1};
downleft = {-1/Sqrt[2],-1/Sqrt[2]};
left = {-1,0};
upleft = {-1/Sqrt[2],1/Sqrt[2]};

(* ---------------------------------- *)
(* By s7 assumption, these do not vary *)
(* ---------------------------------- *)
(* It is important to check for undefined arrows   *)
(* Those marked undefined should be defined later                *)
(* ------------------------------------------------------------ *)
p1=undefined;
p2=none;
```

```
p3=undefined;
p4=undefined;
p5=none;
p6=undefined;
p7=undefined;
p8=up;
p9=undefined;

(* ------------------------------------------------------------------------ *)
(* Non-Double Fill s7 modified assumption                                   *)
(* ------------------------------------------------------------------------ *)
(* Row 5 of S7 Symmetrical Mapping Figure in Dissertation *)
(* Only the p7 changes direction for this case based on 2,2 *)
(* If no B nearby, p7 is left undefined *)
(* X X B X X *)
(* X X B X X *)
(* X o o X X *)
(* X . . X X *)
(* X o . X X *)

If[s7matrix[[5,2]]==2 && s7matrix[[4,2]]==1 && s7matrix[[3,2]]==2,
(* Default Settings *)
If[s7matrix[[2,2]]!=0 && s7matrix[[2,1]]!=0,
p1=downleft; p4=none; p7=up ];
If[s7matrix[[2,2]]==0,
p1=downleft; p4=none; p7=up ];
If[s7matrix[[2,1]]==0,
p1=downleft; p4=none; p7=upleft ];
];
(* ---------- *)
(* Right Side *)
(* ---------- *)
If[s7matrix[[5,4]]==2 && s7matrix[[4,4]]==1 && s7matrix[[3,4]]==2,
If[s7matrix[[2,4]]!=0 && s7matrix[[2,5]]!=0,
p3=downright; p6=none; p9=up ];
If[s7matrix[[2,4]]==0,
p3=downright; p6=none; p9=up ];
If[s7matrix[[2,5]]==0,
p3=downright; p6=none; p9=upright];
];

(* Row 6 of S7 Symmetrical Mapping Figure in Dissertation *)

(* X X B X X *)
(* X X B X X *)
(* X o o X X *)
(* X o . X X *)
(* X . . X X *)
If[s7matrix[[5,2]]==1 && s7matrix[[4,2]]==2 && s7matrix[[3,2]]==2,
(* Default Settings *)
If[s7matrix[[2,2]]!=0 && s7matrix[[2,1]]!=0,
```

```
p1=none; p4=upleft; p7=upleft ];
If[s7matrix[[2,2]]==0,
p1=none; p4=upleft; p7=up ];
If[s7matrix[[2,1]]==0,
p1=none; p4=upleft; p7=upleft ];
];
(* ---------- *)
(* Right Side *)
(* ---------- *)
If[s7matrix[[5,4]]==1 && s7matrix[[4,4]]==2 && s7matrix[[3,4]]==2,
(* Default Settings *)
If[s7matrix[[2,4]]!=0 && s7matrix[[2,5]]!=0,
p3=none; p6=upright; p9=upright ];
If[s7matrix[[2,4]]==0,
p3=none; p6=upright; p9=up ];
If[s7matrix[[2,5]]==0,
p3=none; p6=upright; p9=upright ];
];


(* Row 7 of S7 Symmetrical Mapping *)

(* X X B X X *)
(* X X B X X *)
(* X o o X X *)
(* X . . X X *)
(* X . . X X *)
If[s7matrix[[5,2]]==1 && s7matrix[[4,2]]==1 && s7matrix[[3,2]]==2,
(* Default Settings *)
If[s7matrix[[2,2]]!=0 && s7matrix[[2,1]]!=0,
p1=none; p4=none; p7=up ];
(* X X B X X *)
(* X B B X X *)
(* X o o X X *)
(* X . . X X *)
(* X . . X X *)
If[s7matrix[[2,2]]==0,
p1=none; p4=none; p7=up ];
(* X X B X X *)
(* B X B X X *)
(* X o o X X *)
(* X . . X X *)
(* X . . X X *)
If[s7matrix[[2,1]]==0,
p1=none; p4=none; p7=upleft ];
];

(* Right Side *)
If[s7matrix[[5,4]]==1 && s7matrix[[4,4]]==1 && s7matrix[[3,4]]==2,
(* Default Settings *)
If[s7matrix[[2,4]]!=0 && s7matrix[[2,5]]!=0,
```

```
p3=none; p6=none; p9=up ];
If[s7matrix[[2,4]]==0,
p3=none; p6=none; p9=up ];
If[s7matrix[[2,5]]==0,
p3=none; p6=none; p9=upright ];
];

(* ------------------------------------------------------------------------ *)
(* Double Fill with interior and fill cases using 5by5 info. only with modified*)
(* s7                                                                       *)
(* Here the outer points of 5 by 5 affects mapping.                         *)
(* ------------------------------------------------------------------------ *)


(* X X B X X *)
(* X X B X X *)
(* X o o X X *)
(* X o . X X *)
(* X o . X X *)

(* Left Side *)

If[(s7matrix[[5,2]]==2 && s7matrix[[4,2]]==2 && s7matrix[[3,2]]==2) ,


(* X X B X X *)
(* B B B X X *)
(* B o o X X *)
(* B o . X X *)
(* Z o . X X *)


If[s7matrix[[5,1]]!=0 && s7matrix[[4,1]]==0 && s7matrix[[3,1]]==0 &&
s7matrix[[2,2]]==0 ,
p1=upleft; p4=upleft; p7=up];


(* X X B X X *)
(* B B B X X *)
(* B o o X X *)
(* B o . X X *)
(* B o . X X *)
If[s7matrix[[5,1]]==0 && s7matrix[[4,1]]==0 && s7matrix[[3,1]]==0 &&
s7matrix[[2,2]]==0 ,
p1=left; p4=left; p7=up];

(* X X B X X *)
(* B B B X X *)
(* B o o X X *)
(* B o . X X *)
(* B o . X X *)
```

```
If[s7matrix[[5,1]]!=0 && s7matrix[[4,1]]!=0 && s7matrix[[3,1]]==0 &&
s7matrix[[2,2]]==0 ,
p1=downleft; p4=upleft; p7=up];


(* X X B X X *)
(* B B B X X *)
(* B o o X X *)
(* Z o . X X *)
(* B o . X X *)
If[s7matrix[[5,1]]==0 && s7matrix[[4,1]]!=0 && s7matrix[[3,1]]==0 &&
s7matrix[[2,2]]==0 ,
p1=left; p4=upleft; p7=up];



(* Degenerate *)
(* X X B X X *)
(* B B B X X *)
(* Z o o X X *)
(* B o . X X *)
(* Z o . X X *)
If[s7matrix[[5,1]]!=0 && s7matrix[[4,1]]==0 && s7matrix[[3,1]]!=0 &&
s7matrix[[2,2]]==0 ,
p1=left; p4=left; p7=up];

(* X X B X X *)
(* B B B X X *)
(* Z o o X X *)
(* B o . X X *)
(* B o . X X *)
If[s7matrix[[5,1]]==0 && s7matrix[[4,1]]==0 && s7matrix[[3,1]]!=0 &&
s7matrix[[2,2]]==0 ,
p1=left; p4=left; p7=up];

(* Degenerate *)
(* X X B X X *)
(* B B B X X *)
(* Z o o X X *)
(* Z o . X X *)
(* B o . X X *)
If[s7matrix[[5,1]]==0 && s7matrix[[4,1]]!=0 && s7matrix[[3,1]]!=0 &&
s7matrix[[2,2]]==0 ,
p1=left; p4=left; p7=up];

(* First ROW Done of S7 Symmetrical Mapping in Dissertation figure *)

(* Starting Second ROW *)
(* The first and second row have identical mappings, and will be handled by *)
(* the first 7 *)
(* mappings since s7matrix[[2,1]] is not compared above since its never mapped *)
```

```
(* to                                                                      *)
(* Done with Second ROW *)
(* Starting Third ROW *)
(* ------------------------------------------------------------------------ *)
(* Double Fill with interior and fill cases using 5by5 info. only with modified*)
(* s7                                                                      *)
(* Here the outer points of 5 by 5 affects mapping.                        *)
(* ------------------------------------------------------------------------ *)

(* X X B X X *)
(* B Z B X X *)
(* B o o X X *)
(* B o . X X *)
(* Z o . X X *)


If[s7matrix[[5,1]]!=0 && s7matrix[[4,1]]==0 && s7matrix[[3,1]]==0 &&
s7matrix[[2,1]]==0 ,
p1=upleft; p4=upleft; p7=upleft];

(* X X B X X *)
(* B B B X X *)
(* B o o X X *)
(* B o . X X *)
(* B o . X X *)
If[s7matrix[[5,1]]==0 && s7matrix[[4,1]]==0 && s7matrix[[3,1]]==0 &&
s7matrix[[2,1]]==0 ,
p1=left; p4=left; p7=upleft];

(* X X B X X *)
(* B B B X X *)
(* B o o X X *)
(* Z o . X X *)
(* Z o . X X *)
If[s7matrix[[5,1]]!=0 && s7matrix[[4,1]]!=0 && s7matrix[[3,1]]==0 &&
s7matrix[[2,1]]==0 ,
p1=downleft; p4=upleft; p7=upleft];

(* X X B X X *)
(* B B B X X *)
(* B o o X X *)
(* Z o . X X *)
(* B o . X X *)
If[s7matrix[[5,1]]==0 && s7matrix[[4,1]]!=0 && s7matrix[[3,1]]==0 &&
s7matrix[[2,1]]==0 ,
p1=left; p4=upleft; p7=upleft];

(* Degenerate *)
(* X X B X X *)
(* B Z B X X *)
(* Z o o X X *)
```

```
(* B o . X X *)
(* Z o . X X *)
If[s7matrix[[5,1]]!=0 && s7matrix[[4,1]]==0 && s7matrix[[3,1]]!=0 &&
s7matrix[[2,1]]==0 ,
p1=left; p4=left; p7=upleft];

