CHOICE AND DEVELOPMENT OF A PRECONDITIONER FOR NEWTON-GMRES ALGORITHM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

YUNUS EMRE MUSLUBAŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE MASTER OF SCIENCE
IN
AEROSPACE ENGINEERING

SEPTEMBER 2015

Approval of the thesis:

CHOICE AND DEVELOPMENT OF A PRECONDITIONER FOR NEWTON-GMRES ALGORITHM

submitted by **YUNUS EMRE MUSLUBAŞ** in partial fulfillment of the requirements for the degree of **Master of Science in Aerospace Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Ozan Tekinalp
Head of Department, **Aerospace Engineering** _____

Assoc. Prof. Dr. Sinan Eyi
Supervisor, **Aerospace Engineering Dept., METU** _____

Examining Committee Members:

Prof. Dr. Yusuf Özyörük
Aerospace Engineering Dept., METU _____

Assoc. Prof. Dr. Sinan Eyi
Aerospace Engineering Dept., METU _____

Assoc. Prof. Dr. Burak Aksoylu
Dept. of Mathematics, TOBB ETU _____

Assoc. Prof. Dr. Murat Manguoğlu
Dept. of Computer Engineering, METU _____

Asst. Prof. Dr. Ali Türker Kutay
Aerospace Engineering Dept., METU _____

Date: **07.09.2015**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: YUNUS EMRE MUSLUBAŞ

Signature :

ABSTRACT

CHOICE AND DEVELOPMENT OF A PRECONDITIONER FOR NEWTON-GMRES ALGORITHM

Muslubaş, Yunus Emre

M.S., Department of Aerospace Engineering

Supervisor: Assoc. Prof. Dr. Sinan Eyi

September 2015, 90 pages

This thesis consists of the choice, application and analysis of a preconditioner for a supersonic flow solution through Newton-GMRES (generalized minimal residual) Krylov subspace method and the comparison of the results with unpreconditioned Newton-GMRES method and Newton's methods. Three dimensional Euler equations are used for the analysis. These Euler equations are discretized, then solved using Newton's method and the generalized minimal residual method is used to solve the resulting linear system. The results and the computational time for this Newton-GMRES method approach are then obtained to be compared with those for the same method preconditioned using incomplete lower-upper factorization and the regular Newton's method. The calculation of the Jacobian matrix necessary for the preconditioner and the Newton's method is done analytically.

Keyword: Preconditioning, Newton-GMRES, Jacobian, Computational Times, Euler Equations, Inexact Newton Methods

ÖZ

NEWTON-GMRES ALGORİTMALARINDA ÖNŞARTLANDIRICI SEÇİM VE GELİŞTİRMESİ

Muslubaş, Yunus Emre
Yüksek Lisans, Havacılık ve Uzay Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Sinan Eyi

Eylül 2015, 90 sayfa

Bu tez, Newton-GMRES (genel minimal kalıntı) Krylov altuzay metodu aracılığıyla süpersonik akış çözümü için bir önşartlandırıcı seçimi, uygulaması ve analizi ve buradan elde edilen sonuçların önşartlandırılmamış Newton-GMRES metodu ve Newton metoduyla karşılaştırılmasını içermektedir. Analiz için üç boyutlu Euler denklemleri kullanılmaktadır. Bu Euler denklemleri ayrıklaştırılmış, ardından Newton yöntemiyle çözülürken elde edilen lineer sistemin çözümü için genel minimal kalıntı metodu kullanılmıştır. Bu Newton-GMRES metodu yaklaşımı için sonuçlar ve bilgisayar hesaplama zamanları elde edilmiş ve aynı metodun tamamlanmamış alt-üst faktörizasyon kullanılarak önşartlandırılmış versiyonu ve Newton metodu ile elde edilen sonuç ve zamanlarla karşılaştırılmıştır. Önşartlandırıcı ve Newton metodları için gerekli olan Jacobian matrisinin hesaplanılması analitik olarak yapılmıştır.

Anahtar Kelimeler: Önşartlandırma, Newton-GMRES, Jacobian, Hesaplama Zamanları, Euler Denklemleri

To my patient and loving family

...and all those whose love keeps me going

ACKNOWLEDGMENTS

I would, first of all, like to express my undying gratitude to my family who, no matter what, supported me in all my decisions throughout my studies and has never uttered a discouraging word.

I, as any other student of this beautiful university should, have to offer my thanks to Middle East Technical University as a whole for making me a better person.

My deepest, most heartfelt thanks have to be offered to my advisor Assoc. Prof. Dr. Sinan Eyi, who went far above and beyond the call of duty to help me in my work and without whom this thesis would not be possible.

My friends and all those who love me and have supported me through this arduous process, while I would not have the space here to mention every single one of you, none of you are forgotten.

Lastly, special thanks have to be dedicated to those who have kept me company and shared their wisdom with me every time I needed it, of whom the first ones that come to mind are Engin Leblebici, Mehmet Harun Özkanaktı and Derya Kaya.

This study was supported by The Scientific and Technological Research Council of Turkey (TÜBITAK) with the project number 112M129.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF SYMBOLS

F,G,H	Flux Vectors
Q	Flow Variable Vector
ρ	Density
u, v, w	Components of the Velocity Vector
p	Pressure
e_t	Total Energy Per Unit Volume
γ	Specific Heat Ratio
U,V,W	Contravariant Velocity Components
J	Coordinate Transformation Jacobian
ξ, η,ζ	Curvilinear Coordinates
c	Speed of Sound
R	Residual
A	Matrix
M	Preconditioning Matrix
M	Mach Number
K	Krylov Subspace
v	Krylov Subspace Search Direction
η_k	Relative Tolerance
m,n	Iteration Counters, Matrix Sizes
H,h	Hessenberg Matrix and Its Elements
z	Preconditioned Search Direction
L,U	Lower and Upper Matrices
D	Diagonal of Matrix
l,u,d	Elements of Lower, Upper and Diagonal Matrices
Lfil, lev	Level of Fill
ρ,τ	Level of Fill and Threshold for ILUT(ρ,τ)

LIST OF ABBREVIATIONS

BiCGSTAB	Biconjugate Gradient Stabilized
BC	Boundary Condition
CG	Conjugate Gradient
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
FVM	Finite Volume Method
GMRES	Generalized Minimal Residual Method
ILU	Incomplete Lower Upper Factorization
ILUT	Incomplete Lower Upper Factorization with Thresholding
JFNK	Jacobian-Free Newton-Krylov
MILU	Modified Incomplete Lower Upper Factorization
MUSCL	Monotonic Upstream Centered Scheme Conservation Law
SPAI	Sparse Approximate Inverse
METU	Middle East Technical University
3D	Three Dimensional

CHAPTER 1

INTRODUCTION

1.1 Background

Humanity has always been interested in explaining the matter that surrounds them. From philosophy to chemistry to physics, scientists and great thinkers of their times have tried to understand and make known the mystery of how things work. The collection of all the data and the studies done has led those thinkers to a point that today one can explain most that goes on around them. From how things burn to how our cardiovascular system works, almost every physical or chemical phenomenon can be explained. To achieve that purpose, modern scientists use a plethora of different methods that might be as simple as submerging an object in water to see how much water it displaces or as complex as creating a particle collider with a circumference of 27 kilometers. From amongst an endless variety of subjects, one of the most interesting is how air or water or any other type of matter that behaves like a fluid moves. That particular area of interest is covered by fluid mechanics, especially with its subdiscipline, fluid dynamics. This subdiscipline is concerned with the flow of fluids, from the movement of water in a pipe to how air behaves when passing over the wing of an aircraft at supersonic speeds. It offers the scientists the tools they need to analyze systems and solve practical problems by extending them the help of empirical and semi-empirical laws [1]. These tools, when applied to a certain area of interest such as aircraft design, lets one understand what their design will face under real conditions, which leads to immense savings, negating the need to build prototypes that would fail and waste time and resources or conduct experiments that would take a great amount of work. Thus, to eliminate or at least largely reduce the necessity for such efforts, computational fluid dynamics (CFD) is being used as an essential part of aircraft

design. This type of tool, as the name suggests, uses the relatively cheap computational power of resources that mankind has made available to themselves through algorithms that are generally called flow solution algorithms or flow solvers. These flow solvers use computers to perform the calculations required to simulate the interaction of liquids and gases with surfaces defined by boundary conditions through the application of routines that were too tedious to complete using ordinary human resources.

In contrast to earlier times, the advancement of computational capabilities in modern machines and the development and upgrades of pre-existing solvers or solution algorithms enable the solution of larger linear systems of equations than was possible before. In addition to this, the stiffness of problems to be evaluated in aeronautical practices necessitates careful bounding of errors to maintain stability. This combination leads to more widespread usage of implicit algorithms that have a tendency to obtain more stability and lower residuals while utilizing a low number of iterations compared to those of explicit algorithms. The subject of this thesis, supersonic flows, having the characteristics of high Mach number, high enthalpy and strong entropy gradients, are more unpredictable and harder to model than their subsonic counterparts, requiring a larger number of species and reaction equations. To combat the necessity to use impractically small time steps to obtain numerical stability via explicit methods, implicit methods, while harder to implement, are utilized. These implicit methods lead to stronger convergence in relation to their alternatives. Newton's method, with its quadratic convergence, is one such that is used for the solution of non-linear equations [2]. The pitfall of this method is the necessity to obtain and solve a Jacobian matrix that becomes larger and more complex as the problem chosen to be analyzed gets more complex. The evaluation of this Jacobian matrix is done through the derivation of the residual vector with respect to flow variable vector. While the accuracy of this method is outstanding, if a proper initial solution is not provided, the convergence of the Newton's method may become improbable. Thus, to remove the problem of divergence while keeping the accuracy of this method, other methods are being implemented. The Jacobian-free Newton-Krylov methods are combinations of Newton methods for the solution of nonlinear equations and the Krylov subspace methods for the solution of Newton correction equations. One of these JFNK methods is the Jacobian-free Newton-GMRES method that is used in this

thesis and in various CFD areas, some of which can be observed in the works of Knoll and Keyes [3]. The GMRES method, developed by Saad and Schultz, still needs adequate preconditioning to be successfully applied to the problem at hand as with all other Krylov methods [4]. The application of this preconditioning can differ in method, sometimes necessitating the approximation of the Jacobian to be viable. For this thesis, right preconditioning is aimed to be achieved via incomplete lower-upper factorization.

In this thesis, the performances of Newton's and preconditioned Newton-GMRES method are compared for supersonic flow analysis. For flow analysis, a cell centered finite volume code is developed by using the three dimensional Euler equations. The fluxes are computed using Van-Leer upwind scheme [5]. Flow equations are solved implicitly by Newton and Newton-GMRES methods. The Jacobian matrix needed for Newton's method is evaluated analytically by differentiating Van Leer fluxes for both first and second order discretization. In contrast to this, the Jacobian needed for the preconditioner used in Newton-GMRES method is based on the first order discretization and only its main block-diagonal is kept. This allows for computational efficiency without losing accuracy. In Newton method, the sparse system with the full Jacobian matrix is solved using UMFPACK and PARDISO sparse matrix solver. The boundary conditions are implemented implicitly [6, 7, 8].

1.2 Scope Of The Thesis

As was mentioned in the previous section, flow solvers are very varied in types and applications. The object of this study is to focus on a solver that is being used by many of my peers for various purposes and make it better through the application of a suitable preconditioner. The solver that is chosen for that purpose is the Newton-GMRES solver. The Newton-GMRES solver is widely regarded to be efficient, but has been overcome in terms of speed as other solvers' efficiencies increase. The preconditioning of this solver is aimed to be achieved to lower computational times in its usage for larger problems. For that purpose, an appropriate preconditioner selection is the first step. After the preconditioning process is successfully done, the preconditioned method is to be applied to a problem that takes noticeable time to be solved to better see the difference it makes. The validation of this process is to be done

through the comparison of the results and CPU times with two methods that have been proven successful in previous works, namely the Newton and Newton-GMRES methods [9].

1.3. Literature Research

To provide an understanding to the reader about what has been done in the past regarding the subject of this thesis, a short reminder about the closely related history of CFD will be included here. The beginning point of such an endeavor should be located at the late eighties and early nineties when CFD was gaining favor in terms of implicit methods. These methods allowed researchers to combine the necessary equations of their respective areas with flow equations. Newton's method was one such method that gave results in a lower number of iterations; thus making it less computationally greedy with higher convergence. This led to its implementations in numerous 3D applications. Yet, this usage was later found to be not as useful as previously thought, which Venkatakrishnan demonstrated in his work on solution of transonic flows over an airfoil [10]. While the Newton's method was successfully employed in his work, he stated that a mixture of appropriate preconditioning and Jacobian freezing was necessary to overcome immensely taxing CPU necessities. This was further complicated by the fact that with growing systems the CPU power needed was growing to unreachable points by the technology of that time. This led to an increase in interest towards quasi-Newton methods.

Quasi-Newton method is the collective name given to any method that replaces the exact Jacobian calculated in Newton's method with any approximation. They combine the Newton's method with a linear solver to be applied to the Newton equations that are created in each step of the solution. The convergence of these methods is almost always inferior to Newton's method, yet quasi-Newton methods make for more efficient solvers due to the lack of a Jacobian matrix calculation and storage at every iteration. Instead of the calculation of a Jacobian matrix, quasi-Newton methods simplify or approximate that Jacobian, making the calculations much less demanding on the hardware. One such quasi-Newton method family is Newton-Krylov methods which employ Krylov subspaces. These Krylov subspace methods work through the creation of a sequence of successive matrix powers times the initial residual [11]. Then

this subspace that has been created is used to minimize the residual. These Krylov subspace methods have three prevalent uses in conjugate gradient method, biconjugate gradient method and generalized minimum residual method invented by Yousef Saad in 1986, just when the implicit solvers were coming into more widespread use [12]. The convergence of GMRES method is often well as the method itself works by creating spaces smaller than the matrix that would normally be constructed and checking if the residual is within acceptable limits. This process starts with the creation of the smallest possible space and with each iteration of the inner loop, the space is grown larger to contain the previous space. The combination of this and a clever subspace creation algorithm allows the user to achieve a minimum residual within a manageable number of steps without ever needing to form up a matrix that is equal in size to a full Jacobian necessitated by the Newton's method. While this is true in most practical cases, works of A. Greenbaum, V. Ptak and Z. Strakos if stated in simple terms tell us that for every monotonically decreasing sequence (as with residuals in Krylov methods), one can find a matrix A such that there is a need to calculate residuals equal times in number to the matrix dimensions to find zero residuals [13]. Thus, to alleviate the problem of a possible slow convergence before it arises or to generally make GMRES algorithm faster, less memory-intensive and more robust, preconditioning is done. Yet, the preconditioning process itself is not without its expenses. There is the need to calculate or approximate and store at least one Jacobian matrix for the problem to be used as the basis of a preconditioning matrix. While preconditioning without the explicit computation of any matrix elements can be done through directional derivatives or other more complex methods can be done as suggested by R. Choquet or Y. Chen and C. Shen's works, these preconditioners are always case-dependent and not interchangeable according to the problem at hand [14, 15]. Thus, the usage of the more widespread incomplete lower-upper factorization, ILU methods in preconditioning are seen to be viable as suggested by the studies of several authors on the effects of preconditioning methods on convergence of matrix-free GMRES [16, 17, 18].

1.4. Outline

The first chapter consists of the introductions of the background of the study area, the scope of the thesis and literature research, through the examination of which the reader can understand what has led to the choice of this particular branch of research and the usage of these methods.

The second chapter introduces the flow model and the governing equations. The applied boundary conditions are described. The specifics regarding the usage of the Newton's method are explained.

The third chapter introduces the solution algorithms of Newton and Newton-GMRES methods and explains both in detail.

The fourth chapter focuses on the preconditioning of Newton-GMRES method. The reasons for preconditioning and the choice and application of a suitable preconditioner are discussed.

The fifth chapter compares the Newton's method with preconditioned and regular Newton-GMRES methods in terms of CPU times and convergence histories.

The last chapter provides the reader with a conclusion and future work possibilities.

CHAPTER 2

PROBLEM DEFINITION

The sample problem that was solved in order to showcase the properties of the Newton's, Newton-GMRES and preconditioned Newton-GMRES methods is defined in this chapter. To begin with, the reasoning behind the problem selection should be summarized. In modern day engineering, it is possible to take two distinct paths when dealing with an issue, let's say the design of an aircraft part. One approach is to design depending on previously successful theories and design experience and then test the part under a variety of conditions before usage. This approach is a fairly successful one, especially considering it has been applied for decades until humanity reached a higher form of sentience through the invention of computing machines. Yet, it is also an expensive approach as some parts and their models can be extremely taxing on research funds. Another approach is to apply the same test conditions to the part in a computerized environment to avoid most of the costs included in a real life test situation. When all the problems with the part have been detected, manufacturing a much more prototype to be tested in real life conditions will always be cheaper and much more intelligent. Thus, there is a need to create those test environments where aircraft parts or anything else for that matter can be thoroughly tested. That means we need to take the part, for example a supersonic nozzle and simulate what it will go through while working or in fact, what will go through it! Of course, the answer is "some type of flow" and that brings us to the topic of defining that flow and its characteristics. While modeling this flow, one has to consider the sizes of the grids they want to use, the boundary condition types and of course, how they want to discretize the equations with respect to time. Extremely detailed grids will cost a lot to compute as will extremely detailed equations describing the flow. So, making some small sacrifices without losing too much accuracy is a primary concern in flow modelling. These sacrifices here are using Euler Equations instead of Navier-Stokes

equations in the flow model and trying to achieve good accuracy when possible with smaller grids, which will be obvious to the reader while reading the results. This leads to the elaboration of Euler Equations in the first section.

2.1 Euler Equations

Euler Equations, chosen as they are simpler than Navier-Stokes Equations and thus easier on the computational power needed, are derived from conservation of mass, momentum and energy in a specified control volume. The 3D Euler Equations with steady inviscid flow can be described as below in Cartesian Coordinates:

$$\nabla \cdot \boldsymbol{F} = 0 \tag{1}$$

With

$$Q = \begin{bmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ \rho e_t \end{bmatrix} \text{and } \boldsymbol{F} = \begin{bmatrix} \rho u_1 & \rho u_2 & \rho u_3 \\ \rho u_1^2 + p & \rho u_1 u_2 & \rho u_1 u_3 \\ \rho u_1 u_2 & \rho u_2^2 + p & \rho u_2 u_3 \\ \rho u_3 u_1 & \rho u_3 u_2 & \rho u_3^2 + p \\ (\rho e_t + p)u_1 & (\rho e_t + p)u_1 & (\rho e_t + p)u_1 \end{bmatrix} \tag{2}$$

Which, written with more manageable terms of *u*,*v* and *w* becomes:

$$\frac{\partial \mathrm{F(Q)}}{\partial \mathrm{x}} + \frac{\partial \mathrm{G(Q)}}{\partial \mathrm{y}} + \frac{\partial \mathrm{H(Q)}}{\partial \mathrm{z}} = 0 \tag{3}$$

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho e_t \end{bmatrix} \; and \; F = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ (\rho e_t + p)u \end{bmatrix} \; G = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho wv \\ (\rho e_t + p)v \end{bmatrix} \; H = \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ (\rho e_t + p)w \end{bmatrix} \tag{4}$$

Where vector *Q* defines the flow variables and *F,G* and *H* are the flux vectors. *u, v* and *w* are the velocity components in x,y and z directions. ρ is the density, *p* is pressure and e_t is total energy per unit volume.

The pressure p is obtained from ideal gas relations:

$$p = (\gamma - 1)\rho(e_t - \frac{u^2 + v^2 + w^2}{2})$$

(5)

Where γ is specific heat ratio.

Here, we can convert the Cartesian Coordinates into generalized coordinates using the curvilinear coordinates ξ, η and ζ to work easier. After the transformation, the equation (3) can be written as:

$$\frac{\partial \hat{\mathrm{F}}(\hat{\mathrm{Q}})}{\partial \xi} + \frac{\partial \hat{\mathrm{G}}(\hat{\mathrm{Q}})}{\partial \eta} + \frac{\partial \hat{\mathrm{H}}(\hat{\mathrm{Q}})}{\partial \zeta} = 0$$

(6)

And equation system (4) becomes:

$$\hat{Q} = \frac{1}{\mathrm{J}}\begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho e_t \end{bmatrix} \text{ and } \hat{F} = \frac{1}{J}\begin{bmatrix} \rho U \\ \rho u U + \xi_x p \\ \rho v U + \xi_y p \\ \rho w U + \xi_z p \\ (\rho e_t + p)U \end{bmatrix} \hat{G} = \frac{1}{J}\begin{bmatrix} \rho V \\ \rho u V + \eta_x p \\ \rho v V + \eta_y p \\ \rho w V + \eta_z p \\ (\rho e_t + p)V \end{bmatrix} \hat{H} = \frac{1}{J}\begin{bmatrix} \rho W \\ \rho u W + \zeta_x p \\ \rho v W + \zeta_y p \\ \rho w W + \zeta_z p \\ (\rho e_t + p)W \end{bmatrix}$$

(7)

Where J is the transformation Jacobian:

$$\mathrm{J} = det\begin{bmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{bmatrix}$$

(8)

And *U*, *V* and *W* are the contravariant velocities:

$$U = \xi_x u + \xi_y v + \xi_z w$$

$$V = \eta_x u + \eta_y v + \eta_z w$$

$$W = \zeta_x u + \zeta_y v + \zeta_z w$$

(9)

2.2 Spatial Discretization

The method of spatial discretization used is Finite Volume Method (FVM). The flow domain is distributed onto cells in the center of which are the flow variables. The cell faces store the fluxes while the grid points occupy the same space as the cell corners. Then, the flux balance across a cell will result in the spatial derivatives of the flux vectors:

$$\delta_{\xi}\hat{F} = \hat{F}_{i+\frac{1}{2},j,k} - \hat{F}_{i-\frac{1}{2},j,k}$$
$$\delta_{\eta\xi}\hat{G} = G_{i,j+\frac{1}{2},k} - G_{i,j-\frac{1}{2},k}$$
$$\delta_{\zeta}\hat{H} = \hat{H}_{i,j,k+\frac{1}{2}} - \hat{H}_{i,j,k-\frac{1}{2}} \tag{10}$$

Where the cell faces are described by *i±1/2*, *j±1/2, k±1/2* and which turns Equation (6) into:

$$\frac{\partial\hat{\mathrm{F}}}{\Delta\xi} + \frac{\partial\hat{\mathrm{G}}}{\Delta\eta} + \frac{\partial\hat{\mathrm{H}}}{\Delta\zeta} = 0 \tag{11}$$

Which can be written as follows:

$$\left(\hat{F}_{i+\frac{1}{2},j,k} - \hat{F}_{i-\frac{1}{2},j,k}\right) + \left(\hat{G}_{i,j+\frac{1}{2},k} - \hat{G}_{i,j-\frac{1}{2},k}\right) + \left(\hat{H}_{i,j,k+\frac{1}{2}} - \hat{H}_{i,j,k-\frac{1}{2}}\right) = 0 \tag{12}$$

Interpolating from the flow variables in the cell centers, flow variables in the cell faces can be obtained. Then with those variables, flux can be obtained.

