1-1-2009

Optimal multistage relaxation with automatic parameter selection

Alvin Wong
Ryerson University

Follow this and additional works at: <http://digitalcommons.ryerson.ca/dissertations>

Part of the [Mechanical Engineering Commons](#)

Recommended Citation

Wong, Alvin, "Optimal multistage relaxation with automatic parameter selection" (2009). *Theses and dissertations*. Paper 891.

This Thesis is brought to you for free and open access by Digital Commons @ Ryerson. It has been accepted for inclusion in Theses and dissertations by an authorized administrator of Digital Commons @ Ryerson. For more information, please contact bcameron@ryerson.ca.

OPTIMAL MULTISTAGE RELAXATION WITH AUTOMATIC PARAMETER SELECTION

By

Alvin Wong,
Bachelor of Engineering, Aerospace Engineering,
Ryerson University, 2006

ABSTRACT

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Masters of Applied Science

in the Program of

Mechanical Engineering

Toronto, Ontario, Canada, 2009

© Alvin Wong 2009

I hereby declare that I am the sole author of this thesis or dissertation.

I authorize Ryerson University to lend this thesis or dissertation to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis or dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Optimal Multistage Relaxation with Automatic Parameter Selection.

Masters of Applied Science, Mechanical Engineering, 2009

Alvin Wong

Department of Mechanical Engineering

Ryerson University

ABSTRACT

This research developed a numerical method that solves complicated fluid flow problems without requiring end-user expertise with the solver. This method is capable of obtaining a spatially accurate solution in the same time or better as a skilled user with a conventional solver. An explicit preconditioned multigrid solver was used in this research with a multistage relaxation method. The proposed method utilizes a database with optimized relaxation method parameters for different local flow and mesh conditions. The parameters are optimized for the relaxation method such that the error modes in a complex Fourier series expansion of the residual can be quickly reduced. The convergence time and iteration count of this method was compared against the same solver using default input values, as well as a pre-optimized solver, to simulate a skilled user for various geometries. Improvements in both comparisons were demonstrated.

ACKNOWLEDGEMENTS

This thesis was made possible with the guidance of Dr. Jason V. Lassaline. Your help and advice throughout this experience is greatly appreciated. Thank you for your time and help.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iv
LIST OF TABLES	vii
LIST OF FIGURES	ix
NOMENCLATURE	xi
CHAPTER 1	1
1.1 INTRODUCTION	1
CHAPTER 2	3
2.1 BACKGROUND INFORMATION	3
2.1.1 FINITE VOLUME METHOD	3
2.1.2 GRID GENERATION	6
2.1.3 PRECONDITIONERS	9
2.1.4 RELAXATION METHOD	13
2.1.5 MULTIGRID	23
CHAPTER 3	30
3.1 METHODOLOGY	30
CHAPTER 4	43
4.1 RESULTS	43
4.1.1 MAXIMIZE STEP SIZE	43
4.1.2 MINIMIZE 4 TH ORDER ARTIFICIAL DISSIPATION ON FLAT PLATE	45
4.1.3 SPATIAL ACCURACY COMPARISON	64
4.1.4 MINIMIZE 4 TH ORDER ARTIFICIAL DISSIPATION ON NACA0012 AIRFOIL	65
CHAPTER 5	67
5.1 CONCLUSION	67
APPENDIX	68
MATLAB source code	68
REFERENCES	84

LIST OF TABLES

Table 1 - Tabulated results for 1-D flow problem with RK3	22
Table 2 - Tabulated results for 1-D flow problem with using RK4.....	22
Table 3 - Tabulated results for 1-D flow problem with dt larger than 1.8	23
Table 4 - Input parameters for Optimizer	39
Table 5 - Range of Parameters for Optimizer.....	39
Table 6 - Optimization Input Parameters	44
Table 7 - Optimized Step size, h, coefficients.....	45
Table 8- Flow Parameters for coarse and fine Flow with CFL max at 0.8	46
Table 9 - Optimized Alpha and Beta coefficients for coarse and fine grids	46
Table 10 - Tabulated results for N0012	65

Figure 10 - Simple Multigrid Coarsening	24
Figure 11 - Agglomerated grids	25
Figure 12 - Directional Quasero-2 grids	26
Figure 13 - Quadrants representing the eig-values [7]	27
Figure 14 - Eigenvalues on a fine grid	28
Figure 15 - Eigenvalues on a coarsened grid	28
Figure 16 - Flow diagram	33
Figure 17 - Fourier Footprint	34
Figure 18 - Fourier Footprint with α_4 dissipation	35
Figure 19 - Signal Plot	36
Figure 20 - Database Retrieval Method	40
Figure 21 - Stability and frequency plots for $M=0.15$ with no 4th order artificial dissipation for case 1, 2, and 3	47
Figure 22 - Stability and frequency plots for $M=0.15$ using 4th order artificial dissipation for case 7, 8, and 9	49
Figure 23 - Maximum Aspect Ratio VS Convergence Time at $M=0.2$ and $Re=1e4$	50
Figure 24 - Maximum Aspect ratio VS Convergence time at $M=0.2$ and $Re=1e5$	52
Figure 25 - Maximum Aspect Ratio VS Convergence Time for $M=0.2$ and $Re=1e6$	52
Figure 26 - Maximum Aspect Ratio VS Convergence Time for $M=0.3$ and $Re=1e4$	54
Figure 27 - Maximum Aspect Ratio VS Convergence Time for $M=0.3$ and $Re=1e5$	54
Figure 28 - Maximum Aspect Ratio VS Convergence Time for $M=0.3$ and $Re=1e6$	55
Figure 29 - Maximum Aspect Ratio VS Convergence Time for $M=0.4$ and $Re=1e4$	55
Figure 30 - Maximum Aspect Ratio VS Convergence Time for $M=0.4$ and $Re=1e5$	56
Figure 31 - Maximum Aspect Ratio VS Convergence Time for $M=0.4$ and $Re=1e6$	56
Figure 32 - Maximum Aspect Ratio VS Iteration at $M=0.2$ and $Re=1e4$	57
Figure 33 - Maximum Aspect Ratio VS Iteration at $M=0.2$ and $Re=1e5$	58
Figure 34 - Maximum Aspect Ratio VS Iteration at $M=0.2$ and $Re=1e6$	58
Figure 35 - Maximum Aspect Ratio VS Iteration at $M=0.3$ and $Re=1e4$	60
Figure 36 - Maximum Aspect Ratio VS Iteration at $M=0.3$ and $Re=1e5$	60
Figure 37 - Maximum Aspect Ratio VS Iteration at $M=0.3$ and $Re=1e6$	62

LIST OF FIGURES

Figure 1 - FVM Process	5
Figure 2 - Rectilinear Grid	6
Figure 3 - Variable spaced rectangular grid	7
Figure 4 - Curvilinear Grid [5]	8
Figure 5 - Unstructured Triangular grid	8
Figure 6: Point Jacobi Method	14
Figure 7: Line Gauss-Seidel [7]	17
Figure 8: Runge Kutta Stability [10]	20
Figure 9 - 1-D property flow over test section	21
Figure 10 – Simple Multigrid Coarsening	24
Figure 11 - Agglomerated grids	25
Figure 12 - Directional Coarsened grids	26
Figure 13 - Quadrants representing the eigenvalues [2]	27
Figure 14 - Eigenvalues on a fine grid	28
Figure 15 - Eigenvalues on a coarsened grid	28
Figure 16 - Flow diagram	33
Figure 17 - Fourier Footprint	34
Figure 18 - Fourier Footprint with ϵ_4 dissipation	35
Figure 19 - Sigma Plot	36
Figure 20 - Database Retrieval Method	40
Figure 21 - Stability and frequency plots for $M=0.15$ with no 4th order artificial dissipation for case 1, 2, and 3	47
Figure 22 - Stability and Frequency plots for $M=0.15$ using 4th order artificial dissipation for case 7, 8, and 9	49
Figure 23 - Maximum Aspect Ratio VS Convergence time at $M=0.2$ and $Re=1e4$	50
Figure 24 - Maximum Aspect ratio VS Convergence time at $M=0.2$ and $Re=1e5$	52
Figure 25 - Maximum Aspect Ratio VS Convergence Time for $M=0.2$ and $Re=1e6$	52
Figure 26 - Maximum Aspect Ratio VS Convergence Time for $M=0.3$ and $Re=1e4$	54
Figure 27 - Maximum Aspect Ratio VS Convergence Time for $M=0.3$ and $Re=1e5$	54
Figure 28 - Maximum Aspect Ratio VS Convergence Time for $M=0.3$ and $Re=1e6$	55
Figure 29 - Maximum Aspect Ratio VS Convergence Time for $M=0.4$ and $Re=1e4$	55
Figure 30 - Maximum Aspect Ratio VS Convergence Time for $M=0.4$ and $Re=1e5$	56
Figure 31 - Maximum Aspect Ratio VS Convergence Time for $M=0.4$ and $Re=1e6$	56
Figure 32 - Maximum Aspect Ratio VS Iteration at $M=0.2$ and $Re=1e4$	57
Figure 33 - Maximum Aspect Ratio VS Iteration at $M=0.2$ and $Re=1e5$	58
Figure 34 - Maximum Aspect Ratio VS Iteration at $M=0.2$ and $Re=1e6$	58
Figure 35 - Maximum Aspect Ratio VS Iteration at $M=0.3$ and $Re=1e4$	60
Figure 36 - Maximum Aspect Ratio VS Iteration at $M=0.3$ and $Re=1e5$	60
Figure 37 - Maximum Aspect Ratio VS Iteration at $M=0.3$ and $Re=1e6$	62

Figure 38 - Maximum Aspect Ratio VS Iteration at $M=0.4$ and $Re = 1e4$	62
Figure 39 - Maximum Aspect Ratio VS Iteration at $M=0.4$ and $Re = 1e5$	63
Figure 40 - Maximum Aspect Ratio VS Iteration at $M=0.4$ and $Re = 1e6$	63
Figure 41 - Comparison between Blasius Solution with version B and C	64

NOMENCLATURE

Alphanumeric Symbols

ρ	Density
ϕ	Flow property per unit mass
t	Time
CV	Control Volume
S_ϕ	Source term of flow property ϕ per unit mass
$grad$	Gradient
$\hat{n}$	Unit normal vector
A	Area
V	Volume
Γ	Diffusion coefficient
Q	Solution vector of density, momentum and energy
R	Residual matrix
L_t	Time-stepping method
W	Solution vector of density, momentum and energy
P	Preconditioner
x, y	Two dimensional Cartesian coordinates
A, B	Inviscid flux Jacobians
C, D, E	Viscous flux Jacobians
Δt_{NS}	Time step
Δt_H	Hyperbolic time step
Δt_P	Parabolic time step
CFL	Courant Friedrichs Lewy number
u, v, u', v'	Velocity components in x, y
$\Omega_t, \Omega_x, \Omega_{xx}$	Characteristic variables
Λ	Diagonal eigenvalue matrix
λ	Eigenvalue
ζ	Grid stretching ratio
k	Time step
u	Solution at location i, j on the grid
m	Iterand
n	Number of nodes
f_i	$f(x_i), x_i$ are the vertices of the grid
$g(\theta_\alpha), \sigma$	Amplification factor
α	Wavelength of Fourier modes
R_A	Advective components of residual function
R_D	Dissipative and source components of residual function
α, β	Parameters of the multistage relaxation method
Γ	Switch for β coefficients
h	Multistage relaxation step size
er_μ	Local error
A_x, A_y	Inviscid flux Jacobians in x, y
e_2, e_4	Second and forth difference dissipations

Z	Complex Fourier series representation of the linearized residual vector
$\hat{Q}$	Solution vector time component
θ_x, θ_y	Fourier frequencies in x, y
w_i	Weighting factor

CHAPTER 1

1.1 INTRODUCTION

There are many methods and products used when solving Computation Fluid Dynamics (CFD) problems, some more complex than others. No matter which method or product is used, there are always a great deal of trade-offs. These trade-offs can range anywhere from end-user knowledge requirements to code complexity and stability. User knowledge such as the type of iterative method used or the numerical stability limits of the iterative method must be known. Code complexity can increase the number of input parameters required to use the solver. The desired result from any CFD problem is always a time and spatial accurate solution obtained in the shortest amount of time. The goal of this research is to present a method of solving complex flow problems using an automated selection of optimized solver parameters to obtain a spatial accurate solution in the shortest amount of time. Normally, the user would have to manually select global solver parameters such that a spatially accurate result can be computed in a reasonable time period without the user having prior solver experience and the flexibility to address a wide range of problems. Currently, CFD solvers are either tuned for specific cases or involve conservative settings which introduce large error requiring a large amount of time to compute. It is possible to reduce the time taken by manually tuning specific parameters that control the iterative method along with other methods such as multistage schemes or the application of multigrid and preconditioners. But, even with these methods in place, it is hard to predict how quickly and easily a solver will obtain a solution and who will be operating the solver.

Previous work performed on this topic also focused on optimizing the multistage relaxation scheme. In the paper by K.Hosseini and J. J. Alonso [20], the multistage coefficients were optimized. The 'fmincon' function in MATLAB was used which allows for single variable optimization [22]. The research in [20] was based on minimizing (or maximizing) the convergence time with a set of constraints. This method was only able to find local optima because of the limitations of the gradient-based method [20]. Other works that have been found on this research topic includes C.-H. Tai et al [20] research on residual smoothing of the multistage scheme [21]. The research presented in this paper only focused on Euler equations and did not explore two dimensional flow problems. It

modified the multistage method developed by Van Leer et al [23] by applying a smoothing parameter and an average residual function [21]. This work also showed improvements in convergence time with the aid of multigrid. Both these papers have shown different methods in optimizing convergence time in CFD solvers. The proposed method will also improve convergence time, but will build a relationship between the convergence time and the conditions of stability. Also, multi-objective optimization will be performed on a two-dimensional problem instead of the quasi-1D Euler equations.

By introducing a database of optimized parameters, it becomes the responsibility of the solver to select the best possible combination of parameters to use to obtain a solution. This reduces the burden on the user to research and understand the nature of that specific flow problem and the physical modelling requirements. The results from this research will demonstrate a solver with the ability to choose optimized parameters such that a spatially accurate solution may be obtained in a shorter period of time than conventional methods. The thesis will first describe the solver and existing methods for faster convergence. Then a description of the implication of the proposed method will be described. Finally, results based on flow over a flat plate case, as well as two-dimensional flow past a NACA0012 airfoil, will be presented.

CHAPTER 2

2.1 BACKGROUND INFORMATION

2.1.1 FINITE VOLUME METHOD

To obtain the results mentioned in the introduction, a solver that utilizes the finite volume method with a multistage relaxation scheme was implemented. The solver for this research solves flow problems after generating or loading a grid structure for given geometries. It then forms a spatial discretization of the governing equations using the finite volume method. To determine the steady state solution, a multistage relaxation method is used. To accelerate the convergence speed of the solution, a multigrid scheme was also implemented. Finally, a preconditioner was included in the solver to improve the condition number of the system of equations before they are solved to allow for a smoother iterative method. A section on each of the added attributes will be discussed in a separate section. The focus of this section is to understand the finite volume method (FVM).

FVM integrates a set of discretized governing equations over a control volume (CV). Two common phenomena occurring in fluid dynamics involve diffusion and convection, so the focus of this section will be mainly based on the explanation of the occurrence of diffusion and convection in the governing equations.

To devise an FVM, a basic set of governing equations must be defined. This governing equation may be taken from the general Reynolds transport theorem stated below [7].

$$\frac{\partial(\rho\phi)}{\partial t} + \text{div}(\rho\phi\vec{u}) = \text{div}(\Gamma\text{grad}\phi) + S_\phi \quad (1)$$

In words, the above equation states that the sum of the rate of change in property ϕ of the fluid element and net flux of ϕ through the C.V. boundary is equal to the flux of ϕ through C.V. due to surface diffusion and the rate of change of ϕ due to an external source. The integral form of the steady state diffusion terms of Equation (1) for a control volume is [7]

$$\begin{aligned}
& \int_{CV} \text{div}(\rho \phi \vec{u}) dV - \int_{CV} \text{div}(\Gamma \text{grad} \phi) dV - \int_{CV} S_\phi dV \\
& = \int_A \hat{n} \cdot (\text{div}(\rho \phi \vec{u})) dA - \int_A \hat{n} \cdot (\text{div}(\Gamma \text{grad} \phi)) dA - \int_{CV} S_\phi dV = 0
\end{aligned} \tag{2}$$

Where $\hat{n}$ = outward unit normal

This equation is applied to each control volume and can be iterated from an initial guess using a relaxation method such as Gauss-Seidel, Runge Kutta, or Successive over Relaxation (SOR) until a solution has been found. The solver in this research utilizes the FVM method in combination with a multistage relaxation method that includes the family of classic Runge-Kutta methods. Before applying the FVM scheme, the flow domain must be described with coordinates so it can be calculated. This is where a grid must be implemented. The following section discusses the different types of grids that may be used. After the grid is generated, the FVM scheme is applied to the grid. Figure 1 illustrates how the FVM solver is used and all the existing methods used to accelerate the solution and stabilize the solver. Figure (1) also shows a form of the process with an automated selection solver parameters added. The explicit semi-discrete form of the governing equations represented by FVM is given by the following [13]

$$\frac{dQ}{dt} + R(Q) = 0 \tag{3}$$

This equation represents the governing equations for solution vector Q . The residual function $R(Q)$ in the above equation represents the inviscid and viscous flux contributions, the artificial dissipation terms, and any source terms that may occur. For the steady state solution, time is neglected, so equation (3) becomes [13]

$$R = R(Q) = 0 \tag{4}$$

The vector R given in Equation (4) represents the resulting evaluation of the residual function for solution vector Q . The value of vector R theoretically should reduce to machine zero, but with the application of a relaxation method which will be introduced in a later section, a convergence limit will be set. This will be the form of the governing equations that will be solved by the solver.

Figure 1 - FVM Process

2.1.2 GRID GENERATION

As seen in the FVM section, a grid of locations is required for solving fluid problems using numerical methods. This section will discuss different grid structures and their implications for numerical solvers. When a numerical method computes a solution for a flow problem, it performs calculations on specific locations in the control volume denoted by coordinates. A grid, in the context of this research, refers to a series of points connected by lines to form a systematic structure used to describe the spatial relationship between points. The solver reduces any type of grid into a list of points, edges, and cells to complete the spatial discretization. There are many different types of grids, but they fall within two main categories, structured and unstructured. This section will describe the different types of structured grids followed by the unstructured grids.