(* X X B X X *)
(* B Z B X X *)
(* Z o o X X *)
(* B o . X X *)
(* B o . X X *)
If[s7matrix[[5,1]]==0 && s7matrix[[4,1]]==0 && s7matrix[[3,1]]!=0 &&
s7matrix[[2,1]]==0 ,
p1=left; p4=left; p7=upleft];

(* Degenerate *)
(* X X B X X *)
(* B Z B X X *)
(* Z o o X X *)
(* Z o . X X *)
(* B o . X X *)
If[s7matrix[[5,1]]==0 && s7matrix[[4,1]]!=0 && s7matrix[[3,1]]!=0 &&
s7matrix[[2,1]]==0 ,
p1=left; p4=left; p7=upleft];

(* Done with ROW 3 *)
(* Start ROW 4, think its last *)
(* ------------------------------------------------------------------------- *)
(* Double Fill with interior and fill cases using 5by5 info. only with modified  *)
(* s7                                                                        *)
(* Here the outer points of 5 by 5 affects mapping.                          *)
(* ------------------------------------------------------------------------- *)

(* Degenerate *)
(* X X B X X *)
(* Z Z B X X *)
(* B o o X X *)
(* B o . X X *)
(* Z o . X X *)


If[s7matrix[[5,1]]!=0 && s7matrix[[4,1]]==0 && s7matrix[[3,1]]==0 &&
s7matrix[[2,1]]!=0
  && s7matrix[[2,2]]!=0 ,
p1=upleft; p4=upleft; p7=up];


(* X X B X X *)
(* Z Z B X X *)
(* B o o X X *)
(* B o . X X *)
```

```
(* B o . X X *)
If[s7matrix[[5,1]]==0 && s7matrix[[4,1]]==0 && s7matrix[[3,1]]==0 &&
s7matrix[[2,1]]!=0
  && s7matrix[[2,2]]!=0 ,
p1=left; p4=left; p7=left];

(* X X B X X *)
(* Z Z B X X *)
(* B o o X X *)
(* Z o . X X *)
(* Z o . X X *)
If[s7matrix[[5,1]]!=0 && s7matrix[[4,1]]!=0 && s7matrix[[3,1]]==0 &&
s7matrix[[2,1]]!=0
  && s7matrix[[2,2]]!=0 ,
p1=downleft; p4=upleft; p7=up];

(* Degenerate *)
(* X X B X X *)
(* Z Z B X X *)
(* B o o X X *)
(* Z o . X X *)
(* B o . X X *)
If[s7matrix[[5,1]]==0 && s7matrix[[4,1]]!=0 && s7matrix[[3,1]]==0 &&
s7matrix[[2,1]]!=0
  && s7matrix[[2,2]]!=0 ,
p1=left; p4=upleft; p7=up];

(* Degenerate *)
(* X X B X X *)
(* Z Z B X X *)
(* Z o o X X *)
(* B o . X X *)
(* Z o . X X *)
If[s7matrix[[5,1]]!=0 && s7matrix[[4,1]]==0 && s7matrix[[3,1]]!=0 &&
s7matrix[[2,1]]!=0
  && s7matrix[[2,2]]!=0 ,
p1=left; p4=left; p7=up];

(* X X B X X *)
(* Z Z B X X *)
(* Z o o X X *)
(* B o . X X *)
(* B o . X X *)
If[s7matrix[[5,1]]==0 && s7matrix[[4,1]]==0 && s7matrix[[3,1]]!=0 &&
s7matrix[[2,1]]!=0
  && s7matrix[[2,2]]!=0 ,
p1=left; p4=left; p7=up];

(* Degenerate *)
(* X X B X X *)
(* Z Z B X X *)
```

```
(* Z o o X X *)
(* Z o . X X *)
(* B o . X X *)
If[s7matrix[[5,1]]==0 && s7matrix[[4,1]]!=0 && s7matrix[[3,1]]!=0 &&
s7matrix[[2,1]]!=0
  && s7matrix[[2,2]]!=0 ,
p1=left; p4=left; p7=up];

(* Done with ROW 4 *)

];
(* Right Side *)
If[(s7matrix[[5,4]]==2 && s7matrix[[4,4]]==2 && s7matrix[[3,4]]==2) ,


(* X X B X X *)
(* B B B X X *)
(* B o o X X *)
(* B o . X X *)
(* Z o . X X *)



(* Right Side *)
If[s7matrix[[5,5]]!=0 && s7matrix[[4,5]]==0 && s7matrix[[3,5]]==0 &&
s7matrix[[2,4]]==0 ,
p3=upright; p6=upright; p9=up];

(* X X B X X *)
(* B B B X X *)
(* B o o X X *)
(* B o . X X *)
(* B o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]==0 && s7matrix[[4,5]]==0 && s7matrix[[3,5]]==0 &&
s7matrix[[2,4]]==0 ,
p3=right; p6=right; p9=up];

(* X X B X X *)
(* B B B X X *)
(* B o o X X *)
(* B o . X X *)
(* B o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]!=0 && s7matrix[[4,5]]!=0 && s7matrix[[3,5]]==0 &&
s7matrix[[2,4]]==0 ,
p3=downright; p6=upright; p9=up];

(* X X B X X *)
```

```
(* B B B X X *)
(* B o o X X *)
(* Z o . X X *)
(* B o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]==0 && s7matrix[[4,5]]!=0 && s7matrix[[3,5]]==0 &&
s7matrix[[2,4]]==0 ,
p3=right; p6=upright; p9=up];


(* Degenerate *)
(* X X B X X *)
(* B B B X X *)
(* Z o o X X *)
(* B o . X X *)
(* Z o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]!=0 && s7matrix[[4,5]]==0 && s7matrix[[3,5]]!=0 &&
s7matrix[[2,4]]==0 ,
p3=right; p6=right; p9=up];

(* X X B X X *)
(* B B B X X *)
(* Z o o X X *)
(* B o . X X *)
(* B o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]==0 && s7matrix[[4,5]]==0 && s7matrix[[3,5]]!=0 &&
s7matrix[[2,4]]==0 ,
p3=right; p6=right; p9=up];


(* Degenerate *)
(* X X B X X *)
(* B B B X X *)
(* Z o o X X *)
(* Z o . X X *)
(* B o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]==0 && s7matrix[[4,5]]!=0 && s7matrix[[3,5]]!=0 &&
s7matrix[[2,4]]==0 ,
p3=right; p6=right; p9=up];

(* First ROW Done of S7 Symmetrical Mapping in Dissertation figure *)

(* Starting Second ROW *)
(* The first and second row have identical mappings, and will be handled by the
```

```
first 7 *)
(* mappings since s7matrix[[2,1]] is not compared above since its never mapped
to       *)
(* Done with Second ROW *)
(* Starting Third ROW *)
(* ------------------------------------------------------------------------- *)
(* Double Fill with interior and fill cases using 5by5 info. only with modified  *)
(* s7                                                                        *)
(* Here the outer points of 5 by 5 affects mapping.                          *)
(* ------------------------------------------------------------------------- *)

(* X X B X X *)
(* B Z B X X *)
(* B o o X X *)
(* B o . X X *)
(* Z o . X X *)


(* Right Side *)
If[s7matrix[[5,5]]!=0 && s7matrix[[4,5]]==0 && s7matrix[[3,5]]==0 &&
s7matrix[[2,5]]==0 ,
p3=upright; p6=upright; p9=upright];

(* X X B X X *)
(* B B B X X *)
(* B o o X X *)
(* B o . X X *)
(* B o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]==0 && s7matrix[[4,5]]==0 && s7matrix[[3,5]]==0 &&
s7matrix[[2,5]]==0 ,
p3=right; p6=right; p9=upright];

(* X X B X X *)
(* B B B X X *)
(* B o o X X *)
(* Z o . X X *)
(* Z o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]!=0 && s7matrix[[4,5]]!=0 && s7matrix[[3,5]]==0 &&
s7matrix[[2,5]]==0 ,
p3=downright; p6=upright; p9=upright];

(* X X B X X *)
(* B B B X X *)
(* B o o X X *)
(* Z o . X X *)
(* B o . X X *)
```

```
(* Right Side *)
If[s7matrix[[5,5]]==0 && s7matrix[[4,5]]!=0 && s7matrix[[3,5]]==0 &&
s7matrix[[2,5]]==0 ,
p3=right; p6=upright; p9=upright];


(* Degenerate *)
(* X X B X X *)
(* B Z B X X *)
(* Z o o X X *)
(* B o . X X *)
(* Z o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]!=0 && s7matrix[[4,5]]==0 && s7matrix[[3,5]]!=0 &&
s7matrix[[2,5]]==0 ,
p3=right; p6=right; p9=upright];

(* X X B X X *)
(* B Z B X X *)
(* Z o o X X *)
(* B o . X X *)
(* B o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]==0 && s7matrix[[4,5]]==0 && s7matrix[[3,5]]!=0 &&
s7matrix[[2,5]]==0 ,
p3=right; p6=right; p9=upright];


(* Degenerate *)
(* X X B X X *)
(* B Z B X X *)
(* Z o o X X *)
(* Z o . X X *)
(* B o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]==0 && s7matrix[[4,5]]!=0 && s7matrix[[3,5]]!=0 &&
s7matrix[[2,5]]==0 ,
p3=right; p6=right; p9=upright];

(* Done with ROW 3 *)
(* Start ROW 4, think its last *)
(* ------------------------------------------------------------------------ *)
(* Double Fill with interior and fill cases using 5by5 info. only with modified *)
(* s7                                                                       *)
(* Here the outer points of 5 by 5 affects mapping.                         *)
(* ------------------------------------------------------------------------ *)

(* Degenerate *)
```

```
(* X X B X X *)
(* Z Z B X X *)
(* B o o X X *)
(* B o . X X *)
(* Z o . X X *)