Now, $\hat{F}_{i\pm\frac{1}{2},j,k}, G_{i,j\pm\frac{1}{2},k}\ and\ \hat{H}_{i,j,k\pm\frac{1}{2}}$ can be rewritten with L being left and *R* being right:

$$\hat{F}_{i\pm\frac{1}{2},j,k} = \left[\hat{F}^{+}\left(\hat{Q}^{L}_{i\pm\frac{1}{2},j,k}\right) + \hat{F}^{-}\left(\hat{Q}^{R}_{i\pm\frac{1}{2},j,k}\right)\right]$$
$$\hat{G}_{i,j+\frac{1}{2},k} = \left[\hat{G}^{+}\left(\hat{Q}^{L}_{i,j\pm\frac{1}{2},k}\right) + \hat{G}^{-}\left(\hat{Q}^{R}_{i,j\pm\frac{1}{2},k}\right)\right]$$
$$\hat{H}_{i,j,k\pm\frac{1}{2}} = \left[\hat{H}^{+}\left(\hat{Q}^{L}_{i,j,k\pm\frac{1}{2}}\right) + \hat{H}^{-}\left(\hat{Q}^{R}_{i,j,ki\pm\frac{1}{2}}\right)\right] \tag{13}$$

And *Q* variables can be written in shorter terms via equating variables at cell faces to their counterparts at the nearest cell centers:

$$\hat{Q}^L_{i+\frac{1}{2},j,k} = \hat{Q}_i \ , \qquad \hat{Q}^L_{i,j+\frac{1}{2},k} = \hat{Q}_j \ , \qquad \hat{Q}^L_{i,j,k+\frac{1}{2}} = \hat{Q}_k$$

$$\hat{Q}^R_{i+\frac{1}{2},j,k} = \hat{Q}_{i+1} \ , \qquad \hat{Q}^R_{i,j+\frac{1}{2},k} = \hat{Q}_{j+1} \ , \qquad \hat{Q}^R_{i,j,k+\frac{1}{2}} = \hat{Q}_{k+1} \tag{14}$$

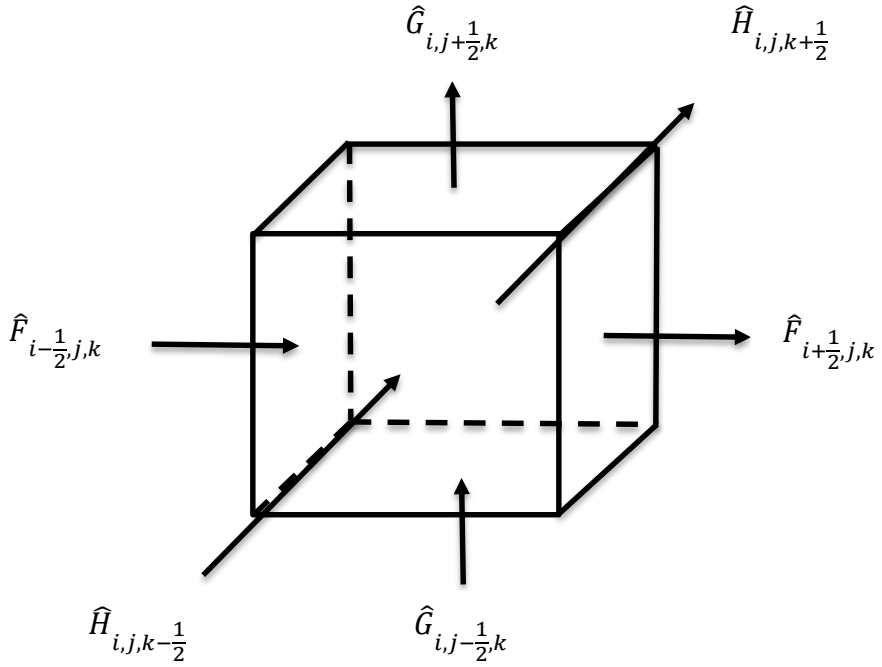

Figure 2.1 Control Volume

2.3 Flux Splitting

After $\hat{F}_{i\pm\frac{1}{2},j,k}$, $G_{i,j\pm\frac{1}{2},k}$ and $\hat{H}_{i,j,k\pm\frac{1}{2}}$ have been formed in terms of $\hat{F}^+, \hat{F}^-, \hat{G}^+, \hat{G}^-, \hat{H}^+$ and $\hat{H}^-$, they can be found via flux splitting. Here, Van Leer method is used for that purpose. The fluxes are split w.r.t. contravariant Mach number, *M* as follows:

Subsonic Case

$$F^{\pm} = \rho c \frac{(M+1)^2}{4}\left(\tilde{k}_1 + \tilde{k}_2 + \tilde{k}_3\right)\begin{bmatrix} 1 \\ \left(\frac{-\widetilde{U} \pm 2c}{\gamma}\right)\tilde{k}_1 + u \\ \left(\frac{-\widetilde{U} \pm 2c}{\gamma}\right)\tilde{k}_2 + v \\ \left(\frac{-\widetilde{U} \pm 2c}{\gamma}\right)\tilde{k}_3 + w \\ \left(\frac{-\widetilde{U} \pm 2c}{\gamma+1}\right)\widetilde{U} + \frac{2a^2}{\gamma^2-1} + \frac{u^2+v^2+w^2}{2} \end{bmatrix} \tag{15}$$

With

$$\widetilde{U} = \frac{u\xi_x + v\xi_y + w\xi_z}{\sqrt{\xi_x^2 + \xi_y^2 + \xi_z^2}}$$

$$\tilde{k}_1 = \frac{\xi_x}{\sqrt{\xi_x^2 + \xi_y^2 + \xi_z^2}}$$

$$\tilde{k}_2 = \frac{\xi_y}{\sqrt{\xi_x^2 + \xi_y^2 + \xi_z^2}}$$

$$\tilde{k}_3 = \frac{\xi_z}{\sqrt{\xi_x^2 + \xi_y^2 + \xi_z^2}} \tag{16}$$

Here, γ is the specific heat ratio and c is the speed of sound. The same can be done for $G^{\pm}$ and $H^{\pm}$by substituting η and ζ for ξ.

For the Supersonic Case,

$$F^{+} = F \; if \; M \geq 1$$
$$F^{+} = 0 \; if \; M \leq 1$$
$$F^{-} = 0 \; if \; M \geq 1$$
$$F^{-} = F \; if \; M \leq 1 \tag{17}$$

This should be followed by the boundary conditions section.

2.4 Boundary Conditions

The boundary conditions can be seen in Figure 2.1.

The problem is solved on one quarter of the problem as the same solution can be applied to the other three quarters. As can be seen in the following figure, this means we have 2 symmetry conditions, 2 wall conditions, 1 inlet and 1 outlet condition. The boundary conditions (BCs) are implemented through the usage of reserved ghost cells being introduced to the edges of the flow.

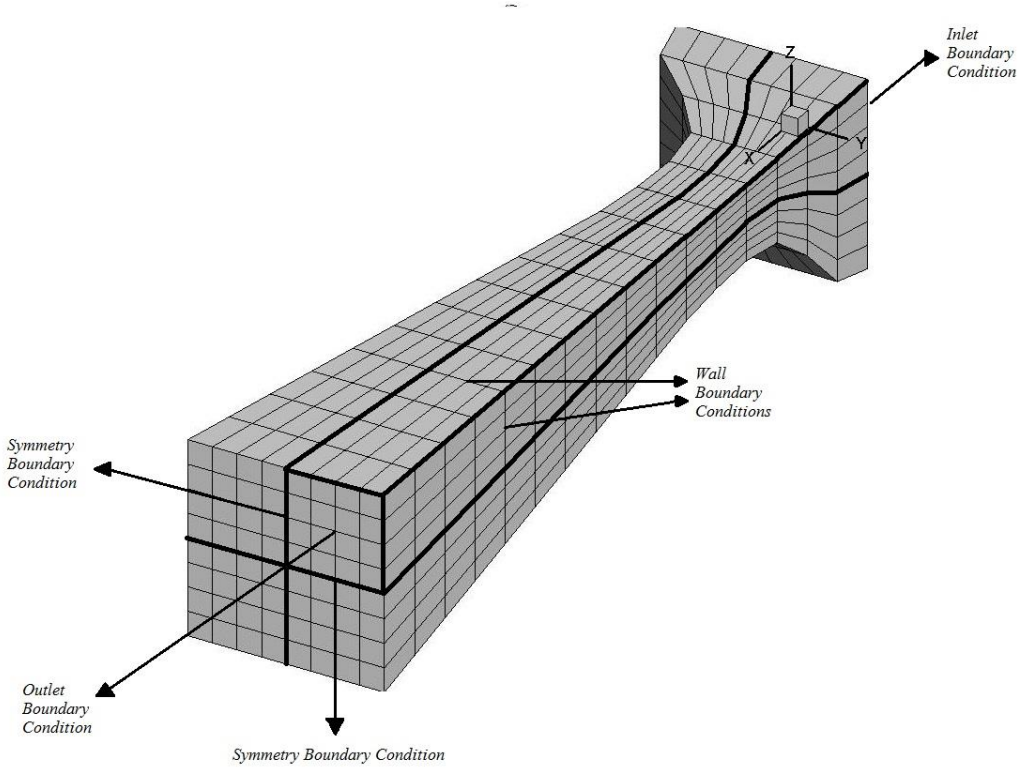

Figure 2.2 All Boundaries

To start with, the symmetry BCs should be discussed. As the solution is the same at every quarter of the geometry according to symmetry planes, the variables to each side of symmetry BCs are taken as equal. Only the normal velocities w.r.t. symmetry planes are reversed according to the dictate of the symmetry planes. This case is valid as long as the entering flow has no angle of attack, which makes it applicable for this thesis.

Next, wall BCs should be elaborated. At the walls, the total energy, density and velocity components are taken from the center of the neighboring interior cells while the normal components of the velocity are equated to zero as there is no mass leak out. For the rest:

$$(\rho u)_{boundary} = (\rho u)_{interior} - \rho_{interior} U_n n_x$$
$$(\rho v)_{boundary} = (\rho v)_{interior} - \rho_{interior} U_n n_y$$
$$(\rho w)_{boundary} = (\rho w)_{interior} - \rho_{interior} U_n n_z$$

n is the unit normal at wall surface.

Next are the inlet BCs. As the inlet flow is subsonic flow, pressure is taken from the inside while total pressure, total temperature, density and Mach number are gotten from the outside. The flow is only in u direction. Then,

$$P_{boundary} = (\gamma - 1)\rho_{inside}(e_t - \frac{u^2 + v^2 + w^2}{2})_{inside}$$
$$\rho_{boundary} = \frac{1}{2}(\gamma P_{boundary} + \sqrt{(\gamma P_{boundary})^2 + 2(\gamma - 1)(\rho u)^2}$$
$$(\rho e_t)_{boundary} = \frac{P_{boundary}}{\gamma - 1} + \frac{(\rho u)^2}{2\rho}$$

Lastly, for the outlet BCs, as the flow is supersonic at this point, all parameters will be taken from the preceding cells.

CHAPTER 3

SOLUTION METHODS

As has been discussed in the introduction in chapter 1, there are various methods to perform flow solutions depending on the desired convergence rate, accuracy, flow type, the computational power at hand and there are even more ways to modify these methods, replacing a part with another or including whatever is deemed necessary for specific needs. For the purpose of this thesis, two solvers have been used to solve the same problem. The Newton's method and Newton-GMRES method are the two solvers, the usage of Newton-GMRES method and its appropriate preconditioning being the focus. The calculation of the Jacobian for each step of the Newton's method is done analytically while the calculation of the first Jacobian for the preconditioning matrix needed for the Newton-GMRES is done analytically and compared to numerical calculation. The numerical and analytical calculations are found to be equivalent which is why the analytical solution is selected due to its far superior speed. The problem is as stated in chapter 2. Detailed explanation of Newton and Newton-GMRES methods follow.

3.1 Newton's method

The spatial discretization of the problem given above give rise to a nonlinear system of equations. Newton's method is the first method we use to solve this system of equations and the method itself can be described as follows:

Starting with an initial guess to a root of a function, a tangent line to the function can be created to achieve an approximation of the function. Then, this tangent line is lengthened to find where it intercepts the x-axis and the process is repeated with the x-intercept that has been obtained. The obtained x-intercept is almost always a better guess for the root of that function and thus, as the process is repeated, better approximations can be obtained. Then, the following can be shown [19]:

Supposing f : [a,b] → **R** is a differentiable function defined on the interval [a,b] with real values. If an initial approximation to a root, say, x_n is obtained, the next approximation to the root can be done through

$$y = f'(x_n)(x - x_n) + f(x_n)$$

(18)

Where f' is the derivative of the function f. Then, the mentioned x-intercept of this line would be the next value to be used as initial guess, that is, x_{n+1}. Setting y = 0 and x = x_{n+1} to obtain a guess at the root results in

$$0 = f'(x_n)(x_{n+1} - x_n) + f(x_n)$$

(19)

Solving this to obtain x_{n+1} results in

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

(20)

Which forms the basis for Newton method. Now, if we were to write these in familiar terms, the main loop that brings forth the solution would be:

$$\hat{Q}_{k+1} = \hat{Q}_k + \Delta\hat{Q}_k$$

(21)

For which, one would need to find $\Delta\hat{Q}_k$ to be able to proceed. Thus, the residual mechanism is introduced. The residual is a vector that will be equal to zero when the exact solution is found, but until then will be as follows:

$$\hat{R}(\hat{Q}) = \frac{\partial\hat{\mathrm{F}}(\hat{Q})}{\delta\xi} + \frac{\partial\hat{\mathrm{G}}(\hat{Q})}{\delta\eta} + \frac{\partial\hat{\mathrm{H}}(\hat{Q})}{\delta\zeta}$$

(22)

Then, this system of nonlinear equations can be transformed into linear equations through a Taylor series expansion keeping only the lowest order term:

$$\hat{R}(\hat{Q}_{k+1}) = \hat{R}(\hat{Q}_k) + \left(\frac{\delta\hat{R}}{\delta\hat{Q}}\right)\Delta\hat{Q}_k$$

(23)

Then, if one assumes iteration will result in zero residual, the residual becomes:

$$\left(\frac{\delta \hat{R}}{\delta \hat{Q}}\right)\Delta \hat{Q}_k = -\hat{R}(\hat{Q}_k)$$

(24)

Hence, the famous linear Ax=b problem that we need to solve is obtained, where A is a matrix and x and b are vectors. Here, $\left(\frac{\delta \hat{R}}{\delta \hat{Q}}\right)$ is the Jacobian matrix, which can be calculated analytically or numerically. The numerical calculation takes far longer due to the size of the Jacobian matrix and thus the analytical way is applied. In Newton's method without any fast linear system solvers, UMFPACK is used to solve the Ax=b equality.

3.2 Newton-GMRES Method

When using Newton's method to solve the systems of nonlinear equations that define our problem, there arises the necessity to solve linear systems where the coefficient is the Jacobian matrix at every step. This creates the problem of computational inefficiency due to repetition of an expensive operation to obtain a large amount of accuracy. The high accuracy is desirable, yet a better method that can achieve an accuracy close to that of Newton's with less computational load is sought for. That is where Krylov subspace methods are needed. The need to avoid computations involving expensive matrix operations leads to the alternative of doing matrix-vector operations and working with the resulting vectors. Those vectors create subspaces called Krylov subspaces. In algebraically understandable terms, if one has a vector b and a matrix A, the first vector in the Krylov subspace would be the multiplication of A^0 with b, the second vector A^1b, the third A^2b and so on. Here, the powers of A represent the times A has been multiplied by itself, that is, A^2=AA. Then, the Krylov subspace of order m that is created using an n-by-n matrix A and a vector b of dimension n would be as follows:

$$K_m(\mathrm{A}, b) = span\{b, \mathrm{A}b, \mathrm{A}^2b, \dots, \mathrm{A}^{r-1}b\}$$

(25)

This subspace can then be used to perform operations that involve large matrices without explicitly forming the matrices themselves. The obvious benefit of this is apparent; one could efficiently take the large numbers of linear equations created by

the Newton's method and solve them at each step of the Newton iterations using Krylov subspaces without ever having to do a matrix-matrix operation. This allows us to take a shortcut through Newton's method and use a Newton-Krylov method instead. The most widely used Krylov subspace methods include Conjugate Gradient (CG), BiConjugate Gradient Stabilized (BiCGSTAB) and Generalized Minimum Residual (GMRES) methods. The method we use in this thesis is the GMRES method coupled with Newton's method. The reason for the choice of GMRES over a method such as BiCGSTAB is due to the latter's increased possibility of termination without convergence. CG, on the other hand, necessitates a symmetric matrix which our matrix is not.

Newton-GMRES method is used to solve the nonlinear systems that are the implicit discretizations of a system of equations. This is done through the linearization of the system at each step of the outer loop which is the Newton's method loop and the iterative solution of the linear system in the inner loop using GMRES. The next step of the outer loop starts with the solution of the previous step. The GMRES algorithm is the composition of four processes: initialization, Krylov subspace orthogonalization, solving a least squares problem and updating the solution. The initialization and update steps are done once each at the beginning and the end of the algorithm while the orthogonalization and least squares solution steps are repeated for every iteration as the Krylov subspace grows.

3.2.1 GMRES Algorithm

The basic GMRES algorithm for the solution of Ax = b is as follows:

Algorithm 1 GMRES

1. Initialization: x_0 is chosen, $r_0 = b - \mathrm{A}x_0$ and $v_1 = r_0/\beta$ where $\beta = \|r_0\|$ is computed.
2. Iteration: For m = 1,2,…
   The search direction vector v_m, if applicable, is preconditioned and the next Krylov subspace search direction v_{m+1} is created by the Arnoldi

orthogonalization. Then the next column of the Hessenberg matrix, H_m is formed:

$$w_m = A v_m$$
$$h_{i,m} = w_m^T v_i, \forall\, i = 1,2,\dots,m$$
$$\hat{v}_{m+1} = w_m - \sum_{i=1}^{m} h_{i,m} v_i$$
$$h_{m+1,m} = \|\hat{v}_{m+1}\|_2$$
$$v_{m+1} = \hat{v}_{m+1}/h_{m+1,m}$$

3. The least squares problem is solved:

$$y_m = \underset{y}{\operatorname{argmin}} \left\| \beta e_1 - \hat{H} y \right\|_2$$

where the minimum value is y_m and e_1 is the first column of the identity matrix of dimension m+1, $(1,0,0,0,\dots,0)^T$. Here, the function that is being minimized is actually the residual, $\|r_m\|_2$. To solve the minimization problem, H_m can be converted into an upper triangular form, making the solution relatively inexpensive and this is done here with QR factorization.

If $\frac{y_m}{\|r_0\|_2} \leq \eta_k$, exit the loop, where η_k is an arbitrary definition of relative tolerance.

4. Update the solution: $x_m = x_0 + V_m y_m$

3.2.2 Convergence of GMRES

The first concern in the application of GMRES is its convergence. The method, as we have discussed, creates a Krylov subspace that grows with every step in size, each iteration containing the subspace before itself. These subspaces are each used to minimize their related residuals, which monotonically decrease as for every new iteration of the algorithm the previous subspace is utilized. In general terms, for a matrix of size *m*, the method would arrive at an exact solution with zero residuals at m iterations, but that would be an extreme use of computational power and excessive inefficiency. Instead, an arbitrary term for relative tolerance to satisfy, η_k is defined as shown above (Alg. 1), which leads to the GMRES algorithm to converge to a

satisfactory residual (around 10^{-17}) in a manageable number of steps (downwards of 1000 steps for 3-D Euler equations for a medium sized grid). This arbitrary term can be chosen between the values of 0 and 1. The value itself translates into a relation between the number of GMRES iterations and Newton iterations necessary to satisfy convergence boundaries. These can be further supported by Trefethen and Bau's work stating [20]:

$$\|r_n\| \leq \left(1 - \frac{\lambda_{min}^2(0.5(\mathrm{A}^{\mathrm{T}} + \mathrm{A}))}{\lambda_{max}(\mathrm{A}^T\mathrm{A})}\right)^{n/2} \|r_0\| \tag{26}$$

For any positive definite matrix A with smallest eigenvalue λ_{min} and largest eigenvalue λ_{max} which pretty much means that for any problem that can at all be solved by a Newton-Krylov method, the solution will be obtained in a smaller number of iterations than solving its full-scale matrix counterpart.

With the resolution of the convergence concerns for GMRES, its main desirability is described next in the form of Jacobian-free calculations.

3.2.3 Matrix-free calculations

Any iterative solution containing the calculation of a Jacobian for a nontrivial system such as 3D Euler equations will face the problem of creation and storage of that Jacobian matrix. The creation of such a Jacobian will be expensive in computational power terms while its storage will cause inefficiency in the solver even when done in sparse matrix form. Fortunately, for unpreconditioned GMRES, no Jacobian formation or storage is necessitated. Instead, first order finite difference approximation can be used to circumvent the process of creating a matrix A to find the product of A with a vector *v* which is necessary as can be seen in Algorithm 1, step 2. The approximation is as follows with *R* being the discretized residual vector and *Q* the flow variables:

$$\mathrm{A}v = \left(\frac{\partial R}{\partial \hat{Q}} + \frac{1}{\Delta t} I\right) v \approx \frac{R\left(\hat{Q} + \epsilon v\right) - R\left(\hat{Q}\right)}{\epsilon} + \frac{1}{\Delta t} v \tag{27}$$

Where ϵ is a value which can be derived through

$$\epsilon = \frac{\epsilon_m}{\|v\|_2^2}$$

(28)

And used to perturb the flow variables with ϵ_m as machine zero. Yet, the definition of ϵ needn't be derived through (28) and can be user defined as long as the round off and truncation errors in finite difference approximation can be kept to a minimum. In our calculations, ϵ is taken to be 10^{-7} with satisfying results as the result of a trial and error process. This leads to the unpreconditioned GMRES being truly matrix free, eliminating a high work load.

3.2.4 Components of GMRES

While showing the algorithm, describing convergence and explaining how GMRES is useful are important, the most important part of this section of chapter 3 is to relay the information on how the components of GMRES work individually and together. As we have explained before, the GMRES is the sum of four steps: initialization, Krylov subspace orthogonalization, solving a least squares problem and update. These four steps are explained below.