The first and simplest type of grid is the uniform structured grid. There are three types of structured grids: Cartesian grids, rectilinear grids, and curvilinear grids. A Cartesian grid is composed of unit squares in 2-D and unit cubes in 3-D. Graph paper is an example of a Cartesian grid. This is the simplest grid to construct, but may not be suitable for complex geometries due to the inability to adapt to the structure of the geometry. Also, a uniform structure grid can be numerically stiff and may take a long time to converge if there is disparity in mesh spacing in one direction versus the other. Numerical stability on a uniform structured grid is limited to a uniform time stepping as all cells are of the same size. A rectilinear grid is one where the grid is broken into parallelograms or rectangles. An example of a rectilinear grid is shown in Figure (2). This grid has equal spacing in the x and y directions. This allows for an easy implementation of the FVM equations because the locations of each vertex can easily be indexed using a matrix of Cartesian coordinates.

Figure 2 - Rectilinear Grid

As can be seen in Figure (2), the neighbouring vertices of any point can be easily located. This type of grid may be simple to implement but generally slow to converge when using explicit

relaxation methods. The convergence rate of this method is dependent upon the stretching of the grid spacing in each direction. The convergence rate tends to decrease (becomes numerically stiffer) as the ratio of the spacing in the x versus y direction increases. After rectangular grids, there are rectilinear grids. These grids are similar to rectangular grids, but the spacing between each cell is not consistent. A rectilinear grid has irregular spacing in both the x and y directions. Figure 3 illustrates a rectilinear grid. Notice that the overall structure is similar to that of a rectangular grid such that the cells are rectangular in shape but the aspect ratio of each cell may differ.

Figure 3 - Variable spaced rectangular grid

This type of grid can focus on specific details on the flow. For instance, this type of grid is very useful when studying the boundary layer of a flow problem because more nodes can be forced into the region where a boundary layer is expected to occur to improve the spatial accuracy of the solution while maintaining the same total number of points. This is useful because when computing a flow solution as the boundary layer is often an important area of study. By focusing on this specific area, the computational load in terms of memory needed to store nodes does not increase, but the details of the specific location increases. This type of grid will be utilized in this research for different flow-over-flat-plate problems.

The last type of structured grid is the curvilinear grid. This grid is similar to the rectangular grid because all angles are perpendicular as shown in Figure 4. The curvilinear grid can use Cartesian coordinates or cylindrical coordinates. Cylindrical coordinates can be implemented simply but using Cartesian coordinates may not be as simple. A transformation to a computational grid will make this problem easier to solve using Cartesian coordinates. A physical curvilinear grid may be transformed into a rectilinear grid using a transformation. The problem is then worked on the computational grid and then transformed back onto the physical grid. This transformation is called mapping. The idea of mapping is to solve the problem on a simpler grid while preserving the original PDE type.

Figure 4 - Curvilinear Grid [5]

The next type of grid structure is the unstructured grid. Unstructured grids lack regular patterns. They are impossible to model using Cartesian coordinates because the vertices are not located in regular positions and thus require the storage of additional connectivity information. Unstructured grids generally involve shapes such as triangles, tetrahedral, and hexahedrons. Figure (5) is an illustration of an unstructured grid.

Figure 5 - Unstructured Triangular grid

Unstructured grids are generally harder to generate than structured grids, but are more spatially accurate for complex geometries. For simple problems such as a flat plate or a single element structure, a structured grid may be the better choice but, for multi element geometries or complex flows, an unstructured grid may capture the physical problem more accurately [6]. A drawback of unstructured grids is that they require more memory to store the extra connectivity information. Compared to the structured grid, where the next vertex may be easily located at $x+\Delta x$ and $y+\Delta y$, the unstructured grid does not have this feature to locate the next vertex. An unstructured grid needs to store the locations and connectivity of all the vertices using separate memory; therefore it will take longer to process and in turn generating longer computational times.

The solver used in this research can use both structured and unstructured grids for its computation. The way the information is stored is based on lists of points, edges, and cells, where:

Point is defined by x, y coordinates in space

Edge is defined by end points and neighbouring cells,

Cell is defined by bounding edges (equivalent to a control volume) [15]

By using this format, the storage method used for creating coarser grids as defined later in the multigrid section is the same. The solver uses a rectilinear grid to solve the flat plate case, and there is also an algorithm to coarsen the mesh while preserving the rectilinear grid, which is used with the multigrid scheme.

This section described the different types of grids; structured and unstructured. Structured grids consist of Cartesian, rectangular, rectilinear, and curvilinear grids and unstructured grids are normally composed using triangles, tetrahedral, and hexahedrons. Structured grids are easier to generate, but slower to converge and normally require a smaller time step when using iterative methods. Unstructured grids use more memory to store the connectivity of every node point on the grid, but are generally faster to converge. All grid types may be used in conjunction with multigrid schemes which will be discussed in later section.

2.1.3 PRECONDITIONERS

Since the discretized flow problem will be solved on a specified grid using a numerical method such as time-stepping schemes or relaxation methods, numerical stability of the solver must be considered. Numerical stability is important to ensure a solution can be obtained by the numerical method. In a steady-state problem, a numerical method is required to solve the residual of the flow problem as shown in Equation (4). The problem is solved either explicitly or implicitly using the governing parameters that control each iterative method. The stability of each problem depends on the eigenvalues of the residual vector and the iterative method itself. Preconditioners are used to condition the residual function so that the eigenvalues obtained from the residual function may be grouped in specific ways such that the time-stepping scheme can effectively dissipate the error from the domain. This is done such that the eigenvalues will be clustered in a region of strong damping away from the origin in the complex plane. A more in-depth look at different types of time-stepping

methods will be discussed in the next section. This section focuses on the different types of preconditioners and the application of the preconditioner in reducing overall computation time. There are many different types of preconditioners, both implicit and explicit. An example of an effective implicit filter is Allmaras's point-implicit block Jacobi preconditioner [1] which proved to be effective in clustering the eigenvalues of the residual away from the origin. Implicit schemes are good at dissipating both high and low frequencies of the Fourier error unlike explicit schemes, but can be harder to implement. Explicit preconditioners are used in the code for the same purpose as the implicit preconditioner mentioned above. There are many effective explicit preconditioners as well including both scalar and matrix preconditioners. This section will explain the difference between each type and its applications.

A common multiplying factor for a preconditioner is the Jacobian of the residual vector. The Jacobian matrix can be analytically derived and has been proven in [2] and [3] to be effective preconditioners. The Jacobian is a matrix, which contains all the partial derivatives of one vector's components versus another vector. In this research, the Jacobian is approximated using the diagonal blocks of dR/dQ to form a point Jacobi method. The basic formation of a preconditioned semi-discrete finite volume scheme has the following formulation:

$$L_t W + PR(W) = 0 \quad (5)$$

where: L_t represents the time-stepping method

W represents the state vector

P represents the preconditioner

$R(W)$ represents the residual of the state vector.

The state vector is the vector that contains the density, momentum, and the total energy terms. In two-dimensions, this vector is represented by

$$W = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix} \quad (6)$$

Note that role of the preconditioner is equivalent to the time-stepping scheme on the state vector. In two-dimensions for a finite difference method, the semi-discrete finite volume equation can be represented using the inviscid and viscous terms as shown below:

$$\frac{\partial W}{\partial t} + A \frac{\partial W}{\partial x} + B \frac{\partial W}{\partial y} = C \frac{\partial^2 W}{\partial x^2} + D \frac{\partial^2 W}{\partial y^2} + E \frac{\partial^2 W}{\partial x \partial y} \quad (7)$$

In the above equation, A and B represents the inviscid flux Jacobians and C, D, and E represents the viscous flux Jacobians. This equation can easily be transformed into Euler equations by neglecting the viscous terms.

The Navier-Stokes equations consist of 2 parts: hyperbolic and parabolic terms. The hyperbolic part is represented by the inviscid terms and the parabolic part is represented by the viscous parts. This affects the maximum time step that can be taken when using iterative methods. The time step can be estimated using both the parabolic and hyperbolic counter parts and their Jacobians.

For a scalar preconditioner, the spectral radii of the Jacobians are taken instead of using the entire Jacobian matrix. The spectral radii of a matrix is the largest eigenvalues of each block of the matrix. This method will reduce the amount of computation required but is not as spatially accurate as the matrix form of the preconditioner. To form the scalar preconditioner, the time steps are divided into hyperbolic and parabolic parts and since the total time step is composed of both the hyperbolic and parabolic time steps, the equation below describes how the time step on the semi-discrete finite volume method is chosen [2].

$$\Delta t_{NS}^{-1} = \Delta t_H^{-1} + \Delta t_P^{-1} \quad (8)$$

The hyperbolic time step is given by [2]

$$\Delta t_H^{-1} = \frac{1}{CFL_H} \left(\frac{\rho(A)}{\Delta x} + \frac{\rho(B)}{\Delta y} \right) \quad (9)$$

where the spectral radii of a Jacobian is represented by $\rho(\text{Jacobian})$.

The parabolic time step is given by [2]

$$\Delta t_P^{-1} = \frac{1}{CFL_P} \left(\frac{4\rho(C)}{\Delta x^2} + \frac{4\rho(D)}{\Delta y^2} + \frac{4\rho(E)}{\Delta x \Delta y} \right) \quad (10)$$

The factor of 4 arises from considering the worst-case scenario of a checkerboard mode, $W_{i,j} = \hat{W}(t)e^{i(\pi i + \pi j)}$, for which the coefficients of the second-difference stencil reinforce each other in both directions.[1] The overall time step Δt_{NS}^{-1} is then defined by both CFL_H and CFL_P . The hyperbolic CFL number defines the stability limit along the imaginary axis and the parabolic CFL number defines the stability on the real axis. By using this local scalar time step, the overall stiffness will be reduced. The scalar preconditioner is equal to the Navier-Stokes time step such that $P_{NS}^{-1} = \Delta t_{NS}^{-1}$. For the Euler equation, the parabolic equations are eliminated so the preconditioner is simply $P_{NS}^{-1} = \Delta t_H^{-1}$. After understanding how the scalar preconditioner is derived, it is also important to understand the stability limit associated with the preconditioner.

The stability limit of the numerical problem depends on both the preconditioner and the numerical dissipation chosen for the problem. Numerical dissipation depends upon the discretization scheme

used, such as forward differencing, central differencing, or upwinding scheme. For a scalar preconditioner, using central differences in space and forward differences in time, as explained in [2]

$$\Delta t \leq \min \left(\frac{2\nu}{a^2}, \frac{\Delta x^2}{2u} \right) \quad (11)$$

This inequality clearly shows that the numerical dissipation is directly related to the time step in the model. If the numerical dissipation is set as per the specifications below, then $\nu = |a| \frac{\Delta x}{2}$ and the CFL limit can be reduced to $\Delta t \leq \frac{\Delta x}{|a|}$. Following the example in [2] the convection-diffusion problem can then be decomposed into the representative eigenvalues for the flux Jacobian “a” matrix and the spectral radius can be used as the maximum time step for a stable solution.

Matrix preconditioners are similar to scalar preconditioners. The main difference between matrix and scalar preconditioners is that the matrix preconditioner does not use the spectral radii of the flux Jacobian. Instead the matrix preconditioner utilizes the entire flux Jacobian matrix including both the viscous and inviscid components when solving Navier Stokes equations. The following equation illustrates how the matrix preconditioner is created.

$$P_M^{-1} = \frac{1}{CFL_H} \left(\frac{A}{\Delta x} + \frac{B}{\Delta y} + \frac{2C}{\Delta x^2} + \frac{2D}{\Delta y^2} \right) \quad (12)$$

As demonstrated in [2], the preconditioner has a similar form to the 2nd/4th difference JST (Jameson Schmidt Turkel) scheme [2]. The solver used in this research uses an artificial dissipation scheme which is equivalent to the JST scheme.

The stability limit for the matrix preconditioner is derived in a similar way as the scalar preconditioner. The one-dimensional convection-diffusion equation will have the following form after decomposition into the eigenvalues. [2]

$$\Omega_t + \Lambda \Omega_x = \frac{\Delta x}{2} |\Lambda| \Omega_{xx} \quad (13)$$

This suggests that the time step can be a maximum of $\Delta t_k = \frac{\Delta x}{|\lambda_k|}$. Note that the scalar stability limit uses the exact same formula with the maximum eigenvalue, $|\lambda|$, instead of the entire eigenvalue matrix Λ . Consideration to the combination of preconditioners and residual is also important. A scalar preconditioner can be applied to both a scalar and matrix residual. The resulting residual will be stable for both configurations, but the matrix preconditioner will provide better dampening of

the error due to the increased spatial accuracy of each cell instead of using the spectral radii. Three configurations of preconditioner and residual that will be stable include the scalar-matrix, matrix-matrix, and scalar-scalar combinations. Also, as illustrated in [2], the combination of matrix preconditioner and residual results in the best clustering of the eigenvalues from the Fourier analysis.

The solver in this research is able to use both matrix and scalar preconditioner with a matrix residual. The solver implements a block Point-Jacobi relaxation preconditioner formed from the block diagonal matrix of the Jacobian matrix $\partial R / \partial w$. This preconditioner is used in conjunction with the implemented multistage relaxation method. The preconditioned form of the governing equations has the form $PR(Q) = 0$ to be consistent with equation (4) for a steady state solution. From this simplified form, it can be seen that the steady state solution does not change by adding a preconditioner as the residual will still equate to zero. A good preconditioner will reduce the number of iterations necessary to compute a feasible solution and is easy to implement.

2.1.4 RELAXATION METHOD

After describing how the preconditioner should be applied, a detail description of the relaxation method will be examined in this section. A relaxation method is an iterative method, which is not time accurate, used to solve the discretized set of steady governing equations [4]. There are many different types of relaxation methods including the Jacobi method, Gauss-Seidel method, Successive Over relaxation method (SOR), or the multistage relaxation method. Each of these methods has an associated convergence rate. The former two methods are closely related in that the convergence rate is based on the characteristic of the A matrix when solving the linear system $Ax=b$. The latter method is considered to be part of a group of iterative methods called predictor-corrector methods. This section will describe the methodology and formulation of each method. The stability and integration with multigrid theory will be discussed in a latter section.

The definition of a relaxation method is one that starts with an initial guess to the solution then advances with either a time marching or iterative method to obtain a steady solution. The method terminates according to satisfaction of a convergence criteria (e.g. L2-norm of the density residual) that is set by the user. For simplicity, all the methods described below will be based on a structured quadrilateral grid.

The simplest relaxation method is the point Jacobi method. Using the initial guess to start of the iterative process, new solutions are generated over subsequent iterations until a solution that satisfies the convergence criteria has been found. Each new solution corresponds to a different instant in time or time step. The point Jacobi method operates using 4 vertices on the current time step (k) to solve for the solution at one vertex (i,j) on the next time step ($k+1$) as illustrated in Figure (6) below.

Figure 6: Point Jacobi Method

The general Point Jacobi equation is [7]

$$u_{i,j}^{(k+1)} = \frac{1}{2(1 + \zeta^2)} \left[u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)} + \zeta^2 (u_{i,j+1}^{(k)} + u_{i,j-1}^{(k)}) \right] \quad (14)$$

Where $\zeta = \frac{\Delta x}{\Delta y}$ = stretching in the grid

k = time step

u = solution at location i,j on the grid

This method is called the Point Jacobi method because it iterates through every point in the mesh, one at a time. This method is advantageous because it is simple to implement but it generally takes longer to converge than other methods such as line Jacobi or Gauss-Seidel. Two other factors affecting convergence speed is the stretching in the grid, ζ , and the nature of the A matrix. Looking at the first factor, ζ , which is a ratio of distance between each vertex in the x direction compared to the distance between vertices in the y direction. For optimal performance, ζ should have a value of 1 which means that $\Delta x = \Delta y$, but that is not always the case especially when applied to viscous

problems. Economical grids used for viscous problems include regions of stretching which introduces numerical stiffness. With a larger value of ζ , the problem becomes more and more stiff, and slower to converge.

Now, note that the point Jacobi method solves every point using the previous iteration's values. This seems to be inefficient because this method does not use the most recent values at all times. For instance, when referring to Figure 6 notation, the method is currently solving for $u_{i,j}$ using $u_{i-1,j}$ and $u_{i+1,j}$ on the horizontal. Then it would move on to solve for $u_{i+1,j}$ using the old solution of $u_{i,j}$ instead of the value most recently calculated. By using such a method, more memory allotment and time will be taken to compute this solution. If points in this method were replaced with the most recent value at each vertex it would reduce the computational memory required and the time taken to iterate to a solution. This method is called the Point Gauss-Seidel method. As proposed Gauss-Seidel is similar to Point Jacobi such that it requires 4 stencil points to calculate a solution, but it differs in that the terms involving the previous solution such as $u_{i-1,j}$ and $u_{i,j-1}$ are at the $k+1$ time step. The formula is as follows [7]:

$$u_{i,j}^{(k+1)} = \frac{1}{2(1 + \zeta^2)} \left[u_{i+1,j}^{(k)} + u_{i-1,j}^{(k+1)} + \zeta^2 (u_{i,j+1}^{(k)} + u_{i,j-1}^{(k+1)}) \right] \quad (15)$$

This formulation will be implemented as such:

$$2u_1^m = u_2^{m-1} + h^2 c \quad (16a)$$

$$-u_{j-1}^m + 2u_j^m = u_{j+1}^{m-1} + h^2 f_j, j=2,3,4,\dots,2n-1 \quad (16b)$$

$$-u_{2n-1}^m + u_{2n}^m = \frac{1}{2} h^2 f_{2n} \quad (16c)$$

Where m = iterand

n = number of nodes

$f_i = f(x_i)$, x_i are the vertices of the grid

The convergence behaviour of these methods can be approximated using Fourier analysis, which is demonstrated further in "Introduction to Multigrid Methods" by P. Wesseling. In this analysis, the author replaced the boundary conditions with periodic boundary conditions so that $u(1) = u(0)$ [8]. From a Fourier analysis, a function $g(\theta_\alpha)$ was introduced as the amplification factor. This amplification factor indicates the growth or decay of the Fourier error modes. For the 1-D periodic boundary condition problem, the amplification factor has a relationship of [7]

$$|g(\theta_\alpha)| = (5 - 4 \cos \theta_\alpha)^{-1/2} \quad (17)$$

This function has a desired value of less than 1 to avoid uncontrolled growth of Fourier modes, but from this relationship, it seems that [7]

$$\max \left\{ |g(\theta_\alpha)| : \theta_\alpha = \frac{\pi\alpha}{n}, \alpha = -n+1, -n+2, \dots, n \right\} \quad (18)$$

where α represents the wavelength of the Fourier modes.