(* Right Side *)
If[s7matrix[[5,5]]!=0 && s7matrix[[4,5]]==0 && s7matrix[[3,5]]==0 &&
s7matrix[[2,5]]!=0
  && s7matrix[[2,4]]!=0 ,
p3=upright; p6=upright; p9=up];

(* X X B X X *)
(* Z Z B X X *)
(* B o o X X *)
(* B o . X X *)
(* B o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]==0 && s7matrix[[4,5]]==0 && s7matrix[[3,5]]==0 &&
s7matrix[[2,5]]!=0
  && s7matrix[[2,4]]!=0 ,

p3=right; p6=right; p9=right];

(* X X B X X *)
(* Z Z B X X *)
(* B o o X X *)
(* Z o . X X *)
(* Z o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]!=0 && s7matrix[[4,5]]!=0 && s7matrix[[3,5]]==0 &&
s7matrix[[2,5]]!=0
  && s7matrix[[2,4]]!=0 ,

p3=downright; p6=upright; p9=up];

(* Degenerate *)
(* X X B X X *)
(* Z Z B X X *)
(* B o o X X *)
(* Z o . X X *)
(* B o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]==0 && s7matrix[[4,5]]!=0 && s7matrix[[3,5]]==0 &&
s7matrix[[2,5]]!=0
  && s7matrix[[2,4]]!=0 ,
```

```
p3=right; p6=upright; p9=up];


(* Degenerate *)
(* X X B X X *)
(* Z Z B X X *)
(* Z o o X X *)
(* B o . X X *)
(* Z o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]!=0 && s7matrix[[4,5]]==0 && s7matrix[[3,5]]!=0 &&
s7matrix[[2,5]]==0 ,
p3=right; p6=right; p9=up];

(* X X B X X *)
(* Z Z B X X *)
(* Z o o X X *)
(* B o . X X *)
(* B o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]==0 && s7matrix[[4,5]]==0 && s7matrix[[3,5]]!=0 &&
s7matrix[[2,5]]!=0
  && s7matrix[[2,4]]!=0 ,
p3=right; p6=right; p9=up];


(* Degenerate *)
(* X X B X X *)
(* Z Z B X X *)
(* Z o o X X *)
(* Z o . X X *)
(* B o . X X *)

(* Right Side *)
If[s7matrix[[5,5]]==0 && s7matrix[[4,5]]!=0 && s7matrix[[3,5]]!=0 &&
s7matrix[[2,5]]!=0
  && s7matrix[[2,4]]!=0 ,
p3=right; p6=right; p9=up];

(* Done with ROW 4 *)

];
dirmat={{p7,p8,p9},{p4,p5,p6},{p1,p2,p3}};
dirmat
);

showfills:=(
(*
fillposlist=Position[thegrid,2];
```

```
*)
fillposlist=correctfillordering;
Do[
matrixi=fillposlist[[ct]][[1]];
matrixj=fillposlist[[ct]][[2]];
ict= matrixj-im-1;
jct=-matrixi+im+1;
oldphysicalpositionvector={ict*deltax,jct*deltax};
newphysicalpositionvector=Rotate2D[oldphysicalpositionvector,N[-theta],{0,0}];
newphysicalicoord=newphysicalpositionvector[[1]];
newphysicaljcoord=newphysicalpositionvector[[2]];
correctp = -N[(Cos[Sqrt[2] Pi physicaltime] Cos[Pi newphysicalicoord] Cos[Pi
newphysicaljcoord])];
If[thegrid[[matrixi,matrixj]]!=2,Print["Fill Error"]];
If[Abs[pressuregrid[[matrixi,matrixj,1,1]]-correctp]>.5,
Print["*Pressure[",matrixi,",",matrixj,"]=",pressuregrid[[matrixi,matrixj,1,1]],
" correctp =",correctp],
Print["Pressure[",matrixi,",",matrixj,"]=",pressuregrid[[matrixi,matrixj,1,1]],"
 correctp =",correctp]]
,{ct,1,Length[fillposlist]}];
);

(* ------------------------------------------------------------------------ *)
(* This procedure will compute the number of fills in the stencil for       *)
(* sorting purposes. Returns 0 if not a useful stencil                      *)
(* ------------------------------------------------------------------------ *)

getnumberoffillsinstencil2[testmatrix_]:=(
localtestmatrix=testmatrix;
lmat=testmatrix;
(* Quads *)
(*-------*)
(*|3 | 2|*)
(*---o---*)
(*|4 | 1|*)
(*-------*)
numberoffillsquad1=5;
numberoffillsquad2=5;
numberoffillsquad3=5;
numberoffillsquad4=5;
(* ---------------------------------- *)
(* Determine which 2 by 2 stencil to use *)
(* ---------------------------------- *)
(* x x x x x x x *)
(* x x x x x x x *)
(* x x x x x x x *)
(* x x x o . x x *)
(* x x x . . x x *)
(* x x x x x x x *)
(* x x x x x x x *)
(* Count number of fills in quad 1, 0 means don't use     *)
```

```
(* Trying to minimize number of fills in each stencil     *)
(* ------------------------------------------------------ *)
If[lmat[[5,5]]==1 || lmat[[5,4]]==1 || lmat[[4,5]]==1,
numberoffillsquad1=1;
If[lmat[[5,5]]==2, numberoffillsquad1++];
If[lmat[[4,5]]==2, numberoffillsquad1++];
If[lmat[[5,4]]==2, numberoffillsquad1++];
];
(* x x x x x x x *)
(* x x x x x x x *)
(* x x x . . x x *)
(* x x x o . x x *)
(* x x x x x x x *)
(* x x x x x x x *)
(* x x x x x x x *)
(* Count number of fills in quad 2, 0 means don't use     *)
(* Trying to minimize number of fills in each stencil     *)
(* ------------------------------------------------------ *)
If[lmat[[3,5]]==1 || lmat[[3,4]]==1 || lmat[[4,5]]==1,
numberoffillsquad2=1;
If[lmat[[3,5]]==2, numberoffillsquad2++];
If[lmat[[4,5]]==2, numberoffillsquad2++];
If[lmat[[3,4]]==2, numberoffillsquad2++];
];
(* x x x x x x x *)
(* x x x x x x x *)
(* x x . . x x x *)
(* x x . o x x x *)
(* x x x x x x x *)
(* x x x x x x x *)
(* x x x x x x x *)
(* Count number of fills in quad 3, 0 means don't use     *)
(* Trying to minimize number of fills in each stencil     *)
(* ------------------------------------------------------ *)
If[lmat[[4,3]]==1 || lmat[[3,3]]==1 || lmat[[3,4]]==1,
numberoffillsquad3=1;
If[lmat[[4,3]]==2, numberoffillsquad3++];
If[lmat[[3,3]]==2, numberoffillsquad3++];
If[lmat[[3,4]]==2, numberoffillsquad3++];
];
(* x x x x x x x *)
(* x x x x x x x *)
(* x x x x x x x *)
(* x x . o x x x *)
(* x x . . x x x *)
(* x x x x x x x *)
(* x x x x x x x *)
(* Count number of fills in quad 4, 0 means don't use     *)
(* Trying to minimize number of fills in each stencil     *)
(* ------------------------------------------------------ *)
If[lmat[[5,3]]==1 || lmat[[5,4]]==1 || lmat[[4,3]]==1,
```

```
numberoffillsquad4=1;
If[lmat[[5,3]]==2, numberoffillsquad4++];
If[lmat[[4,3]]==2, numberoffillsquad4++];
If[lmat[[5,4]]==2, numberoffillsquad4++];
];

bestquadnumber=0;
(* ---------------------------------------------------------------- *)
(* Now determine which quadrant has the fewest number of fill points *)
(* ---------------------------------------------------------------- *)
(* Quad 1 *)
(* ------ *)
If[numberoffillsquad1 <= numberoffillsquad2 &&
   numberoffillsquad1 <= numberoffillsquad3 &&
   numberoffillsquad1 <= numberoffillsquad4 &&
   numberoffillsquad1 <5,
bestquadnumber=1;numberoffills=numberoffillsquad1];

(* Quad 2 *)
(* ------ *)
If[numberoffillsquad2 <= numberoffillsquad1 &&
   numberoffillsquad2 <= numberoffillsquad3 &&
   numberoffillsquad2 <= numberoffillsquad4 &&
   numberoffillsquad2 <5,
bestquadnumber=2;numberoffills=numberoffillsquad2];

(* Quad 3 *)
(* ------ *)
If[numberoffillsquad3 <= numberoffillsquad2 &&
   numberoffillsquad3 <= numberoffillsquad1 &&
   numberoffillsquad3 <= numberoffillsquad4 &&
   numberoffillsquad3 <5,
bestquadnumber=3;numberoffills=numberoffillsquad3];

(* Quad 4 *)
(* ------ *)
If[numberoffillsquad4 <= numberoffillsquad2 &&
   numberoffillsquad4 <= numberoffillsquad3 &&
   numberoffillsquad4 <= numberoffillsquad1 &&
   numberoffillsquad4 <5,
bestquadnumber=4;numberoffills=numberoffillsquad4];

(* ---------------------------------------------------------------- *)
(* At this point, bestquadnumber is the correct quadrant to use with this *)
(* fill point.  If 0, then this fill point is not needed            *)
(* numberoffills contains number of fills using the correct quadrant      *)
(* Return this pair of information                                  *)
(* ---------------------------------------------------------------- *)