Initialization

To solve any Ax = b linear equation system using definite or indefinite Newton methods, one needs a good starting point of x_0. At the initialization step, this x_0 is chosen after which the first residual $r_0 = b - Ax_0$ and $v_1 = r_0/\beta$ is computed with $\beta = \|r_0\|_2$. In terms of a flow solution with R as the discretized residual vector and Q as the flow variables, $\Delta\hat{Q}_k^m$ at m=0 would be our x_0 with m signifying the GMRES iteration step and k signifying Newton iteration step and the first residual would turn into:

$$r_k^0 = \frac{\partial \hat{R}}{\partial \hat{Q}_k}\Delta\hat{Q}_k^0 - \hat{R}(\hat{Q}_k)$$

(29)

The next step would be carrying out the Arnoldi iteration.

Arnoldi iteration

To find the largest eigenvalue of an m-by-m matrix A, one can use the power iteration. Taking a random vector v as the beginning point, one computes Av, A^2v, A^3v, and so

on while recording the resultant vector on v at each iteration. The iteration at some point converges to the eigenvector representing the largest eigenvalue of the matrix A. If one was to instead keep the Av^m products at each iteration instead, a Krylov subspace would be generated. The one minor problem with that subspace would be that the columns of this subspace (kept in matrix form) would not be orthogonal. The Krylov subspace being created without proper orthogonalization would lead to linear dependency between the subsequent subspaces that each contain the previous subspace. A method such as Gram-Schmidt Orthogonalization could be used to create an orthogonal basis that we need but that would have the problem of instability. To alleviate this problem, the Arnoldi iteration is done instead at every iteration m of GMRES and an orthonormal basis for the Krylov subspace is created. This Krylov subspace will then be used to search for solutions that minimize the residuals. These solutions are generally found within an acceptable number of iterations without needing to create a subspace as big as the matrix that we need to solve for. If we were to have an m-by-n matrix A_n and a nonzero vector x that is m-long, the Arnoldi iteration creates a Krylov subspace that is characterized by:

$$\text{Col span (V)} = \text{span } (x, Ax, A^2x, ..., A^{m-1}x)$$

The Arnoldi iteration amounts to the following:

For each m of the GMRES iteration until the expected residual value is reached, the vector v_m called the search direction is taken (and preconditioning is applied if available), the corresponding column of the upper Hessenberg matrix H_m is found and the next search direction v_{m+1} is created by the Arnoldi orthogonalization. Here, it becomes necessary to introduce the reader to Hessenberg matrices to continue. If V_n is the m-by-n matrix with the columns being the orthonormal basis $\{v_1, v_2, \ldots, v_n\}$, then $H_n \equiv V_n^T A V_n$ with the entries $h_{i,j}$ that are created by the Arnoldi algorithm is the upper n-by-n Hessenberg matrix. This Hessenberg matrix can be seen to be the representation in the basis formed by the Arnoldi vectors of the orthogonal projection of A onto the Krylov subspace K. A sample upper Hessenberg matrix would be:

$$\mathrm{H} = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \dots & h_{1,n} \\ h_{2,1} & h_{2,2} & h_{2,3} & \dots & h_{2,n} \\ 0 & h_{3,2} & h_{3,3} & \dots & h_{3,n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & h_{n,n-1} & h_{n,n} \end{bmatrix}$$

After next search direction is created, the next column of the Hessenberg matrix is formed.

$$w_m = \mathrm{A}v_m$$
$$h_{i,m} = w_m^T v_i \,, \forall\, i = 1,2, \dots, m$$
$$\hat{v}_{m+1} = w_m - \sum_{i=1}^{m} h_{i,m} v_i$$
$$h_{m+1,m} = \|\hat{v}_{m+1}\|_2$$
$$v_{m+1} = \hat{v}_{m+1} / h_{m+1,m}$$

Then the least squares problem is faced.

Least Square Problem

The $\mathrm{v_m}$ vector and the $\overline{\mathrm{H}}_\mathrm{m}$ matrix, which is $\mathrm{H_m}$ with one extra row that has $h_{m+1,m}$ as its only nonzero element satisfy the following equation:

$$\mathrm{AV_m} = \mathrm{V_{m+1}}\overline{\mathrm{H}}_\mathrm{m} \tag{30}$$

Then, the least square problem that is

$$\min_p \|r_0 - \mathrm{A}p\|$$

Needs to be solved. If $p = \mathrm{V_m} y_m$ is set, then the problem to be minimized can be turned into

$$\min_{V_m y_m} \|\beta v_1 - \mathrm{AV_m} y_m\| \tag{31}$$

With $\beta = \|r_0\|$. Then, using (30), one can turn (31) into

$$\min_{V_m y_m} \|\mathrm{V_{m+1}}(\beta e_1 - \overline{\mathrm{H}}_\mathrm{m} y_m)\| \tag{32}$$

Where e_1 is $\{1,0,0,\ldots,0\}^T$ with the dimension of m+1. As V_{m+1} is known to be orthonormal, the problem becomes:

$$\min_{y_m} \|(\beta e_1 - \bar{H}_m y_m)\| \tag{33}$$

Here, QR factorization obtained with a Givens Rotation is used to obtain the needed y_m as QR factorization is both computationally light and easy to implement with a Hessenberg matrix as our upper Hessenberg matrix is close to an upper triangular matrix. To minimize y_m in $\|(\beta e_1 - \bar{H}_m y_m)\|$ through a QR factorization, one needs to write $\bar{H}_m$ as $\bar{H}_m = QR$ where Q is an orthonormal matrix and R is a triangular matrix. Through this factorization, y_m is obtained and used to update the current GMRES iteration for the next step.

Update

If the ρ_m value obtained as the result of Eqn. 33 in the previous step is lesser than the determined threshold, that is, $\frac{\rho_m}{\|r_0\|_2} \leq \eta_k$, the inner loop is exited to update the solution using it:

$$x_m = x_0 + V_m y_m \tag{34}$$

Or in terms of a discretized flow solution:

$$\Delta\hat{Q}_k^m = \Delta\hat{Q}_k^0 + V_m y_m \tag{35}$$

The inner loop that is the summation of all these parts is done until the residual vector obtained can satisfy stopping criteria, which can be a limited number of iterations if swift convergence can't be obtained or a predetermined threshold such as

$$\|r_k^m\| \leq \eta_k \left\|\hat{R}(\hat{Q}_k)\right\| \tag{36}$$

With the GMRES algorithm fully detailed, preconditioning process can now be explained.

CHAPTER 4

PRECONDITIONING

Preconditioning is the process of transforming a poorly formed problem into a form that can more easily be solved using numerical methods. Generally in doing this, one takes a system of equations that can as in our case be stored in matrix form such as a Jacobian and applies a "preconditioning matrix" M to those equations. This application takes the form of matrix operations. While using a method that boasts of its matrix-free property, it is very important that these matrix operations be easy to do, store and the matrices easy enough to create. The preconditioning is done to speed up calculations and decrease computational loads on hardware and yet if it is not done properly, the reverse of that can happen. That is why the choice and implementation of a preconditioning method is of utmost importance when using an iterative method. In the following sections, how preconditioners work, what they cost, what their types are, what we have chosen to implement and why will be explained in detail.

4.1. Reasons for Preconditioning

It is known that iterative methods have convergence rates that depend on the problem they are trying to solve, such as the properties of a coefficient matrix like a Jacobian. This means that if one could transform those matrices into matrices that have better properties with the same solutions, the convergence rates could be improved. The matrices that we call the "preconditioners" are effectively how those transformations are introduced into our iterative solvers. For example, if one was trying to solve the generic linear system Ax = b where A is the matrix with unfavorable properties, the following transformation could be done:

$$M^{-1}\mathrm{A}x = M^{-1}b \tag{37}$$

With which the solution of A$x = b$ would remain unchanged while computations done using M^{-1}A could be more efficient. Many iterative solvers, especially those that use inexact Newton methods as their basis need preconditioning to be truly effective given their inclination to converge slower if a poor starting point is selected. When creating a preconditioner, there are two methods of going about this business: One is finding a matrix M that would have more favorable properties than A while successfully imitating A in terms of solution obtained and finding its inverse or the mathematical equivalent of its inverse after. The other is to find a matrix M that approximates A^{-1} while still having more favorable properties than A and not having to calculate an inverse for M. Most preconditioners do the first as the process of inverting the preconditioning matrix can be mostly circumvented at least in classical terms via the usage of intelligent manipulation.

4.2 The Cost of Using Preconditioners

As the reasoning behind the usage of preconditioners is first and foremost to lower computational times, the amount of computational power lost in using those preconditioners in the first place is a primary consideration. The two main aspects of "the cost" of these preconditioners come in terms of setting up a preconditioner and applying it at every iteration. If the convergence time gained in the usage of such a process is larger than the time lost for its application, *ceteris paribus* the preconditioning is considered successful. While some preconditioners such as an SSOR preconditioner may not need a setup phase at all, most preconditioners such as incomplete factorizations involve a large amount of work to be used. The return for the time and power used for this process can be harvested through its usage over iterations or the preconditioner itself can be used in various different linear systems after being formed.

4.3 Types of Preconditioning

There are multiple ways to precondition a problem and to classify the process of preconditioning, two main avenues must be specified: Whether it is left or right preconditioning and the method of preconditioning used, such as a Jacobi preconditioner or an ILU factorization.

4.3.1. Left and Right Preconditioning

The transformation of the linear system Ax = b into another linear system $\mathrm{M}^{-1}\mathrm{A}x = \mathrm{M}^{-1}b$ is not the only transformation that is available for use. In fact, if that were the case, many iterative solvers that use the benefits of preconditioning would become obsolete due to the forced removal of properties that are inherent in the matrices they use. For example, the symmetricity of a matrix could be lost with any kind of preconditioning application, which would lead to the usefulness of a specific method being eliminated. Instead, one could modify Ax = b by splitting the preconditioning matrix into two parts; a right and a left preconditioner. The transformation of the preconditioning matrix M into $\mathrm{M}_1\mathrm{M}_2$ would transform the above equation system as follows:

$$\mathrm{M}_1^{-1}\mathrm{A}\mathrm{M}_2^{-1}(\mathrm{M}_2 x) = \mathrm{M}_1^{-1}b \tag{38}$$

Then, with M_1 named the left preconditioner and M_2 named the right preconditioner, one could insert simple steps in their algorithm enabling the use of one or the other as long as the preconditioning matrix M is nonsingular. For a right preconditioned system, the two steps necessary would be the implementation of:

1- The solution of the system using a new vector y,

$$\mathrm{M}_2^{-1}\mathrm{A}y = b$$

2- The rollback step from y to x,

$$x = \mathrm{M}_2^{-1}y$$

And for a left preconditioned system, one would be solving

$$\mathrm{M}_1^{-1}(\mathrm{A}x - b) = 0$$

While the right preconditioning looks a little harder to implement, it has the added bonus of never modifying the residual of the linear system $r = b - \mathrm{A}x$, which means the stopping criteria for the preconditioned solver do not need to be redefined or based on relative convergence. In this thesis, right preconditioning is applied for that reason. Here, the algorithm for right-preconditioned GMRES is given.

Algorithm 2 GMRES

1. Initialization: x_0 is chosen, $r_0 = b - \mathrm{A}x_0$ and $v_1 = r_0/\beta$ where $\beta = \|r_0\|$ is computed.
2. Iteration: For m = 1,2,…

   The search direction vector v_m is preconditioned via multiplication with the right preconditioning matrix and the next Krylov subspace search direction v_{m+1} is created by the Arnoldi orthogonalization. Then the next column of the Hessenberg matrix, H_m is formed:

$$z_m = M^{-1}v_m$$
$$w_m = Az_m$$
$$h_{i,m} = w_m^T v_i, \forall\, i = 1,2,\dots,m$$
$$\hat{v}_{m+1} = w_m - \sum_{i=1}^{m} h_{i,m}v_i$$
$$h_{m+1,m} = \|\hat{v}_{m+1}\|_2$$
$$v_{m+1} = \hat{v}_{m+1}/h_{m+1,m}$$

3. The least squares problem is solved:

$$y_m = \underset{y}{\operatorname{argmin}} \left\|\beta e_1 - \hat{H}y\right\|_2$$

   where the minimum value is y_m and e_1 is the first column of the identity matrix of dimension m+1, $(1,0,0,0,\dots,0)^T$.

   If $\frac{y_m}{\|r_0\|_2} \leq \eta_k$, exit the loop, where η_k is an arbitrary definition of relative tolerance.

4. Update the solution: $x_m = x_0 + M^{-1}V_m y_m$

The two steps necessary for the right preconditioning can be seen above. With the question of preconditioning side answered, the preconditioning methods can be examined.

4.3.2. Preconditioning Methods

By Saad's words, "a preconditioner is any form of implicit or explicit modification of an original linear system which makes it easier to solve by a given iterative method"

[21]. As such, one could use any variety of commonly used preconditioning methods as well as any other method that they could derive from their knowledge of the physical problem at hand and call it their own. In this section, information regarding the common methods for preconditioning of large sparse matrices will be conveyed.

4.3.2.1. Jacobi Method

Jacobi preconditioner, also known as the diagonal preconditioner is the simplest preconditioner that can be created. The preconditioning matrix is selected to be the main diagonal of the linear system to be preconditioned. It is useful when dealing with matrices that are diagonally dominant. Also, for matrices that are more complex, yet are still mainly dominant near the diagonal such as with problems with more than one variable per node, a block version of this method can be applied. As is obvious, the Jacobi preconditioners are applicable to none but the simplest linear systems.

4.3.2.2 Symmetric Successive Over-Relaxation Method

The symmetric successive over-relaxation (SSOR) method is close to Jacobi preconditioners in terms of calculation necessary to compute a preconditioning matrix. It also necessitates the presence of a symmetrical matrix to be preconditioned. If one was to decompose matrix A as shown:

$$\mathrm{A} = \mathrm{D} + \mathrm{L} + \mathrm{L}^{\mathrm{T}} \tag{39}$$

With its diagonal, lower and upper triangular submatrices, then the SSOR matrix definition would be:

$$\mathrm{M} = (\mathrm{D} + \mathrm{L})\mathrm{D}^{-1}(\mathrm{D} + \mathrm{L})^{\mathrm{T}} \tag{40}$$

Which can be parameterized using ω:

$$\mathrm{M}(\omega) = \frac{1}{2-\omega}\left(\frac{1}{\omega}\mathrm{D} + \mathrm{L}\right)\left(\frac{1}{\omega}\mathrm{D}^{-1}\right)\left(\frac{1}{\omega}\mathrm{D} + \mathrm{L}\right)^{T} \tag{41}$$

The optimal value of this parameter ω can be computed using spectral information from the original matrix but the process of this computation makes the method

unreasonably expensive. The SSOR preconditioning matrix is factorized as can be observed above, which makes it similar to the other factorization based methods in those terms. This takes us to the discussion of ILU factorization preconditioners.

4.3.2.3. Incomplete Lower-Upper Factorization Preconditioners

Using a large sparse matrix A, the incomplete lower-upper factorization of that matrix would yield two sparse matrices, a lower triangular sparse matrix L and an upper triangular sparse matrix U. The difference of the factorized matrices and the original matrix A, R=LU-A has to satisfy specific constraints, such as mimicking the zero pattern of the original matrix A, that is, having zeroes at the same locations as A. There are many types of these factorizations, in fact without a specification one could construct an infinite number of factorizations for a matrix. Thus, there are many types of ILU preconditioners used in ILU preconditioning. This section will discuss these in detail.

ILU factorizations, in contrast to the previous preconditioners we have discussed, necessitate computations that are complex and as such, they are prone to breakdowns due to zero pivots (zero entries in the main diagonal) and can result in indefinite matrices due to negative pivots. In these cases, the substitution of a user-selected positive number can be an effective solution. As we have discussed before, preconditioning is almost always a costly process and with the ILU factorization computations, the cost can be equal to an iteration of the solver. This cost may be depreciated through the usage of the preconditioner for more than one linear system, such as in successive time steps or if the solver takes several iterations to converge and the number of those iterations can be lowered.

Incomplete factorizations can be done in many different forms. The original matrix A can be factorized as A = LU in which case the problem can be solved normally (Figure 4.1). The factorization can also be done as $A = (D + L)D^{-1}(D + U)$ with D diagonal and L and U triangular matrices. This case can be solved with a slightly altered approach (Figure 4.2) [22].

Let A = LU where Ax = b

for i=1,2,…

$z_i = l_{ii}^{-1}(b_i - \sum_{j<i} l_{ij} z_j)$

for i=n,n-1,…

$x_i = u_{ii}^{-1}(z_i - \sum_{j>i} u_{ij} x_j)$

Figure 4.1: Preconditioner solve for Ax = b and A = LU

Let $A = (\mathrm{D} + \mathrm{L})(\mathrm{I} + \mathrm{D}^{-1}\mathrm{U})$ where Ax = b

for i=1,2,…

$z_i = d_{ii}^{-1}(b_i - \sum_{j<i} l_{ij} z_j)$

for i=n,n-1,…

$x_i = z_i - d_{ii}^{-1} \sum_{j>i} u_{ij} x_j$

Figure 4.2: Preconditioner solve for Ax = b and $\mathrm{A} = (\mathrm{D} + \mathrm{L})(\mathrm{I} + \mathrm{D}^{-1}\mathrm{U})$

As can be seen in figure 4.2, $\mathrm{A} = (\mathrm{D} + \mathrm{L})\mathrm{D}^{-1}(\mathrm{D} + \mathrm{U})$ can be shortened into two specific forms:

1- $(\mathrm{D}+\mathrm{L})\ z = b$, $(\mathrm{I}+\mathrm{D}^{-1}\mathrm{U})\ x = z$ and

2- $(\mathrm{I}+\mathrm{L}\ \mathrm{D}^{-1})\ z = b$, $(\mathrm{D}+\mathrm{U})\ x = z$

In both forms, the diagonal is used two times and in both, only divisions are done using D. This leads to the storage of D^{-1} being the easiest option in memory terms, yet is also the cause for the zero pivot breakdowns.

Having shown how preconditioner solves for factorizations are done, information about how they are calculated should be relayed. The most widespread incomplete factorizations are done through the preservation of a set of physical positions on the original matrix and using this set of positions through the factorization to produce matrices with all zero entries excluding the set. This set of positions is generally chosen

to be the nonzero pattern of the original matrix. Any position that is zero in A but filled with an entry in the factorization is called fill-in. According to the complexity of the factorization, these fill-in's can be discarded or kept. If all fill-in is discarded, the result of the factorization is called ILU(0) factorization. If the fill-in caused by the original matrix is kept, the factorization would be an ILU(1) factorization. As such, if the fill-in caused by level k of the factorization is kept (as in, from ILU(k)), the factorization would be named ILU(k+1). This number, *k+1*, is called the level of fill. The incomplete factorization process with a set *S* of predefined locations, can be described as follows:

$$for\ all\ k,i,j\ >\ k{:}\ a_{i,j} \leftarrow \begin{cases} a_{i,j} - a_{i,k}a_{k,k}^{-1}a_{k,j} & if(i,j) \in S \\ a_{i,j} & otherwise \end{cases}$$

In terms of storage space, ILU(0) takes at maximum the same amount of storage space as the original matrix does. With each increasing level of fill, the storage necessity increases. SPARSKIT by Youssef Saad offers the following algorithm for the usage of ILU(p) for any *p*, as shown by John Gatsis [23], [24]:

Algorithm 3 SPARSKIT ILU(p) factorization algorithm

Define a shifted level of fill-in: $\hat{p} = p + 1$
for i = 1; n do
  for k = 1; i-1 do
    if $lev_{ik} \leq \hat{p}$ then
      $\phi\ a_{ik}/a_{kk}$
      for j = 1,n do
        $lev_{ij}^{*} = lev_{ik} + lev_{kj}$
        if $lev_{ij} = 0$ then
          % Fill is unassigned
          if $lev_{ij}^{*} \leq \hat{p}$ then
            $a_{ij} = -\phi a_{kj}$
            $lev_{ij} = lev_{ij}^{*}$
          end if
        else
          % Existing fill
          $a_{ij} \leftarrow a_{ij} \leftarrow \phi a_{kj}$
          $lev_{ij} \leftarrow \min(lev_{ij}, lev_{ij}^{*})$
        end if
      end for % Index j

```
    end if
  end for % Index k
end for % Index i
```

While higher levels of fill do indeed require a higher amount of computational power and storage space to be expended, they are generally better than their lower level of fill counterparts. An optimization relating the computational power lost due to the level of fill and accuracy gained can be done. There is no "fixed" optimal level of fill for ILU(p) factorization preconditioners, every problem has its own optimal preconditioner.

Modified ILU (MILU) preconditioners add another step to the process their ILU counterparts use. With MILU, at the end of the discard step for every row, the discarded entries are summed and this sum is added to the diagonal entry. This way, the row sums of the LU matrix are equal to those of A. This leads to the MILU preconditioners being especially useful when considering PDEs. When considering other problems, MILU preconditioners are not much different than their ILU cousins.

The ILU(p) factorizations, that is, ILU factorizations with only level of fill considerations, do not consider numerical properties of the entries they are dropping. The entries that are being dropped can be important to the solution of the system and thus another factor when considering dropping entries should be decided upon. One such factor can be creating a threshold for values that are to be dropped, such as the relatively small elements. Also, a strategy for defining the number of elements to be kept in each row can also be introduced. The combination of these two would create the ILUT, ILU with threshold. This strategy can be seen on the next algorithm.

Algorithm 4 ILUT algorithm

for i = 1; n do
$w = a_{i*}$
for k = 1; i-1 and $w_k \neq 0$ do
$w_k = w_k / a_{kk}$
Apply dropping rule to w_k
$if\ w_k \neq 0\ then$
$w = w - w_k u_{k*}$
Endif
Enddo
Apply dropping rule to row w
$l_{i,j} = w_j\ for\ j = 1, \dots, i-1$
$u_{i,j} = w_j\ for\ j = 1, \dots, n$
$w = 0$
Enddo

Here, for a factorization with considerations for a value size drop threshold and a value count drop threshold, ILU(ρ,τ) would use the two guidelines:

1- First, if an element of the matrix, w_k is under a tolerance limit τ that has been obtained through the multiplication of τ and the original norm of its row, the element is dropped.
2- At the next application, the elements that are under the tolerance limit are dropped again. In addition to that, the algorithm keeps the ρ largest elements of the lower matrix and ρ largest elements of the upper matrix of each row in addition to the diagonal. This allows for the control of number of elements in each row according to the user's desires.

This approach is one of the best when the numerical values themselves are more important than the number of values.

If a pivoting strategy such as picking the entry with the largest value in a row to be the diagonal element in that row is applied, the method turns into ILUTP (ILUT with pivoting).

If the linear system is made up of blocks of data instead of randomly filled, block fill ILU (BFILU) methods can be used. Here, this was tried but ultimately not used due to the amount of data that was being discarded consistently lead to divergence.