This would result in a value of $|g(0)| = 1$ which does not satisfy the desired convergence criteria of $|g(\theta_\alpha)| < 1$. This means that the solution does not decay and that the method does not converge. But taking a closer look, due to the periodic boundary conditions, $\alpha = 0$ is not required to decay during iteration. Therefore an extra constraint needs to be added to Equation (18). After the addition of this constraint, Equation (18) becomes

$$\max \left\{ |g(\theta_\alpha)| : \theta_\alpha = \frac{\pi\alpha}{n}, \alpha = -n+1, -n+2, \dots, n, \alpha \neq 0 \right\} = |g(\theta_1)| \quad (19a)$$

$$\max \{|g(\theta_\alpha)|\} = \{1 - 2\theta_1^2 + O(\theta_1^4)\}^{-1/2} \quad (19b)$$

$$\max \{|g(\theta_\alpha)|\} = 1 - 4\pi^2 h^2 + O(h^4) \quad (19c)$$

So it can be seen that the rate of convergence decreases as h decreases, where h is the time step. From the above formula, it can be seen that with larger wavelength Fourier modes, the solution decays slower compared to smaller wavelength Fourier modes. To address the problem of the different wavelengths of the Fourier modes, the method of multigrid was introduced. The purpose of multigrid is to approximate the long wavelength part of the error on the fine grid as a short wavelength on coarser grids. The non-smooth or rough part is reduced with a smaller number of iterations on the finest grid [8].

The Gauss-Seidel method may be modified to compute one row of vertices at a time. This is then called line Gauss-Seidel. This method of computing a line of solutions at a time is more efficient than computing every vertex individually per iteration. This method uses an implicit solver instead of the previous explicit solvers. The formulation of this method is as follows:

$$u_{i-1,j}^{k+1} - 2(1 + \zeta^2)u_{i,j}^{k+1} + u_{i+1,j}^{k+1} = -\zeta(u_{i,j+1}^k - \zeta^2 u_{i,j-1}^k) \quad (20)$$

When expanded into a linear system, solving for u^{k+1} involves inverting a tri diagonal matrix which can be done efficiently.

Figure (7) shows how the line Gauss-Seidel operates. As shown, the horizontal nodes are calculated at once using tri diagonal matrices. Each row is calculated using the previous row of the same iteration and the following row from the previous iteration. Along with this variation, there are other variations of the Gauss-Seidel method such as Block Gauss-Seidel, and Alternating Zebra Gauss-Seidel. P. Wesseling in "Introduction to Multigrid Methods" further explains these methods [8].

Figure 7: Line Gauss-Seidel [7]

A multistage relaxation method using "m" stages or sub iterations can be written as follows to solve a problem of the form of Equation (4)

$$Q_0 = Q^n \quad (21)$$

$$Q_j = Q_0 - \alpha_j h P \left(R_A Q_{j-1} + \sum_{k=0}^{j-1} \Gamma_{j,k} R_D Q_k \right) \quad (22)$$

$$Q^{(n+1)} = Q_m \quad (23)$$

The solution vector Q represents the solution vector at different time steps or stages. A 0 subscript denotes the initial solution at the current time step and before the start of the multistage method. A j subscript denotes the solution at the j-th iteration of the multistage method. The m subscript represents the solution at the end of the multistage method. The superscript n is the solution at the current time step and n+1 represents the solution at the next time step. The factor h is the time step taken for this iteration from n to n+1, while the factor $\alpha_j h$ can be thought of as a fractional time step.

The time step h is also known as the CFL number. Matrix P is the preconditioner and R_A and R_D contain the inviscid flux terms and remaining diffusive residual terms respectively. The last factor Γ is similar to the β terms presented in Equations (14 and 15) and can be constructed using new parameters β_j with the definition

$$\Gamma_{j,k} = \begin{cases} \beta_j & \text{if } k = j - 1 \\ (1 - \beta_j)\Gamma_{j-1,k} & \text{if } k \neq j - 1 \end{cases} \quad (24)$$

The values of α_j and β_j are restricted to the following values.

$$0 \leq \alpha_j \leq 1 \quad \forall j < m \quad (25a)$$

$$0 \leq \beta_j \leq 1 \quad \forall j > 1 \quad (25b)$$

$$0 < h \quad (25c)$$

The above inequality for α_j and β_j allows for $2(m-1)$ multistage parameters which can be tuned by the end-user to adjust the convergence performance. Also affecting the convergence performance is the second and forth difference dissipation terms k_2 and k_4 . The value of the second and forth dissipation terms directly affect the convergence rate as well as the spatial accuracy of the solution provided by the solver.

The multistage method described in this research falls under the predictor-corrector family of iterative methods. A simple example of predictor-corrector methods is the classical Runge-Kutta method which is a subset of the multistage relaxation method. This method iterates by using multiple stages where each stage uses different time steps. The Runge-Kutta method is a special subset of predictor-corrector methods because it was constructed based on a desired numerical stability limit. The following example utilizes a Runge-Kutta method to demonstrate how multistage relaxation methods are applied. Knowing the system of ordinary difference equations that result from applying the spatial discretization to the governing equations, one can solve for the system eigenvalues, λ , and in turn find the principle roots of the time marching equations denoted σ . The amplification function σ is equivalent to Wesseling's g function explained above.. Wesseling describes the amplification factor as $g(\theta_\alpha)$ while in this example it is described as $\sigma(\lambda h)$ where $\lambda(\theta)$. Further explanation of the stability analysis using the eigenvalues may be found in reference [9]. The focus now is to describe the formulation of the Runge Kutta method and its application as a multistage relaxation method. Now, the special aspect of the Runge Kutta method is that there is

just one σ -root produced for each λ -root so that $\sigma(\lambda h)$ corresponds to the Taylor series expansion of $e^{\lambda h}$ up to the truncation error of the method [9]. The higher the level of the Runge Kutta method the greater the spatial accuracy but also the greater computation time required. The principle σ -root can be represented by

$$\sigma = 1 + \lambda h + \frac{1}{2} \lambda^2 h^2 + \dots + \frac{1}{k!} \lambda^k h^k \quad (26)$$

This formula alone does not ensure k^{th} order spatial accuracy. To ensure that level of spatial accuracy, another constraint must be considered. This constraint is that the local error be order k

$$er_{\mu} = O(h^{k+1}) \quad (27)$$

The method follows a general algorithm of predictor followed by corrector. There are many types of predictor-corrector methods whose difference is only in the number of principle σ -root terms carried throughout the calculations. For example, the RK4 method follows the pattern of Euler Predictor, Euler Corrector, Leapfrog Predictor, and Milne Corrector [9]. In terms of formulations, the RK method has the general form of m steps or stages

$$u_0 = u^{(n)} \quad (28a)$$

$$u^{(n+\alpha_0)} = u_1 = u^{(n)} + \Delta t (\beta_{0,0} u_0') \quad (28b)$$

$$u^{(n+\alpha_1)} = u_2 = u^{(n)} + \Delta t (\beta_{1,0} u_0' + \beta_{1,1} u_1') \quad (28c)$$

⋮

$$u^{(n+\alpha_{k-1})} = u_{k-1} = u^{(n)} + \Delta t \sum_{j=0}^{k-2} (\beta_{k,j} u_j') \quad (28d)$$

$$u^{(n+1)} = u_m = u^{(n)} + \Delta t \sum_{j=0}^{m-1} (\beta_{m,j} u_j') \quad (28e)$$

Also, the restriction on β is

$$\alpha_i = \sum_{j=0}^i \beta_{i,j} \quad (29)$$

Where α = time sampling (intermediate time steps)

β = scaling coefficient

n = iterand

The RK methods may vary in number of steps but the steps most commonly used are 3, 4, and 5 denoted RK3, RK4 and RK5. The main difference between each of these RK methods is the size and shape of the stability limit contour. Figure 8 below illustrates the stability limits of each RK method.

Figure 8: Runge Kutta Stability [10]

As illustrated, the RK2 method is only stable for the negative real region of the complex λh -plane as compared to RK3 which extends into the positive real region. For RK4, it extends along the positive real axis proving to be more stable than the previous two RK methods. The disadvantage for RK4 is the processing time required to compute a solution. The iteration cost increases with increasing number of stages however it may be possible to use a larger time step. Therefore, a compromise between spatial accuracy, stability, computational resources, and time must be made. Generally, the RK4 method is widely accepted as the best compromise for the four criteria.

To better illustrate the Runge Kutta method, the example of the 1-D problem will be solved using the RK3 method. This example represents a convective-diffusive 1-D transfer of a property ϕ in a domain shown in Figure 9 [24]. In this example, the test section was 1 meter in length divided into 5 sections of equal length. The boundary conditions are $\phi_0 = 1$ and $\phi_L = 0$ at $x = 0$ and $x = L = 1\text{m}$ respectively. Also, a velocity of 0.1m/s was introduced along with a density of 1 kg/m^3 and a diffusion coefficient of 0.1kg/m/s [24]. Figure 9 below illustrates the boundary conditions for this flow problem.

Figure 9 - 1-D property flow over test section

The solution from the RK3 method should yield the same results as those from a direct solve of the linear problem. To implement the RK3 method in the simplest form, 6 β coefficients are needed.

The values of these coefficients are as follows,

$$\beta_{0,0} = 1/3$$

$$\beta_{1,0} = 0 \quad \beta_{1,1} = 1/2$$

$$\beta_{2,0} = 0 \quad \beta_{2,1} = 0 \quad \beta_{2,2} = 1$$

Applying these values to the RK3 method yields the following results after 37 iterations at a pseudo-time step of $\Delta t = 1.5$. The convergence limit was set to $1e-10$.

$$\begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \end{bmatrix} = \begin{bmatrix} 0.9421 \\ 0.8006 \\ 0.6276 \\ 0.4163 \\ 0.1579 \end{bmatrix}$$

These results are the exactly same as those obtained by a linear solve. The time taken to compute this solution at this specific time step was 0.225 seconds.

Note that the number of iterations and the time taken to converge changes as the pseudo-time step changes. This example was computed multiple times with different dt values to illustrate the convergence rate and the number of iterations taken at each pseudo-time step. The results are tabulated below, in Table 1.

Δt	Time taken(s)	Number of iterations
1.8	0.200868	30
1.7	0.205258	32
1.5	0.225622	37
1.0	0.311243	56
0.5	0.466403	111

Table 1 - Tabulated results for 1-D flow problem with RK3

For values higher than $\Delta t = 1.8$, the method became unstable and the results did not converge, and, for values lower than $\Delta t = 0.5$, the time taken and number of iterations kept increasing. This analysis shows the stability limit on the pseudo-time step limits to this problem to be approximately 1.8. This time step limit should vary with different RK schemes. To show this, the same example was computed using a RK4 scheme with β coefficients of the following values

$$\beta_{0,0} = 1/4$$

$$\beta_{1,0} = 0 \quad \beta_{1,1} = 1/3$$

$$\beta_{2,0} = 0 \quad \beta_{2,1} = 0 \quad \beta_{2,2} = 1/2$$

$$\beta_{3,0} = 0 \quad \beta_{3,1} = 0 \quad \beta_{3,2} = 0 \quad \beta_{3,3} = 1$$

The same values of Δt were used to compute the solution using the RK4 scheme and the following results were presented.

Δt	Time taken(s)	Number of iterations
1.8	0.195154	27
1.7	0.205923	29
1.5	0.210294	33
1.0	0.280905	49
0.5	0.472953	97

Table 2 - Tabulated results for 1-D flow problem with using RK4

Note that the times taken for both methods are approximately the same, but the number of iterations decreased for RK4. Also, for RK4 a larger stability limit is expected, so values of Δt larger than 1.8 were used and the following results were calculated.

Δt	Time taken(s)	Number of iterations
2.2	0.165809	22
2.1	0.170882	23
2.0	0.172548	24
1.9	0.174725	25

Table 3 - Tabulated results for 1-D flow problem with Δt larger than 1.8

For the RK4 method, the pseudo-time step can be increased to a value of 2.2 that results in a smaller number of iterations and faster time taken to compute the solution as compared to RK3. For a simple problem such as the 1-D heat conduction through a rod with only 5 nodes, the complexity of the method is not properly represented; therefore it seems that the difference in expense between RK3 and RK4 is minimal. For larger problems, the complexity of RK4 is far greater than RK3 that will increase the time taken to compute the solution although the RK4 method will be able to implement a larger pseudo-time step. The spatial accuracy in both methods would be the same as the mesh is identical for both examples.

2.1.5 MULTIGRID

After exploring the different types of grids and the different types of relaxation methods, this section will discuss and study the purpose of using multigrid methods. In short, a multigrid method is an iterative method used to accelerate the convergence of the solution by solving a forced form of the equations on a sequence of coarse grids such that the low frequency Fourier mode errors present on the finest grid will be removed quickly. Multiple grids are used with relaxation methods to further increase the convergence rate of the solution by disassembling the flow problem over a series of grids starting with the original grid, which would be considered the finest grid. After the solver iterates on the fine grid, the solver will then proceed to compute a solution correction on a coarse version of that grid. The purpose of this step is to remap the low frequency Fourier errors so that they become high frequency errors and the relaxation method can be tuned to quickly propagate or damp high frequency error modes, depending on the method chosen by the user. There are many different types of multigrid coarsening strategies including basic coarsening, agglomeration, and directional coarsening. Basic coarsening involves removing entire lines of the grid so that the general structure of the grid remains the same but with fewer nodes. Agglomeration involves merging of cells using a heuristic algorithm and depending on how the cells are fused; the iterative method may become less stiff on coarser grids. Directional coarsening is

implemented to reduce the stiffness in grids and involves coarsening in specific directions. This section will discuss the three mentioned multigrid methods.

The first method is coarsening the grid simply by removing every other line from the fine grid. This method will preserve the original structure as well as the stiffness. Figure 10 illustrates a two level multigrid process.

Figure 10 – Simple Multigrid Coarsening

As illustrated above, it can be seen that the computational load on the coarser grid is significantly reduced, but the aspect ratio of the x and y directions stays the same. This suggests that the stiffness of the coarser grid will not be a significant improvement on the fine grid. This multigrid method is the simplest form of multigrid because it requires the least modification to the original grid. This is an effective method when solving for grids that do not have a lot of stretching in the cells but is only applicable to structured meshes.

The next type of multigrid is the agglomeration method [18]. Agglomeration describes the act or process of gathering into a mass [11]. In terms of numerical methods and multigrid, agglomeration refers to the merging of cells and the removal of points and edges. Unlike coarsening by removing grid lines and retaining the original structure, agglomeration removes cells such that the neighbouring cells will merge to create a new cell. The following illustration, Figure 11, demonstrates a two level agglomerated grid.

Figure 11 - Agglomerated grids

Notice that by agglomerating the cells, the aspect ratio of each cell differs as the grid becomes coarser. This method can reduce some of the numerical stiffness and will allow for a quicker convergence in the solution. This method is very effective and easy to implement because the original nodes are still super imposed upon the fine grid so there are no new coordinates to store.

Finally, the last method described here is directional coarsening. Directional coarsening allows for the most control on how to coarsen the grid. As shown Figure 12, by removing every other horizontal line in the coarser grid, the aspect ratio between fine and the coarse grid have been reduced by half. This reduces the numerical stiffness and can be applied to areas with the most stretching of the cells. Directional coarsening is good for structured grids where a specific direction can easily be described. This method requires a more complex approach (directionally coarsened agglomeration) for unstructured grids due to the fact that unstructured grids are composed of geometries that have no prescribed directions.

Figure 12 - Directional Coarsened grids

All three methods were created for the purpose of propagating Fourier frequencies out of the domain or damping the error as quickly as possible. This is done because, on each level of the grid, the higher frequency errors will be diminished faster than the low frequencies. So, to accelerate the convergence of the overall solution, coarser grids are used to remap the low frequency errors so that they become high frequencies on the coarser grid. The same iterative method is then applied on all grid levels so that the previous low frequency modes will be diminished faster. This will aid the overall convergence of the solution. Multigrid cannot improve the convergence rate of a solution on its own. This must be coupled with a high-frequency damping relaxation method for an effective solution to improve the convergence rate of the solution.

Multigrid works to quickly eliminate the lower Fourier frequencies as mentioned above, but the details of how this is done was omitted in the above sections. As described in the Fourier analysis section of the report below, the governing equations can be converted into a complex Fourier series. The relaxation method is tuned to eliminate the high frequency modes, but it is slower to eliminate the lower frequency Fourier modes from the domain. A multigrid acts to transform the lower

modes into higher frequency modes by expanding the dimensions of each cell of the grid as the grid becomes coarser. To further understand this concept, attention must be given to the Fourier analysis and the graph of the amplification factor $|\sigma(\lambda h)|$ of the relaxation method. The plot of the amplification is shown in Figure 13 and is divided into four quadrants. As the figure illustrates, the graph is in terms of Fourier wave angle θ in the x and y direction for a 2-d problem.

Figure 13 - Quadrants representing the eigenvalues [2]

In the above Figure 13, H_x and H_y represents the high frequencies in x and y, respectively. The L_x and L_y represents the lower frequencies in x and y. An efficient relaxation method will damp the error from the three shaded quadrants on the grid leaving the quadrant with the low frequencies for the next level of the multigrid scheme. Iterating the solution on the coarse grid effectively maps the $L_x L_y$ region to the entire four quadrants, doubling the frequency in both the x and y direction. Iterating with the same relaxation method on the coarse grid thus eliminates the higher frequency components of the original low frequency region. This may be applied recursively by adding more coarse grids to the sequence thereby providing more effective damping of all error modes. Figure 14 illustrates that by coarsening the grid and reducing the number of nodes, the remaining low frequency eigenvalues will be magnified on the coarse Fourier quadrant plot.

Figure 14 - Eigenvalues on a fine grid

On the fine grid, as shown above, the green areas represent the frequency modes that the relaxation scheme will dissipate, with example system eigenvalues λ indicated by dots. The white area is the low frequency area which on this level of the grid, the relaxation method will handle poorly. Figure 15 illustrates this same system mapped to the coarse grid. Figures 14 and 15 combined show how high frequency eigenvalues are eliminated.