{bestquadnumber,numberoffills}
);
```

```
(* ------------------------------------------------------------------------ *)
(* This procedure will compute the number of fills in the stencil for       *)
(* sorting purposes. Returns 0 if not a s8 or s7 stencil                    *)
(* ------------------------------------------------------------------------ *)

getnumberoffillsinstencil[testmatrix_]:=(
localtestmatrix=testmatrix;
isgood=False;
isneeded=False;
ignore=False;
dolater=False;
rotations=xxx;
s8formrotations=False;
s7formrotations=False;
MatrixForm[testmatrix];
numberoffills=0;
Do[
(* -------------------------- *)
(* Test Diagonal s8 assumption *)
(* -------------------------- *)
If[localtestmatrix[[4,4]]==1,
  If[localtestmatrix[[1,1]]==0 && localtestmatrix[[2,2]]==0 &&
localtestmatrix[[5,5]]==1,
(* ------------------------------------------------------------------------ *)
(* It is s8, now count the number of fills in this 3 by 3 canonical stencil *)
(* ------------------------------------------------------------------------ *)
numberoffills8=1;
  If[localtestmatrix[[3,4]]==2,numberoffills8++];
  If[localtestmatrix[[3,5]]==2,numberoffills8++];
  If[localtestmatrix[[4,3]]==2,numberoffills8++];
  If[localtestmatrix[[5,3]]==2,numberoffills8++];
  If[localtestmatrix[[4,5]]==2,numberoffills8++];
  If[localtestmatrix[[5,4]]==2,numberoffills8++];
       isgood=True;rotations=ct-1; s8formrotations=ct-1];
  isneeded=True];
(* -------------------------- *)
(* Test Vertical s7 assumption *)
(* -------------------------- *)
If[localtestmatrix[[4,3]]==1,
  If[localtestmatrix[[1,3]]==0 && localtestmatrix[[2,3]]==0 &&
localtestmatrix[[5,3]]==1,
(* ------------------------------------------------------------------------ *)
(* It is s7, now count the number of fills in this 3 by 3 canonical stencil *)
(* ------------------------------------------------------------------------ *)
numberoffills7=1;
  If[localtestmatrix[[3,2]]==2,numberoffills7++];
  If[localtestmatrix[[3,4]]==2,numberoffills7++];
  If[localtestmatrix[[4,2]]==2,numberoffills7++];
  If[localtestmatrix[[4,4]]==2,numberoffills7++];
  If[localtestmatrix[[5,2]]==2,numberoffills7++];
```

```
  If[localtestmatrix[[5,4]]==2,numberoffills7++];
       isgood=True;rotations=ct-1; s7formrotations=ct-1];
  isneeded=True];
subtestmatrix=rotate5by5[localtestmatrix,1];
localtestmatrix=subtestmatrix;
,{ct,1,4}];
If[isneeded && Not[isgood],
Print["s8 or s7 assumption not holding at location
",matrixi,matrixj,localtestmatrix];
dolater=True];
If[Not[isneeded] && Not[isgood], Print["Ignore this fill"];ignore=True];

(* ------------------------------------------------------- *)
(* If both s8 and s7 occur, this will give preference to s8 *)
(* ------------------------------------------------------- *)
If[NumberQ[s8formrotations],numberoffills=numberoffills8,
If[NumberQ[s7formrotations],numberoffills=numberoffills7]];
numberoffills
);

(* ------------------------------------------------------------------------ *)
(* This procedure will return a list of fill points in the order of         *)
(* minimizing the boundary terms used.                                      *)
(* It will do this by ordering the fill points in the order of minimal fill  *)
(* points in the spatial interpolation stencil                              *)
(* ------------------------------------------------------------------------ *)
minimizeboundary2:=(


(* ------------------------------------------------------------- *)
(* Gather all the fill points in entire grid whether needed or not *)
(* ------------------------------------------------------------- *)
fillposlist=Position[thegrid,2];

fillctlist={};

Do[
(* ------------------------------------------------------ *)
(* Get the location of the fill point in matrix coordinates *)
(* ------------------------------------------------------ *)
matrixi=fillposlist[[ct]][[1]];
matrixj=fillposlist[[ct]][[2]];
(* ------------------------------------------------ *)
(* Get the 7 by 7 stencil with the fill in the center *)
(* ------------------------------------------------ *)
stencil7by7=SubMatrix[thegrid,{matrixi-3,matrixj-3},{7,7}];
(* ------------------------------------------------ *)
(* Count the number of fills in the s8 or s7 stencil *)
(* ------------------------------------------------ *)
{bestquadnumber,numberoffills}=getnumberoffillsinstencil2[stencil7by7];
(* ------------------------------------------------------------------------ *)
```

```
(* If bestquadnumber is zero, this fill has no adjacent interior; ignore it *)
(* Otherwise, make a list of all legal stencil fill counts for sorting      *)
(* ------------------------------------------------------------------------ *)
If[bestquadnumber!=0, AppendTo[fillctlist,{numberoffills,ct,bestquadnumber}]];
,{ct,1,Length[fillposlist]}];


(* ------------------------------------------------------------------------ *)
(* At this point have a list that needs sorted, each element has 3  parts *)
(* 1. Number of fills    2. index of fill point in fillposlist            *)
(* 3. correct quadrant to use with this fill point                        *)
(* ------------------------------------------------------------------------ *)

minimizeboundarylist=Sort[fillctlist];
maximizeboundarylist=Reverse[minimizeboundarylist];

returnlist={};

(* ------------------------------------------------------------------------ *)
(* Favor the interior data information by using stencils with fewest number *)
(* of fills.                                                                *)
(* ------------------------------------------------------------------------ *)
Do[
index=minimizeboundarylist[[ct]][[2]];
quadtouse=minimizeboundarylist[[ct]][[3]];
fillpointlocation=Append[fillposlist[[index]],quadtouse];
AppendTo[returnlist,fillpointlocation];
,{ct,1,Length[maximizeboundarylist]}];
(*
Do[
index=maximizeboundarylist[[ct]][[2]];
quadtouse=maximizeboundarylist[[ct]][[3]];
fillpointlocation=Append[fillposlist[[index]],quadtouse];
AppendTo[returnlist,fillpointlocation];
,{ct,1,Length[maximizeboundarylist]}];
*)
returnlist
);

(* ------------------------------------------------------------------------ *)
(* This procedure will return a list of fill points in the order of         *)
(* minimizing the boundary terms used.                                      *)
(* ------------------------------------------------------------------------ *)
minimizeboundary:=(


(* ------------------------------------------------------------- *)
(* Gather all the fill points in entire grid whether needed or not *)
(* ------------------------------------------------------------- *)
fillposlist=Position[thegrid,2];
```

```
fillctlist={};

Do[
(* ------------------------------------------------------- *)
(* Get the location of the fill point in matrix coordinates *)
(* ------------------------------------------------------- *)
matrixi=fillposlist[[ct]][[1]];
matrixj=fillposlist[[ct]][[2]];
(* ------------------------------------------------- *)
(* Get the 5 by 5stencil with the fill in the center *)
(* ------------------------------------------------- *)
stencil5by5=SubMatrix[thegrid,{matrixi-2,matrixj-2},{5,5}];
(* ------------------------------------------------- *)
(* Count the number of fills in the s8 or s7 stencil *)
(* ------------------------------------------------- *)
numberoffills=getnumberoffillsinstencil[stencil5by5];
(* ------------------------------------------------------------------------- *)
(* If numberoffills is zero, this fill has no s8 or s7 stencil so ignore it *)
(* Otherwise, make a list of all legal stencil fill counts for sorting      *)
(* ------------------------------------------------------------------------- *)
If[numberoffills>0, AppendTo[fillctlist,{numberoffills,ct}]];
,{ct,1,Length[fillposlist]}];


(* ------------------------------------------------------------------------ *)
(* At this point have a list that needs sorted, each element has two parts *)
(* 1. Number of fills    2. index of fill point in fillposlist            *)
(* ------------------------------------------------------------------------ *)

minimizeboundarylist=Sort[fillctlist];
maximizeboundarylist=Reverse[minimizeboundarylist];

returnlist={};

(* ------------------------------------------------------------------------ *)
(* Favor the interior data information by using stencils with fewest number *)
(* of fills.                                                                *)
(* ------------------------------------------------------------------------ *)
Do[
index=minimizeboundarylist[[ct]][[2]];
fillpointlocation=fillposlist[[index]];
AppendTo[returnlist,fillpointlocation];
,{ct,1,Length[maximizeboundarylist]}];
returnlist
);

(* ======================================================================== *)


(* ------------------------------------------------------------------ *)
(* This procedure will compute the normal and tangential derivatives *)
```

```
(* of a function.                                                    *)
(* ------------------------------------------------------------- *)
computederivative[thefunction_,dnormal_,dtangent_]:=(
Clear[ff,x,y];
mygrad[ff_]:= {D[ff,x],D[ff,y]};
derfntau[0,0] = thefunction;
(* ------------------------------------------------------------- *)
(* Take multiple directional derivatives in direction normal      *)
(* ------------------------------------------------------------- *)
Do[
derfntau[ct,0] = Simplify[mygrad[derfntau[ct-1,0]].{normx,normy}];
,{ct,1,dnormal}];
(* ----------------------- *)
(* The tangent . normal = 0 *)
(* ----------------------- *)
tangx=-normy;
tangy=normx;
(* ------------------------------------------------------------- *)
(* Take multiple directional derivatives in direction tau, tangent *)
(* ------------------------------------------------------------- *)
(*
derfntau[0] = derfn[dnormal];
*)
Do[
derfntau[dnormal,ct] = Simplify[mygrad[derfntau[dnormal,ct-1]].{tangx,tangy}];
,{ct,1,dtangent}];

Print["Collecting..."];
(*
Collect[derfntau[dtangent],pdata[_,_,_,_]]
FullSimplify[derfntau[dtangent]]
*)
(*
Collect[derfntau[dtangent],{D[HX0[_,_],{x,_},{y,_}],D[HX1[_,_],{x,_},{y,_}],D[HY
0[_,_],{x,_},{y,_}],D[HY1[_,_],{x,_},{y,_}]},FullSimplify]
Collect[derfntau[dtangent],D[thefunction,{x,_},{y,_}],FullSimplify]
Collect[derfntau[dtangent],variablelistp,Factor]
derfntau[dtangent]
Collect[derfntau[dtangent],{Derivative[_, _][HX0][_, _],Derivative[_,
_][HX1][_, _],Derivative[_, _][HY0][_, _],Derivative[_, _][HY1][_,
_]},FullSimplify]
*)
thehead=Head[thefunction];
Collect[derfntau[dnormal,dtangent],{Derivative[_,_][thehead][x,y]},FullSimplify]
);