If the factorizations are set aside and the inverse of the original matrix A is directly approximated, this type of method is called an approximate inverse method (or sparse approximate inverse if used with sparse matrices). While useful, the computational power necessary for this method offsets its efficiency in our use.

4.3.3 Ordering

Preconditioners that use incomplete factorization, such as those that are used in this study, depend on the ordering of the equations that are being solved. Using ordering can improve stability of the preconditioner or enable the utilization of parallelization. Yet, the disadvantage of ordering is that it can result in elements with low numerical quality, affecting the convergence. For the purpose of this thesis, the popular Reverse Cuthill McKee ordering has been investigated and decided not to be used as the historical data regarding ordered preconditioners is not completely decisive. Both positive and negative results have been collected from ordered preconditioners. Still, historical data suggests that for a nonsymmetric problem, RCM can be highly beneficial and thus, this is part of the future work that is to be investigated.

In this work, unordered ILU preconditioners ILU(0), ILU(1), MILU(0), ILU(ρ,τ) have been used with the specifics as given in the results. ILU(p) with $p>1$ was tried and found to constantly diverge so it was discarded and the computational expense of and our inexperience with SPAI lead to us not being able to use it.

CHAPTER 5

RESULTS

This chapter collects all the results obtained through the usage of different methods and compares these results. The Newton's method, unpreconditioned Newton-GMRES method and Newton-GMRES method with block-diagonal Jacobian matrix modified with incomplete factorization as the preconditioning matrix have been applied to the problem at hand. Their separate and average CPU times, iteration counts and relative residual histories have been showcased here. The parameters affecting Newton-GMRES and the preconditioner algorithms are discussed. The meshes used and the dependence of the algorithm on the meshes are analyzed.

The Newton-GMRES algorithm used is developed through manipulation of the one used in [9]. The preconditioning algorithm used is developed through the mixing of Newton-GMRES algorithm and the routines used in SPARSKIT and their manipulation [23]. Both are developed in Fortran77 and run with ifort. The machine specifics used are as follows: AMD Opteron^{TM} Processor 6378 at 2400 MHz with 2048 kB cache and 8 cores, using 1 out of 64 available nodes with a total of 256 GB available RAM.

5.1 Sample Problem Solution and Comparisons

The sample problem to be used in comparing preconditioned and unpreconditioned algorithms is as stated in problem definition in chapter 2, the solution of 3D Euler equations for supersonic flow on a nozzle. This selection is done with ease of access to results, ease of convergence and definitive difference between convergence times in mind. The algorithm could be applied with some effort to other problems, which is one of the intended future projects.

For different η_k values that regulate the relation of GMRES iterations and Newton iterations and different grid sizes, the CPU times, iterations needed for convergence

and relative residual histories are shown. It should be stated that the results from here to the last section are based on the CPU times for one trial done for each grid type and forcing factor done on the same server with no other running processes. In the last section of this chapter, all references to CPU times refer to average CPU times obtained through multiple trials (ranging from 3 to 10 trials according to variability of results). The effect of level of fill and threshold value used in the preconditioner is showcased. All spatial discretizations for Newton-GMRES methods are done in first order as the main focus is on drawing attention to the amount of computational efficiency that can be gained while keeping the same accuracy and as that order increases, the efficiency decreases with very little accuracy gain. At the end of the results section, some samples of second order discretization are given to prove this statement. The relative residual of all methods including Newton's and Newton-GMRES methods are converged down to 10^{-14}. Newton-GMRES method allows far more convergence, yet the Newton's Method becomes prohibitively time-consuming if further convergence is aimed. It must be said here that this residual is normalized, as in, the residuals shown are based on the original calculated residual. From this point on, all references to residuals refer to relative residuals based on the original residuals.

5.1.1 Grid Sizes, Shapes and Dependencies

The grid sizes and shapes are given in Table 5.1.

Table 5.1 Grid sizes

Coarse Grid	17x5x5 = 425
Medium Grid	33x9x9 = 2673
Fine Grid	65x17x17 = 18785

The most important problem that was faced in this study was declaring array sizes for different uses. Creating a high number of arrays and storing them until the end of each GMRES iteration was inefficient, yet storing data in arrays that were not going to be used again would lead to array size mismatch errors in the algorithm. The algorithm would crash at mesh sizes larger than 65x17x9 due to that. This problem was detected and solved by a trial-and-error process to find why this was happening and the grid size is not a problem now, yet it can be seen as a warning indicator that the same

problem could be faced if the algorithm is applied to a much larger grid. For the purpose of this study, however, all intended processes have been applied successfully. The grid shapes used with the sizes as shown in Table 5.1 follow in Figures 5.1 through 5.3.

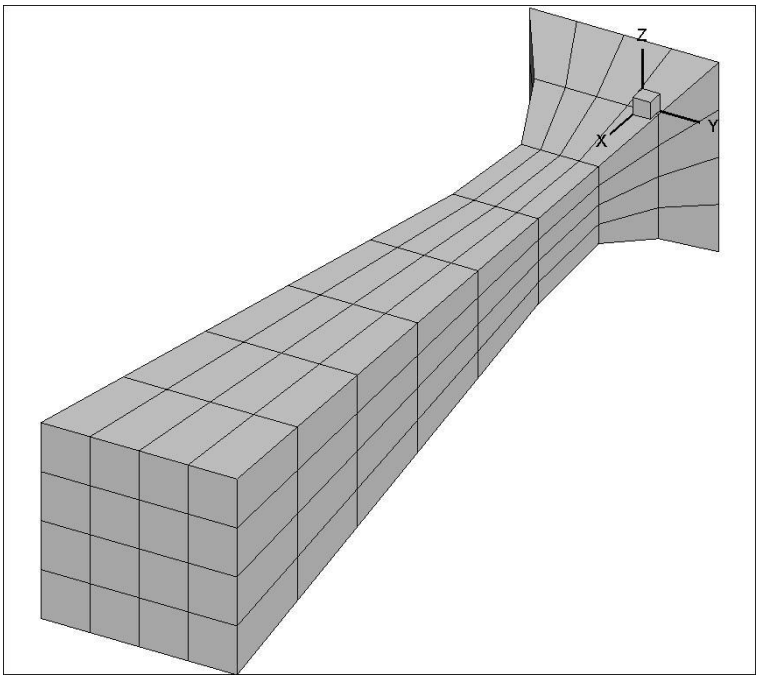

Figure 5.1 The Coarse Grid (17x5x5)

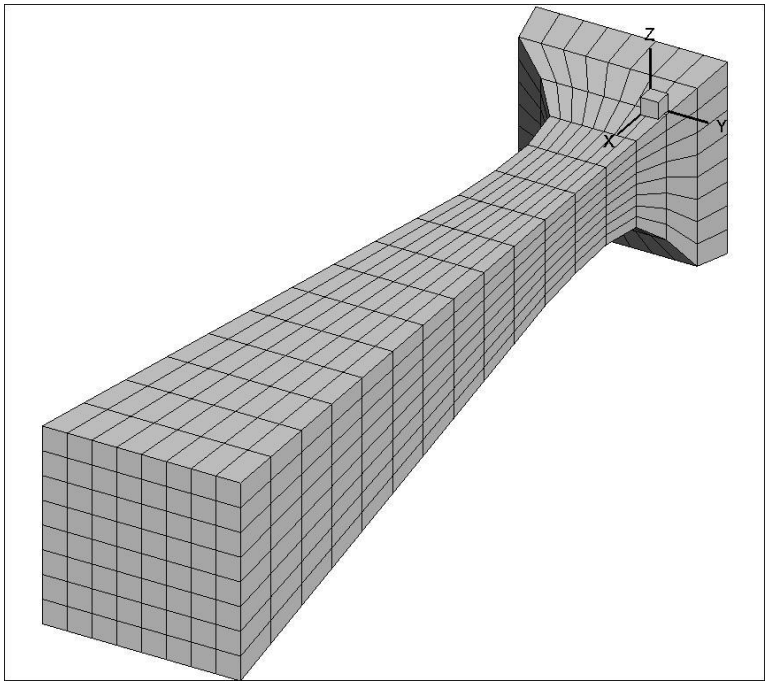

Figure 5.2 The Medium Grid (33x9x9)

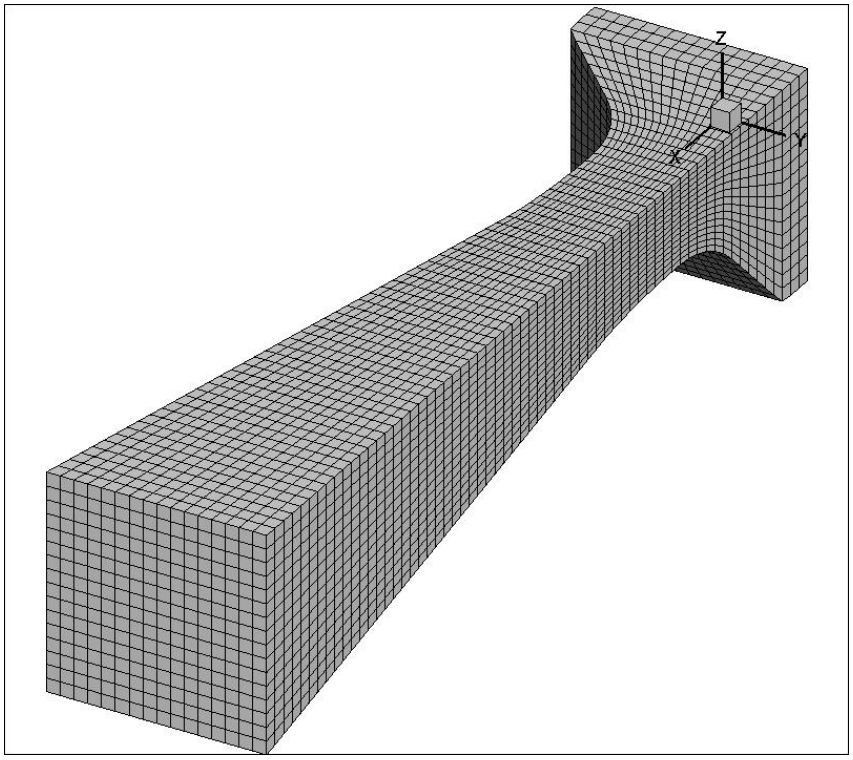

Figure 5.3 The Fine Grid (65x17x17)

Next, the dependency of the results on the meshes should be shown before comparing the results with each other.

The dependency is showcased for first order discretization on preconditioned Newton-GMRES method on Figure 5.4. Mach number versus location in normalized position on grid is plotted for this purpose, which could in fact be replaced by any number of variables determined by the solution. It is selected merely for the ease of inspection it provides.

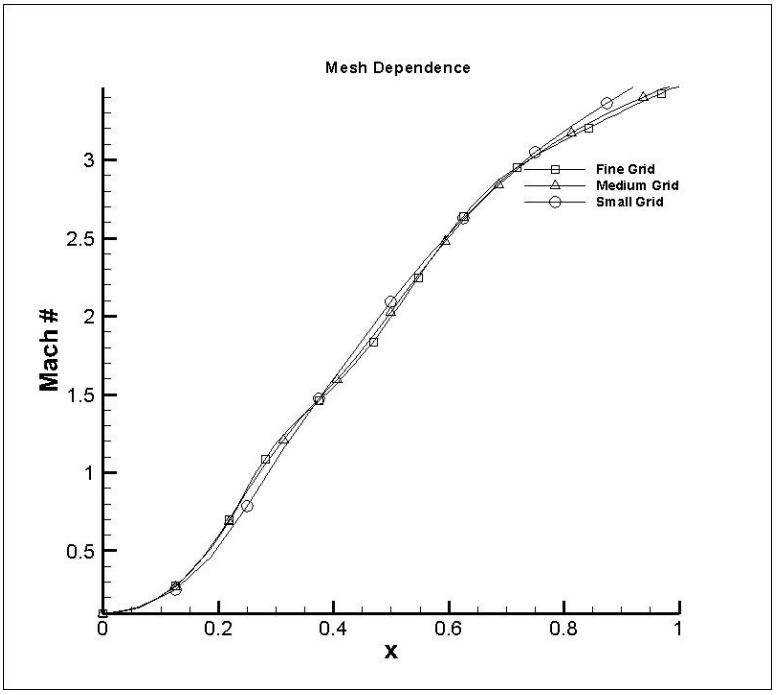

Figure 5.4 Grid Dependency of Preconditioned Newton-GMRES Method

It can be seen in Figure 5.4 that the algorithm is clearly dependent on the mesh used. As the fineness of the mesh increases, the necessary computational power to converge to a result increases but the accuracy also increases. Yet, it can be observed that the difference between fine and medium mesh solutions is lower than that of medium and coarse mesh solutions and that all solutions overlap for a significant amount of the plot. This means that with an even finer mesh, the results will not differ very much from those obtained with the fine mesh.

5.1.2 Comparison of Results

The results for the solution of the problem are as indicated in the following pages. They include CPU time, iteration and residual history comparisons for Newton's method, Newton-GMRES method and Newton-GMRES method with ILU(0), ILU(1), MILU(0) and ILUT(ρ,τ) factorized block-diagonal first order approximated Jacobian preconditioner. Average Wall clocks and CPU times for all methods are included in the end to enable comparison. ILU(k) with higher values of k lead to non-convergence due to memory issues. The results are shown for different values of η_k at 0.4 and 0.6 for one trial. Deviation from these values can achieve higher or lower convergence

times but what is important is that the trend of superiority between preconditioned vs. unpreconditioned solver results stay the same at all values of η_k. The results are also analyzed according to grid sizes. For the ILUT(ρ,τ) preconditioner, the level of fill is selected to be 5 due to the block-diagonal characteristic of the computed Jacobian while the threshold for dismissing values is selected to be 0.0001 so as to keep all significant values. Increasing the threshold further results in the loss of important values and a loss of convergence and thus that value is not changed. The effect of changing the level of fill to a lower value of 4 and through this, removing parts of the Jacobian is showcased. Increasing the level of fill does not make a difference in this problem as the Jacobian does not have more elements. The block-diagonal format of the Jacobian matrix for fine grid can be seen in Figure 5.5 and 5.6.

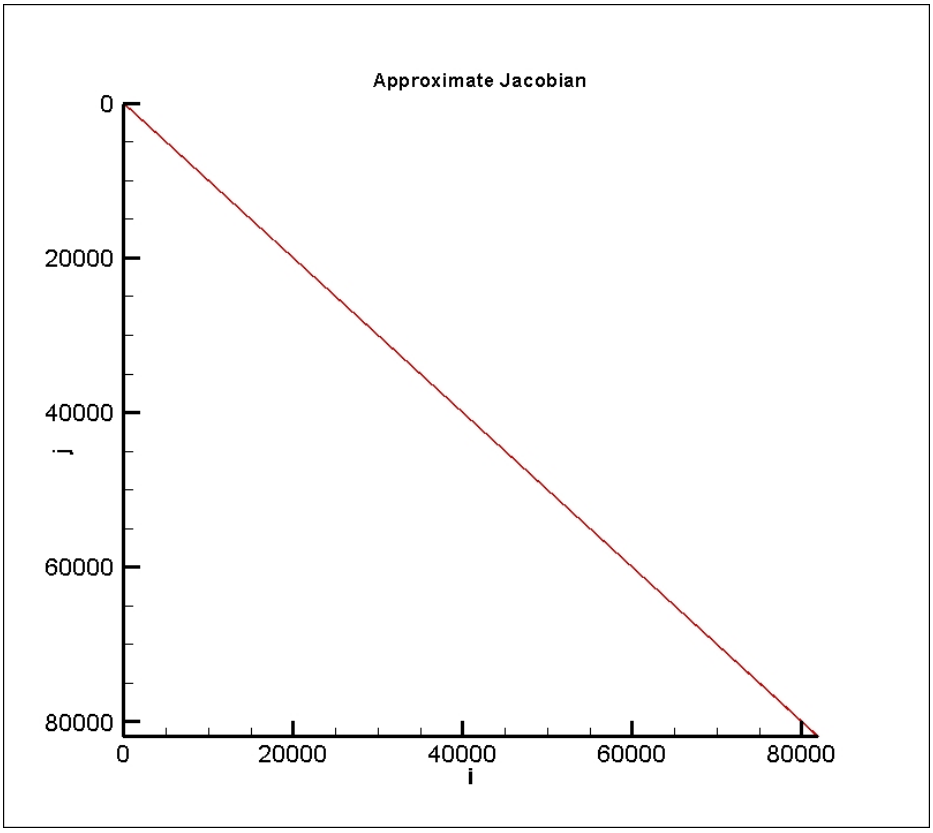

Figure 5.5 Approximate Jacobian (Whole)

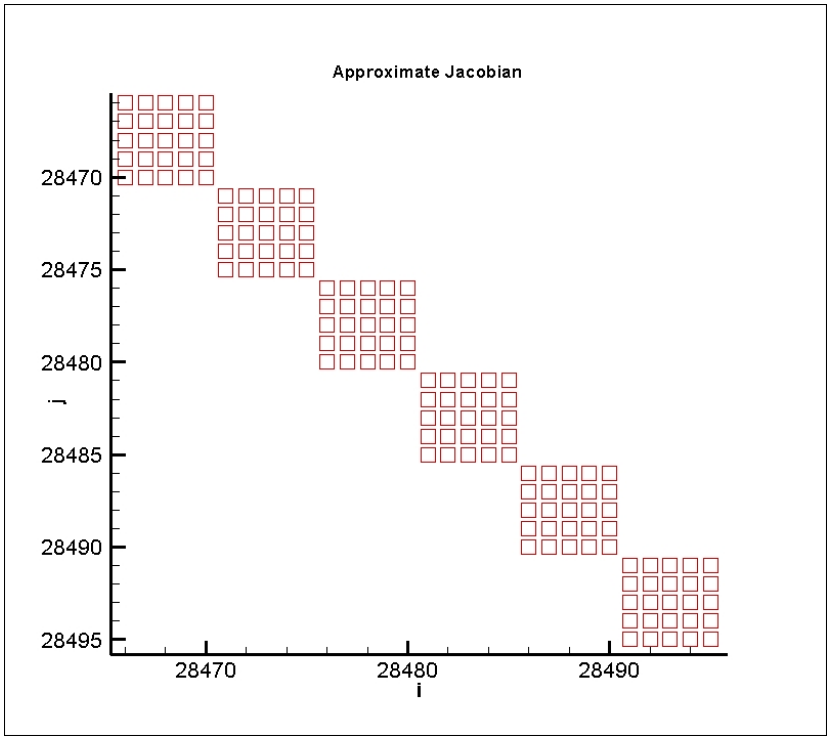

Figure 5.6 Approximate Jacobian (Zoomed)

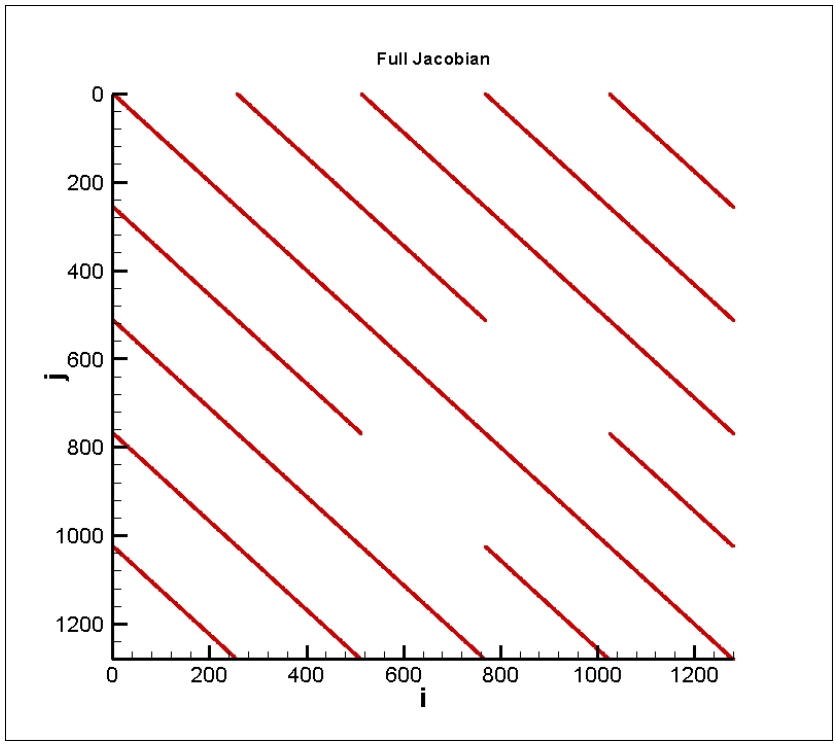

Figure 5.7 Full Jacobian

In Figure 5.7, the full Jacobian for a coarse grid can be seen for comparison with Figure 5.5, where only the block diagonal is kept. The distribution of data in the matrices for coarse, medium and fine meshes have the same pattern. It can easily be seen that a large portion of the full Jacobian has been removed for easy calculation. This of course leads to a preconditioner that is not as good as it could be if the full Jacobian was used as the basis, yet the results show that the current implementation of the method was still satisfactory for the purpose of this study. The following figures, 5.8 and 5.9 show the eigenvalue distributions before and after preconditioning for matrix A using ILU(0) and ILUT methods, respectively.