Figure 15 - Eigenvalues on a coarsened grid

A multigrid can be used with both explicit and implicit time marching schemes. An explicit scheme offers low operation count, low storage requirements, and good parallel scalability, but they suffer from the limited stability imposed by the CFL condition [2]. Implicit schemes can offer unconditional

stability but are more computationally intensive, require a heavy storage overhead, and are more difficult to parallelize efficiently. Also, it is impractical to directly invert the solution matrix for complicated problems so different factorizations such as LU or ADI must be used. But by using these factorization methods, the time steps are limited to a certain value, so the unconditional stability clause is limited to a certain range of time steps. Because of the overly complex structure of the implicit scheme, the solver used for this research uses an explicit multistage relaxation scheme with a multigrid scheme.

CHAPTER 3

3.1 METHODOLOGY

This experiment was conducted on an explicit multigrid solver as explained in the background section. The explicit solver includes all the conditioning that will accelerate the solver such that a solution can be obtained in the shortest period of time with the most spatial accuracy. This section will explain the methodology of the experiment, as well as the methodology used to compare results.

When solving numerical problems, the governing equations must be written in the form of an algebraic expression for the computer to solve the problem. As explained in the background section of the report, the simplified algebraic expression that needs to be solved is as follows.

$$dQ/dt = R(Q) \quad (34)$$

For steady state problems, the $\frac{dQ}{dt}$ term is zero, therefore the expression becomes

$$R(Q) = 0 \quad (35)$$

where: Q is the solution vector, and

$R(Q)$ is the residual function

A block Jacobi preconditioner was also used with the explicit solver, so the actual equation becomes

$$PR(Q) = 0 \quad (36)$$

It is important to remember that this expression is non-linear due to the attributes of $R(Q)$. Since the purpose of this research is to optimize the stability and convergence time for the multistage relaxation method for fixed and varying grids, the parameters that control the method such as α, β , and the step size h will be optimized. This research will optimize the multistage method for minimum convergence time subject to the constraint of numerical stability. To be able to predict convergence rates and numerical stability of the solver, a Fourier analysis must be performed. Looking at the current set of equations to be solved, it is impossible to perform a complex Fourier series because it is non-linear. A requirement for Fourier analysis is that the equations must be linear. Therefore, to predict the stability and convergence rate of a solver, the governing equations

must be linearized, in addition to being discretized in both space and time. For this particular problem, a Fourier analysis is completed for a representative structured mesh location. A better understanding of this concept can be obtained by revisiting the equations in a non-linear form. As described above, $R(Q)$ represents the residual of a solution vector. The given Q vector represents the solution at every node location in the mesh. For a two-dimensional structured mesh, Q can be represented at a point as the sub-vector $Q_{i,j}$. The residual sub-vector at this location then becomes

$$R_{i,j} = R_{i,j}(Q) \quad (37)$$

The above equation is still in a non-linear form and the solution vector cannot be factored from the residual function. Linearizing the above equation about the local solution using a truncated Taylor series allows us to write the residual vector as the product of a matrix and the solution vector as shown below,

$$\Delta R_{i,j} \approx \frac{\partial R}{\partial Q} \cdot \Delta Q_{i,j} \quad (38)$$

The solution at an interior node (x_i, y_i) can be represented by a complex Fourier series in space as follows

$$Q_{i,j} = \hat{Q}(t) e^{i(k_x \Delta x i + k_y \Delta y j)} = \hat{Q}(t) e^{i(\theta_x i + \theta_y j)} [16] \quad (39)$$

Where: k_x, k_y are wave numbers in the x and y directions

Δ_x, Δ_y are mesh spacing in the x and y directions

θ_x, θ_y are the wave angle in the x and y directions

This representation illustrates both the amplitude and spatial components of the local solution vector Q . Combining this with the spatial discretization introduced by the structured mesh results in a complex Fourier series representation of the linearized residual vector R , for which each Fourier mode, for each location in the mesh, can be written as

$$Z = (e_2 - 2e_4(\cos \theta_x + \cos \theta_y - 2)) (|A_x|(\cos \theta_x - 1) + |A_y|(\cos \theta_y - 1)) - i(A_x \sin \theta_x + A_y \sin \theta_y) [13] \quad (40)$$

where: e_2, e_4 represents the switch that acts to blend the second and forth dissipation

A_x, A_y represent the inviscid flux Jacobians

The above equation is a complex equation with both real and imaginary components. The real components of this equation represent the artificial dissipation in the solution while the imaginary part represents convection from the inviscid flux terms. In this research, only the inviscid flux and artificial dissipation terms are considered in the above linearization. The assumption to ignore the viscous terms is valid due to the high Reynolds numbers considered in this application of the governing equations which will effectively reduce the relative importance of the viscous terms from the governing equations. On highly stretched meshes, the contribution of the viscous terms to numerical stiffness is significantly smaller than the inviscid terms. Thus, it is neglected without distorting the model dramatically. Excluding regions with shocks, the dissipation parameters tend to follow the trend $e_2=0$ and $e_4=k_4$ on the finest mesh in the multigrid sequence. For regions with shocks the dissipation switches to approximately $e_2 = 1$ and $e_4 = 0$. For the coarse grids in the multigrid sequence, the spatial accuracy of the solution is not important; hence $e_2=k_2=1$ and $e_4=0$ is used which is equivalent to 1st order upwinding on a structured mesh.

The residual vector block at location (x_i, y_j) for each Fourier mode can be written as

$$R(Q_{i,j}) = Z\hat{Q}_{i,j} \quad (41)$$

With the addition of the Block Jacobi preconditioner, the linearized, semi-discrete form of each Fourier mode is [17]

$$\frac{d\hat{Q}}{dt} = PZ\hat{Q} \quad (42)$$

To be able to determine the stability and convergence rate, the eigenvalues of the PZ matrix will be used along with the σ - λ relationship of the relaxation method. Recall that the Z matrix is a complex Fourier expansion of each mode that represents the residual of the solution matrix calculated at each node point in the mesh. Since the residual matrix depends on the spatial discretization and flow conditions, the complex PZ matrix also depends on such conditions. It is important to remember that the PZ matrix is a linearization about the local flow solution so the conditions that affect the PZ matrix are local conditions and not global conditions such as boundary conditions. In addition, when linearizing the equations, artificial dissipation terms were included in the Z matrix so

on top of the localized conditions, the PZ matrix will also depend on the dissipation coefficients. From this information, it can be concluded that the eigenvalues of the PZ matrix will depend on local conditions such as Mach number and flow alignment angle. It will also depend on the local mesh dimensions, Δx and Δy . Lastly, the eigenvalues also depend on the dissipation scaling terms e_2 and e_4 . Figure 16 is a diagram of the representative local flow conditions on each mesh cell.

Figure 16 - Flow diagram

Since the procedure is desired to work with non-dimensionalized values, the flow can be decomposed into its non-dimensionalized components shown below

$$u' = M \cos(\theta) \quad (43a)$$

$$v' = M \sin(\theta) \quad (43b)$$

Where θ is the flow alignment angle.

The Mach number is already non-dimensionalized but the grid spacing can be represented by an aspect ratio, $\frac{\Delta x}{\Delta y}$. Finally, the dissipation coefficients e_2 and e_4 must also be considered. These dissipation terms are already non-dimensional.

Recall the relaxation method controlling parameters α , β , and h . These three parameters influence the stability and convergence rate of this relaxation method. The relationship between the eigenvalues λ and relaxation method is given by the amplification factor σ . The general relationship between amplification factor σ and the eigenvalues of the complex form of the residual is given by

$$|\sigma(\lambda h)| \quad (44)$$

The conditions for stability in this expression requires

$$|\sigma| \leq 1$$

The specific forms of each RK method was described in the background section of the report. Knowing the eigenvalues of the complex residual and the σ relationship, the numeric stability can be plotted as a Fourier footprint as described in the previous section. Three pieces of information can be obtained from a Fourier footprint. Recall that the Fourier footprint is the plot of the eigenvalues versus σ on the complex plane. The first attribute is the boundary where $|\sigma| = 1$, this is represented by the black outline as shown in Figure 17 below.

Figure 17 - Fourier Footprint

The second attribute is the contour flood plot of amplification factor within the stable region. Finally, the last and most important attribute is the blue line which represents the boundary of all eigenvalues of matrix PZ and scaled by h. Since the relaxation method is more efficient with low sigma values, this Fourier footprint can be modified two ways to accommodate the desired configuration. The first way is to increase the step size, h, which will increase the size of the boundary containing the eigenvalues of the PZ matrix. This method allows for a much larger step size that may possibly allow the relaxation method to eliminate the errors by convection. This method of reducing the errors will only be feasible if the problem has non-reflective boundaries so these error modes can be carried out of the domain. By increasing h the blue boundary will become closer to the black boundary which represents $\sigma = 1$. This can become a problem because the non-

linear terms may make the method unstable as σ approaches 1. Therefore, although it is desirable to use as large a time step h as possible, there are upper limits on h which may not be easily determined.

The second method of eliminating Fourier error modes is to damp the amplitudes of the Fourier error modes at each iteration. Increasing the dissipation within the scheme will stretch the blue boundary that is currently a circle as seen in Figure 17. By stretching the circle, it can include lower sigma values that are desirable. This alteration can be done by increasing the artificial dissipation in the system which is controlled by the e_4 parameter in the linearized equation. By adding dissipation to the equations, the eigenvalues can be better fit within regions of increased damping. Figure 18 is an example of a stretched eigenvalue boundary with fourth-difference dissipation.

Figure 18 - Fourier Footprint with e_4 dissipation

This figure shows that the blue boundary that was a circle in Figure 16 is now stretched due to an increase in e_4 from 0 to 0.05. The solid blue line represents the boundary of all high frequency error modes while the dashed blue line represents the boundary of all Fourier modes. It shows that the damping values are improved compared to the previous figure. However, too much artificial dissipation in the system will alter the original problem and reduce the spatial accuracy of the solution. Based on these two methods of eliminating error modes, an optimal medium must be reached such that a solution may use the largest possible time step as well as the minimum amount of dissipation.

Notice that there will be regions within the eigenvalue footprint where σ will always have a value of 1, particularly near the origin. Thus, the relaxation method can only effectively reduce some error-mode frequencies. Comparing Figures (16) and (17), the stability boundary shape also changed. This boundary shape and contour plot is controlled by α and β parameters in the relaxation method. By altering the stability boundary shape, a better fit to the eigenvalues can be done without altering the step size, h , or the artificial dissipation e_4 . Examples illustrating the change in stability boundary shapes of the contour plots are presented in the results section. Looking at the figure below which represents amplification factor plotted with respect to the Fourier frequencies in x and y from 0 to π within the PZ eigenvalue boundary, it is apparent that there are four quadrants.

Figure 19 - Sigma Plot

These quadrants are divided by complex angles in x and y . These quadrants are divided as follows:

Quadrant 1 $0 < \theta_x, \theta_y < \frac{\pi}{2}$

Quadrant 2 $\frac{\pi}{2} < \theta_x < \pi$ and $0 < \theta_y < \frac{\pi}{2}$

Quadrant 3 $\frac{\pi}{2} < \theta_x, \theta_y < \pi$ and finally,

Quadrant 4 $\frac{\pi}{2} < \theta_y < \pi$ and $0 < \theta_x < \frac{\pi}{2}$

Quadrant 3 is well damped while quadrants 2 and 4 vary from well-damped to poorly damped. Quadrant 1 contains low frequency errors and the damping is poor. This poses a problem because all modes of error will need to be eliminated. Providing a better fit of the small magnitude σ values to the eigenvalues of PZ eigenvalues will reduce the σ magnitudes in quadrants 2 and 4. By applying multigrid to the solver, the relaxation method can be optimized to eliminate error from quadrant 2, 3, and 4 while leaving quadrant 1 to the next level of the multigrid.

For the multistage relaxation method, the amplification factor is controlled by various α , β , and h values while the eigenvalues of PZ are controlled by the local flow conditions, spatial discretization as well as artificial dissipation. To compute a spatial accurate solution with stability and fast convergence time, the optimal relaxation parameters should be used as well as a minimal amount of dissipation. The minimal amount of dissipation refers to the smallest amount of artificial dissipation required to satisfy numerical stability. A database of optimized dissipation and relaxation method parameters for different types of flow conditions was constructed. These parameters are optimized using the MATLAB function `fgoalattain`. This MATLAB function is a multi-objective goal attainment function [14]. It allows the user to enter “soft” goals for multiple inputs and optimize for values that satisfy the constraints. In MATLAB, the `fgoalattain` function will reduce a set of nonlinear functions $f_i(x)$ below a set of goals f_i^* . [14] The variable used to represent the attainment “slack” factor used in this optimization function is γ . [13] This attainment factor will be minimized with respect to a parameter vector, x . In this problem, the vector x is $[h \ e_4 \ \alpha_1 \ \dots \ \alpha_m \ \beta_2 \ \dots \ \beta_m]$ where h represents the step size, e_4 represents the 4th order dissipation factor, and α and β represents the tuning parameters of the multistage relaxation method. Without scaling the functions, the goal attainment function will minimize the maximum of $f_i(x) - f_i^*$. For this problem the parameters involved in the optimization have different importance to the overall problem so a weighting factor w_i is used. With the weighting factor, the inequality that represents the constraint in this problem becomes [14]

$$f_i(x) - w_i f_i^*; \forall i \quad [13] \tag{45}$$

As explained earlier, we desire as large a time step as possible, which leads to minimizing $f_1 = -h$. For spatial accuracy, we wish to minimize the amount of artificial dissipation which leads to minimizing $f_2 = e_4$. A third factor is a measure of the average damping over the high frequency domain of error modes. This is computed using an area-weighted average over the high-frequency modes $f_3 =$

$\frac{\int_{HF} |\sigma| d\theta}{\int_{HF} d\theta}$ that we wish to minimize. These three functions will provide the multiple objectives for the fgoalattain function to optimize.

Using fgoalattain, a database of optimized parameters was constructed based on a range of flow conditions. Using this database, the solver can automatically select the nearest set of parameters which match the local flow conditions at each local node in the mesh. The user defines the interval at which optimal parameters are updated. Table 4 describes the input parameters and their nomenclature within the optimizing script.

Input Value	Description
RKN	This is the value that defines the configuration of the multistage relaxation scheme to be 3, 4, or 5 stage. This will affect the overall shape of the stability plot.
CFLgoal	This value defines a soft maximum value of the step size in the optimizing function fgoalattain. A CFLgoal = Inf means there is no limit to the CFL value
K4goal	This is used to control the maximum 4 th difference dissipation used in the solver where the optimal value is as small as possible. This will alter the stretching of the PZ eigenvalues in the stability plot.
Sigmagoal	This is a “soft” constraint set out in the fgoalattain function to guide the minimization of the average high-frequency damping.
Hardk4	This is the hard limit on the 4 th order dissipation. To turn off 4 th order dissipation, simply set hardk4 =0. To turn on 4 th order dissipation, set hardk4 =1. This is required to keep artificial dissipation under a maximum value.
M	This is the local inflow Mach number

AOA	This is the local flow alignment
Dydx	This is the aspect ratio of cell stretching defined based on spatial discretization. This value represents $\frac{\Delta y}{\Delta x}$

Table 4 - Input parameters for Optimizer

The range of input parameters used to calculate the respective eigenvalues and σ values are tabulated below in Table 5.

Parameter	Range of values
RKN	3 to 5
Mach (M)	0.05 to 1.3
Angle of attack (AOA)	0 for aligned flow and 45° for non aligned flows
Dydx	1e-4 to 1e-6
Hardk4	Set to 0 for coarse grids Set to 1 for fine grid
k4goal	Set to 0 for coarse grids Set to 0.1 for fine grid

Table 5 - Range of Parameters for Optimizer

By systematically creating a database of different optimal parameters the solver will have the ability to pick and choose the closet set of optimized parameters to the local flow and mesh conditions. Note that each location in the mesh can use different relaxation parameters provided care is taken to ensure that the numerical method remains conservative. This will reduce the number of iterations necessary to eliminate all Fourier error modes while locally minimizing the addition of artificial dissipation. The next issue is the retrieval of the parameters from the database when needed. As there are a large number of parameters for different flow conditions and for different grid types, the time it takes to retrieve information must also be considered.

Now that the data has been stored, a unique retrieval method must be created such that the time it takes to retrieve the information does not contribute largely to the computation time. The data should be retrieved in an organized manner. The database is structured such that it first searches for Mach number, and then retrieves the closet matching RKN number followed by the alignment of the flow. Lastly it searches for the best aspect ratio of the grid cell and will return a suitable set of α ,

β , and CFL values. The following tree structure, Figure 20 illustrates how the database works to retrieve information.

Figure 20 - Database Retrieval Method

Note however, this database has been constructed based on parameters optimized based on Fourier analysis only. It cannot be applied to flow problems with a different structure. It represents cases that best describe each flow condition based on Fourier analysis of inviscid flows using an explicit multistage relaxation scheme. The database used for this research was an open source database owned by Oracle named Berkeley DB [12]. Version 4.6 of Berkeley DB was used because creating a database application is beyond the scope of this particular research. Berkeley DB is fast, scalable, and uses the same format as the application, so no translation is necessary. Also, after implementation, Berkeley DB does not need to be maintained or configured. It provides a reasonable initial implementation in relation to other data retrieval methods. Berkeley DB is a database that is stable and well known. It is reliable and can be used interfaced with the research; therefore, it was chosen to support the data structure in this research.

Now that a database has been set and optimized parameters are available, it is important to compare the results obtained between the proposed method and other proven existing methods. This research will compare two other methods to the proposed method and these methods are listed below.

1. Version A, where the user has no pre-existing knowledge about the solver and chooses safe parameters.
2. Version B, where the user is experienced with a specific model and has knowledge of what best global parameters to set for this specific problem.
3. Version C, where the user has no expert knowledge about the solver but the database is applied and the optimal parameters are automatically chosen at a specified interval.

Version A was conducted to demonstrate a realistic case where the user would not know the specific details of the solver. This is a common problem due to the fact that CFD cases are very specific and significant experience with a solver is required to know how to converge each case as quickly as possible. The common user will not have this background knowledge and will need to run the solver with caution. A step size value $h=0.8$ is the most commonly used default value for this relaxation method. This version of the solver will be given global flow parameters and a default value of $e_4 = 0.05$ for the artificial dissipation on the fine grid of the multistage scheme. The solver will then use the given initial conditions and iterate using fixed α and β values at each node for every iteration. Since this method requires no prior solver knowledge, the user is able to focus on the physics of the problem instead and provide accurate physical parameters such as boundary conditions. Keep in mind, this method is only capable of solving problems that can be linearized with Fourier analysis and cannot be applied generically to any flow problem.