(* ------------------------------------------------- *)
(* Procedure to manually build the derpntau function *)
(* ------------------------------------------------- *)

buildderf[infunction_,nder_,tder_]:=(
```

```
Clear[f,dx,dy,de,thecoef];
de=Table[Table[Table[Table[0,
 {dx,0,degree}]
,{dy,0,degree}]
,{ict,1,2}]
,{jct,1,2}];

(*
Print["In buildderf de = ",de];
*)

thefunc=infunction;
(* ------------------------------------------ *)
(* Handle the degenerate case of no derivatives *)
(* ------------------------------------------ *)
If[nder==0 && tder==0,
(* ------------------------------------------------------------ *)
(* Loop through all the data elements in the stencil adding the  *)
(* HX HY thecoef factor                                          *)
(* ------------------------------------------------------------ *)
Do[Do[
Do[Do[
termx=Symbol["HX"<>ToString[ict]<>"D"<>ToString[dx]][x];
termy=Symbol["HY"<>ToString[jct]<>"D"<>ToString[dy]][y];
de[[2-jct,ict+1,dx+1,dy+1]]+=(1) D[termx,{x,0}] D[termy,{y,0}];
,{dx,0,degree}]
,{dy,0,degree}]
,{ict,0,1}]
,{jct,0,1}]
];
If[Not[nder==0 && tder==0],
(* ------------------------------------------------------------------ *)
(* Calculate the normal and tangential derivative of undefined function *)
(* ------------------------------------------------------------------ *)
getcoeffunc=computederivative[f[x,y],nder,tder];

(* ------------------------------------------------ *)
(* Loop through all terms in the function's derivative *)
(* ------------------------------------------------ *)
Do[
dyct=(Length[getcoeffunc]-1)-dxct;
(* ------------------------------------------------------------------ *)
(* This is the coefficients that are for now comment to prevent symbol *)
(* explosion                                                          *)
(* ------------------------------------------------------------------ *)
thecoef[dxct,dyct]=Coefficient[getcoeffunc,Derivative[dxct,dyct][f][x,y]];
(*
thecoefdefinition[dxct,dyct]=Coefficient[getcoeffunc,Derivative[dxct,dyct][f][x,
y]];
*)
```

```
(* ------------------------------------------------------------ *)
(* Loop through all the data elements in the stencil adding the  *)
(* HX HY thecoef factor                                          *)
(* ------------------------------------------------------------ *)
Do[Do[
Do[Do[
termx=Symbol["HX"<>ToString[ict]<>"D"<>ToString[dx]][x];
termy=Symbol["HY"<>ToString[jct]<>"D"<>ToString[dy]][y];
(*
Print["Doing ",dxct,dyct,ict,jct,dx,dy,thecoef[dxct,dyct], D[termx,{x,dxct}],
D[termy,{y,dyct}]];
*)
de[[2-jct,ict+1,dx+1,dy+1]]+=thecoef[dxct,dyct] D[termx,{x,dxct}]
D[termy,{y,dyct}];
,{dx,0,degree}]
,{dy,0,degree}]
,{ict,0,1}]
,{jct,0,1}]
,{dxct,0,Length[getcoeffunc]-1}];
];
(* ------------------------------------------------------ *)
(* The complete partial f /over partial n partial tau is *)
(* ------------------------------------------------------ *)
(*
infunction[x_,y_] := Sum[Sum[Sum[Sum[
*)
Sum[Sum[Sum[Sum[pvarordata[2-jct,ict+1,dx+1,dy+1] de[[2-jct,ict+1,dx+1,dy+1]]
,{dx,0,degree}]
,{dy,0,degree}]
,{ict,0,1}]
,{jct,0,1}]
);

(* ------------------------------------------------------------------------ *)
(* Build all the equations required for solving all the data at a particular *)
(* fill point.                                                              *)
(* ------------------------------------------------------------------------ *)

buildequationsforafillpoint:=(
makeherm1d;

Clear[x,y,normx,normy,tangx,tangy];
equationct=1;
derpntaulist={};
deruntaulist={};
dervntaulist={};
nandtlist={{1,0},{3,0},{1,1},{3,1}};
(*
Do[
dnorm=nandtlist[[nandtct]][[1]];
dtau=nandtlist[[nandtct]][[2]];
```

```
*)
Do[Do[
If[equationct<=(degree+1)^2,
(* take derivatives in normal direction 2n+1 with n=0,1,2,3 ... *)
(* ---------------------------------------------------------------------- *)
(* Pressure wall boundary normal derivative = 0                           *)
(* partial^{2n+1} p / partial N^{2n+1} partial T^{t} = 0 for n,t = 0,1,2,3 *)
(* ---------------------------------------------------------------------- *)
Print["dnorm: ",dnorm," dtau : ",dtau];
thepequation=buildderf[prhs[x,y],dnorm,dtau];
AppendTo[derpntaulist,thepequation==0];
Clear[a1,b1,c1,d1];
AppendTo[deruntaulist,(thepequation/. {pvarordata[a1:_,b1:_,c1:_,d1:_]->tangx
uvarordata[a1,b1,c1,d1] + tangy vvarordata[a1,b1,c1,d1]})==0];
thevequation=buildderf[prhs[x,y],dnorm-1,dtau];
Clear[a1,b1,c1,d1];
AppendTo[dervntaulist,(thevequation/. {pvarordata[a1:_,b1:_,c1:_,d1:_]->normx
uvarordata[a1,b1,c1,d1] + normy vvarordata[a1,b1,c1,d1]})==0];
equationct++;
]
(*
,{nandtct,1,Length[nandtlist]}];
*)
,{dnorm,1,2*(degree+1),2}]
,{dtau,0,2*(degree+1)}];

(* -------------------------- *)
(* Define the H's symbolically *)
(* -------------------------- *)

Clear[x0,x1,y0,y1,xl,yl];
Clear["HX0*","HX1*","HY0*","HY1*"];

Do[
newsymbol1=Symbol["HX0D"<>ToString[dx]];
newsymbol1[x_]=Simplify[Coefficient[hermpoly1d[x],fdata[dx,x0]]];
newsymbol2=Symbol["HX1D"<>ToString[dx]];
newsymbol2[x_]=Simplify[Coefficient[hermpoly1d[x],fdata[dx,x1]]];
newsymbol3=Symbol["HY0D"<>ToString[dx]];
newsymbol3[y_]=newsymbol1[x] /. {x->y,x0->y0,x1->y1};
newsymbol4=Symbol["HY1D"<>ToString[dx]];
newsymbol4[y_]=newsymbol2[x] /. {x->y,x0->y0,x1->y1};
,{dx,0,degree}];

x0=-deltax/2;
x1= deltax/2;
y0=-deltax/2;
y1= deltax/2;

(* ------------------ *)
(* Define the thecoef *)
```

```
(* ------------------ *)
(*
thecoef[dx_,dy_]:=thecoefdefinition[dx,dy];
*)


);

makeherm1d:=(
If[Not[ValueQ[degree]],
degree=Input["Enter the degree: "];
];
Clear[x0,x1,fdiv];
ptlist=Join[Table[x0,{i,0,degree}],Table[x1,{i,0,degree}]];
Clear[fdiv];
If[Not[ListQ[ptlist]],
ptlist=Input["Enter the list of point locations"]];
Do[
Print["Column: ",column];
Do[
subptlist=Take[ptlist,{subparts,subparts+column}];
Print["fdiv[",Flatten[subptlist],"]"];
(* ---------------------- *)
(* Now assign fdiv[subptlist] *)
(* ---------------------- *)
If[Length[subptlist]==1,
fdiv[subptlist]=fdata[0,First[subptlist]]];
(* ------------------------------------------------------------------ *)
(* If the last and first index is at the same location then this is  *)
(* a derivative term divided difference and needs specially assigned *)
(* ------------------------------------------------------------------ *)
If[Length[subptlist]>1,
If[First[subptlist]===Last[subptlist],
fdiv[subptlist]=fdata[column,First[subptlist]]/(Length[subptlist]-1)!]];
(* ------------------------------------------------------------------ *)
(* If not, then apply Divided Difference Recurrence Relation to general *)
(* case.                                                              *)
(* ------------------------------------------------------------------ *)
If[Length[subptlist]>1,
If[First[subptlist]=!=Last[subptlist],
fdiv[subptlist]=(fdiv[Drop[subptlist,1]] - fdiv[Drop[subptlist,-1]])/
(Last[subptlist] -
First[subptlist]);
]];
(*
Print["Assigned : ",fdiv[subptlist]]
*)
,{subparts,1,Length[ptlist]-column}]
,{column,0,Length[ptlist]-1}];

(*
```

```
degree=2;
ptlist={x0,x0,x0,x1,x1,x1};
buildtab;
*)

hermpoly1d[x_]:=Collect[Sum[ Product[(x-ptlist[[j]]),{j,1,n-1}]
fdiv[Table[ptlist[[ptct]],{ptct,1,n}]] ,{n,1,Length[ptlist]}],fdata[_,_]];
);

(*
normxtemp[1,1]:=0;
normytemp[1,1]:=0;

normxtemp[1,1]:=-.707107;
normytemp[1,1]:= .707107;
normxtemp[2,1]:= .707107;
normytemp[2,1]:= .707107;
normxtemp[1,2]:= .707107;
normytemp[1,2]:=-.707107;

xcoord2[1,1]:=-0.664214;
ycoord2[1,1]:=0.25;

xcoord2[2,1]:=-0.707107;
ycoord2[2,1]:=-0.707107;

xcoord2[1,2]:=-0.207107;
ycoord2[1,2]:=0.707107;

xcoord2[1,1]:=-Sqrt[2]+.25;
ycoord2[1,1]:=-.25;

xcoord2[1,1]:=-.9+.25;
ycoord2[1,1]:=0.264214;
xcoord2[1,1]:=-.9+.25;
ycoord2[1,1]:=fy[xcoord2[1,1]];
y-(0.707107) = 1 ( x - -0.207107)
fy[-.9+.25]
fy[fx_]:= 1 ( fx - -0.207107) + (0.707107)
Clear[normxtemp,normytemp,xcoord2,ycoord2];

column=1;
Do[
Print["Row : ",ct," Column : ",column," = ",pmatrix[[ct,column]]]
,{ct,1,Dimensions[pmatrix][[1]]}]

oper[a_,b_]:=Expand[(nx dx + ny dy)^a (ny dx - nx dy)^b];
oper[a_,b_]:=Expand[(nx Derivative[1,0] + ny Derivative[0,1])^a (ny
Derivative[1,0]- nx Derivative[0,1])^b];
/. {Derivative[1,0]^c:_ ->Derivative[c,0], Derivative[0,1]^d:_
->Derivative[d,0]}
```