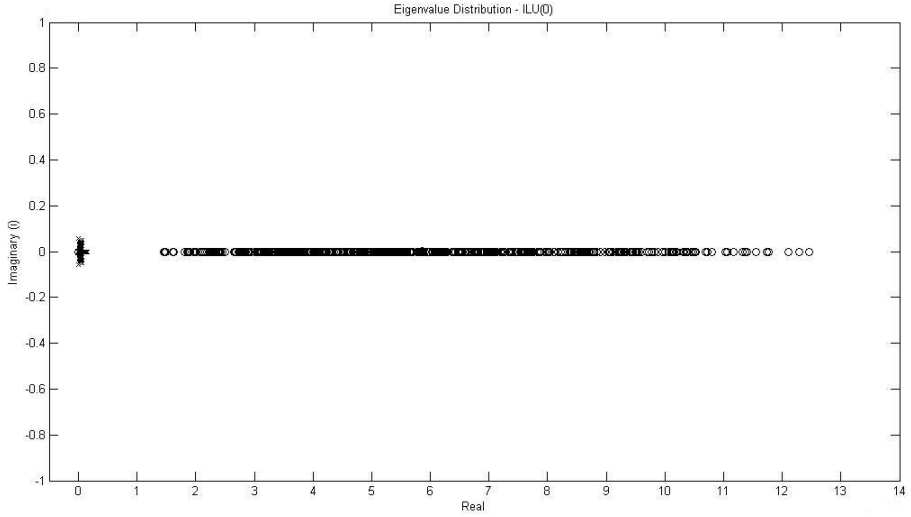

Figure 5.8 Eigenvalue Distribution for ILU(0)

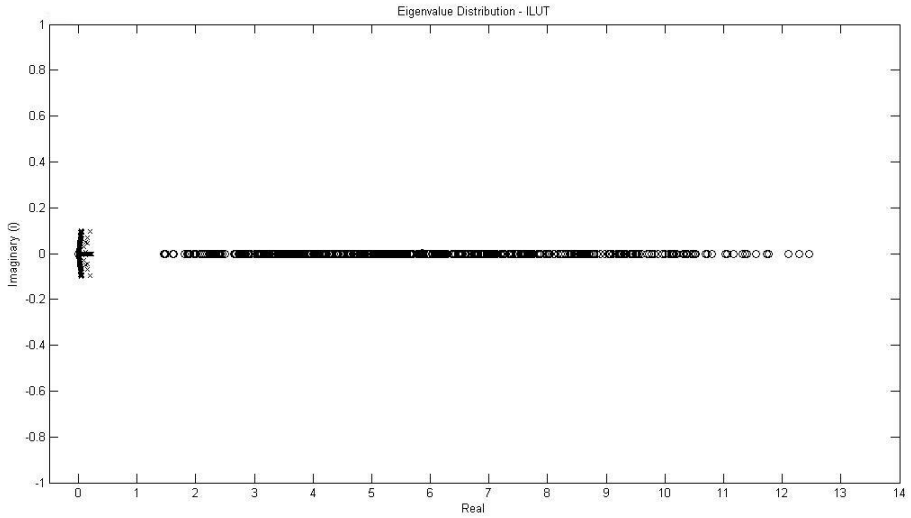

Figure 5.9 Eigenvalue Distribution for ILUT

The eigenvalue distributions in Figures 5.8 and 5.9 represent the clustering of eigenvalues after preconditioning. The spread-out eigenvalues, shown with circles from 1.6 to 13 in both figures represent the matrix before preconditioning is applied. The clustered eigenvalues to the left from 0.1 to 0.4 in both figures represent the matrix after preconditioning takes place. Also, it is difficult to observe from the figures, but the imaginary parts of unpreconditioned eigenvalues are far smaller than their preconditioned counterparts. Ideally, the eigenvalue distribution would be much closer to 1, yet even under the current circumstances the preconditioned clusters are still much closer overall to unity. This leads to a better overall convergence for the linear solver, GMRES. It was also found that the condition number improves by around 10^3 after preconditioning, that is, the problem becomes well-conditioned with a condition number $\kappa \approx 23$ for coarse grid and similar for the other grids.

5.1.2.1 Graphical Comparison for Different Grids at $\eta_k = 0.4$

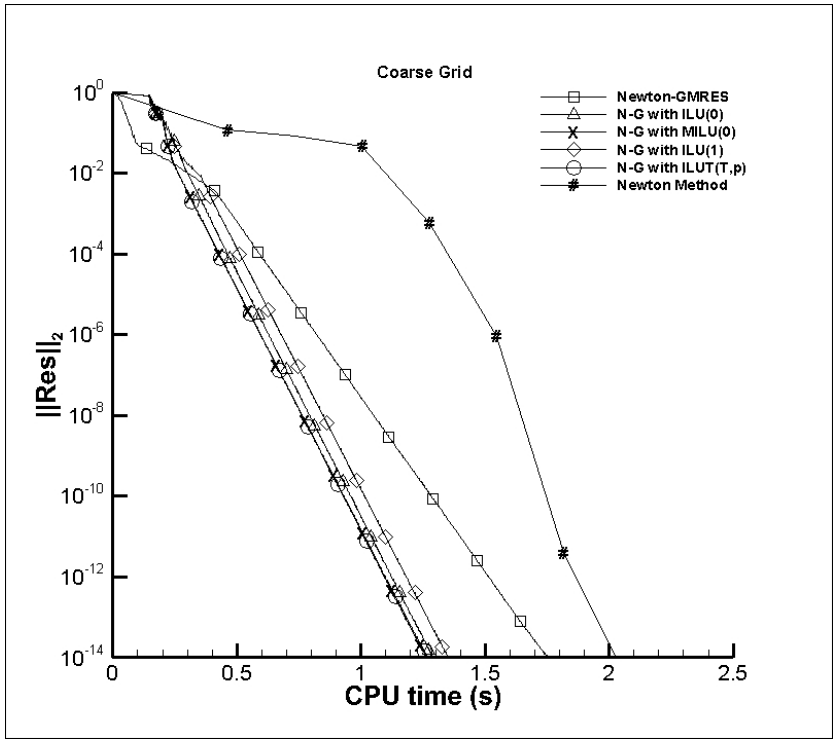

Figure 5.10 Convergence History Comparison for Coarse Grid and $\eta_k = 0.4$

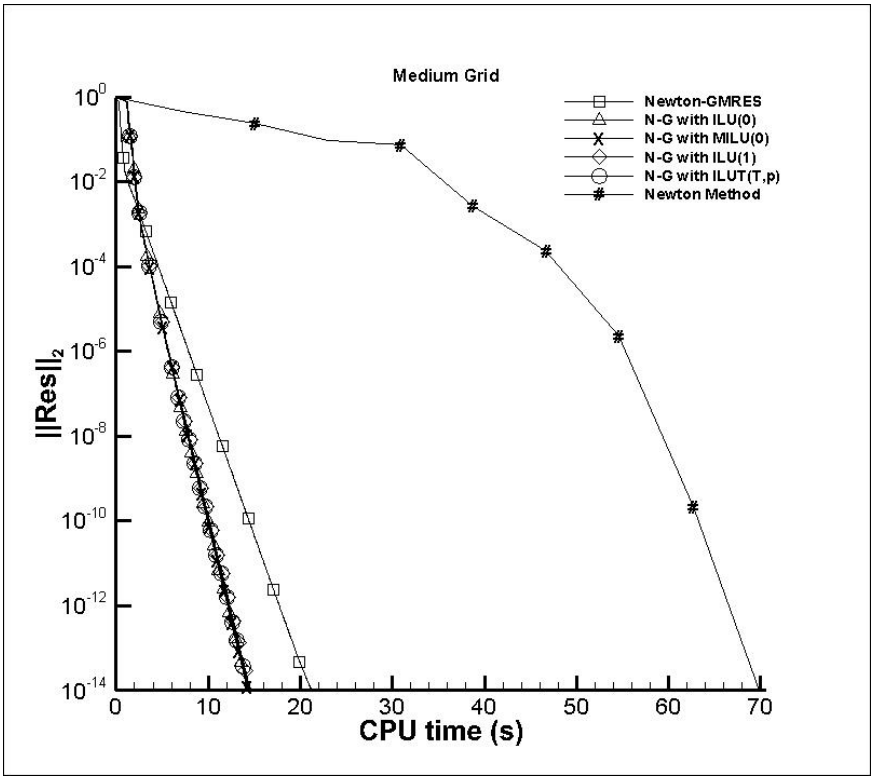

Figure 5.11 Convergence History Comparison for Medium Grid and $\eta_k = 0.4$

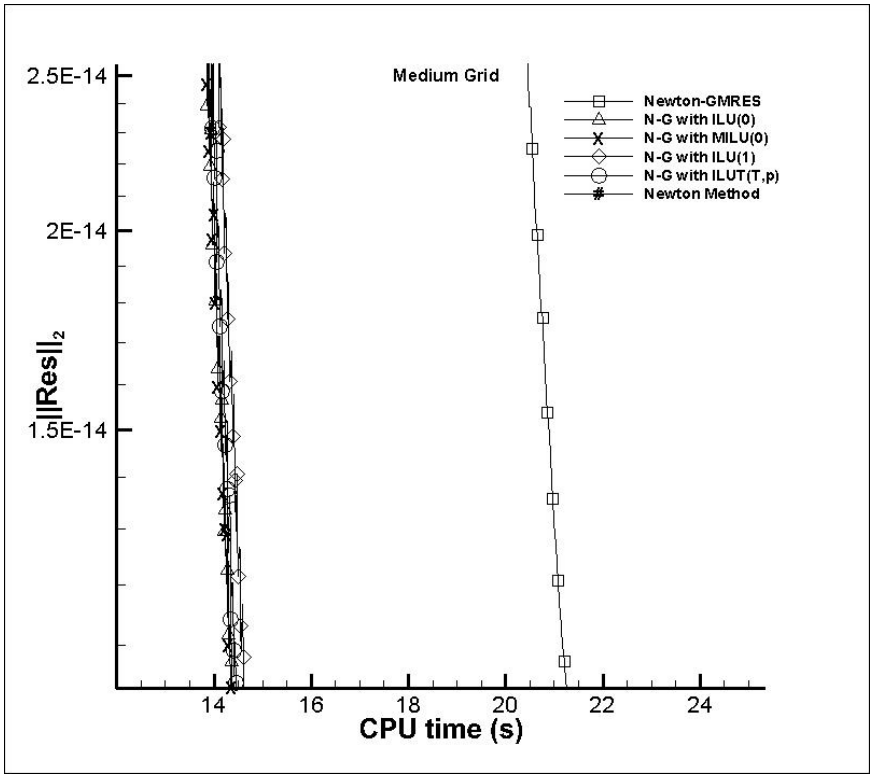

Figure 5.12 Convergence History Comparison for Medium Grid and $\eta_k = 0.4$ (Zoomed)

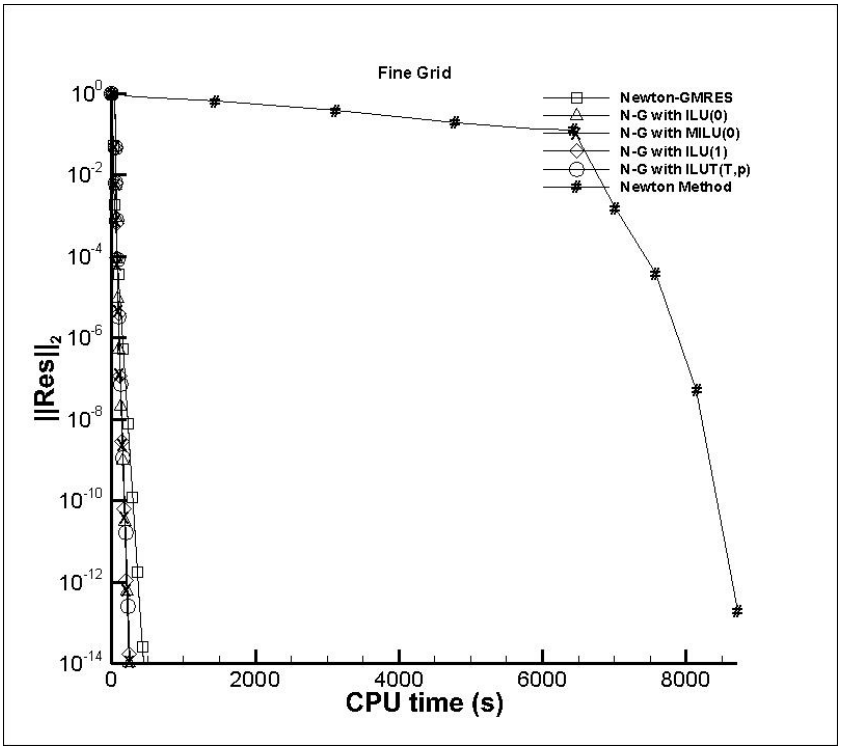

Figure 5.13 Convergence History Comparison for Fine Grid and $\eta_k = 0.4$

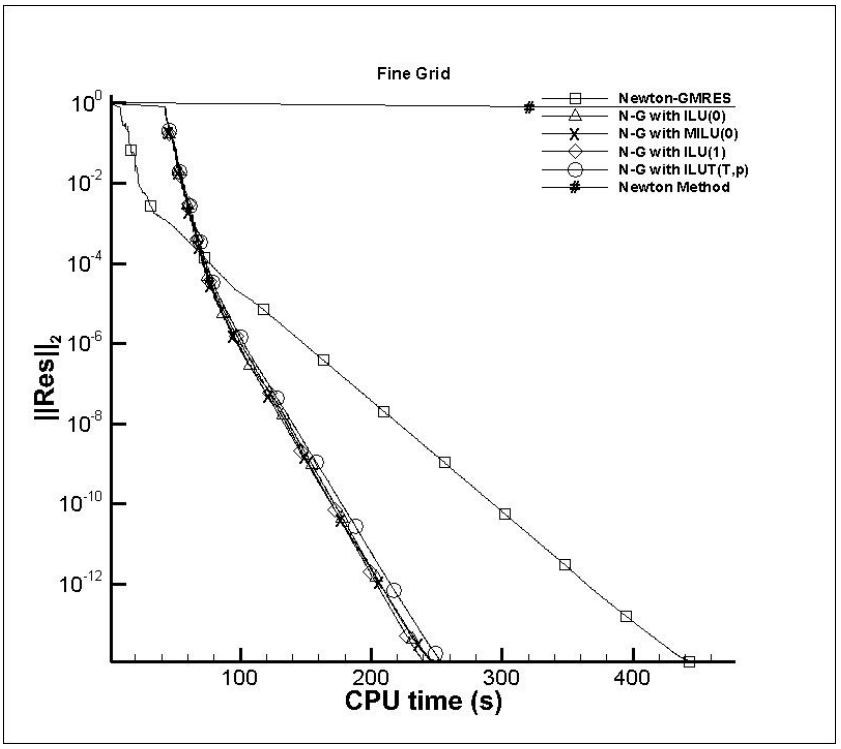

Figure 5.14 Convergence History Comparison for Fine Grid and $\eta_k = 0.4$ (Zoomed for Comparison with Newton-GMRES)

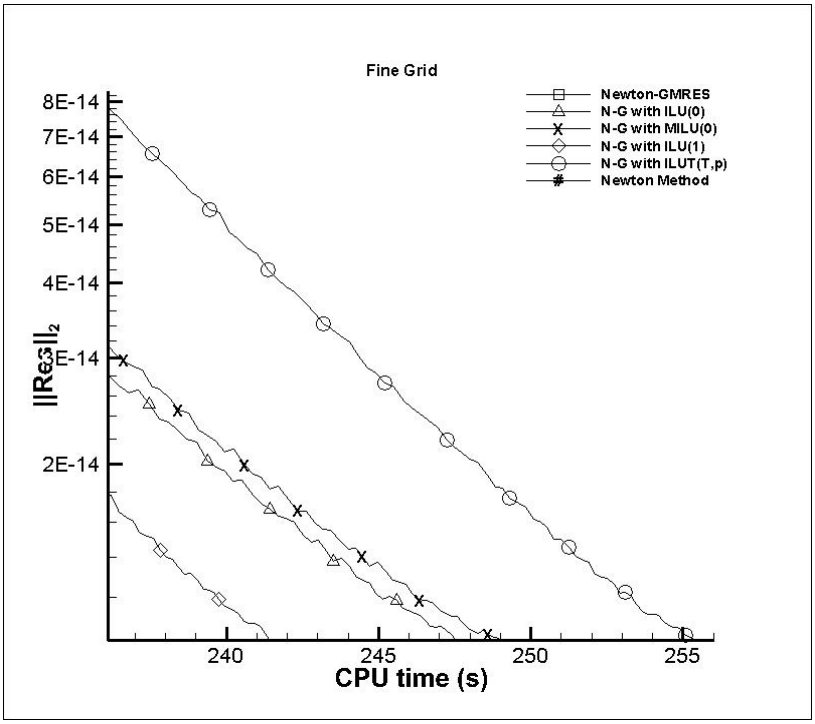

Figure 5.15 Convergence History Comparison for Fine Grid and $\eta_k = 0.4$
(Zoomed for Comparison of Preconditioners)

The convergence history of Newton's method for fine grid in Figure 5.13 could especially have been excluded due to its unwieldy nature. While its Newton-GMRES counterparts converge in approximately five minutes in worst case scenarios, Newton's method takes more than an hour to converge and thus does not represent a viable method in solving a problem with an extremely large matrix. Otherwise, it follows the trend of previous grid types.

In Figures 5.11, 5.12 and 5.14, it can easily be seen that the CPU times of preconditioned Newton-GMRES methods always are as expected the lowest among methods used. It can also be stated that MILU(0) outperforms its counterparts by a small margin from Figure 5.10 and 5.11 (as shown in the tables 5.2 and 5.3 under the next header). The only trend that doesn't fit this description is the one seen in fine grid, in Figure 5.15. This is at part due to CPU load at the time of running the algorithm, as it affects the results, albeit at a miniscule level. The other reason is ILU(1) factorizing the obtained matrix one more time than the other methods used, leading to a better

Jacobian matrix in general. The iteration counts that relate to these methods can be seen in the following Figures; 5.16, 5.17 and 5.18.

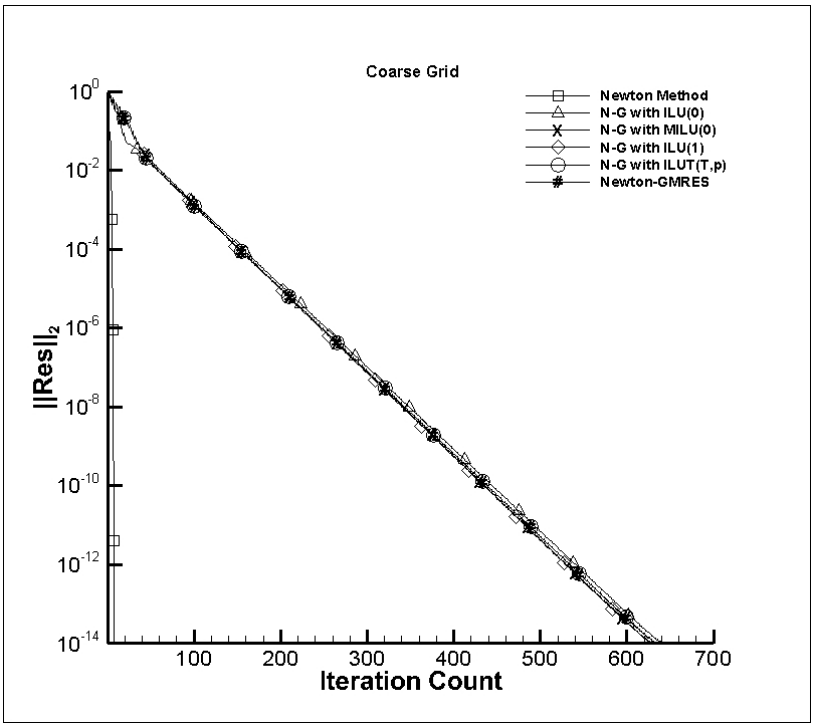

Figure 5.16 Iteration Count Comparison for Coarse Grid and $\eta_k = 0.4$

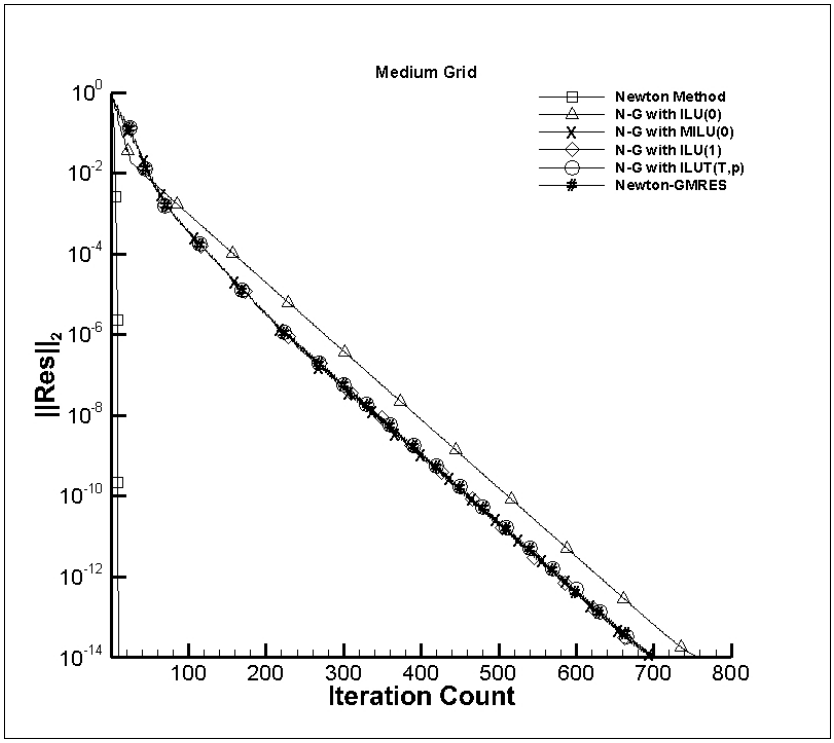

Figure 5.17 Iteration Count Comparison for Medium Grid and $\eta_k = 0.4$

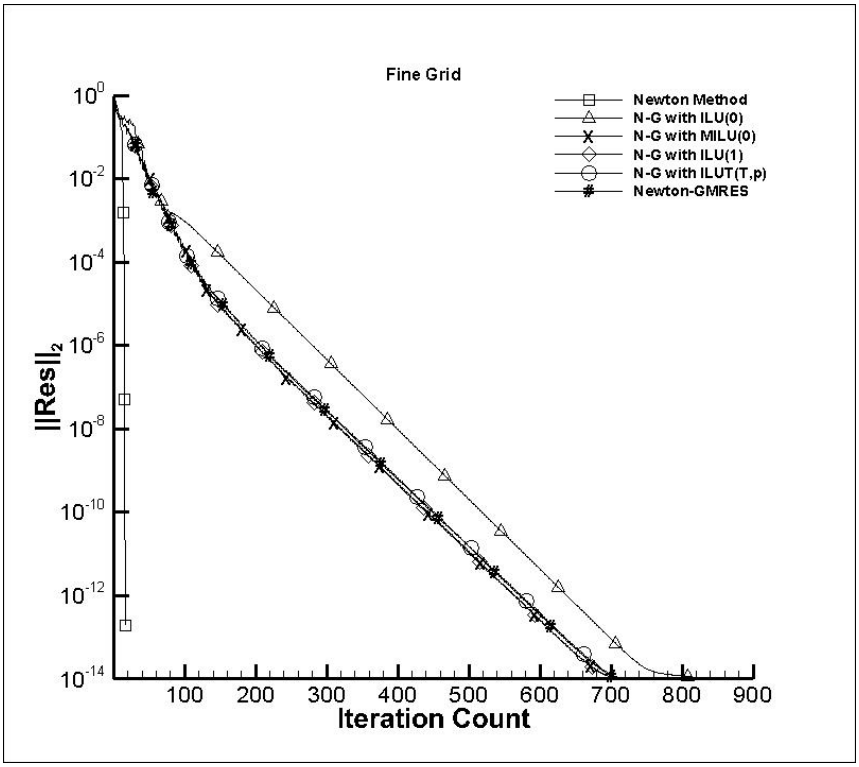

Figure 5.18 Iteration Count Comparison for Fine Grid and $\eta_k = 0.4$

One can conclude from the comparison of these iteration count graphs and the CPU time graphs given before that iteration counts do not directly relate to the speed differences of individual iterative methods. While Newton's method always converges in sub-twenty iterations in given grids, it can take hours to converge. Similar to that, ILU(0) preconditioned Newton-GMRES method has the highest needed amount of iterations to converge while it is also among the fastest methods to converge. In addition to these, it can be seen that preconditioned and unpreconditioned Newton-GMRES methods have a similar trend of iterations needed to be completed.