Version B will demonstrate a user with background knowledge on the solver and the specific problem being solved. A globally set optimal initial step size as well as α and β terms will be assigned to the solver. This is possible because the problem can be run multiple times with different combinations of parameters until the optimized set of parameters have been found. This version will simulate the solver being run by a skilled user with an in-depth amount of knowledge on the specific case. A user with extensive background knowledge on a specific case will be able to predict the best possible set of parameters to use to minimize the time required to converge the solution. This method can only be applicable for a small group of users and is generally not user friendly. Also, it is very case specific and cannot be used as a consistent method to solve all flow problems.

Version C is the proposed automatic parameter selection. It begins using the same optimal set of parameters as version B, but it then updates the local relaxation method parameters at intervals defined by the user. The solver will use the database containing optimized relaxation method parameters based on local flow and mesh conditions. This method should provide results with as much stability and equal or better convergence time as version B as the provided database will have all the optimal parameters for each local flow conditions. This is equivalent to having a skilled user manually input the parameters for a wide range of problems. The disadvantage of this method is the time taken to retrieve the information from the database. As this research is not focused on optimizing database performance, the above-mentioned database will be used in the default setting and overhead time is expected to occur. However, the performance of this method can be gauged based on iteration count as compared to version B and an indication of performance can be obtained. Counting the number of iterations used for each method will give an indication to actual solver time without the overhead time of choosing parameters from the database. This is because the number of iterations represents the number of operations that the solver needs to perform and the smaller amount of operations needed results in a faster convergence time. This method can be used by any user with or without pre-existing knowledge of the solver and problem because by selecting parameters based on local flow conditions, the solver is not case specific and can be applied to any problem as long as sufficient data exists in the database.

All three versions of the solver were compared based on number of iterations and convergence time. These two attributes provide a direct measurement of the convergence behaviour of each version of the solver. The source code that generated the results is presented in the Appendix.

CHAPTER 4

4.1 RESULTS

The results presented in this report are focused on version C of the solver as a full understanding of the capabilities and performance of the automated selection method was the bases of this research. Version C of the solver was initially tested with two different optimizations, which were step size based and dissipation based. The optimized values were stored into two separate databases. The first one includes optimized values by maximizing step size, h , allowing for convection of the error modes out of the domain which is effective if the boundaries conditions are non-reflective. In this method of optimization, the step size was allowed to increase with no limit on the maximum dissipation used. The second database includes optimized values by using a minimal amount of dissipation along with a maximized step size to allow for both damping and convection of the error modes to occur. The second optimization minimized the 4th order dissipation used and maximized the damping of high-frequency error modes while maintaining a step size goal of 0.8. This step size value is considered a conservative step size. As explained above, an optimized goal is just a target used by fgoalattain and not a hard limit to that variable. This section of the report will illustrate the results given from both databases. The primary geometry used in this research was the basic flat plate. This simple case was used to save computation time for version A and B since version C can be applied to any flow problem, this case will demonstrate the effectiveness of implementing an automated selection process. To demonstrate the flexibility of version C, the NACA0012 airfoil was also used.

4.1.1 MAXIMIZE STEP SIZE

The first run was based on optimizing the solver performance by propagating the error out of the domain as quickly as possible. To accomplish this, the database was populated with parameters created based on maximum step size. The flow problem studied in this test was the attached flow over a flat plate. The optimizer was run multiple times with different flow conditions to test the flexibility of the code. Table 6 is a sample table containing some of the combinations of parameters

used as mesh cell flow conditions. This table generated optimal α and β parameters that the solver can use to decrease the convergence time.

RKN (RKN)	Inflow Mach Number (M)	Flow Alignment	Aspect ratio (dydx)	K4 Dissipation (k4)
5	0.5	0	1.00E-04	0.05
4	0.5	0	1.00E-04	0.05
3	0.5	0	1.00E-04	0.05
2	0.5	0	1.00E-04	0.05
2	0.5	$\pi/4$	1.00E-04	0.05
3	0.5	$\pi/4$	1.00E-04	0.05
4	0.5	$\pi/4$	1.00E-04	0.05
5	0.5	$\pi/4$	1.00E-04	0.05
2	0.5	$\pi/2$	1.00E-05	0.05
3	0.5	$\pi/2$	1.00E-05	0.05
4	0.5	$\pi/2$	1.00E-05	0.05
5	0.5	$\pi/2$	1.00E-05	0.05

Table 6 - Optimization Input Parameters

The resulting α and β terms are tabulated below in Table 7. This table shows that at different flow alignments and aspect ratios, there are different optimal α and β terms to use in the relaxation scheme.

Alphas	Beta	CFL	e4
[0.1239, 0.22925, 0.37573, 0.70989, 1]	[1 0.80777, 0.30394, 0.74692, 0.10755]	1.5688	0.30572
[0.30106, 0.42577, 0.71976, 1]	[1, 0.69304, 0.08488, 0.16239]	1.0722	0.34386
[0.2086, 0.58422, 1]	[1, 1, 0.22023]	0.82891	0.37059
[0.49996, 1]	[1, 0.47554]	0.46227	0.42561
[1, 1]	[1, 0.23755]	0.64392	0.1979
[0.80563, 0.88634, 1]	[1, 0, 0]	1.0795	0.06569 4
[0.43858, 0.37536, 1, 1]	[1, 0.19138, 0.60758, 0.10072]	1.3559	0.13896
[0.49071, 0.32088, 0.70384, 0.9887, 1]	[1, 0, 0.75153, 0, 0]	1.6082	0.07163 3
[1, 1]	[1, 0.23755]	0.64387	0.19791
[0.80457, 0.89602, 1]	[1, 0, 0]	1.07232	0.06574 6
[0.43829, 0.37546, 1, 1]	[1, 0.19163, 0.60743, 0.10049]	1.3548	0.13914
[0.49095, 0.32076, 0.70402, 0.98847, 1]	[1, 0, 0.75158, 0, 0]	1.6084	0.07158 8

Table 7 - Optimized Step size, h, coefficients

By running this preliminary test, it could be seen that the flow parameters greatly influence the choice of α and β , which was discussed in the methods section of the thesis. By applying different α and β terms to different locations on the grid based upon the current local flow values, the relaxation method can provide a better Fourier footprint to fit the high frequency error modes and converge faster to a steady state solution. The results were compared between version B and C. The interval at which the parameters were chosen depended on the user as explained in the methods section. The α and β terms were chosen locally based upon the current flow conditions at each node of the grid for version C. It was found that the solver convergence behavior was inconsistent with version C diverging on some cases while version B converged. It was believed that this was a result of using too large a step size which allowed the transient portion of the solution to become unbounded.

4.1.2 MINIMIZE 4TH ORDER ARTIFICIAL DISSIPATION ON FLAT PLATE

The second series of test cases involved running the optimizer based on a combination of minimizing the dissipation used for a fixed step size goal. This was done by setting the step size goal to be h

=0.8 in the fgoalattain function. The α and β coefficients were again found for the different combinations of flow parameters. Sample results for α and β terms when optimizing for minimal dissipation are presented in Tables 8 and 9.

Run	RKN	Mach Number	Flow Alignment	Dydx	Grid Type
1	3	0.15	Aligned	1.00E-04	coarse
2	4	0.15	Aligned	1.00E-04	coarse
3	5	0.15	Aligned	1.00E-04	coarse
4	3	0.15	Not Aligned	1.00E-04	coarse
5	4	0.15	Not Aligned	1.00E-04	coarse
6	5	0.15	Not Aligned	1.00E-04	coarse
7	3	0.15	Aligned	1.00E-04	fine
8	4	0.15	Aligned	1.00E-04	fine
9	5	0.15	Aligned	1.00E-04	fine
10	3	0.15	Not Aligned	1.00E-04	fine
11	4	0.15	Not Aligned	1.00E-04	fine
12	5	0.15	Not Aligned	1.00E-04	fine

Table 8- Flow Parameters for coarse and fine Flow with CFL max at 0.8

Run	alpha	Beta	e2	e4
1	[0.239, 0.495, 1]	[1, 1, 0.285]	1	0
2	[0.402, 0.292, 0.824, 1]	[1, 0, 0.594, 0.225]	1	0
3	[0.255, 0.236, 0.465, 0.652, 1]	[1, 0.381, 0.217, 0.513, 0.0321]	1	0
4	[0.273, 0.579, 1]	[1, 0.804, 0.302]	1	0
5	[0.350, 0.351, 1, 1]	[1, 0.302, 0.826, 0.067]	1	0
6	[0.407, 0.363, 0.550, 1, 1]	[1, 8.06e-04, 0.18, 0.41, 0.001]	1	0
7	[0.5614 0.3980 1 1]	[1 0.1468 0.2911 0]	0	0.05
8	[0.8056 0.8649 1]	[1 0 0]	0	0.05
9	[0.3304 0.1672 0.5126 0.5000 1]	[1 0 0.6785 0.4332 0]	0	0.05
10	[0.654, 0.501, 1]	[1, 0, 0.218]	0	0.05
11	[0.3873 0.3018 0.6988 1]	[1 0.6667 0.1190 0]	0	0.05
12	[0.5574 0.2895 0.7391 0.9166 1]	[1 0 0.6424 0 0]	0	0.05

Table 9 - Optimized Alpha and Beta coefficients for coarse and fine grids

Each run generated a plot of the Fourier footprint of the method which showed the stability limit of σ along with a contour plot of the frequency of σ within the limit of numerical stability. Figure 21 represents the stability and frequency plots of flow condition of case 1 through 3.

Figure 21 - Stability and frequency plots for $M=0.15$ with no 4th order artificial dissipation for case 1, 2, and 3

Notice how the PZ eigenvalue boundary is always a circle. Since $e_4 = 0$, this boundary will always be circular, but notice the transformation of the outer boundary of the contour plots. This shape is based on α and β terms used for each relaxation method. Notice as the number of stages (RKN) used in the method increases the high frequency regions on the frequency plot were able to include more low amplitude factor values. Figure 22 represents the stability and frequency of case 7 through 9.

Notice the difference in PZ eigenvalue boundaries between the two sets of plots. With the addition of 4th order dissipation, the relaxation method was able to better fit the Fourier error modes within the stability and frequency plots resulting with better damping and convection of the error terms out of the domain. The above results are in support of the research to optimize relaxation method parameters α , β , and h with the 4th order dissipation e_4 . It clearly showed that by altering α , β the overall shape of the boundary $|\sigma| = 1$ changes. The step size, h , scales the PZ eigenvalue boundary while the 4th order dissipation stretches the circular boundary to an egg shaped geometry which allows for a better fit to include lower magnitude σ .

After understanding and proving the critical relationship between relaxation method and 4th order dissipation to the convergence and stability of a solver of a problem, results for version A, B, and C in respect to convergence time for the flat plate case are presented. Recall that the problem has been discretized using Cartesian coordinates using the finite volume method so the flat plate case was able to use a simple rectangular mesh. Results from different global inflow conditions were calculated and the results are compared based upon convergence time and the number of iterations used to reach convergence. Comparing convergence times will directly show the cost of arriving at the same solution using the different methods. Comparing number of iterations will illustrate the number of operations needed to obtain the solution, excluding any cost related to automatic parameter selection. The number of iterations is useful to show the difference between version B, which represents the knowledgeable user, and version C, which represents the automated selection process. Problems involving different Mach and Reynolds numbers were used. The Mach number ranged from 0.2 to 0.4 and the Reynolds number ranged from 10^4 to 10^6 . Figure 23 illustrates the difference between each version of the solver at $M = 0.2$ and $Re = 10^4$. A numerical convergence of

$1e^{-10}$ was used as the convergence limit. It also shows version C with parameters chosen every 100 and 1000 intervals. This is important because the frequency of parameters chosen can add

Figure 22 - Stability and Frequency plots for $M=0.15$ using 4th order artificial dissipation for case 7, 8, and 9

Figure 23 - Maximum Aspect Ratio VS Convergence time at $M=0.2$ and $Re=1e4$

overhead to the overall convergence time. So it is imperative to choose an appropriate interval to update the optimized parameters so the solver uses updated parameters and so the interaction between the solver and the database does not hinder the overall convergence time by a significant amount.

Note that there is a spike in the graph. This spike illustrates a case where the solution is unable to converge and oscillates between two solutions. This may be caused by a number of reasons, which can include: a bug in the solver, limited data near that region, or even the solver using parameters that are close to the stability limit. The exact cause of the anomaly is uncertain and can be researched by re-running the solver at conditions slightly to the left and right of the anomaly to diagnose the exact cause of the error. Also notice the last point on the graph for version C also increases suddenly. This is a consistent behaviour for all the results presented. This could be caused by lack of data points in the database for the automated parameter selection method to choose from, causing an unstable numerical method.

Recall, version A represents a user with no knowledge of the flow problem or the solver and default values are used for the step size. Version B represents a user with extensive knowledge of this

specific problem and is familiar with the solver so optimal parameters are chosen as an initial value. Finally, version C of the solver does not require any pre-existing knowledge of either the solver or the problem because a database with optimal parameters will be used for all the different local flow conditions. The aspect ratio was allowed to increase as the number of grid lines in the boundary layer was forced to increase from 5 to 12. The above graph shows that there is a distinct difference between version A with version B and C. Clearly, version A takes much longer to converge to the same solution. Also, as the aspect ratio increased, the convergence time also increased, as Figure 20 shows. This is due to the increased stiffness as the mesh cells stretch in a single direction. It also contributes to the numerical stability of the solver. As the problem becomes extremely stiff, the solver becomes unstable. Notice, however, the spike in version C for both choosing parameters every 100 and 1000 iterations. This spike signifies a divergence for that particular solution. This could be caused by the lack of accurate parameters in the database for the solver to select resulting in divergence. This can be improved by increasing the data points in the database.

To explore the maximum stiffness that the solver can properly solve, the Reynolds number was increased to 10^5 . By increasing the Reynolds number the problem becomes stiffer and higher aspect ratios can be obtained. This will test the solver's ability to solve problems that tend to become more unstable. With the Reynolds number at 10^5 Figure 24 shows the convergence time of the same three versions of the solver.

The results at Reynolds number $Re=10^5$ is consistent with the previous case. The convergence time for both versions B and C are substantially better than version A. It is important to understand that version B and version C should not be compared side by side because version B assumes a user with extensive knowledge. This means version B will have the best global optimized parameters when running the solver for that specific problem. To compare versions B and C, the number of iterations used to reach the solution should be used as a means of comparison. This will be further discussed in the second half of the results presented. Next, the Reynolds number was increased again to 10^6 which becomes turbulent flow for this specific case. The results are shown in Figure 25. Note that the solver used for this experiment is not adequate in handling turbulent flows, but by using a large Reynolds number, the behaviour of the solvers should be the same and the ability to solve extremely stiff problems still exists. It is important to understand that the purpose for using $Re=10^6$ is to observe the behaviour of the solver when solving extremely stiff meshes.

Figure 24 - Maximum Aspect ratio VS Convergence time at $M=0.2$ and $Re=1e5$

Figure 25 - Maximum Aspect Ratio VS Convergence Time for $M=0.2$ and $Re=1e6$

Notice how the trend stays the same in Figures 23 to 25. Also, notice the increased convergence time for version B while version C stayed consistent, but keep in mind that versions B and C cannot be compared based on convergence time due to external factors. These external factors would require optimizing the database selection method as well as the sample size of the database. Also, version B assumes a knowledgeable user with adequate experience with both the solver and the problem. With the current results, version C has shown reasonable improvements from version. It is now important to see the behaviour of the solver under a higher Mach number. This is necessary to demonstrate the flexibility of the solver and the database selection method. The next set of results are presented in Figures 26 to 28 for $M=0.3$ with Reynolds number varying again from $Re=10^4$ to 10^6 .

Comparing the three figures illustrated in Figures 26 to 28, a consistent trend can be found as the Reynolds numbers change even for a higher Mach number. Version A is still the slowest to converge while increasing the period at which parameters are chosen for version C actually increased the convergence time. This may be because the parameters used in the relaxation method may no longer be optimal for the current solution due to the delay in updating parameters. Version B seems to consistently have the best performance in laminar flow and a similar trend can be seen in the turbulent figure with $Re=10^6$. Both versions B and C become unstable at large degrees of stiffness, with version C typically exhibiting convergence problems earlier than version B. This may be due to insufficient data in the database for version C to select and by selecting data that is far from the optimal value, the solver may become unstable. This could be resolved by increasing the data points in the database. The above mentioned cases have had desirable results but to ensure that these behaviours are consistent, a third case was constructed using $M=0.4$. If the solver is consistent then similar results will be seen. The results are presented in Figures 29 to 31.

As expected, the results from $M=0.4$ is consistent with the other previously presented results. The difference between each version is most noticeable at $M=0.4$. This is because at higher Mach numbers, the maximum cell aspect ratios of the grids used increases and by using a default set of parameters, the solution will take longer to converge. A skilled user is assumed to know which values are optimal for each grid, which is why the convergence time is consistently lower than version A. The results obtained in version B are only possible with a user that can provide the best relaxation parameters by experience. Version C is able to maintain a lower convergence time than Version A as well as maintain its flexibility and user friendliness. From the results presented, it

Figure 26 - Maximum Aspect Ratio VS Convergence Time for $M=0.3$ and $Re=1e4$

Figure 27 - Maximum Aspect Ratio VS Convergence Time for $M=0.3$ and $Re=1e5$

Figure 28 - Maximum Aspect Ratio VS Convergence Time for $M=0.3$ and $Re=1e6$

Figure 29 - Maximum Aspect Ratio VS Convergence Time for $M=0.4$ and $Re=1e4$

Figure 30 - Maximum Aspect Ratio VS Convergence Time for $M=0.4$ and $Re=1e5$

Figure 31 - Maximum Aspect Ratio VS Convergence Time for $M=0.4$ and $Re=1e6$

Figure 32 - Maximum Aspect Ratio VS Iteration at $M=0.2$ and $Re=1e4$

clearly shows that version C is capable of solving fluid problems with the efficiency of a skilled user while maintaining its user friendliness. Improvements can be made with respect to convergence time by improving the database search algorithm or even including more optimized parameters such that local optimization can be performed with best “fit” data.

After comparing the convergence time, which clearly illustrated the ability of version C to reproduce a spatially accurate solution in a shorter period of time than version A, the number of iterations used for each of the previous runs was examined. Since the cost of a residual evaluation is the same for each version of the solver, this was done to determine if the extended convergence time for version C when compared to version B was due to the database selection method or actual increased convergence time. If version C uses more iterations than version B, then it means that the parameters selected by version C during the automated parameter update are not the best set of parameters possible. Whereas if the iteration count is lower than version B then it shows the increased convergence time is due to inefficiencies in the database selection method and not the core focus of this research. This next set of results, Figures 32 to 34, are based on the previous conditions and was generated simultaneously as the previous results so an accurate representation

Figure 33 - Maximum Aspect Ratio VS Iteration at $M=0.2$ and $Re = 1e5$

Figure 34 - Maximum Aspect Ratio VS Iteration at $M=0.2$ and $Re = 1e6$

of the number of iterations used can be related to the convergence time. At $M=0.2$ and the various Reynolds numbers, the following number of iterations were recorded for versions A, B, and C.