```
Through[operator[p[x,y]]] /. {[p[x,y]]->[p][x,y]}

*)
showneighbors[mi_,mj_]:=(
Clear[ldx,ldy,ict,jct];
milocal=mi;
mjlocal=mj;
(*
fillposlist=correctfillordering;
Do[
matrixi=fillposlist[[ct]][[1]];
matrixj=fillposlist[[ct]][[2]];
*)
Do[Do[
matrixi=milocal+ictloop;
matrixj=mjlocal+jctloop;
ict= matrixj-im-1;
jct=-matrixi+im+1;
oldphysicalpositionvector={ict*deltax,jct*deltax};
(* ------------------------------------------------ *)
(* Need to unrotate coordinates to get solution here *)
(* ------------------------------------------------ *)
newphysicalpositionvector=Rotate2D[oldphysicalpositionvector,N[-theta],{0,0}];
(*
newphysicalpositionvector=Rotate2D[oldphysicalpositionvector,N[theta],{0,0}];
*)
newphysicalicoord=newphysicalpositionvector[[1]];
newphysicaljcoord=newphysicalpositionvector[[2]];
Do[Do[
correctp = -N[(D[Cos[Sqrt[2] Pi physicaltime] Cos[Pi x] Cos[Pi
y],{x,ldx},{y,ldy}] /. {x->newphysicalicoord,y->newphysicaljcoord})];
(*
If[thegrid[[matrixi,matrixj]]!=2,Print["Fill Error"]];
*)
If[Abs[pressuregrid[[matrixi,matrixj,ldx+1,ldy+1]]-correctp]>.5,
Print["*Pressure[",matrixi,",",matrixj,",",ldx,",",ldy,"]=",pressuregrid[[matrix
i,matrixj,ldx+1,ldy+1]]," correctp =",correctp],
Print["Pressure[",matrixi,",",matrixj,",",ldx,",",ldy,"]=",pressuregrid[[matrixi
,matrixj,ldx+1,ldy+1]]," correctp =",correctp]]
,{ldx,0,degree}]
,{ldy,0,degree}]
,{ictloop,0,2}]
,{jctloop,0,2}];
);

(* ---------------------------------------- *)
(* Perform Gaussian Elimination with pivoting *)
(* From Cormen p. 754                       *)
(* ---------------------------------------- *)
lupdecomposition[matrix_]:=
Module[{a,n,pm,p,kp,k,i,j},
```

```
a=matrix;
n=Dimensions[a][[1]];
pm=Table[i,{i,1,n}];
Do[
   p=0;
   Do[
      If[Abs[a[[i,k]]]> p, p = Abs[a[[i,k]]]; kp = i];
   ,{i,k,n}];

   If[p=0,Print["Singular Matrix"]; Break];
   {pm[[k]],pm[[kp]]}={pm[[kp]],pm[[k]]};
   Do[
      {a[[k,i]],a[[kp,i]]}={a[[kp,i]],a[[k,i]]}
   ,{i,1,n}];
   Do[
      a[[i,k]] = a[[i,k]]/a[[k,k]];
      Do[
         a[[i,j]]=a[[i,j]]-(a[[i,k]] a[[k,j]])
      ,{j,k+1,n}];
   ,{i,k+1,n}];
,{k,1,n-1}];
{a,pm}
];

(* ------------------------------------------------------------------------ *)
(* Provide the solution to the linear system that was divided into an LU-Decom*)
(* ------------------------------------------------------------------------ *)
lupsolve[a_,pm_,b_]:=Module[{n,x,y,i,j},
n=Dimensions[a][[1]];
x=Table[0,{i,1,n}];
y=Table[0,{i,1,n}];
Do[
y[[i]]=b[[pm[[i]]]] - Sum[ a[[i,j]] y[[j]] , {j,1,i-1}];
,{i,1,n}];
Do[
x[[i]]= (y[[i]] - Sum[ a[[i,j]] x[[j]] ,{j,i+1,n}])/a[[i,i]]
,{i,n,1,-1}];
x
];

(* ---------------------------------------------------------------------- *)
(* This routine replaces LinearSolve since it is more numerically stable  *)
(* ---------------------------------------------------------------------- *)

solvesystem[matrix_,rhsvector_]:=Module[{sings,conditionnumber,det,a,pm,lhsvecto
r},
sings=SingularValues[N[matrix]][[2]];
conditionnumber=Sqrt[Max[sings]/Min[sings]];
det=Det[matrix];
Print["Matrix Condition #:",conditionnumber," and determinant :",N[det]];
{a,pm}=lupdecomposition[matrix];
```

```
lhsvector=lupsolve[a,pm,rhsvector];
Flatten[lhsvector]
];

(* ------------------------------------------------- *)
(* This routine uses low level C to compute the LU solve *)
(* ------------------------------------------------- *)

solvesysteminc[matrix_,rhsvector_]:=Module[{sings,conditionnumber,det,a,pm,lhsve
ctor},
Install["compiledsolvesystem.exe"];
(*
sings=SingularValues[N[matrix]][[2]];
conditionnumber=Sqrt[Max[sings]/Min[sings]];
det=Det[N[matrix]];
Print["Matrix Condition #:",conditionnumber," and determinant :",N[det]];
*)
lhsvector=compiledsolvesystem[Flatten[N[matrix]],Flatten[N[rhsvector]],Dimension
s[matrix][[1]]];
Print["Done with solve"];
Flatten[lhsvector]
];

(* ------------------------------------------------------------------ *)
(* This uses p. 763 Cormen to solve system in C by getting inverse then *)
(* multiplying in MMA for symbolic RHS *)
(* ---------------------------------- *)