The Mach contour, pressure distribution, temperature distribution and velocity vector comparisons between Newton and preconditioned Newton-GMRES methods are given for fine mesh in Figures 5.19 through 5.26:

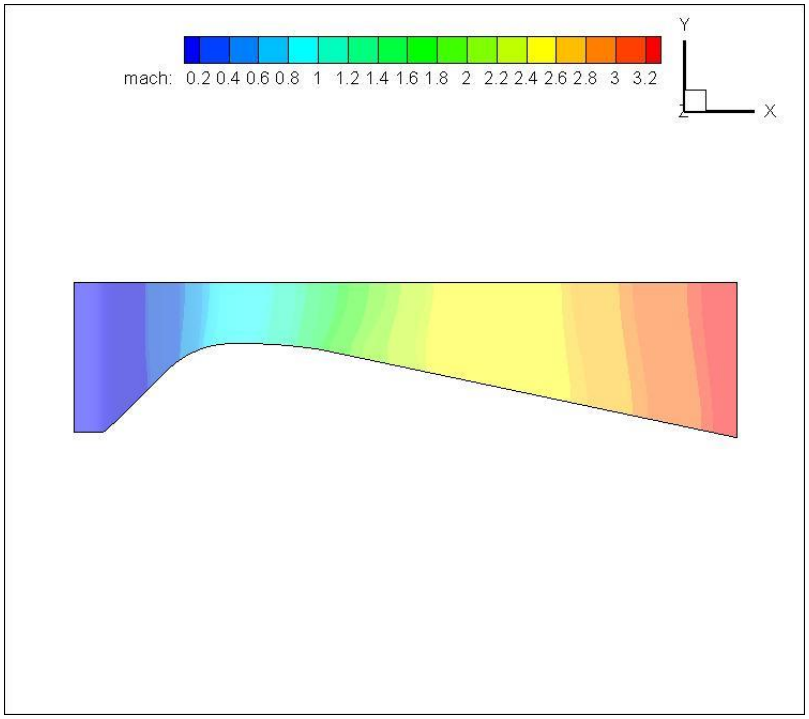

Figure 5.19 Mach Contour for Newton's method

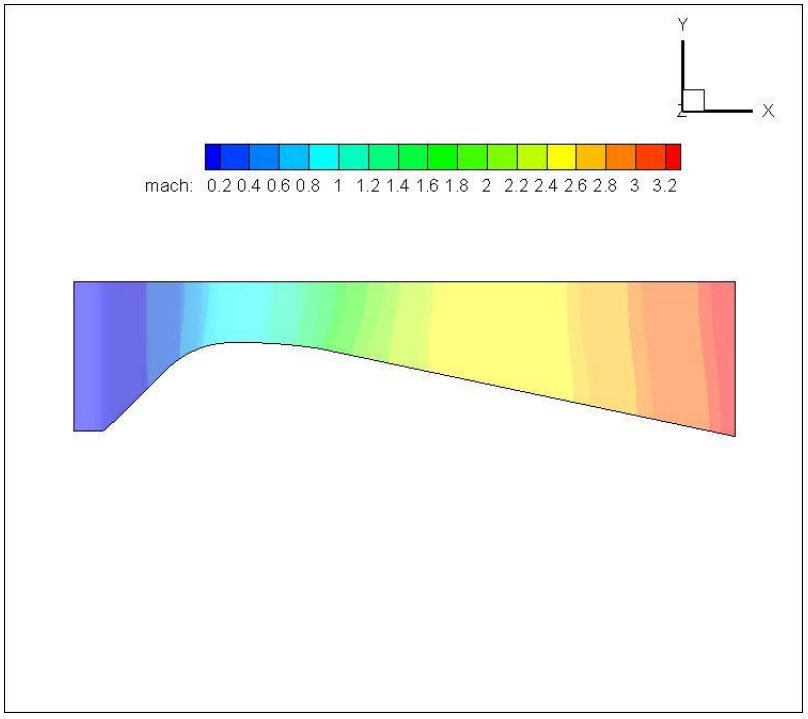

Figure 5.20 Mach Contour for Preconditioned Newton-GMRES Method

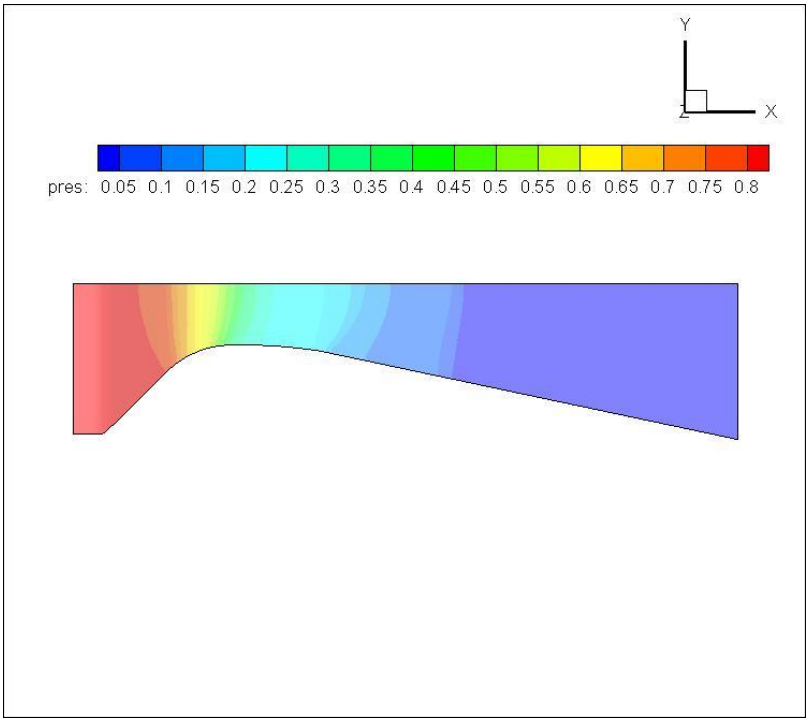

Figure 5.21 Pressure Distribution for Newton's method

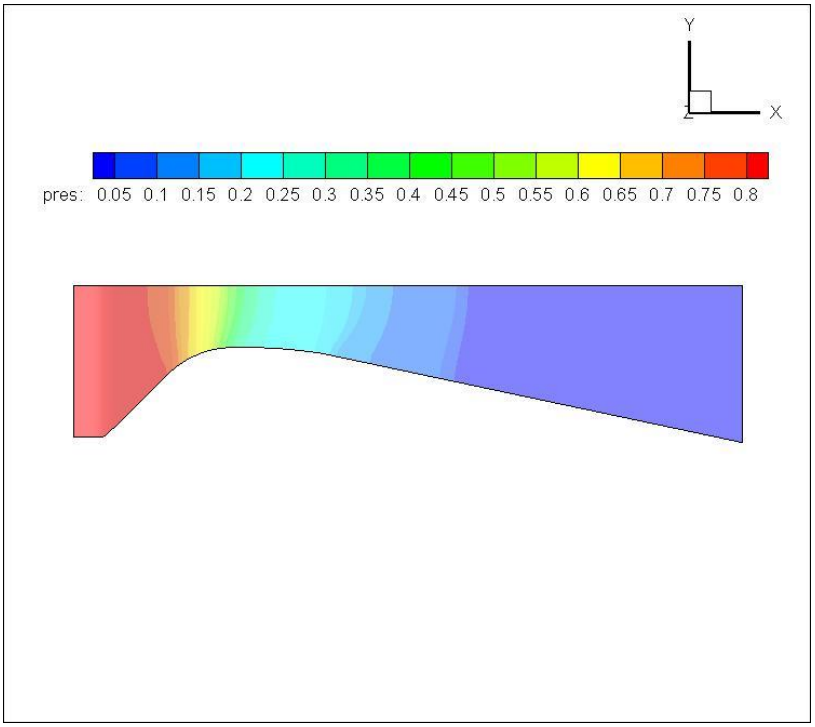

Figure 5.22 Pressure Distribution for Preconditioned Newton-GMRES Method

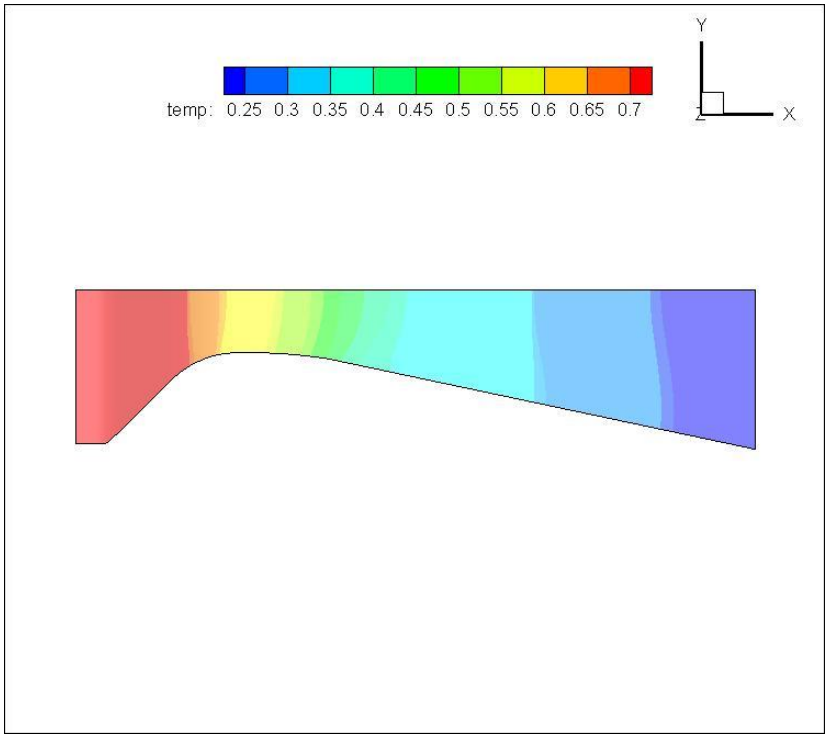

Figure 5.23 Temperature Distribution for Newton's method

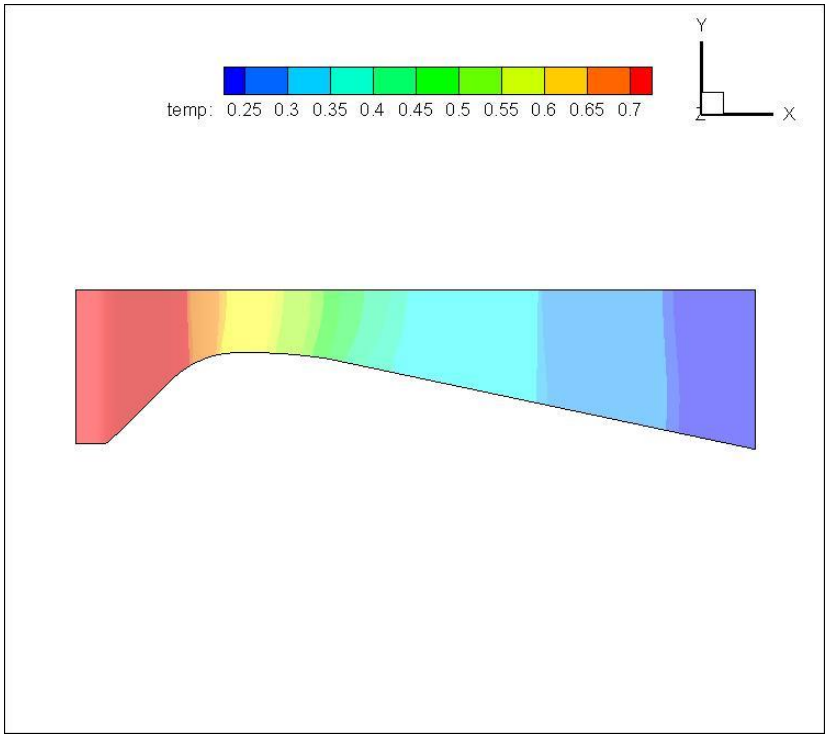

Figure 5.24 Temperature Distribution for Preconditioned Newton-GMRES Method

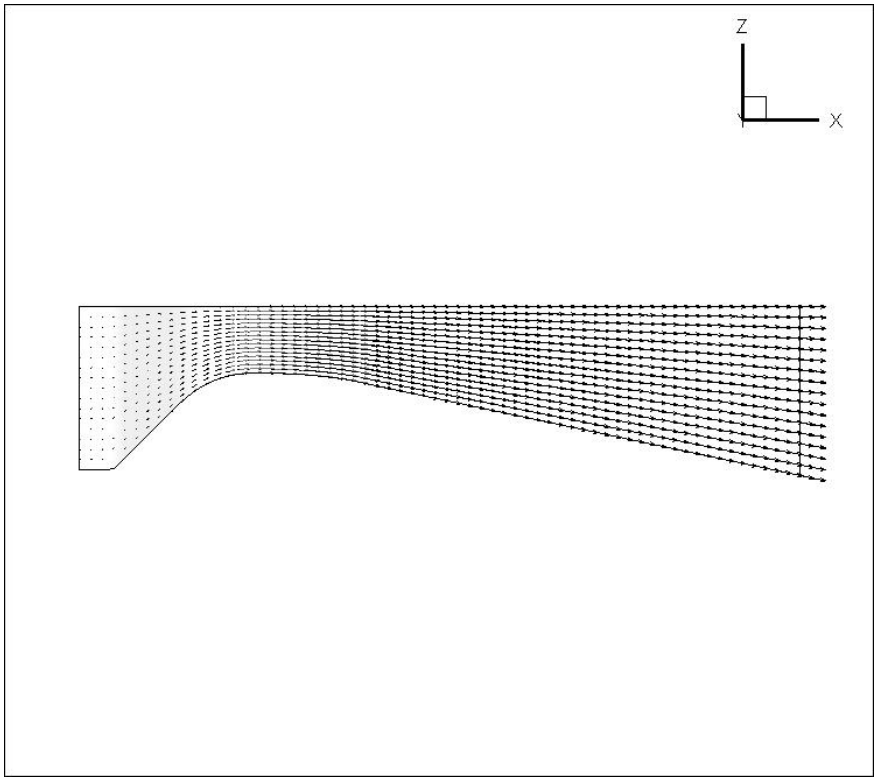

Figure 5.25 Velocity Vectors for Newton's method

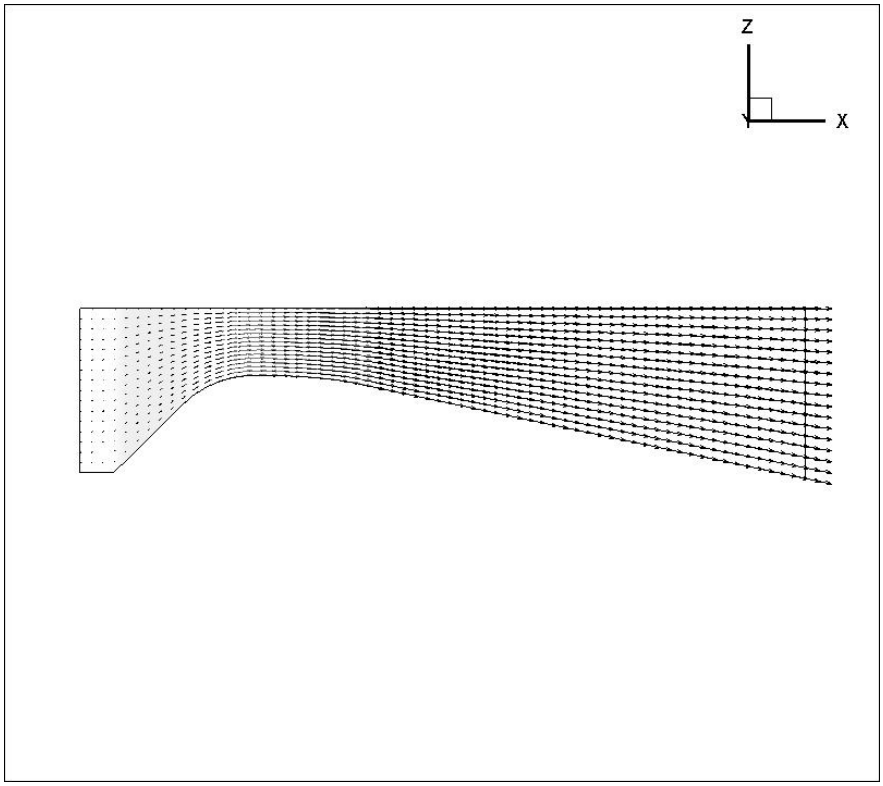

Figure 5.26 Velocity Vectors for Preconditioned Newton-GMRES Method

Section 5.1.2.2 will elaborate on the computational power differences.

5.1.2.2 Numerical Comparison for Different Grids at $\eta_k = 0.4$

The comparison of CPU times and outer (Newton) iteration counts can be seen as tabulated below:

Table 5.2 CPU and Iteration Count Comparisons for $\eta_k = 0.4$

	Coarse		Medium		Fine	
	CPU Time (seconds)	Outer iteration Count	CPU Time (seconds)	Outer iteration Count	CPU Time (seconds)	Outer iteration Count
Newton's method	2,18	8	70,05	10	8710,52	17
Newton - GMRES	1,76	635	21,51	703	450,12	720
ILU(0) Preconditioned Newton- GMRES	1,31	644	14,45	760	245,37	850
MILU(0) Preconditioned Newton- GMRES	1,26	628	14,40	704	246,07	710
ILU(1) Preconditioned Newton- GMRES	1,36	629	14,65	700	239,89	711
ILUT(0.0001,5) Preconditioned Newton- GMRES	1,27	634	14,46	705	253,01	721

Here in Table 5.2, it can much more clearly be seen that preconditioned Newton-GMRES algorithm is superior to its Newton's method and unpreconditioned Newton-GMRES method counterparts. With a level of fill of 5, value threshold of 0.0001 and an η_k value of 0.4, the ILUT preconditioned algorithm performs around 38.58% better than its unpreconditioned counterpart with a coarse mesh, again 48.75% better with a medium mesh and around 77.91% better with a fine mesh. As is expected, the preconditioner results in a small amount of gain in coarse and medium meshes while its benefit in usage with larger meshes can easily be seen, effectively halving the computational time used. The outer iteration count also can be seen to decreasingly increase for every type of mesh applied, which is the result of the inner iterations

giving better approximations than their unmodified counterparts. Another interesting fact that can be obtained from the table is that there is not a single best preconditioner to be applied under every condition. Instead, preconditioners should be chosen according to each specific application. However, if every small performance increase is not being considered, ILU based preconditioners can be said to perform in a uniform manner. It can be deduced that here the calculation and usage of the first partial Jacobian for the preconditioning purpose takes a negligible amount of time in contrast to the amount of time it returns to the user. One concern here is that if the grid was to get much larger and the number of iterations be likewise increased, the Jacobian freezing that we apply would become insufficient as a preconditioning matrix and the need for the calculation of another Jacobian would arise. This could still be bounded by a preset number of total new Jacobian calculations and so regulated to be manageable. Table 5.3 sums the performance gain in each preconditioner used, using a performance factor defined as the preconditioned algorithm CPU time divided by the Newton-GMRES CPU time; meaning values closer to zero display better performance.

Table 5.3 Performance Comparisons for $\eta k = 0.4$

Method	Coarse		Medium		Fine	
	CPU time (s)	Perf. Factor	CPU time (s)	Perf. Factor	CPU time (s)	Perf. Factor
Newton - GMRES	1,76	1	21,51	1	450,12	1
ILU(0) Preconditioned Newton- GMRES	1,31	0,744	14,45	0,671	245,37	0,545
MILU(0) Preconditioned Newton- GMRES	1,26	<u>0,715</u>	14,4	<u>0,669</u>	246,07	0,546
ILU(1) Preconditioned Newton- GMRES	1,36	0,772	14,65	0,681	239,89	<u>0,532</u>
ILUT(0.0001,5) Preconditioned Newton- GMRES	1,27	0,721	14,46	0,672	253,01	0,562

Best performances are underlined.

5.1.2.3 Graphical Comparison for Different Grids at $\eta_k = 0.6$

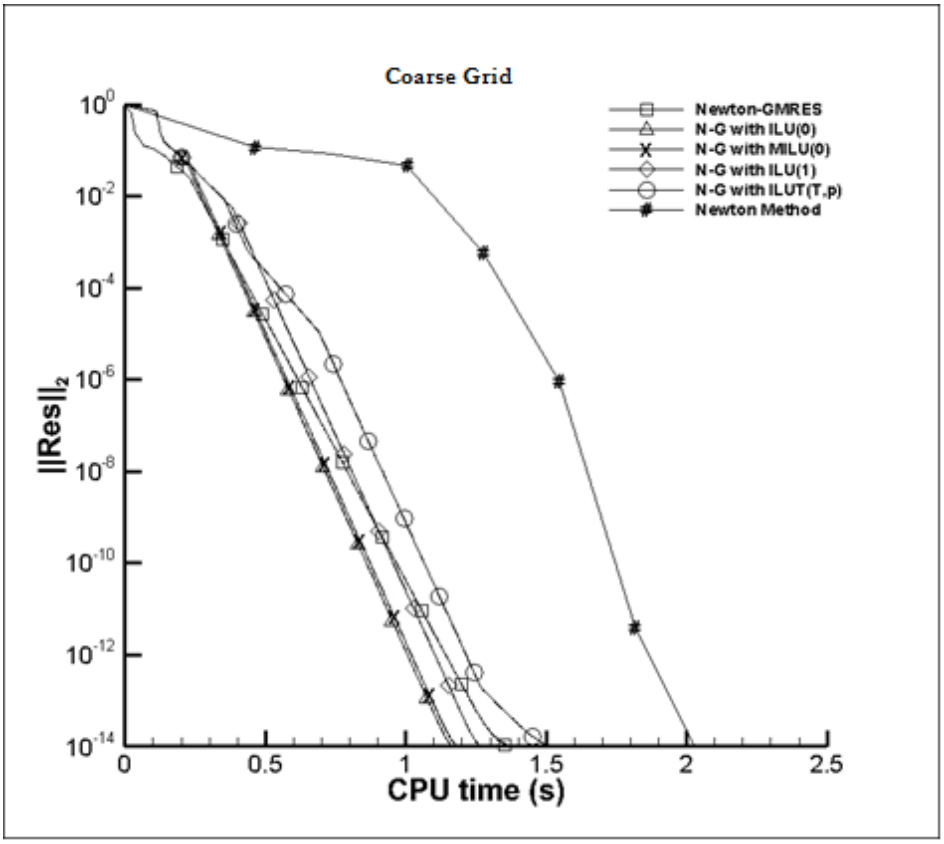

Figure 5.27 Convergence History Comparison for Coarse Grid and $\eta_k = 0.6$

In Figure 5.27, it can be observed that the preconditioned Newton-GMRES is following the trend created by its predecessors in terms of residual history. As the grid is coarse and the calculations are extremely fast, convergence time differences between methods are expectedly small. Yet, it can be seen that ILU(0) and MILU(0) methods have a small amount of superiority over their competitors in this smaller grid. Still, the differences are so small that conclusions such as definitively stating ILUT is always slower than other ILU based preconditioners tested here in regards to coarse grids would be wrong.