Notice with a Reynolds number of $1e4$, the number of iterations spiked to limit for version C. This was reflected in the convergence graph as well. This could be caused by the lack of data points in that region which caused the solver to diverge. To improve convergence, more data points could be generated. However, looking at the three graphs presented as Figures 32 to 34, version C uses less iteration than both version A and B consistently. This suggests that the increased convergence time is a result of a slow data retrieval method and not a slow relaxation method. This proves that version C is capable of generating spatial accurate results faster than version B if the data selection method can be optimized. Optimization of the data selection method is beyond the scope of this research. To ensure that the solver does use less iteration for different conditions, results from Mach numbers of 0.3 and 0.4 were also created based on the number of iterations. Figures 35 to 37 show the consistency of version C with obtaining a solution with less iteration than both version A and B.

The trends of iteration count are consistent with the exception of Figure 35. With Reynolds number at $1e4$, it seems version C is able to perform better with more parameter selection. All the results so far have supported a better convergence time and iteration count when version C chooses optimal parameters at a more frequent interval. This finding is reasonable, because the solver works closer to optimal parameters when the parameters are chosen more frequently as the solution evolves. With a small pool of data to select from, it may be more advantageous to select parameters more frequently because the overhead time when selecting the parameters is small but with a large database, the overhead time will increase. This will affect the overall convergence time of the solution. It also seems version C performs much better under highly stiff grids when compared to version A or B. Recall the graph on the right of Figure 26 which represented a Reynolds number of 10^6 , is not an accurate representation of the flow because it is turbulent, but by increasing the Reynolds number, the stiffness was able to increase. With the increased stiffness, it shows that version C is far superior when compared to both version A and B. This result is presented in both the figures representing $M=0.2$ and $M=0.3$. These results also support the flexibility of version C's ability to handle problems with different degrees of stiffness.

Figure 35 - Maximum Aspect Ratio VS Iteration at $M=0.3$ and $Re = 1e4$

Figure 36 - Maximum Aspect Ratio VS Iteration at $M=0.3$ and $Re = 1e5$

The last case studies the iteration behaviour at $M=0.4$. The same trend as the above two presented cases is expected. Figures 38 to 40 below illustrates the results at $M=0.4$.

Looking at all the figures that represents iteration count for the different versions of the solver, it shows that version C actually performs better for stiffer grids. When using a lower Reynolds number, version C takes more iteration to reach a solution when selecting parameters every 1000 iteration, but it quickly decreases the number of iterations compared to version B when the Reynolds number and aspect ratio increased. Looking at Reynolds number at 10^6 , it clearly shows that version C out performs both versions A and B by a large amount for all cases. This result is desired because stiffness in flow problems are common and by selecting optimal parameters locally at given intervals, the stiffness has been locally reduced. Overall, from the results presented, version C of the solver performed better than version A in all aspects. The convergence time and iteration count for version C out performed version A so it can be concluded that automated selection of optimized relaxation parameters is better than using default values. Next, by looking at iteration count, version C also out performed version B because the iteration count was consistently lower when choosing optimized parameters every 100 iterations. Also, version C provided better results as the aspect ratio increased and the problem becomes stiffer. It showed how a user with no prior knowledge of the problem or solver can utilize a database and solve the problem with less iteration than a skilled user with that particular problem. It also showed that stiffness in numerical problems can be reduced further with the automated selection process.

Figure 37 - Maximum Aspect Ratio VS Iteration at $M=0.3$ and $Re = 1e6$

Figure 38 - Maximum Aspect Ratio VS Iteration at $M=0.4$ and $Re = 1e4$

Figure 39 - Maximum Aspect Ratio VS Iteration at $M=0.4$ and $Re = 1e5$

Figure 40 - Maximum Aspect Ratio VS Iteration at $M=0.4$ and $Re = 1e6$

4.1.3 SPATIAL ACCURACY COMPARISON

This next section provides a comparison of version A, B, and C to the analytical Blasius solution. It is important to acknowledge the spatial accuracy of the solver to show the solution obtained is an accurate representation of the actual solution at steady state. The velocity profile at the end of the flat plate determined using version B (identical to A) and C of the solver is plotted below in Figure 41.

Figure 41 - Comparison between Blasius Solution with version B and C

The line labeled Blasius is Blasius' analytical solution to laminar flow over a flat plate. This flow problem involves a plate is 1 [m] long with inflow speed of $M=0.3$ and Reynolds number of $1E5$. The maximum mesh cell aspect ratio was 18443. Both cases use a maximum 4th difference artificial dissipation, e_4 , of 0.05. Artificial dissipation causes the discrepancy between the numerical and

analytical solution at the upper limit of the boundary layer. Version C automatically selects lower e_4 where possible. The analytical solution to the skin friction coefficient is $C_f=2.10E-3$ for this case. Using version B, the computed skin friction coefficient is $2.00E-3$. Using version C, the computed skin friction coefficient is $2.10E-3$ which illustrates that an accurate answer can be obtained in less time using version C. For this particular example version C of the solver finished in only 67% of the time required for version B. Figure 41 illustrates typical solutions obtained by using all three versions of the solver for the range of flat plate flow problems tested in this research. This shows that version C of the solver is able to obtain a solution with spatial accuracy that is equal to or better than the original version of the solver.

4.1.4 MINIMIZE 4TH ORDER ARTIFICIAL DISSIPATION ON NACA0012 AIRFOIL

After acknowledging the abilities of the solver using an automated selection process, the application of this process to different flow geometries must be studied. This was completed by using a two-dimensional mesh about a NACA0012 airfoil. The meshed used to construct the NACA0012 airfoil was a structured C mesh with a sharp trailing edge. The off-wall spacing varied from $1e^{-4}$ to $1e^{-6}$. Version C of the solver was applied to the NACA0012 airfoil and compared with version B. Again, both convergence times and iteration count were studied. Using a mesh with an off wall spacing of 10^{-4} near the airfoil, the following table of results (Table 10) were obtained.

Case	Mach	RE	Angle of attack (deg)	Version B		Version C	
				Conv. time (s)	Iterations	Conv. time (s)	Iterations
1	0.3	1.00E+04	0	556.542	3767	Did not Converge	3190
2	0.3	1.00E+04	1	554.486	3533	462.724	2601
3	0.5	1.00E+04	0	566.736	3796	471.114	2809
4	0.5	1.00E+04	1	543.243	3475	454.698	2612

Table 10 - Tabulated results for N0012

The angle of attack in Table 10 refers to the global angle of attack. Note the local flow directions will be different at each mesh location. By using an airfoil, the inflow conditions are different than the flat plate case. The NACA0012 airfoil was first meshed as necessary for any CFD solver and the maximum aspect ratio was recorded as 18078. The table above shows that for aligned and non-aligned flows, version C of the solver selecting optimal parameters every 100 iteration is better in both convergence time and iteration count at $M=0.3$ and $M=0.5$ and Reynolds number at 10^4 . These

results are expected because based on the flat plate case, as the aspect ratio increased, the performance of version C becomes better than version B. In addition to having improved results, the same database used for the flat plate case was applied to solving this problem. Being able to utilize the same database to different geometries suggests that only one database needs to be constructed. This suggests the overhead time needed to construct the database only needs to be performed once, whereas a different skilled user is required to produce the results given by version B. Using the same database is possible because the solver uses local flow conditions regardless of the geometry used or the boundary conditions. That makes version C of the solver very versatile and flexible for any flow problem that needs to be solved. Notice also, that there is a divergence in one of the solutions provided by version C. The reason for the divergence in case 1 for version C may be caused by insufficient data points for the solver to effectively solve the flow in that configuration which resulted in a divergence. This can be fixed by increasing the data points in the database. Smaller samples of results were computed with the NACA0012 airfoil because the purpose of using the airfoil was to illustrate the versatility of version C of the solver. To obtain results for various flow conditions with the NACA0012 airfoil, more optimized data points are required.

Overall, the automated selection process provided results that are comparable in convergence time when compared to a skilled user and exceeded the minimum number of iterations required. It was able to produce such results using generic default conditions which simulate a user with no knowledge of the solver for this flow problem. The automated selection process also demonstrated the versatility of the solver by applying multiple geometries to the same database and producing results that exceed the performance of a skilled user.

1	0.2	1.00E+01	2.43E+03	3.75	4.62E+02	3001
2	0.3	1.00E+04	2.43E+04	1	2.43E+04	3002
3	0.2	1.00E+04	2.43E+04	0	2.43E+04	3003
4	0.2	1.00E+04	2.43E+04	1	2.43E+04	3004

Table 10 - Tabulated results for NACA0012

CHAPTER 5

5.1 CONCLUSION

This research has demonstrated that an explicit multistage solver that utilizes a preconditioner and multigrid schemes to reduce convergence time can be improved with the addition of a database containing optimized solver parameters. The first objective of this research was to determine if optimizing local flow conditions at each mesh location was effective in reducing the convergence time while maintaining stability of the solver. The second objective was to create a method such that users with no prior solver or problem knowledge can still obtain a solution by using common default parameters in the same or a shorter amount of time as a skilled user. The method allows the solver to automatically select 'best fit' parameters on intervals defined by the user. The results clearly illustrate the reduced number of iterations used to obtain the solution using the automatic parameter selection version of the solver when compared to a skilled user or common default input values. From the flat plate case presented, it can be concluded that the implementation of a database containing such parameters is able to effectively reduce the computation time required to obtain a solution. Comparisons were made with two separate cases that represented a user with no prior knowledge where conservative input parameters were used and a skilled user with the best initial input parameters. Also, by applying this method to a different geometry, it was demonstrated that this method is generic and can be applied to different flow conditions provided the problem is laminar and can be linearized using Fourier analysis. The overhead time for this method can be reduced by optimizing the data retrieval method. This will aid in the overall convergence time of the solution. It was found that, by increasing the number of data points in the database, the solver is able to choose more suitable parameters which will prevent divergence. Overall, this research has successfully demonstrated the ability of a solver with automatic parameter selection to perform on par or better than version B of the solver. With a larger database and an optimized data retrieval method, this method will reduce the time taken to solve more complex problems.

APPENDIX

MATLAB source code

The following source code was used to generate the optimal solver parameters beginning with the function 'Opt2D'. Source code is documented within the comments of the source code. Additional supported functions are provided for functions:

Euler2D

Simple2D

rk2sigma

rk3sigma

rk4sigma

Hull2D

Sigma2D

```

1 %% Opt2D
2 %% Description
3 % Optimize multistage coefficients for a method with RKN stages
4 %% Input parameters
5 %
6 % * RKN - Number of stages (eg. 3)
7 % * LambdaFxn - Hook to function for calculating eigenvalues of difference
8 % equation (eg. @Euler2D)
9 % * CFLgoal - Desired CFL. Set to Inf for maximum possible.
10 % * k4goal - Desired 4th diff dissipation coefficient.
11 % * sigmagoal - Desired average high-frequency amplification factor.
12 % * hardk4 - Set to non-zero to constrain k4<k4goal.
13 % * M - cell local Mach number
14 % * AOA - cell local angle of attack relative to x-dir
15 % * dydx - cell ratio of height to width (inverse aspect ratio).
16 %% Output parameters
17 %
18 % * alpha - optimal alpha (scaling) coefficients
19 % * beta - optimal beta (dissipation) coefficients
20 % * CFL - optimal CFL
21 % * e4 - optimal kappa4 coefficient (if return variable not provided,
22 % assume 2nd diff dissipation)
23 %% Definition
24 function [alpha,beta,CFL,e4]=Opt2D(RKN,LambdaFxn,CFLgoal,k4goal,...
25     sigmagoal,hardk4,M,AOA,dydx)
26
27 % If no e4 parameter requested assume 2nd diff dissipation only!
28 if (nargout<=3)
29     k4goal=0;
30 end
31
32 ii=1;
33 %CFL
34 x0(ii)=1;
35 xLB(ii)=0.0;
36 xUB(ii)=Inf;
37 ii=ii+1;
38 % alpha and beta with alpha(RKN)=1 and beta(1)=1
39 for jj=1:RKN-1
40     x0(ii+jj-1)=1/(RKN-(jj-1));
41     xLB(ii+jj-1)=0.0;
42     xUB(ii+jj-1)=1.0;
43     x0(ii+RKN-1+jj-1)=1;
44     xLB(ii+RKN-1+jj-1)=0;
45     xUB(ii+RKN-1+jj-1)=1;
46 end
47 ii=ii+2*(RKN-1);
48 if (k4goal~=0)
49     % kappa4
50     x0(ii)=0.1;
51     xLB(ii)=0.0;
52     xUB(ii)=1.0;
53     ii=ii+1;
54 end
55
56 if (isinf(CFLgoal))
57     display('Using fmincon');
58     Options=optimset('fmincon');
59 else
60     display('Using fgoalattain');
61     Options=optimset('fgoalattain');
62 end
63 Options=optimset(Options,'LargeScale','off');
64 %Options=optimset(Options,'TypicalX',x0);
65 Options=optimset(Options,'MaxFunEvals',1000*max(size(x0)));
66 Options=optimset(Options,'Display','iter');
67 Options=optimset(Options,'GradObj','on');
68 Options=optimset(Options,'GradConstr','off');
69 Options=optimset(Options,'TolX',1e-3);
70 Options=optimset(Options,'TolCon',1e-10);
71 Options=optimset(Options,'TolFun',1e-3);

```



```

72
73 if (isinf(CFLgoal) && k4goal==0)
74     display(['Optimizing for maximum CFL with 2nd diff dissipation']);
75     [x,fval,exitflag,output]=fmincon(@CFLObjFxn,x0,[],[],[],[],[],xLB,xUB,...
76         @SigmaCon2ndFxn,Options,RKN,LambdaFxn,M,AOA,dydx);
77 elseif (isinf(CFLgoal) && k4goal~=0)
78     display(['Optimizing for maximum CFL and minimum kappa4 ',...
79         'with 4th diff dissipation']);
80     goal=[-100,k4goal];
81     weight=abs(goal);
82     if (hardk4~=0)
83         weight(2)=0;
84     end
85     [x,fval,attainfactor,exitflag,output]=fgoalattain(@CFLDissObjFxn,x0,...
86         goal,weight,[],[],[],[],xLB,xUB,@SigmaCon4thFxn,Options,RKN,...
87         LambdaFxn,M,AOA,dydx);
88     display(['attainfactor=',num2str(attainfactor)]);
89     display(['goal=',num2str(goal)]);
90 elseif (k4goal==0)
91     display(['Optimizing for maximum CFL and minimum average sigma ',...
92         'with 2nd diff diss']);
93     goal=[-CFLgoal,sigmagoal];
94     weight=abs(goal);
95     [x,fval,attainfactor,exitflag,output]=fgoalattain(@CFLSigmaObjFxn,...
96         x0,goal,weight,[],[],[],[],xLB,xUB,@SigmaCon2ndFxn,Options,RKN,...
97         LambdaFxn,M,AOA,dydx);
98     display(['attainfactor=',num2str(attainfactor)]);
99     display(['goal=',num2str(goal)]);
100 else
101     display(['Optimizing for maximum CFL, minimum kappa4 and ',...
102         'minimum average sigma with 4th diff diss']);
103     goal=[-CFLgoal,k4goal,sigmagoal];
104     weight=abs(goal);
105     if (hardk4~=0)
106         weight(2)=0;
107     end
108     [x,fval,attainfactor,exitflag,output]=fgoalattain(@CFLDissSigmaObjFxn,...
109         x0,goal,weight,[],[],[],[],xLB,xUB,@SigmaCon4thFxn,Options,RKN,...
110         LambdaFxn,M,AOA,dydx);
111     display(['attainfactor=',num2str(attainfactor)]);
112     display(['goal=',num2str(goal)]);
113 end
114
115 display(['fval=',num2str(fval)]);
116 display(['exitflag=',num2str(exitflag)]);
117 output.iterations
118 output.funcCount
119 output.algorithm
120
121 if (nargout>3)
122     [alpha,beta,CFL,e4]=Decode(x,RKN);
123 else
124     [alpha,beta,CFL]=Decode(x,RKN);
125 end
126
127 display(['alpha=',num2str(alpha)]);
128 display(['beta=',num2str(beta)]);
129 display(['CFL=',num2str(CFL)]);
130 if (k4goal~=0)
131     display(['e4=',num2str(e4)]);
132 end
133
134
135 %% Supporting sub-functions
136
137 function [alpha,beta,CFL,e4]=Decode(x,RKN)
138     ii=1;
139     CFL=x(ii); ii=ii+1;
140     beta(1)=1.0;
141     for jj=1:RKN-1
142         alpha(jj)=min(1,max(0,x(ii+jj-1)));

```

```

143     beta(jj+1)=min(1,max(0,x(ii+RKN-1+jj-1)));
144     if (beta(jj+1)<1e-4)
145         beta(jj+1)=0.0;
146     end
147 end
148 alpha(RKN)=1.0;
149 ii=ii+2*(RKN-1);
150 if (nargout>3)
151     e4=x(ii);
152     ii=ii+1;
153 end
154 end
155
156 function [f,g]=CFLObjFxn(x,RKN,LambdaFxn,varargin)
157     [alpha,beta,CFL]=Decode(x,RKN);
158     f(1)=-CFL;
159     if nargout>1
160         N=max(size(x));
161         g=zeros(N,1);
162         g(1,1)=-1;
163     end
164 end
165
166 function [f,g]=CFLDissObjFxn(x,RKN,LambdaFxn,varargin)
167     [alpha,beta,CFL,e4]=Decode(x,RKN);
168     f(1)=-CFL;
169     f(2)=e4;
170     if nargout>1
171         N=max(size(x));
172         g=zeros(N,2);
173         g(1,1)=-1; g(2,2)=1;
174     end
175 end
176
177 function [f,g]=CFLSigmaObjFxn(x,RKN,LambdaFxn,varargin)
178     [alpha,beta,CFL]=Decode(x,RKN);
179     f(1)=-CFL;
180     f(2)=SigmaObj(alpha,beta,CFL,1,0,LambdaFxn,varargin{:});
181     if nargout>1
182         N=max(size(x));
183         g=zeros(N,2);
184         g(1,1)=-1;
185         for ii=1:N
186             xpeps=x;
187             epsilon=1e-6;
188             if (abs(x(ii))>epsilon)
189                 epsilon=epsilon*abs(x(ii));
190             end
191             xpeps(ii)=x(ii)+epsilon;
192             invepsilon=1.0/epsilon;
193             [alpha,beta,CFL]=Decode(xpeps,RKN);
194             g(ii,2)=invepsilon*(SigmaObj(alpha,beta,CFL,1,0,...
195                 LambdaFxn,varargin{:})-f(2));
196         end
197     end
198 end
199
200 function [f,g]=CFLDissSigmaObjFxn(x,RKN,LambdaFxn,varargin)
201     [alpha,beta,CFL,e4]=Decode(x,RKN);
202     f(1)=-CFL;
203     f(2)=e4;
204     f(3)=SigmaObj(alpha,beta,CFL,0,e4,LambdaFxn,varargin{:});
205     if nargout>1
206         N=max(size(x));
207         g=zeros(N,3);
208         g(1,1)=-1; g(2,2)=1;
209         for ii=1:N
210             xpeps=x;
211             epsilon=1e-6;
212             if (abs(x(ii))>epsilon)
213                 epsilon=epsilon*abs(x(ii));