solvesystemincnofc[matrix_,rhsvector_]:=Module[{sings,conditionnumber,det,a,pm,l
hsvector,lumatrixandpm,lumatrix},
(*
sings=SingularValues[N[matrix]][[2]];
conditionnumber=Sqrt[Max[sings]/Min[sings]];
det=Det[N[matrix]];
Print["Matrix Condition #:",conditionnumber," and determinant :",N[det]];
*)
Install["compiledinv.exe"];
n=Dimensions[matrix][[1]];
invmatrix=Partition[compiledinv[Flatten[N[matrix]],n],n];
lhsvector=invmatrix.rhsvector;
Flatten[lhsvector]
];
```

Bibliography

[1] Abromowitz,M.;Stegan,I.A. *Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables*, U.S. Dept. of Commerce, National Bureau of Standards, Applied Mathematics Series, 55, 1964.

[2] Adams,J.C.;Brainerd,W.S.;Martin,J.T.;Smith,B.T.;Wagener,J.K. *FORTRAN 90 Handbook*, McGraw-Hill, Inc., New York, 1992.

[3] "Aeronautics & Space Transportation Technology: Three Pillars for Success." Office of Aeronautics & Space Transportation Technology, NASA Headquarters Brochure, 1997.

[4] Agboola, O. "The Influence of Turbulence and Blade Geometry on the Acoustics of Turbomachinery." A Ph.D. Thesis, The University of Alabama, 1998.

[5] Anderson,D.A.;Tannehill,J.C.;Pletcher,R.H. *Computational Fluid Mechanics and Heat Transfer*, McGraw-Hill, New York, 1984.

[6] Anton,H. *Elementary Linear Algebra 5e* John Wiley & Sons, 1987.

[7] Abarbanel,S.; Ditkowski,A.;Yefet,A. *Bounded Error Schemes for the Wave Equation on Complex Domain.* Tel-Aviv University, ISRAEL, NASA Contract No. NAS1-19480 ICASE, 1995.

[8] Bangalore,A.;Morris,P.J.;Long,L.N."A Parallel Three-dimensional Computational Aeroacoustics Method Using Non-Linear Disturbance Equations", AIAA 96-1728, 1996.

[9] Batchelor,G.K. *An Introduction to Fluid Dynamics*, Cambridge University Press, 1967.

[10] Berger,M.J.;LeVeque,R.J. "An Adaptive Cartesian Mesh Algorithm for the Euler Equations in Arbitrary Geometries" AIAA-89-41777, 1989.

[11] Berger,M.J.;Oliger,J. "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations." *J. Comp. Physics*, Vol. 53, pp. 484-512, 1984.

[12] Berger,M.J,;LeVeque,R.L."Stable Boundary Conditions For Cartesian Grid Calculations." *Computing Systems in Engineering*, Vol. 1, Nos. 2-4, pp. 305-11, 1990.

[13] Bojanov,B.D.;Hakopian,H.A.;Sahakian,A.A. *Spline Functions and Multivariate Interpolations*, Kluwer Academic Publishers, 1993.

[14] Brualdi,R.A. *Introductory Combinatorics* North-Holland, 1992.

[15] Burden,R.L.;Faires,J.D. *Numerical Analysis: Fourth Edition* PWS-Kent Publishing Company, 1989.

[16] Cengel,Y.A.;Boles,M.A. *Thermodynamics: An Engineering Approach*, McGraw-Hill, New York, 1989.

[17] Coirier,W.J. "An Adaptively-Refined, Cartesian, Cell-Based Scheme for the Euler and Navier-Stokes Equations." NASA TM 106754, 1994.

[18] Collatz, L. *The Numerical Treatment of Differential Equations*, Springer-Verlag, 1960.

[19] Colonius,T.;Lele,S.K.;Moin,P. "Sound Generation in a Mixing Layer", *J. Fluid Mech.* Vol.330, pp.375-409, 1997.

[20] Cormen,T.H.;Leiserson,C.E.;Rivest,R.L. *Introduction to Algorithms.* McGraw-Hill Book Company, 1992.

[21] Cox,D.;Little,J.;O'Shea,D. *Ideals, Varieties, and Algorithms.* Springer-Verlag, 1992.

[22] Crighton,D.G."Basic Principles of Aerodynamic Noise Generation." *Prog. Aerospace Sci.*, Vol. 16, No. 1, pp. 31-96, 1975.

[23] Cullen,C.G. *Linear Algebra and Differential Equations*, PWS-Kent Publishing, 1991.

[24] Davis,A.D. *Classical Mechanics*, Academic Press, 1986.

[25] Ditkowski,A. "Bounded-Error Finite Difference Schemes for Initial Boundary Value Problems on Complex Domains." Ph.D. Thesis, Tel-Aviv University, 1997.

[26] Farin,G. *Curves and Surfaces for Computer Aided Geometric Design:A practical guide*,Academic Press, 1993.

[27] Forrer, H. "Second Order Accurate Boundary Treatment for Cartesian Grid Methods." Research Report No. 96-13, Seminar fur Angewandte Mathematik, Eidgenossische Technische Hochschule, CH-8092 Zurich, Switzerland, 1996.

[28] Fulks,W. *Advanced Calculus: an introduction to analysis*, John Wiley & Sons, 1978.

[29] Ganzha,V.G. *Numerical Solutions for Partial Differential Equations: Problem Solving Using Mathematica*, CRC Press, New York, 1996.

[30] Garabedian,P.R. *Partial Differential Equations*, Chelsea Publishing Company, New York, N.Y., 1986.

[31] Gerald,C.F.;Wheatley,P.O. *Applied Numerical Analysis.* Addison-Wesley Publishing Company, 1994.

[32] Godfrey,A.G.; Mitchell,C.R. and Walters,R.W. "Practical aspects of spatially high accurate methods". AIAA-92-0054, 1992.

[33] Goodrich, J.W. "Application of a New High Order Finite Difference Scheme to Acoustic Propagation With the Linearized Euler Equation." NASA TM 106454, 1993.

[34] Goodrich, J.W. "An Approach to the Development of Numerical Algorithms for First Order Linear Hyperbolic Systems in Multiple Space Dimensions:The Constant Coefficient Case." NASA TM 106928, 1995.

[35] Goodrich,J.W. "Accurate Finite Difference Algorithms." NASA TM 107377, 1996.

[36] Goodrich,J.W.;Hagstrom,T. "Accurate Algorithms and Radiation Boundary Conditions for Linearized Euler Equations." AIAA 96-1660, 1996.

[37] Goodrich,J.W. "High Accuracy Finite Difference Algorithms for Computational Aeroacoustics." AIAA 97-1584, 1997.

[38] Goodrich,J.W.;Hagstrom,T. "A Comparison of Two Accurate Boundary Treatments for Computational Aeroacoustics." AIAA 97-1585, 1997.

[39] Goodrich,J.W.;Dyson,R.W. "Automated Development of Accurate Algorithms and Efficient Codes for Computational Aeroacoustics." Computational Aerosiences Conference, NASA Ames, 1998.

[40] Goodrich,J.W.;Hardin,J. "Accurate Finite Difference Algorithms for Computational Aeroacoustics", *CFD Review 96*, John Wiley, 1996.

[41] Gottlieb,D.;Turkel,E. "Dissipative Two-Four Methods for Time-Dependent Problems." *Mathematics of Computation*, Vol 30, Number 136, 1976.

[42] Hagstrom,T. "On high-order radiation boundary conditions" *IMA Volume to Computational Wave Propagation*, 1996.

[43] Hagstrom,T. "On the convergence of local approximations to pseudodifferential operators with applications, Proceedings of the Third International Conference on Mathematical and Numerical Aspects of Wave Propagation Phenomena, E. Becache, G. Cohen, P. Joly and J. Roberts, eds." *SIAM* , pp. 474, 1995.

[44] Hagstrom,T. "Much Ado About Nothing – Radiation Boundary Conditions at Artificial Boundaries for Computational Aeroacoustics", *ICOMP News*, Vol. 3, No.2, 1997.

[45] Hagstrom,T. Private Communication, 1998.

[46] Halliday,D.;Resnick,R. *Fundamentals of Physics*, John Wiley & Sons, 1986.

[47] Hamming,R.W. *Numerical Methods for Scientists and Engineers* McGraw-Hill, 1973.

[48] Hardin,J.C. "Introduction to Computational Aeroacoustics", *Computational Fluid Dynamics Review*, 1996.

[49] Hart,J.F. *Computer Approximations*, John Wiley & Sons,Inc.,New York, 1968

[50] Harten,A.;Engquist,B.;Osher,S.;Chakravarthy,S.R."Uniformly Higher Order Accurate Essentially Non-oscillatory Schemes, III." *Journal of Computational Physics*, Vol 71, No.2, 1987.

[51] Helzer,G. "Gröebner Bases " *The Mathematica Journal*, Vol. 5 Issue 1, 1995.

[52] Hixon,R.;Shih,S.H.;Mankbadi,R.R. " Evaluation of Boundary Conditions for Computational Aeroacoustics" *AIAA Journal* , 33, 2006-2012, 1995.

[53] Hixon,R. "Evaluation of a High-Accuracy MacCormack-Type Scheme Using Benchmark Problems." *NASA CR 202324, ICOMP-97-03*, 1997.

[54] Hodgman,C.D.;Weast,R.C.; Wallace,C.W.;Selby,S.M.;eds. *Handbook of Chemistry and Physics* Chemical Rubber Publishing Co., 1954.

[55] Hu,F.Q. "On Absorbing Boundary Conditions for Linearized Euler Equations by a Perfectly Matched Layer" , *J. Comp. Phys.*, 129, pp.201-219, 1996.

[56] Hu,F.Q.;Hussaini,M.Y.;Mathey,J.L. "Low Dissipation and Low Dispersion Runge-Kutta Schemes for Computational Aeroacoustics." *J. Comput. Phys.*, 124, p. 177, 1996.

[57] Huff,D.L. "Fan Noise Prediction: Status and Needs." *AIAA-98-0177* , 1998.

[58] Janna,W.S. *Introduction to Fluid Mechanics*, PWS-Kent Publishing Company, 1987.

[59] Johansen,H. *A Cartesian Grid Embedded Boundary Method for Poisson's Equation on Irregular Domains*, 1997.

[60] Knuth,D.E. *The TeXbook.* American Mathematical Society and Addison-Wesley Publishing Company, 1986.

[61] Koepf,W. "Efficient Computation of Chebyshev Polynomials in Computer Algebra." Konrad-Zuse-Zentrum fur Informationstechnik Berlin (ZIB), Takustr. 7, D-14195, koepf@zib.de

[62] Kreiss,H.O.; Lorenz,J. *Initial-Boundary Value Problems and the Navier-Stokes Equations.* Academic Press, 1989.

[63] Kreiss,H.O.; Oliger, J. "Comparison of Accurate Methods for the Integration of Hyperbolic Equations," *Tellus*, vol. 24, pp. 199, 1972.

[64] Krishnamurthy,E.V. *Error-free Polynomial Matrix Computations*, Springer-Verlag, 1985.

[65] Kurbatskii,K.A.;Tam,C.K.W. "Cartesian Boundary Treatment of Curved Walls for High-Order Computational Aeroacoustics Schemes." *AIAA Journal*, Vol. 35, 133-140, 1997.

[66] Kwon,K.H.;Littlejohn,L.L. *Classification of classical orthogonal polynomials* 1993 preprint series No. 27, http://www.garc.snu.ac.kr/print/pre93/pre93_27.html

[67] Lamport,L. *A Document Preparation System: LaTeX: User's Guide & Reference Manual* Addison-Wesley Publishing Company, 1986.

[68] Lax,P.D.;Wendroff,B. "Systems of Conservation Laws." *Comm. Pure Appl. Math*, 13, pp.217, 1960.

[69] Lele,S.K. "Compact Finite Difference Schemes with Spectral-like Resolution." *Journal of Computational Physics*, Vol. 130, 1992.

[70] Lele,S.K. *Computational Aeroacoustics: A Review*, 35th Aerospace Sciences Meeting & Exhibit, 1997.

[71] Leighton,F.T. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, 1992.

[72] Lichtblau,D. *The Mathematica Journal*, Wolfram Researh, volume 7, issue 1, p.26, 1997.

[73] Lorentz,R.A. *Multivariate Birkhoff Interpolation*, Springer-Verlag, 1992.

[74] MacCormack,R.W. *Lecture Notes in Physics*, Vol. 8, Springer-Verlag, New York/Berlin, pp. 151, 1971.

[75] MacCormack,R.W. *Numerical Solution of the Interaction of a Shock Wave with a Laminar Boundary Layer,* Proc. 2nd Internat. Conf. on Numerical Methods in Fluid Dynamics, Lec. Notes in Phys., M. Holt, Editor, Springer-Verlag, New York, 1974.