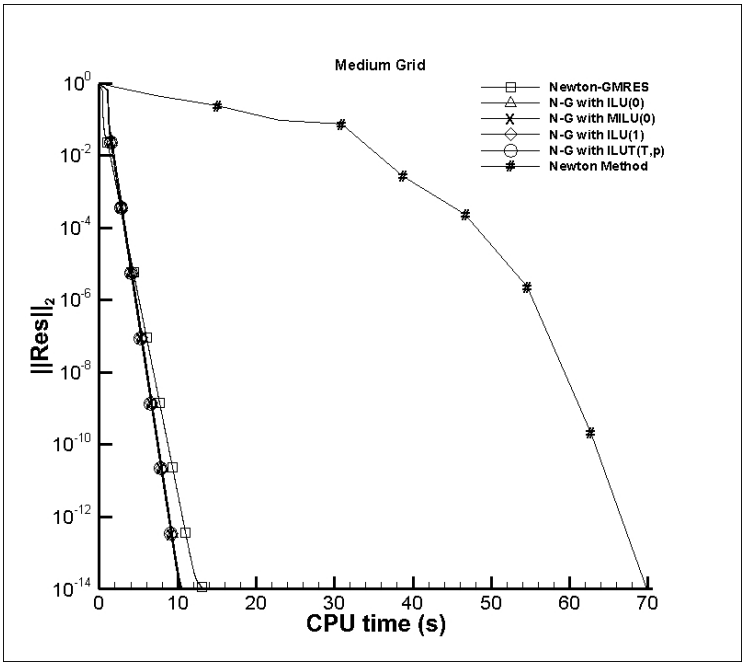

Figure 5.28 Convergence History Comparison for Medium Grid and $\eta_k = 0.6$

In Figure 5.28, we can see that the preconditioning process results in noticeable difference over the unpreconditioned version. If we were to inspect Figure 5.29 closely:

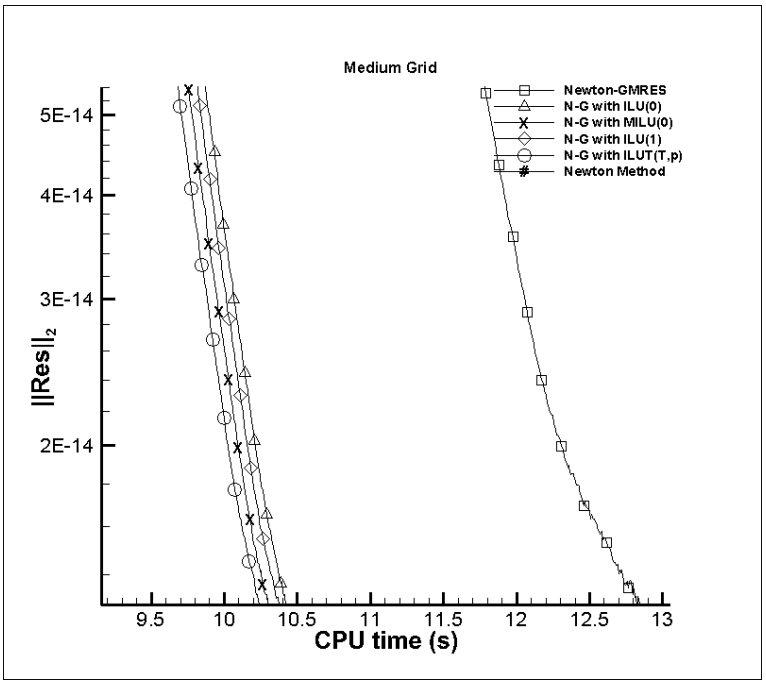

Figure 5.29 Convergence History Comparison for Medium Grid and $\eta_k = 0.6$ (Zoomed)

We would be able to see that ILUT performed slightly better than its competitors in this grid. In contrast, ILU(0) was the slowest this time around.

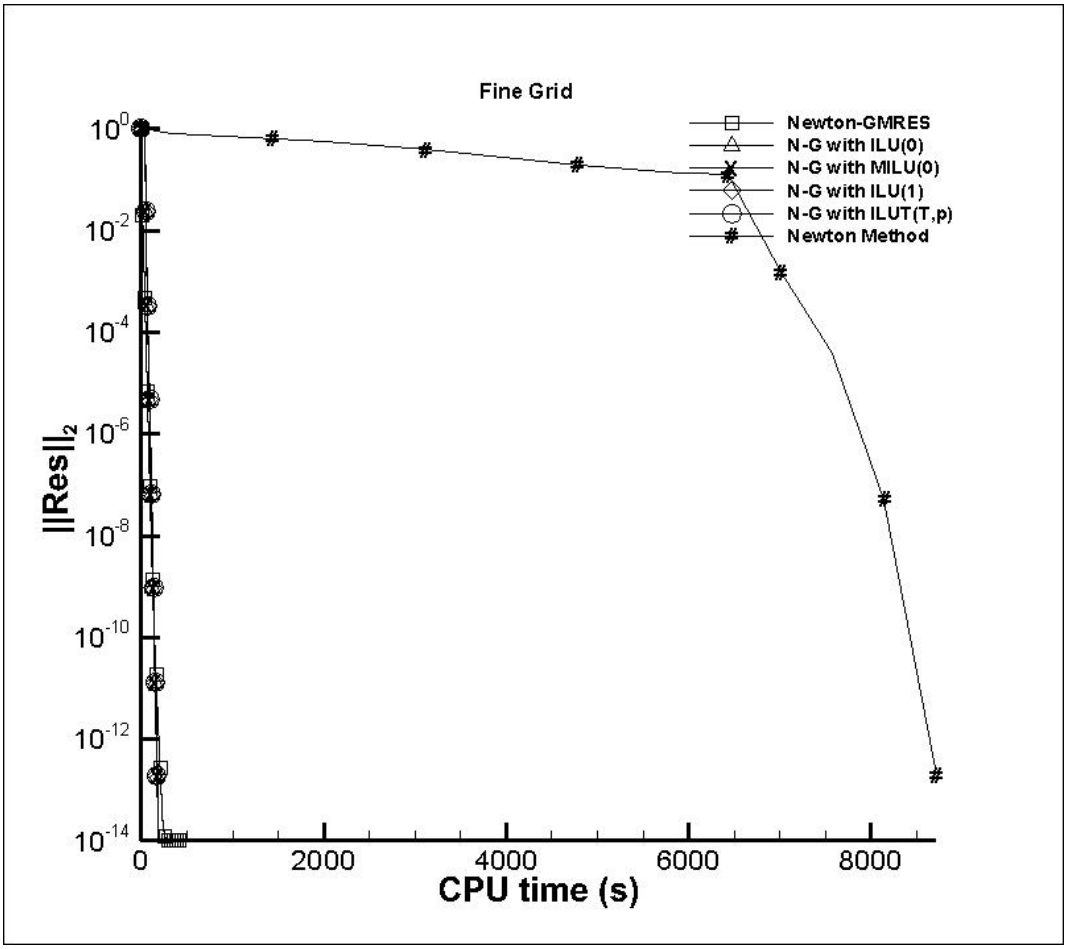

Figure 5.30 Convergence History Comparison for Fine Grid and $\eta_k = 0.6$

Again, for display purposes, we include the CPU time plot without extracting Newton's method in Figure 5.30. The following Figures, 5.31 and 5.32 are magnified to show the Newton-GMRES process results:

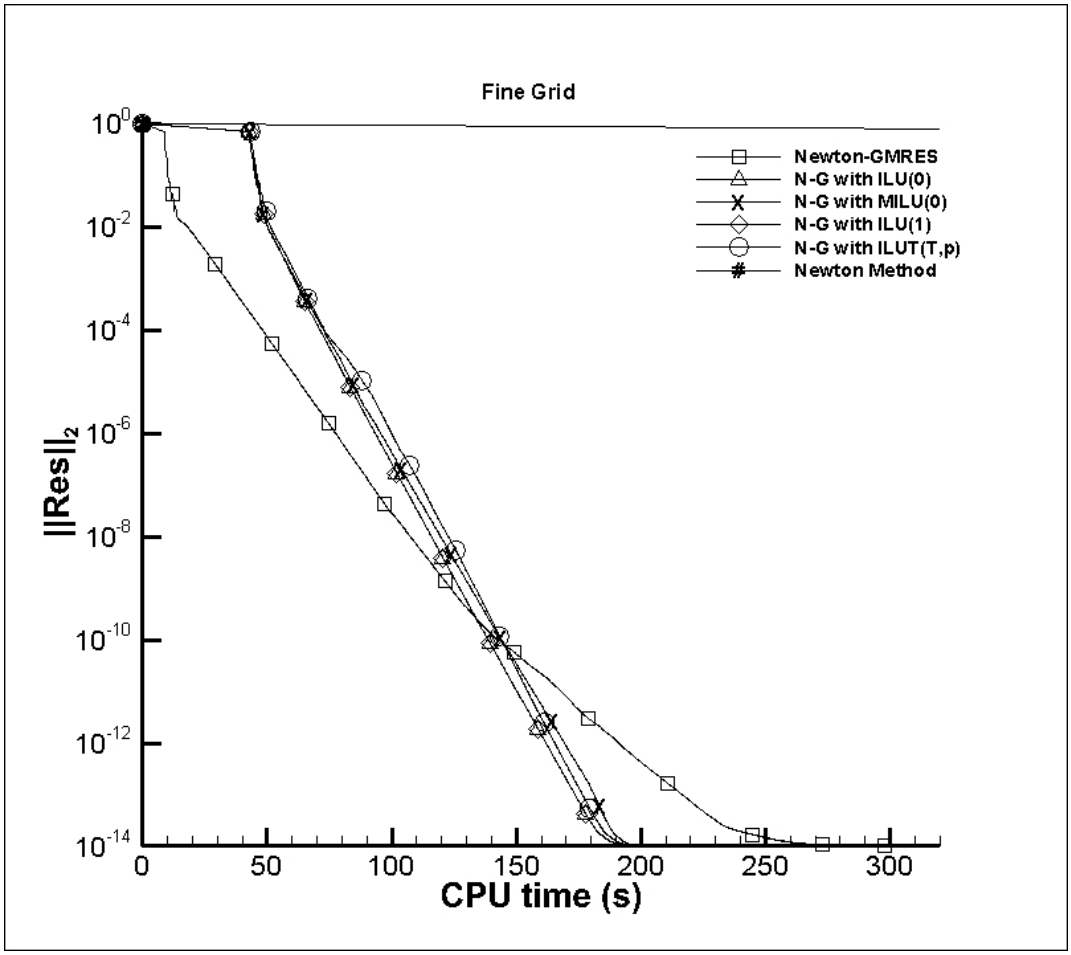

Figure 5.31 Convergence History Comparison for Fine Grid and $\eta_k = 0.6$ (Zoomed)

In Figure 5.31, it can easily be observed that the performance increase brought by preconditioning. The residual drops faster than preconditioned residual in Newton-GMRES until it reaches a certain amount, after which it slows its search in the Krylov subspace. One more magnification to this plot can tell which methods are the fastest in this grid.

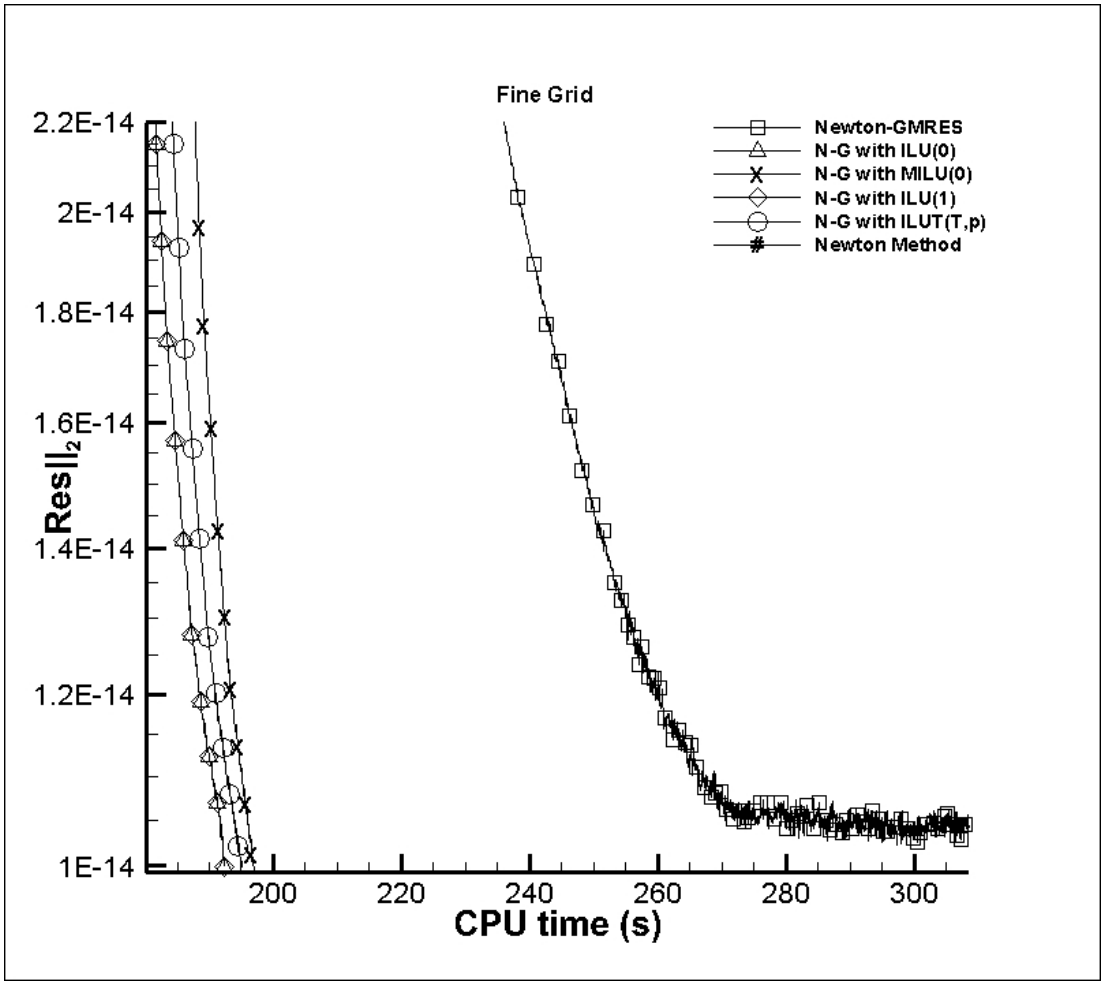

Figure 5.32 Convergence History Comparison for Fine Grid and $\eta_k = 0.6$
(Zoomed for Comparison of Preconditioners)

In Figure 5.32, it can be seen that ILU(0) and ILU(1) are the fastest preconditioners for our system at $\eta_k = 0.6$ value and the largest grid. It is an interesting observation that the preconditioners result in even closer CPU times this time around. In fact, ILU(0) and ILU(1) take almost exactly the same amount of time. One can also observe that the Newton-GMRES algorithm stalls around 10^{-14}. This passes a few more seconds later but is a rather interesting observation. The exact reasoning behind it is open to interpretation, yet it is known that iterative solvers are prone to this kind of behavior when operating at residuals this low.

The related iteration counts are displayed as follows in Figures 5.33 through 5.35 for reference:

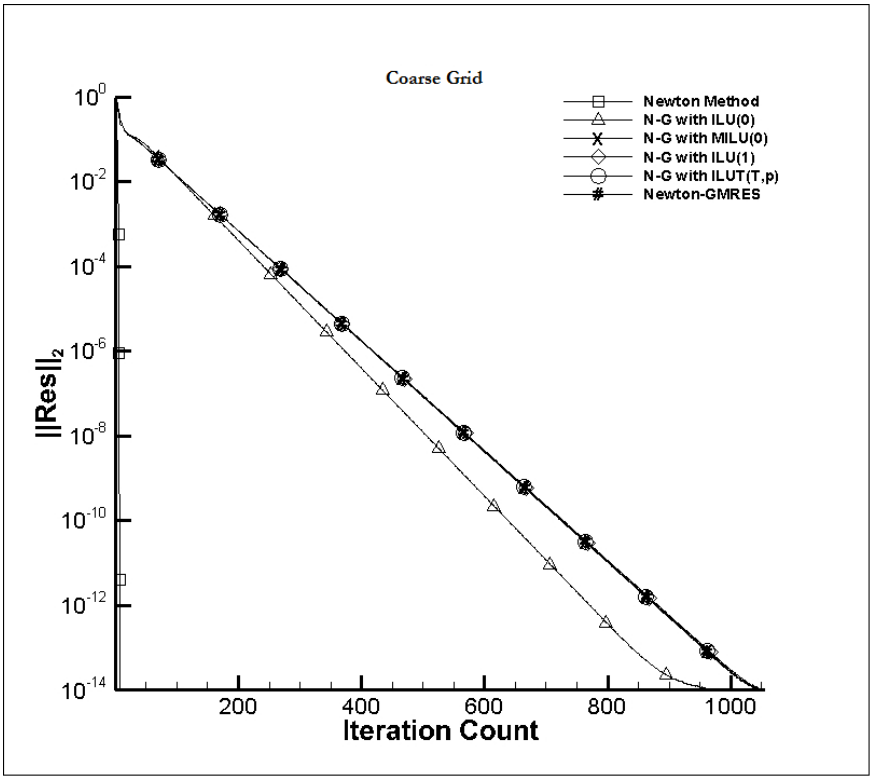

Figure 5.33 Iteration Count Comparison for Coarse Grid and $\eta_k = 0.6$

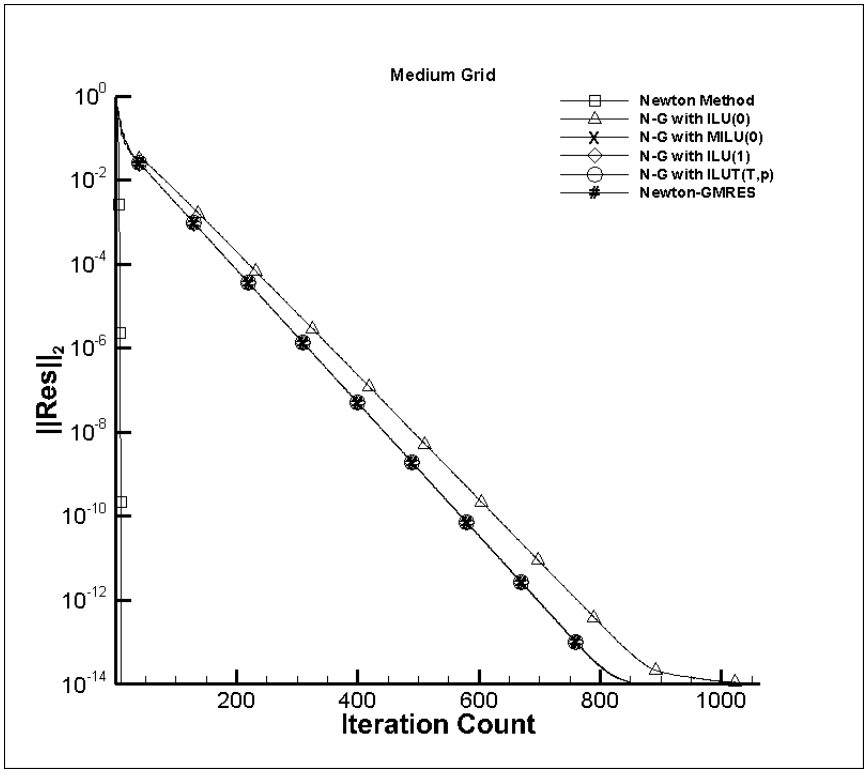

Figure 5.34 Iteration Count Comparison for Medium Grid and $\eta_k = 0.6$

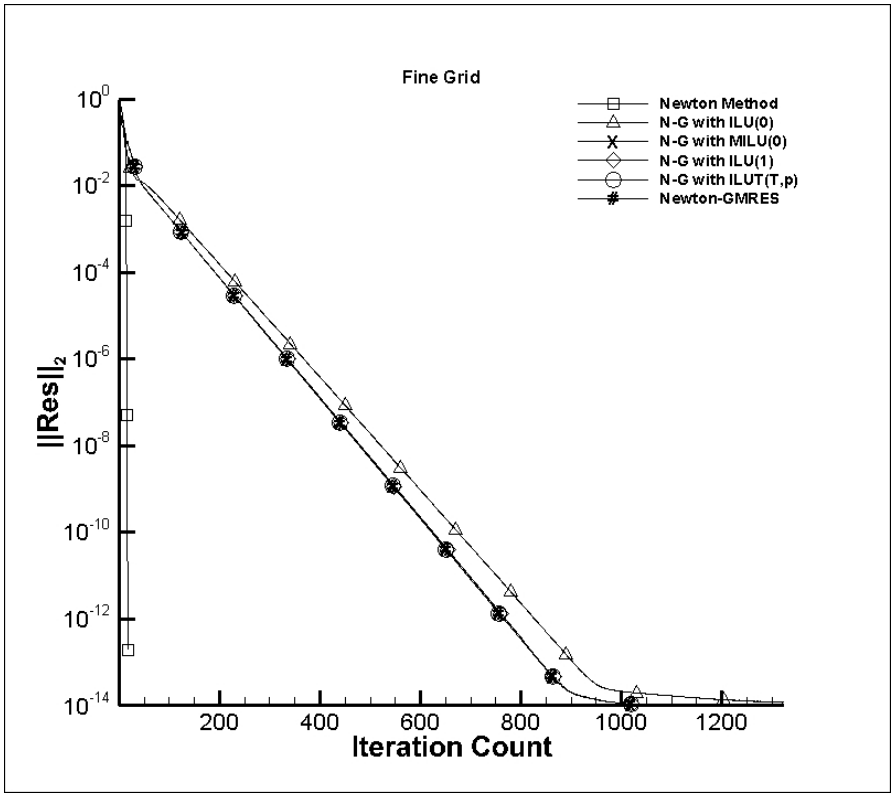

Figure 5.35 Iteration Count Comparison for Fine Grid and $\eta_k = 0.6$

The residual history in this case of $\eta_k = 0.6$ behaves in the same way as before in $\eta_k = 0.4$ with the one change being in that the residual for Newton's method cannot even be manipulated enough to reach 10^{-14} level. Still, the Newton-GMRES methods can converge down to 10^{-17} level with ease. The scale for the residual has been kept constant throughout the plots between 1 and 10^{-14} to enable easier comparison with Newton's method and all the curves are known to follow their shown trends below that value. The contours given in the previous sections will not be repeated here so as not to fall to duplicity.