```

```

214         end
215         xpeps(ii)=x(ii)+epsilon;
216         invepsilon=1.0/epsilon;
217         [alpha,beta,CFL,e4]=Decode(xpeps,RKN);
218         g(ii,3)=invepsilon*(SigmaObj(alpha,beta,CFL,0,e4,...
219             LambdaFxn,varargin{:})-f(3));
220     end
221 end
222 end
223
224 function [c,ceq,gc,gceq] = SigmaCon4thFxn(x,RKN,LambdaFxn,varargin)
225     [alpha,beta,CFL,e4]=Decode(x,RKN);
226     [c]=SigmaCon(alpha,beta,CFL,0,e4,LambdaFxn,varargin{:});
227     ceq=[];
228     if nargin>2
229         N=max(size(x));
230         M=max(size(c));
231         gc=zeros(N,M);
232         for ii=1:N
233             xpeps=x;
234             epsilon=1e-6;
235             if (abs(x(ii))>epsilon)
236                 epsilon=epsilon*abs(x(ii));
237             end
238             xpeps(ii)=x(ii)+epsilon;
239             invepsilon=1.0/epsilon;
240             [alpha,beta,CFL,e4]=Decode(xpeps,RKN);
241             gc(ii,1:M)=transpose(invepsilon*(SigmaCon(alpha,beta,...
242                 CFL,0,e4,LambdaFxn,varargin{:})-c));
243         end
244         gceq=[];
245     end
246 end
247
248 function [c,ceq,gc,gceq] = SigmaCon2ndFxn(x,RKN,LambdaFxn,varargin)
249     [alpha,beta,CFL]=Decode(x,RKN);
250     [c]=SigmaCon(alpha,beta,CFL,1,0,LambdaFxn,varargin{:});
251     ceq=[];
252     if nargin>2
253         N=max(size(x));
254         M=max(size(c));
255         gc=zeros(N,M);
256         for ii=1:N
257             xpeps=x;
258             epsilon=1e-6;
259             if (abs(x(ii))>epsilon)
260                 epsilon=epsilon*abs(x(ii));
261             end
262             xpeps(ii)=x(ii)+epsilon;
263             invepsilon=1.0/epsilon;
264             [alpha,beta,CFL]=Decode(xpeps,RKN);
265             gc(ii,1:M)=transpose(invepsilon*(SigmaObj(alpha,beta,...
266                 CFL,1,0,LambdaFxn,varargin{:})-c));
267         end
268         gceq=[];
269     end
270 end
271
272 function [f] = SigmaObj(alpha,beta,CFL,e2,e4,lambdafxn,varargin)
273
274     RKN=max(size(alpha));
275     [sigmafxn]=sprintf('rk%dsigma',RKN);
276
277     % Integrate sigma over high-frequency range only.
278     % Sigma for RK is symmetrical about real axis, so only
279     % need to check one half
280
281     f1=0;
282     % High-High frequencies
283     f1=f1+dblquad(@sigma_wrt_theta,pi/2,pi,pi/2,pi,[],[],sigmafxn,...
284         lambdafxn,alpha,beta,CFL,e2,e4,varargin{:});

```



```

285 % High-Low frequencies
286 f1=f1+dblquad(@sigma_wrt_theta,pi/2,pi,0,pi/2,[],[],sigmafxn,...
287     lambdafxn,alpha,beta,CFL,e2,e4,varargin{:});
288 % Low-High frequencies
289 f1=f1+dblquad(@sigma_wrt_theta,0,pi/2,0,pi,[],[],sigmafxn,...
290     lambdafxn,alpha,beta,CFL,e2,e4,varargin{:});
291 %D=max(size(feval(lambdafxn,1,0,0,0,varargin{:})));
292 D=1;
293 scale=D*3*pi/2*pi/2;
294 f=f1/scale;
295 end
296
297 function [sigma] = sigma_wrt_theta(thetax,thetay,sigmafxn,lambdafxn,...
298     alpha,beta,CFL,e2,e4,varargin)
299 N=max(size(thetax));
300 sigma=zeros(N,1);
301 for ii=1:N
302     lambda=CFL*feval(lambdafxn,e2,e4,thetax(ii),thetay,varargin{:});
303     sigma(ii)=max(feval(sigmafxn,real(lambda),imag(lambda),...
304         alpha,beta));
305 end
306 end
307
308 function [c] = SigmaCon(alpha,beta,CFL,e2,e4,lambdafxn,varargin)
309 RKN=max(size(alpha));
310 [sigmafxn]=sprintf('rk%dsigma',RKN);
311
312 n=15; % Increase this number if the constraints are being violated
313
314 % Sigma for RK is symmetrical about real axis, so only
315 % need to check one half
316
317 % Some sigmas >1 can occur close to zero and might be missed.
318 % Generate a range that includes some small values near zero.
319 thetax=[0,1e-5,1e-4,1e-3,1e-2,pi/n:(pi-pi/n)/n:pi];
320 thetay=[0,1e-5,1e-4,1e-3,1e-2,pi/n:(pi-pi/n)/n:pi];
321 N=max(size(thetax));
322 D=1;
323
324 c=0;
325
326 for ii=1:N
327     for jj=1:N
328         lambda=CFL*feval(lambdafxn,e2,e4,thetax(ii),thetay(jj),...
329             varargin{:});
330         c=max(c,max(feval(sigmafxn,real(lambda),imag(lambda),...
331             alpha,beta)));
332     end
333 end
334 c=c-1;
335 end
336
337 function [f]=RHSEvals(beta)
338 RKN=max(size(beta));
339 f=0;
340 for ii=1:RKN
341     if (beta(ii)==0)
342         f=f+0.075;
343     else
344         f=f+1;
345     end
346 end
347 f=f/RKN;
348 end
349
350 end

```

```

1  %% Euler2D
2  %% Description
3  % Calculate eigenvalues of linearized 2D Euler equations for given
4  % Fourier angle.
5  % Input parameters
6  %
7  % * e2 - 2nd difference artificial dissipation coefficient
8  % * e4 - 4th difference artificial dissipation coefficient
9  % * thetax - Fourier angle in x
10 % * thetay - Fourier angle in y
11 % * varargin - additional arguments to pass additional parameters:
12 %% Additional input parameters
13 % * varargin{1}(1) - local Mach number (default 1.0)
14 % * varargin{2}(1) - local flow angle (default pi/4)
15 % * varargin{3}(1) - local mesh spacing ratio (default 1.0)
16 % * varargin{4}(1) - local gas specific heat ratio (default 1.4)
17 %% Definition
18 function [lambda] = Euler2D(e2,e4,thetax,thetay,varargin)
19
20 gamma=1.4;
21 M=1.0;
22 AOA=pi/4;
23 dydx=1.0;
24 if (length(varargin)>=1)
25     M=varargin{1}(1);
26 end
27 if (length(varargin)>=2)
28     AOA=varargin{2}(1);
29 end
30 if (length(varargin)>=3)
31     dydx=varargin{3}(1);
32 end
33 if (length(varargin)>=4)
34     gamma=varargin{4}(1);
35 end
36 rho=1.0;
37 u=M*cos(AOA);
38 v=M*sin(AOA);
39 p=1.0;
40 c=sqrt(gamma*p/rho);
41 Ax=[u,rho,0,0;u,0,1/rho,0;0,u,0,rho*c^2,0,u]*dydx;
42 Ay=[v,0,rho,0;v,0,0,0;v,1/rho,0,0,rho*c^2,v];
43 Lx=[1,0,rho/2/c,rho/2/c;0,0,0.5,-0.5;0,-1,0,0;0,0,rho*c/2,rho*c/2];
44 Ly=[1,0,rho/2/c,rho/2/c;0,1,0,0;0,0,0.5,-0.5;0,0,rho*c/2,rho*c/2];
45 Lxinv=[1,0,0,-1/c^2;0,0,-1,0;0,1,0,1/rho/c;0,-1,0,1/rho/c];
46 Lyinv=[1,0,0,-1/c^2;0,1,0,0;0,0,1,1/rho/c;0,0,-1,1/rho/c];
47 Lambdax=diag([abs(u),abs(u),abs(u+c),abs(u-c)]);
48 Lambday=diag([abs(v),abs(v),abs(v+c),abs(v-c)]);
49 absAx=Lx*Lambdax*Lxinv*dydx;
50 absAy=Ly*Lambday*Lyinv;
51 lambda=Simple2D(e2,e4,thetax,thetay,Ax,Ay,absAx,absAy);

```

```

1 %% Simple2D
2 %% Description
3 % Calculate eigenvalues of linearized 2D equations for given
4 % Fourier angle.
5 %% Input parameters
6 %
7 % * e2 - 2nd difference artificial dissipation coefficient
8 % * e4 - 4th difference artificial dissipation coefficient
9 % * thetax - Fourier angle in x
10 % * thetay - Fourier angle in y
11 % * varargin - additional arguments to pass additional parameters:
12 %% Additional input parameters
13 % * varargin{1} - flux jacobian in x
14 % * varargin{2} - flux jacobian in y
15 % * varargin{3} - abs flux jacobian in x
16 % * varargin{4} - abs flux jacobian in y
17
18 %% Definition
19 function [lambda] = Simple2D(e2,e4,thetax,thetay,varargin)
20
21 cx=cos(thetax);
22 sx=sin(thetax);
23 cy=cos(thetay);
24 sy=sin(thetay);
25 if (length(varargin)==0)
26     lambda=Simple1D(e2,e4,thetax,varargin{:});
27     return
28 end
29 Ax=varargin{1};
30 Ay=varargin{2};
31 if (length(varargin)>=3)
32     absAx=varargin{3};
33     absAy=varargin{4};
34 else
35     [Tx,Dx]=eig(Ax);
36     [Ty,Dy]=eig(Ay);
37     absAx=Tx*abs(Dx)*inv(Tx);
38     absAy=Ty*abs(Dy)*inv(Ty);
39 end
40
41 Z=(e2-2*e4*(cx+cy-2))*(absAx*(cx-1)+absAy*(cy-1))-i*(Ax*sx+Ay*sy);
42 P=absAx+absAy;
43 lambda=eig(inv(P)*Z);

```



```

1  %% rk2sigma
2  %% Description
3  % Calculate sigma-lambda relationship for RKN=2.
4  % Generated symbolically.
5  %% Input parameters
6  %
7  % * x - real component of eigenvalue lambda
8  % * y - imag component of eigenvalue lambda
9  % * alpha - vector of alpha coefficients
10 % * beta - vector of beta coefficients
11 %
12 function [sigma]=rk2sigma(x,y,alpha,beta)
13 [sigma]=abs(i.*y.*alpha(2)+x.*alpha(2).*beta(2)...
14 -y.^2.*alpha(1).*alpha(2)+i.*y.*alpha(1).*x.*alpha(2).*beta(2)...
15 +i.*x.*alpha(1).*beta(1).*y.*alpha(2)...
16 +x.^2.*alpha(1).*beta(1).*alpha(2).*beta(2)...
17 +1+x.*alpha(2).*beta(1)-x.*alpha(2).*beta(1).*beta(2));

```

```

1  %% rk3sigma
2  %% Description
3  % Calculate sigma-lambda relationship for RKN=3.
4  % Generated symbolically.
5  %% Input parameters
6  %
7  % * x - real component of eigenvalue lambda
8  % * y - imag component of eigenvalue lambda
9  % * alpha - vector of alpha coefficients
10 % * beta - vector of beta coefficients
11 %
12
13 function [sigma]=rk3sigma(x,y,alpha,beta)
14 [sigma]=abs(1+x.*alpha(3).*beta(3)-y.^2.*alpha(2).*alpha(3)...
15 +i.*x.^2.*alpha(1).*beta(1).*y.*alpha(2).*alpha(3).*beta(3)...
16 +i.*x.^2.*alpha(1).*beta(1).*alpha(2).*beta(2).*y.*alpha(3)...
17 +i.*y.*alpha(3)...
18 +i.*y.*alpha(1).*x.^2.*alpha(2).*beta(2).*alpha(3).*beta(3)...
19 +i.*x.*alpha(2).*beta(2).*y.*alpha(3)...
20 +i.*y.*alpha(2).*x.*alpha(3).*beta(3)...
21 +x.^2.*alpha(2).*beta(2).*alpha(3).*beta(3)+x.*alpha(3).*beta(1)...
22 -x.^2.*alpha(2).*beta(1).*beta(2).*alpha(3).*beta(3)...
23 +x.^2.*alpha(2).*beta(1).*alpha(3).*beta(3)...
24 -x.^2.*alpha(1).*beta(1).*alpha(3).*beta(2).*beta(3)...
25 +x.^2.*alpha(1).*beta(1).*alpha(3).*beta(2)...
26 -x.*alpha(1).*beta(1).*y.^2.*alpha(2).*alpha(3)...
27 +i.*y.*alpha(1).*x.*alpha(3).*beta(2)...
28 +i.*x.*alpha(2).*beta(1).*y.*alpha(3)...
29 -y.^2.*alpha(1).*x.*alpha(2).*beta(2).*alpha(3)...
30 -y.^2.*alpha(1).*alpha(2).*x.*alpha(3).*beta(3)...
31 -x.*alpha(3).*beta(1).*beta(2)-x.*alpha(3).*beta(1).*beta(3)...
32 +x.*alpha(3).*beta(1).*beta(2).*beta(3)...
33 -i.*y.^3.*alpha(1).*alpha(2).*alpha(3)...
34 +x.^3.*alpha(1).*beta(1).*alpha(2).*beta(2).*alpha(3).*beta(3)...
35 -i.*x.*alpha(2).*beta(1).*beta(2).*y.*alpha(3)...
36 -i.*y.*alpha(1).*x.*alpha(3).*beta(2).*beta(3)...
37 +x.*alpha(3).*beta(2)-x.*alpha(3).*beta(2).*beta(3));