[76] Mayr,E.; Meyer,A. "The complexity of the word problem for commutative semigroups and polynomial ideals." *Adv. Math.* 46, 305-329, 1982.

[77] Melton,J.E.;Berger,M.J.;Aftosmis,M.J.;Wong,M.D. "Development and Application of a 3D Cartesian Grid Euler Method." *NASA CP 3291*, pp.225-249, 1995.

[78] Melton,J.E.;Berger,M.J.;Aftosmis,M.J.;Wong,M.D. "3D Applications of a Cartesian Grid Euler Method." *AIAA-95-0853*, 1995.

[79] Melton,J.E.;Enomoto,F.Y. "3D Automatic Cartesian Grid Generation for Euler Flows." *AIAA-93-3386-CP*, 1993.

[80] M. Metcalf. *Effective FORTRAN 77* Oxford Science Publications, 1985.

[81] Mitchell,B.E.;Lele,S.K.;Moin,P. *Direct Computation of the Sound Generated by Subsonic and Supersonic Axisymmetric Jets*, Report No. TF-66, Thermosciences Division, Department of Mechanical Engineering, Stanford University, 1995.

[82] Morris,P.J.;Long,L.N.;Bangalore,A.;Chyczewski,T.; Lockard,D.;Ozyoruk,Y. *Experiences in the Practical Application of Computational Aeroacoustics*, Fluids Engineering Division Conference, Volume 3, ASME, 1996.

[83] Morse,P.M.;Ingard,K.U. *Theoretical Acoustics*, McGraw-Hill Book Company, 1968.

[84] Nilsson,J.W. *Electric Circuits.* Addison-Wesley, 1990.

[85] O'Neil,P.V. *Advanced Engineering Mathematics.* Wadsworth Publishing Company, 1991.

[86] Ozyoruk,Y. "Sound Radiation From Ducted Fans Using Computational Aeroacoustics on Parallel Computers", Ph.D. Thesis, Pennsylvania State Univeristy, 1995.

[87] Ozyoruk,Y.;Long,L.N. " A New Efficient Algorithm for Computational Aeroacoustics on Parallel Processors ", *Journal of Computational Physics*, Vol. 125, pp. 135-149, 1996.

[88] Pierce,A.D. *Acoustics: An introduction to its physical principles and applications*, McGraw-Hill Book Company, 1981.

[89] Powell, M.J.D., *Approximation theory and methods*, Cambridge University Press, 1981.

[90] Priestley,A. *Roe's Schemes, Euler Equations, Cartesian Grids, Non-Cartesian Geometries, Rigid Walls and all that* Numerical Analysis Report 14/87, 1987.

[91] Quirk,J.J. "An Alternative to Unstructured Grids for Computing Gas Dynamics Flows Around Arbitrarily Complex Two-Dimensional Bodies" *ICASE Report No. 92-7.*, 1992.

[92] Rangwalla,A.A.;Rai,M.M. "A Multizone High-Order Finite-Difference Method for the Navier-Stokes Equations." *AIAA-95-1706-CP*, 1995.

[93] Rumsey,C.L. "Computation of Acoustic Waves Through Sliding Zone Interfaces Using an Euler/Navier-Stokes Code." *AIAA-96-1752*, 1996.

[94] Saff,E.B.;Snider,A.D. *Fundamentals of Complex Analysis for Mathematics, science and engineering.* Prentice-Hall, Englewood Cliffs, N.J., 1993.

[95] Sedgewick,R. *Algorithms*, Addison-Wesley, 1988.

[96] Skudrzyk,E. *The Foundations of Acoustics*, Springer-Verlag, New York, 1971.

[97] Smith,G.D. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*, Clarendon Press, 1985.

[98] Smith,M.J. *Aircraft Noise*, Cambridge University Press, 1989.

[99] Stewart,G.W. *Introduction to Matrix Computations*, Academic Press, 1973.

[100] Strikwerda,J.C. *Finite Difference Schemes and Partial Differential Equations.* Wadsworth & Brooks/Cole Advanced Books & Software, 1989.

[101] Swokowski,E.W. *Calculus with Analytic Geometry.* PWS-KENT Publishing Company, 1988.

[102] Szego,G. "Orthogonal Polynomials." *American Mathematical Society Colloquium Publications Volume XXIII*, 1959.

[103] Takewaki,H.;Nishiguchi,A.;Yabe,T."Cubic Interpolated Pseudo-particle Method (CIP) for Solving Hyperbolic-Type Equations." *Journal of Computational Physics*, Vol. 61, pp. 261-268, 1985.

[104] Takewaki,H.;Yabe,T. "The Cubic-Interpolated Pseudo Particle (CIP) Method: Application to Nonlinear and Multi-dimensional Hyperbolic Equations." *Journal of Computational Physics*, Vol. 70, pp. 355-372, 1987.

[105] Tam,C.K.W.;Webb,J.C. "Dispersion-Relation-Preserving Finite Difference Schemes for Computational Acoustics." *Journal of Computational Physics*, Vol. 107, 1993.

[106] Tam,C.K.W.;Dong,Z. "Wall Boundary Conditions for High-Order Finite-Difference Schemes in Computational Aeroacoustics." *Theoret. Comput. Fluid Dynamics*, 6, pp. 303-322, 1994.

[107] Tam,C.K.W."Computation Aeroacoustics: Issues and Methods." *AIAA Journal*, Vol. 33, No. 10, 1995.

[108] Tam,C.K.W. "Advances in Numerical Boundary Conditions for Computational Aeroacoustics." *AIAA-97-1774*, 1997.

[109] Tam,C.K.W.;Konstantin,A.;Kurbatskii,A.;Fang,J. "Numerical Boundary Conditions for Computational Aeroacoustics Benchmark Problems", *NASA CP 3352*, pp. 191-219, 1997.

[110] Taylor,A.E. *Advanced Calculus*, Ginn and Company, 1955.

[111] Thompson,J.F. *Numerical Grid Generation.* Elsevier Science Publishing Company, Inc., 1982.

[112] Trefethen,L.N. "Group Velocity In Finite Difference Schemes." *SIAM Review*, Vol.24, No. 2, 1982.

[113] Trim,D.W. *Applied Partial Differential Equations*, PWS-Kent Publishing Company, 1990.

[114] *Unix V: The Quick Reference Guide* . Order Number 311544-002.NKR Computer Seminars. Intel Scientific Computers, 1989.

[115] Varga,R. *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, N.J., 1962.

[116] Vemuri,V.;Karplus,W.J. *Digital Computer Treatment of Partial Differential Equations* Prentice-Hall Series in Computational Mathematics, 1981.

[117] Viswanathan,K.;Sankar,L.N."Toward the Direct Calculation of Noise: FluidAcoustic Coupled Approach." *AIAA Journal*, Vol. 33, No.12, pp 2271-2279, 1995.

[118] Viswanathan,K.;Sankar,L.N. *A Comparative Study of Upwind and MacCormack Schemes for CAA Benchmark Problems*, ICASE/LaRC Workshop on Benchmark Problems in Computational Aeroacoustics, NASA CP 3300, Hampton , Va., p. 185-195, 1995.

[119] Wagner,D.B. *Power Programming with Mathematica* McGraw-Hill, New York, 1996.

[120] Winston,P.H. *Artificial Intelligence*, Addison-Wesley, 1984.

[121] Wolfram,S. *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley Publishing Company, 1991.

[122] Wolfram,S. *The Mathematica Book, 3rd ed.* Wolfram Media/Cambridge University Press, 1996.

[123] Wolfram Research *Mathematica 3.0 : Standard Add-On Packages* Cambridge University Press, 1996.

[124] Yabe,T. "A Universal Cubic Interpolation Solver for Compressible and Incompressible Fluids." *Shock Waves* Vol.1, pp. 187-195, 1991.

[125] Young,D.M.;Gregory,R.T. *A Survey of Numerical Mathematics*, Dover Publications, 1988.

[126] Zachmanoglou,E.C.;Thoe,D.W. *Introduction to Partial Differential Equations with Applications.* Dover Publications, New York, 1986.

[127] Zauderer,E. *Partial Differential Equations of Applied Mathematics.* John Wiley & Sons, 1989.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1999	Technical Memorandum

4. TITLE AND SUBTITLE

An Automated Code Generator for Three-Dimensional Acoustic Wave Propagation With Geometrically Complex Solid Wall Boundaries

5. FUNDING NUMBERS

WU–538–03–11–00

6. AUTHOR(S)

Rodger William Dyson, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
John H. Glenn Research Center at Lewis Field
Cleveland, Ohio 44135–3191

8. PERFORMING ORGANIZATION REPORT NUMBER

E–11691

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546–0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA TM—1999-209182

11. SUPPLEMENTARY NOTES

This report was submitted as a dissertation in partial fulfillment of the requirements for the degree Doctor of Philosophy to Case Western Reserve University, Cleveland, Ohio, May 1999. Responsible person, Rodger William Dyson, Jr., organization code 5940, (216) 433–9083.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Categories: 64, 71, 61, and 45 Distribution: Nonstandard

This publication is available from the NASA Center for AeroSpace Information, (301) 621–0390.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Finding the sources of noise generation in a turbofan propulsion system requires a computational tool that has sufficient fidelity to simulate steep gradients in the flow field and sufficient efficiency to run on today's computer systems. The goal of this dissertation was to develop an automated code generator for the creation of software that numerically solves the linearized Euler equations on Cartesian grids in three dimensional spatial domains containing bodies with complex shapes. It is based upon the recently developed Modified Expansion Solution Approximation (MESA) series of explicit finite-difference schemes that provide spectral-like resolution with extraordinary efficiency. The accuracy of these methods can, in theory, be arbitrarily high in both space and time, without the significant inefficiencies of Runge-Kutta based schemes. The complexity of coding these schemes was, however, very high, resulting in code that could not compile or took so long to write in FORTRAN that they were rendered impractical. Therefore, a tool in Mathematica was developed that could automatically code the MESA schemes into FORTRAN and the MESA schemes themselves were reformulated into a very simple form-making them practical to use without automation or very powerful with it. A method for automatically creating the MESA propagation schemes and their FORTRAN code in two and three spatial dimensions is shown with up to 29th order accuracy in space and time. Also, a method for treating solid wall boundaries in two dimensions is shown with up to 11th order accuracy on grid aligned boundaries and with up to 2nd order accuracy on generalized boundaries. Finally, an automated method for parallelizing these approaches on large scale parallel computers with near perfect scalability is presented. All these methods are combined to form a turnkey code generation tool in Mathematica that once provided the CAD geometry file can automatically simulate the acoustical physics by replacing the traditionally labor intensive tasks of grid generation, algorithm development, FORTRAN coding, and wall boundary treatments with automated algorithmic procedures

14. SUBJECT TERMS	15. NUMBER OF PAGES
Cartesian grid; Finite difference; Computational aeroacoustics; Parallel computing; Complex geometry; High order accuracy	356
	16. PRICE CODE: A16

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	