```

```

1  %% rk4sigma
2  %% Description
3  % Calculate sigma-lambda relationship for RKN=3.
4  % Generated symbolically.
5  %% Input parameters
6  %
7  % * x - real component of eigenvalue lambda
8  % * y - imag component of eigenvalue lambda
9  % * alpha - vector of alpha coefficients
10 % * beta - vector of beta coefficients
11 %
12 function [sigma]=rk4sigma(x,y,alpha,beta)
13 [sigma]=abs(1-x.^2.*alpha(2).*beta(1).*alpha(4).*beta(3).*beta(4)...
14 +x.*alpha(4).*beta(1).*beta(2).*beta(3)...
15 -x.*alpha(4).*beta(1).*beta(2).*beta(3).*beta(4)...
16 -x.^2.*alpha(1).*beta(1).*y.^2.*alpha(2).*alpha(3).*alpha(4).*beta(4)...
17 -x.^2.*alpha(1).*beta(1).*y.^2.*alpha(2).*alpha(3).*beta(3).*alpha(4)...
18 -x.^2.*alpha(1).*beta(1).*alpha(2).*beta(2).*y.^2.*alpha(3).*alpha(4)...
19 +x.^4.*alpha(1).*beta(1).*alpha(2).*beta(2).*alpha(3).*beta(3).*alpha(4).*beta(4)...
20 +x.^3.*alpha(1).*beta(1).*alpha(3).*beta(2).*alpha(4).*beta(4)...
21 +x.^2.*alpha(1).*beta(1).*alpha(4).*beta(2)...
22 -x.^2.*alpha(1).*beta(1).*alpha(4).*beta(2).*beta(4)...
23 -x.^2.*alpha(1).*beta(1).*alpha(4).*beta(2).*beta(3)...
24 +x.^2.*alpha(1).*beta(1).*alpha(4).*beta(2).*beta(3).*beta(4)...
25 +x.^3.*alpha(2).*beta(1).*alpha(3).*beta(3).*alpha(4).*beta(4)...
26 +x.*alpha(2).*beta(1).*beta(2).*y.^2.*alpha(3).*alpha(4)...
27 +x.^2.*alpha(2).*beta(1).*beta(2).*alpha(4).*beta(3).*beta(4)...
28 +x.^2.*alpha(3).*beta(1).*alpha(4).*beta(4)...
29 -x.^2.*alpha(3).*beta(1).*beta(3).*alpha(4).*beta(4)...
30 -x.^2.*alpha(3).*beta(1).*beta(2).*alpha(4).*beta(4)...
31 +x.^2.*alpha(3).*beta(1).*beta(2).*beta(3).*alpha(4).*beta(4)...
32 -x.*alpha(4).*beta(1).*beta(4)+x.*alpha(4).*beta(1).*beta(3).*beta(4)...
33 +x.*alpha(4).*beta(1).*beta(2).*beta(4)...
34 -y.^2.*alpha(1).*x.^2.*alpha(2).*beta(2).*alpha(3).*beta(3).*alpha(4)...
35 +y.^2.*alpha(1).*alpha(2).*x.*alpha(4).*beta(3).*beta(4)...
36 +y.^4.*alpha(1).*alpha(2).*alpha(3).*alpha(4)...
37 -y.^2.*alpha(1).*x.*alpha(3).*beta(2).*alpha(4)...
38 +i.*y.*alpha(1).*x.^3.*alpha(2).*beta(2).*alpha(3).*beta(3).*alpha(4).*beta(4)...
39 +i.*y.*alpha(1).*x.^2.*alpha(2).*beta(2).*alpha(4).*beta(3)...
40 -x.^2.*alpha(2).*beta(1).*beta(2).*alpha(4).*beta(3)...
41 +x.*alpha(4).*beta(4)+i.*y.*alpha(4)...
42 -x.*alpha(2).*beta(1).*y.^2.*alpha(3).*alpha(4)...
43 -y.^2.*alpha(2).*x.*alpha(3).*beta(3).*alpha(4)...
44 -x.*alpha(4).*beta(1).*beta(3)-x.*alpha(4).*beta(1).*beta(2)...
45 +x.*alpha(4).*beta(3)-x.*alpha(4).*beta(3).*beta(4)...
46 -x.*alpha(4).*beta(2).*beta(4)-x.*alpha(4).*beta(2).*beta(3)...
47 +x.^3.*alpha(1).*beta(1).*alpha(2).*beta(2).*alpha(4).*beta(3)...
48 -x.^3.*alpha(1).*beta(1).*alpha(2).*beta(2).*alpha(4).*beta(3).*beta(4)...
49 -y.^2.*alpha(3).*alpha(4)+x.^2.*alpha(3).*beta(3).*alpha(4).*beta(4)...
50 +i.*y.*alpha(3).*x.*alpha(4).*beta(4)+i.*x.*alpha(3).*beta(3).*y.*alpha(4)...
51 -x.*alpha(2).*beta(2).*y.^2.*alpha(3).*alpha(4)...
52 +x.^3.*alpha(2).*beta(2).*alpha(3).*beta(3).*alpha(4).*beta(4)...
53 +x.^2.*alpha(2).*beta(2).*alpha(4).*beta(3)...
54 -x.^2.*alpha(2).*beta(2).*alpha(4).*beta(3).*beta(4)...
55 +x.^2.*alpha(3).*beta(2).*alpha(4).*beta(4)...
56 -x.^2.*alpha(3).*beta(2).*beta(3).*alpha(4).*beta(4)...
57 +x.*alpha(4).*beta(2)+x.*alpha(4).*beta(2).*beta(3).*beta(4)...
58 -y.^2.*alpha(2).*alpha(3).*x.*alpha(4).*beta(4)...
59 +i.*y.*alpha(2).*x.^2.*alpha(3).*beta(3).*alpha(4).*beta(4)...
60 +i.*y.*alpha(2).*x.*alpha(4).*beta(3)...
61 +i.*x.^2.*alpha(2).*beta(2).*y.*alpha(3).*alpha(4).*beta(4)...
62 +i.*x.^2.*alpha(2).*beta(2).*alpha(3).*beta(3).*y.*alpha(4)...
63 +i.*x.*alpha(3).*beta(2).*y.*alpha(4)-i.*y.^3.*alpha(2).*alpha(3).*alpha(4)...
64 -i.*y.*alpha(2).*x.*alpha(4).*beta(3).*beta(4)...
65 -i.*x.*alpha(3).*beta(2).*beta(3).*y.*alpha(4)...
66 +i.*x.*alpha(3).*beta(1).*y.*alpha(4)+x.*alpha(4).*beta(1)...
67 +x.^2.*alpha(2).*beta(1).*alpha(4).*beta(3)...
68 -x.^3.*alpha(1).*beta(1).*alpha(3).*beta(2).*alpha(4).*beta(3).*beta(4)...
69 -x.^3.*alpha(2).*beta(1).*beta(2).*alpha(3).*beta(3).*alpha(4).*beta(4)...
70 -y.^2.*alpha(1).*alpha(2).*x.^2.*alpha(3).*beta(3).*alpha(4).*beta(4)...
71 +i.*y.*alpha(1).*x.^2.*alpha(3).*beta(2).*alpha(4).*beta(4)...

```



```

72 -y.^2.*alpha(1).*alpha(2).*x.*alpha(4).*beta(3)...
73 +i.*y.*alpha(1).*x.*alpha(4).*beta(2)...
74 +i.*y.*alpha(1).*x.*alpha(4).*beta(2).*beta(3).*beta(4)...
75 +i.*x.^3.*alpha(1).*beta(1).*alpha(2).*beta(2).*y.*alpha(3).*alpha(4).*beta(4)...
76 +i.*x.^3.*alpha(1).*beta(1).*alpha(2).*beta(2).*alpha(3).*beta(3).*y.*alpha(4)...
77 +i.*x.^2.*alpha(1).*beta(1).*alpha(3).*beta(2).*y.*alpha(4)...
78 +i.*x.^3.*alpha(1).*beta(1).*y.*alpha(2).*alpha(3).*beta(3).*alpha(4).*beta(4)...
79 +i.*x.^2.*alpha(1).*beta(1).*y.*alpha(2).*alpha(4).*beta(3)...
80 +i.*x.^2.*alpha(2).*beta(1).*y.*alpha(3).*alpha(4).*beta(4)...
81 +i.*x.^2.*alpha(2).*beta(1).*alpha(3).*beta(3).*y.*alpha(4)...
82 +i.*x.*alpha(3).*beta(1).*beta(2).*beta(3).*y.*alpha(4)...
83 +y.^2.*alpha(1).*x.*alpha(3).*beta(2).*beta(3).*alpha(4)...
84 -y.^2.*alpha(1).*x.^2.*alpha(2).*beta(2).*alpha(3).*alpha(4).*beta(4)...
85 -i.*y.^3.*alpha(1).*x.*alpha(2).*beta(2).*alpha(3).*alpha(4)...
86 -i.*y.*alpha(1).*x.^2.*alpha(2).*beta(2).*alpha(4).*beta(3).*beta(4)...
87 -i.*y.*alpha(1).*x.^2.*alpha(3).*beta(2).*beta(3).*alpha(4).*beta(4)...
88 -i.*y.*alpha(1).*x.*alpha(4).*beta(2).*beta(4)...
89 -i.*y.*alpha(1).*x.*alpha(4).*beta(2).*beta(3)...
90 -i.*x.^2.*alpha(1).*beta(1).*alpha(3).*beta(2).*beta(3).*y.*alpha(4)...
91 -i.*x.^2.*alpha(1).*beta(1).*y.*alpha(2).*alpha(4).*beta(3).*beta(4)...
92 -i.*x.*alpha(1).*beta(1).*y.^3.*alpha(2).*alpha(3).*alpha(4)...
93 -i.*x.^2.*alpha(2).*beta(1).*beta(2).*y.*alpha(3).*alpha(4).*beta(4)...
94 -i.*x.^2.*alpha(2).*beta(1).*beta(2).*alpha(3).*beta(3).*y.*alpha(4)...
95 -i.*x.*alpha(3).*beta(1).*beta(3).*y.*alpha(4)...
96 -i.*x.*alpha(3).*beta(1).*beta(2).*y.*alpha(4)...
97 -i.*y.^3.*alpha(1).*alpha(2).*alpha(3).*x.*alpha(4).*beta(4)...
98 -i.*y.^3.*alpha(1).*alpha(2).*x.*alpha(3).*beta(3).*alpha(4));

```

```

1  function Euler2DHull(CFL,e2,e4,colour,varargin)
2
3  Hull2D(CFL,e2,e4,@Euler2D,colour,varargin{:});
4
5  %% Hull2D
6  %% Description
7  % Plots convex hull of all eigenvalues scaled by CFL coefficient.
8  %% Input parameters
9  %
10 % * CFL - CFL coefficient
11 % * e2 - 2nd difference artificial dissipation coefficient
12 % * e4 - 4th difference artificial dissipation coefficient
13 % * lambdafxn - Hook to function for calculating eigenvalues of difference
14 % equation (eg. @Euler2D)
15 % * colour - boolean for colour plot
16 % * varargin - additional arguments to pass to lambdafxn
17
18 %% Definition
19 function Hull2D(CFL,e2,e4,lambdafxn,colour,varargin)
20
21 % A script to calculate all the sigma eigenvalues for a range of k's
22
23 format long;
24 %colour=1;
25
26 n=25;
27 thetax=-pi:2*pi/n:pi;
28 thetay=-pi:2*pi/n:pi;
29 [N]=max(size(thetax));
30 allreal=zeros(N*N,1);
31 allimag=zeros(N*N,1);
32 minlambda=10.0;
33
34 hold on;
35 set(gca,'linewidth',2.0);
36 set(gca,'fontsize',14);
37 set(gca,'fontweight','demi');
38 set(gca,'xtickmode','auto');
39 set(gca,'ytickmode','auto');
40 kk=0;
41 kkhi=0;
42 for ii=1:1:N
43     for jj=1:1:N
44         eeig=CFL*feval(lambdafxn,e2,e4,thetax(ii),thetay(jj),varargin{:});
45         [nr,nc]=size(eeig);
46         for e=1:1:nr
47             x=real(eeig(e));
48             y=imag(eeig(e));
49             allreal(kk+e,1)=x;
50             allimag(kk+e,1)=y;
51             minlambda=min(minlambda,x);
52         end
53         if thetax(ii) >= pi/2 | thetax(ii) <= -pi/2 | thetay(jj) >= pi/2 | thetay(jj) <= -pi/2
54             for e=1:1:nr
55                 x=real(eeig(e));
56                 y=imag(eeig(e));
57                 allhireal(kkhi+e,1)=x;
58                 allhiimag(kkhi+e,1)=y;
59             end
60             kkhi=kkhi+1;
61             % if colour ~= 0
62             %     plotcolour='bo';
63             % else
64             %     plotcolour='ko';
65             % end
66             % for e=1:1:nr
67             %     x=real(eeig(e));
68             %     y=imag(eeig(e));
69             %     plot(x,y,plotcolour);
70             %     plot(x,-y,plotcolour);
71             % end

```

```

72     end
73     kk=kk+1;
74 end
75 end
76 display(['min lambda=', num2str(minlambda)]);
77 k=convhull(allreal, allimag);
78 khi=convhull(allhireal, allhiimag);
79 if colour ~= 0
80     plot(allreal(k), allimag(k), 'LineWidth', 2.0, 'Color', 'Blue', 'LineStyle', '--');
81     plot(allhireal(khi), allhiimag(khi), 'LineWidth', 2.0, 'Color', 'Blue', 'LineStyle', '-');
82 else
83     plot(allreal(k), allimag(k), 'LineWidth', 2.0, 'Color', 'Black', 'LineStyle', '--');
84     plot(allhireal(khi), allhiimag(khi), 'LineWidth', 2.0, 'Color', 'Black', 'LineStyle', '-');
85 end
86 %xlim([min(allreal(k)) max(allreal(k))]);
87 %ylim([min(allimag(k)) max(allimag(k))]);

```



```

function Euler2DSigma(alpha,beta,CFL,e2,e4,colour,varargin)

Sigma2D(alpha,beta,CFL,e2,e4,@Euler2D,colour,varargin{:});

%% Sigma2D
%% Description
% Plots magnitude of sigma for all eigenvalues over a range of Fourier
% modes
%% Input parameters
%
% * alpha - vector of alpha coefficients
% * beta - vector of beta coefficients
% * CFL - CFL coefficient
% * e2 - 2nd difference artificial dissipation coefficient
% * e4 - 4th difference artificial dissipation coefficient
% * lambdafxn - Hook to function for calculating eigenvalues of difference
% equation (eg. @Euler2D)
% * colour - boolean for colour plot
% * varargin - additional arguments to pass to lambdafxn

%% Definition
function Sigma2D(alpha,beta,CFL,e2,e4,lambdafxn,colour,varargin)

% A script to calculate all the sigma eigenvalues for a range of k's

hold on;
xlim([0 pi]);
ylim([0 pi]);
xlabel('\theta_x','fontsize',14);
ylabel('\theta_y','fontsize',14);
axis square;
set(gca,'linewidth',2.0);
set(gca,'fontsize',14);
set(gca,'fontweight','demi');
set(gca,'fontname','Symbol');
set(gca,'xtickmode','manual');
set(gca,'xtick',[0;pi]);
set(gca,'xticklabel',{'0';'p'});
set(gca,'ytickmode','manual');
set(gca,'ytick',[0;pi]);
set(gca,'yticklabel',{'0';'p'});

RKN=max(size(alpha));
[rksigma]=sprintf('rk%dsigma',RKN);

n=150;
thetax=0:pi/n:pi;
thetay=0:pi/n:pi;
[N]=max(size(thetax));

maxsigma=0;
abssigma=zeros(N,N);
for ii=1:N
    for jj=1:N
        eeig=CFL*feval(lambdafxn,e2,e4,thetax(ii),thetay(jj),varargin{:});
        [nr,nc]=size(eeig);
        for e=1:nr
            x=real(eeig(e));
            y=imag(eeig(e));
            temp=feval(rksigma,x,y,alpha,beta);
            abssigma(ii,jj)=max(abssigma(ii,jj),temp);
        end
        if thetax(ii)>= pi/2 | thetay(jj)>=pi/2
            maxsigma=max(maxsigma,abssigma(ii,jj));
        end
    end
end

mycolormap=zeros(64,3);
if colour == 1
    for ii=1:1:64

```

```

mycolormap(ii,1)=(ii-1)/63;
mycolormap(ii,2)=1.0-(ii-1)/63;
mycolormap(ii,3)=0.0;
end
else
for ii=1:1:64
mycolormap(ii,1)=1.0-(ii-1)/63*0.5;
mycolormap(ii,2)=mycolormap(ii,1);
mycolormap(ii,3)=mycolormap(ii,1);
end
end
colormap(mycolormap);
[C,H]=contourf(thetax,thetay,abssigma,[0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1]);
%set(H(:),'LineStyle','none');
%axis equal tight;
[h]=colorbar('EastOutside');
set(h,'YLim',[0 1]);
[htitle]=get(h,'Title');
set(htitle,'String','|\sigma|','FontSize',14,'FontWeight','demi');
set(H(:),'linewidth',1.0);
%if colour==1
% clabel(C,h,'fontsize',13,'fontweight','bold','color','white');
%else
% clabel(C,h,'fontsize',13,'fontweight','bold','color','black');
%end
line([0 pi],[pi/2 pi/2],[0 0],'Color','k','LineStyle','--');
line([pi/2 pi/2],[0 pi],[0 0],'Color','k','LineStyle','--');
hold off;
display(['Max sigma: ',num2str(maxsigma)]);

```

REFERENCES

1. Allmaras, S. "Analysis of a local matrix preconditioner for the 2-D Navier-Stokes Equations." AIAA Paper 93-3330-CP, 11th Computational Fluid Dynamics Conference, Orlando, FL, 1993.
2. Pierce, Niles A., Giles, Michael B. "Preconditioned Multigrid Methods for Compressible Flow Calculations on Stretched Meshes." Oxford university Computing Laboratory, Numerical Analysis Group, Oxford OX1 3QD, United Kingdom. Received April 29, 1996; revised May 29, 1997.
3. Pierce, Niles A., Giles, Michael B. "Preconditioning compressible flow calculations on stretched meshes." Oxford Univ., United Kingdom. AIAA 34th Aerospace Sciences Meeting and Exhibit, Reno, NV. Jan 15-18, 1996.
4. [Weisstein, Eric W. "Relaxation Methods." From *MathWorld*--A Wolfram Web Resource.](http://mathworld.wolfram.com/RelaxationMethods.html)
<http://mathworld.wolfram.com/RelaxationMethods.html>
5. <http://www.answers.com/topic/regular-grid>
6. Darmofal, D. "Structured vs. Unstructured Grids", *MIT Opencourseware*, Lecture 48, 2005.
http://ocw.mit.edu/NR/rdonlyres/Aeronautics-and-Astronautics/16-100Fall-2005/D63339BE-639F-4FB0-A971-D3262778B192/0/16100lectre48_cj.pdf
7. Walsh, P. "Computational Fluid Dynamics and Heat Transfer – 1st Edition" *Lecture Notes* 2006.
8. Wesseling, P. "Introduction to Multigrid Methods" *ICASE Report* No. 95-11, February 1995.
9. Lomax, Pulliam, Zingg "Fundamentals of Computational Fluid Dynamics" 1999.
10. WarBurton, Tim "Numerical Methods for Partial Differential Equations" Course: CAAM 452 Spring 2005 Lecture 4 www.caam.rice.edu/~caam452/CAAM452Lecture4b.ppt
11. "Agglomeration." The American Heritage® Dictionary of the English Language, Fourth Edition, Copyright © 2006 by Houghton Mifflin Company. Published by Houghton Mifflin Company.
12. Oracle. "Oracle Berkeley DB" Copyright 2006, Oracle. All Rights Reserved.
<http://www.oracle.com/database/docs/berkeley-db-datasheet.pdf>
13. Lassaline, J.V. "Optimal Multistage Relaxation Coefficients for Multigrid Flow solvers." Ryerson University 2007.

14. MATLAB Documentation. "Optimization Toolbox – fgoalattain." Referenced January 11, 2009.
15. Lassaline, J.V. "A Navier-Stokes equation solver using agglomerated multigrid featuring directional coarsening and line-implicit smoothing." University of Toronto 2003.
16. Lassaline, J.V. "A Navier-Stokes Equation Solver Using Agglomerated Multigrid Featuring Directional Coarsening and Line-Implicit Smoothing" PhD thesis, University of Toronto, 2003.
17. Pierce, N. A., Giles, M.B., Jameson, A., and Martinelli, L. "Accelerating three-dimensional Navier-Stokes calculations." AIAA Paper 97-1953, AIAA, 1997.
18. Mavriplis D. J., Venkatakrishnan V. "Agglomeration multigrid for two dimensional viscous flows." *Comput. Fluids*, 24, pp. 553-570, 1995.
19. "relaxation method." (2009). In *Encyclopedia Britannica*. Retrieved January 11, 2009, from Encyclopdeia Britannica
Online: <http://www.britannica.com/EBchecked/topic/496925/relaxation-method>
20. Hosseini, K. and Alonso, J. J., "Optimization of multistage coefficients for explicit multigrid flow solvers." AIAA Paper 2003-3075, AIAA, 2003.
21. Tai, C.-H., Sheu, J.-H., and Leer, B. "Optimal multistage schemes for Euler equations with residual smoothing." *AIAA J.*, 33:1008-1016, 1995.
22. MATLAB Documentation. "Optimization Toolbox – fmincon." Referenced January 26, 2009.
23. Van Leer, B., Tai, C.-H., and Powell, K.H., "Design of Optimally Smotthing Multi-stage Scheme for the Euler Equations." AIAA Paper 89-1923-CP, June 1989.
24. Versteeg, H. K., and Malalasekera, W. "An introduction to computational fluid dynamics The finite volume method." Longman Scientific & Technical. Longman House Burnt Mill, Harlow Essex CM20 2JE, England. 1995.