5.1.2.4 Numerical Comparison for Different Grids at $\eta_k = 0.6$

The comparison of CPU times and outer (Newton) iteration counts can be seen as tabulated below in Table 5.4:

Table 5.4 CPU and Iteration Count Comparisons for $\eta_k = 0.6$

	Coarse		Medium		Fine	
	CPU Time (seconds)	Outer iteration Count	CPU Time (seconds)	Outer iteration Count	CPU Time (seconds)	Outer iteration Count
Newton's method	2,18	8	70,05	10	8710,52	17
Newton - GMRES	1,32	1046	12,80	855	259,11	968
ILU(0) Preconditioned Newton- GMRES	1,14	969	10,45	1050	191,51	1290
MILU(0) Preconditioned Newton- GMRES	1,16	1045	10,35	854	195,61	967
ILU(1) Preconditioned Newton- GMRES	1,24	1056	10,41	854	191,49	969
ILUT(0.0001,5) Preconditioned Newton- GMRES	1,46	1048	10,29	853	193,31	967

In Table 5.4, the data is consistent with the case of $\eta_k = 0.4$ at least in terms of preconditioning generally speeding up the algorithm. Yet, there is one case where performance increase was not seen. In the case of ILUT preconditioner coupled with a coarse grid, the solver actually took longer than it would left unpreconditioned. That is due to the value of η_k which determines the relation of the outer and inner iterations. If the iterations are left to be less reliant on the linear solver and more on the Newton step, the positive effect of the preconditioner can become negligible in relation to its negative effect. With that exception in mind, the preconditioning process speeds up the convergence across all grid sizes, this being more apparent the larger the grid grows. It can also be seen that increasing the forcing term has sped up the algorithm for all grid sizes. The optimization of this forcing term coupled with the right preconditioner for your grid size can lead to optimized routines. The following Table 5.5 will clarify the table above in terms of performance between preconditioners:

Table 5.5 Performance Comparisons for ηk = 0.6

Method	Coarse		Medium		Fine	
	CPU time (s)	Perf. Factor	CPU time (s)	Perf. Factor	CPU time (s)	Perf. Factor
Newton - GMRES	1,32	1	12,8	1	259,11	1
ILU(0) Preconditioned Newton- GMRES	1,14	0,863	10,45	0,816	191,51	0,739
MILU(0) Preconditioned Newton- GMRES	1,16	0,878	10,35	0,808	195,61	0,754
ILU(1) Preconditioned Newton- GMRES	1,24	0,939	10,41	0,813	191,49	0,739
ILUT(0.0001,5) Preconditioned Newton- GMRES	1,46	1,106	10,29	0,803	193,31	0,746

Lower performance factor means better performance. Best performances are underlined. If these results are compared to those obtained from ηk = 0.4:

Table 5.6 Performance Comparisons of different ηk values

Method	Coarse		Medium		Fine	
	ηk = 0.4	ηk = 0.6	ηk = 0.4	ηk = 0.6	ηk = 0.4	ηk = 0.6
Newton - GMRES	1	1	1	1	1	1
ILU(0) Preconditioned Newton- GMRES	0,744	0,863	0,671	0,816	0,545	0,739
MILU(0) Preconditioned Newton- GMRES	0,715	0,878	0,669	0,808	0,546	0,754
ILU(1) Preconditioned Newton- GMRES	0,772	0,939	0,681	0,813	0,532	0,739
ILUT(0.0001,5) Preconditioned Newton- GMRES	0,721	1,106	0,672	0,803	0,562	0,746

It can be seen that the ILU(1) preconditioner is the fastest in both values of ηk when considering larger grids. As the grid gets smaller, the choice becomes harder, generally leaning towards simpler preconditioners (ILU(0) and MILU(0)). Both parts of result is to be expected. The only unexpected part is the low convergence rate of ILUT, which we expected to be higher. The simplest reason for that can be a problem in choosing

the level of fill or threshold for that preconditioner. The following section relays a simple test to determine the viability of using a lower level of fill. Before concluding, it can also be stated that a higher value of η_k results in a generally faster algorithm.

5.1.2.5 Graphical Comparison for Level of Fill = 4 and 5 and $\eta_k = 0.6$

For the purpose of showcasing the difference level of fill makes for the ILUT preconditioner, the results of the flow solution at level of fill = 4 with a medium grid is compared to the flow solution at level of fill = 5 with the same grid. The graphical comparison is as follows through Figures 5.36 to 5.43:

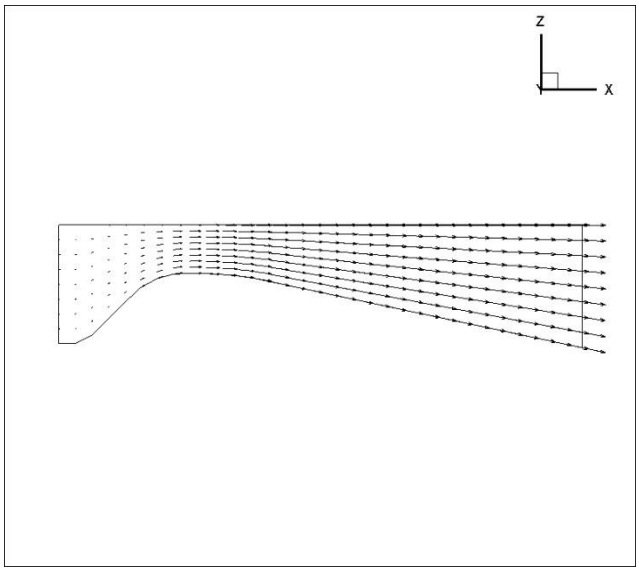

Figure 5.36 Mach Contours for Lfil = 4

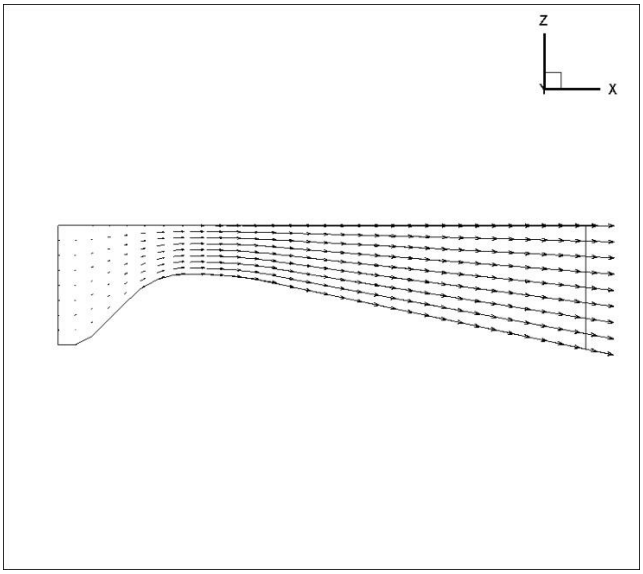

Figure 5.37 Mach Contours for Lfil = 5

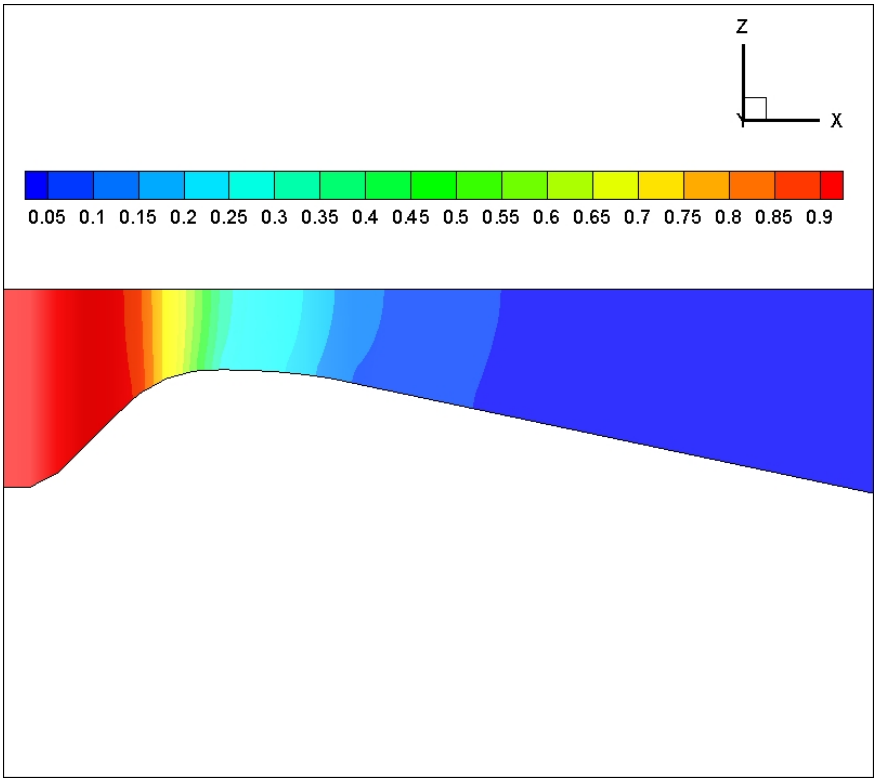

Figure 5.38 Pressure Distribution for Lfil = 4

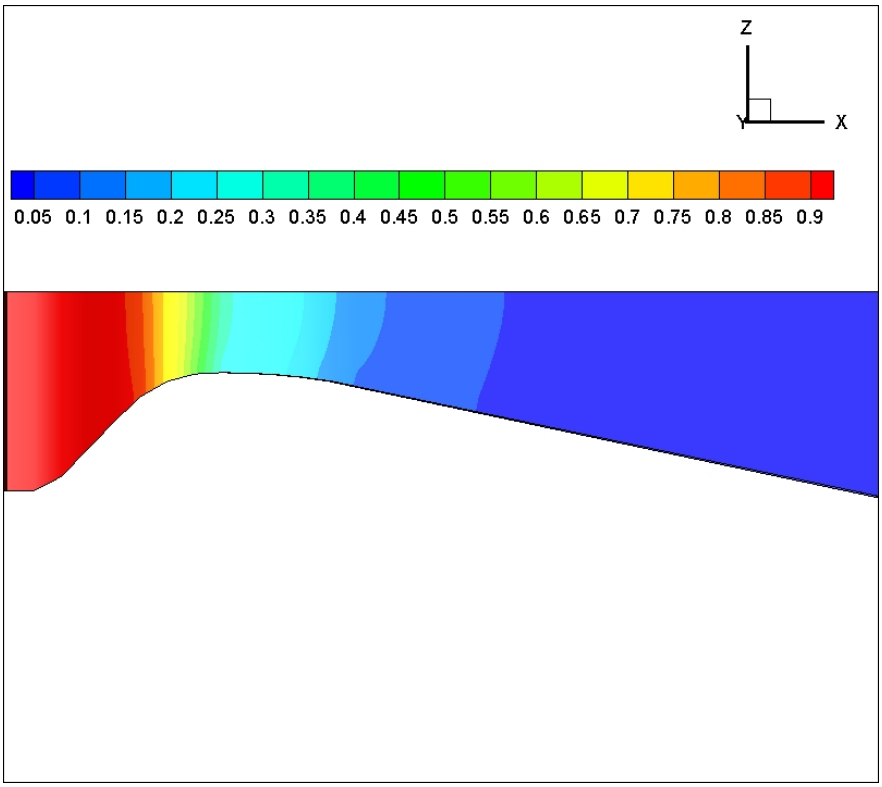

Figure 5.39 Pressure Distribution for Lfil = 5

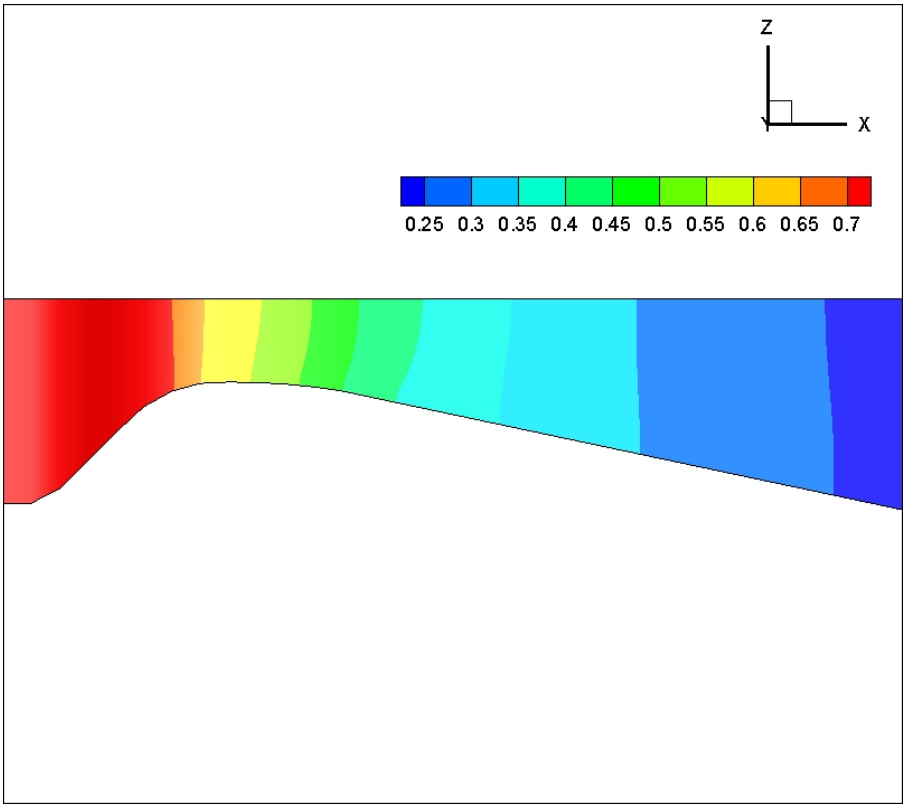

Figure 5.40 Temperature Distribution for Lfil = 4

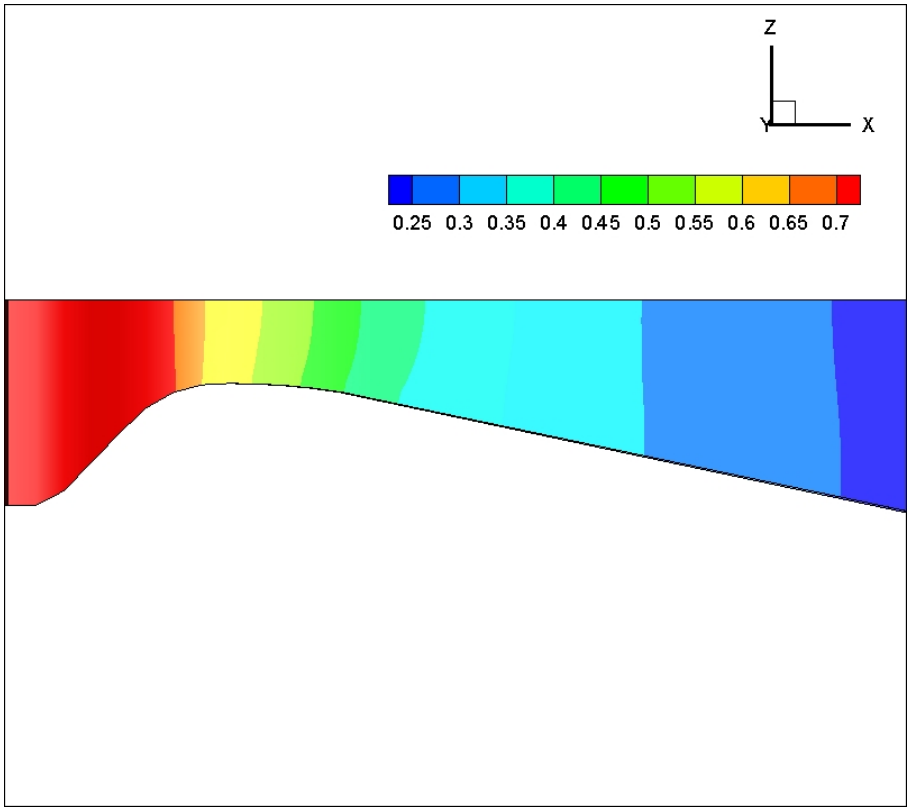

Figure 5.41 Temperature Distribution for Lfil = 5

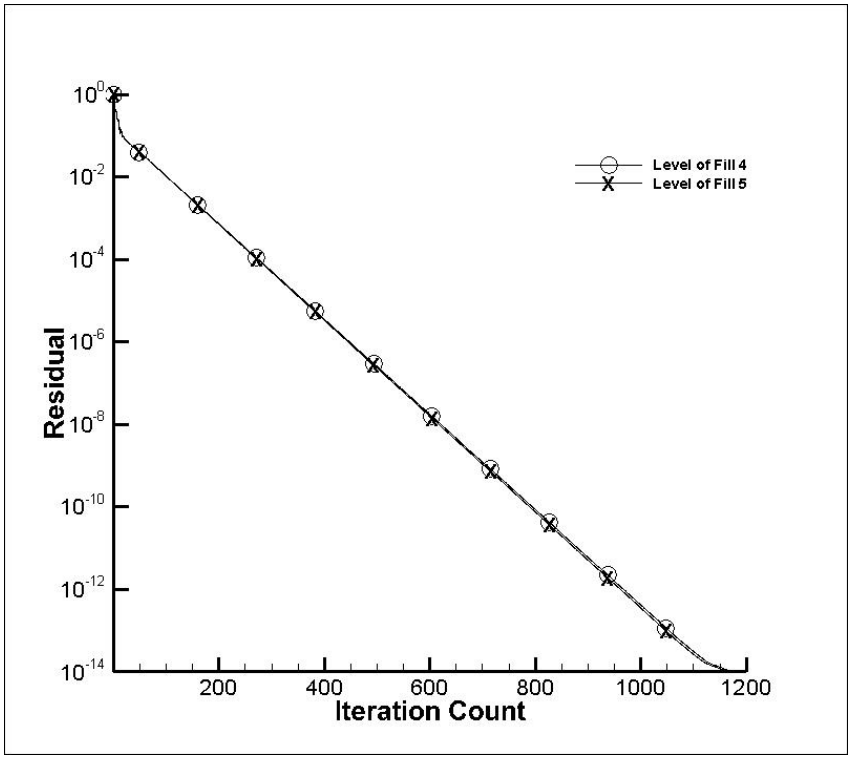

Figure 5.42 Residual Histories for Lfil = 4 and 5

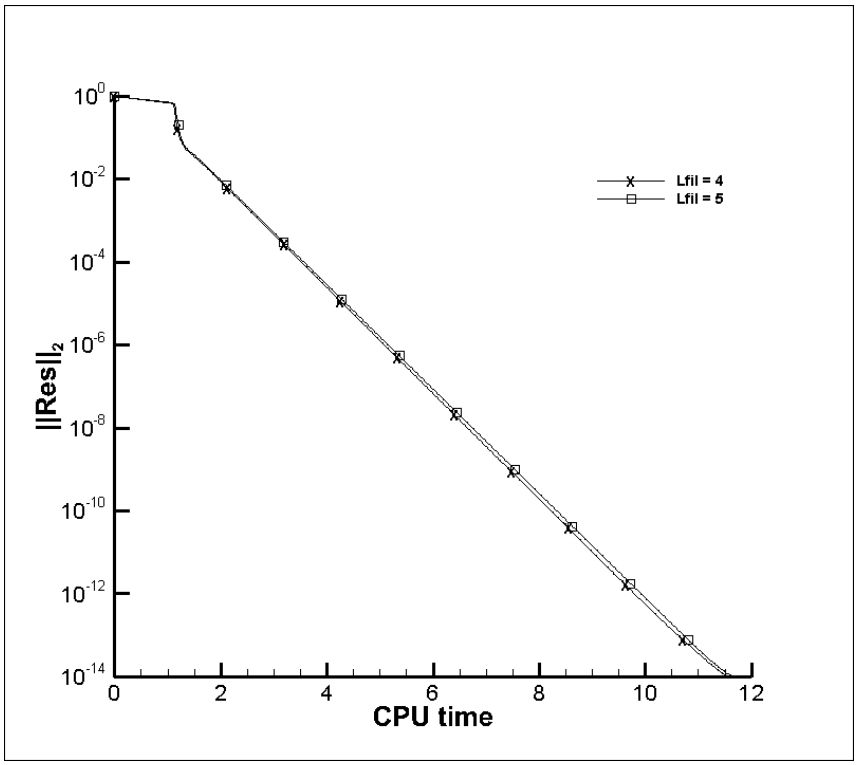

Figure 5.43 CPU Time Comparison for Lfil = 4 and 5

Here, it can be seen that lowering the value of level of fill on medium grid does not have any significant effects on residual histories iteration count-wise or with respect to CPU time.

5.1.2.6 Numerical Comparison for Level of Fill = 4 and 5 and ηk=0.6

The comparison of CPU times and outer (Newton) iteration counts can be seen as tabulated below in Table 5.7:

Table 5.7 CPU and Iteration Count Comparisons for Level of Fill = 4 and 5

	Coarse		Medium		Fine	
	CPU Time (seconds)	Outer iteration Count	CPU Time (seconds)	Outer iteration Count	CPU Time (seconds)	Outer iteration Count
Level of Fill = 4	1,35	1055	11,51	1162	-	-
Level of Fill = 5	1,46	1048	11,55	1151	193,31	967

As can be seen from Table 5.7, decreasing the level of fill results in a somewhat faster convergence in medium and coarse meshes while the algorithm does not achieve convergence on a fine mesh. The non-convergence and the computational speed improvement are due to the same reason. As the mesh fineness increases, the number of elements the threshold algorithm drops increases. For a coarse mesh, the dropped number of elements will be low which in turn will speed up the overall solution while not affecting convergence whereas for a fine mesh, the dropped number of elements will be higher which can result in a matrix that cannot be solved. This means the level of fill carries great importance when improving matrix sizes and mesh fineness. Also, it was observed that while the algorithm with level of fill 4 reaches 10^{-14} residual level in just a few steps after the level of 10^{-12}, the algorithm with level of fill 5 lingers for a long time at 10^{-12} residual level, which is the main cause for its time loss. For a real engineering problem, a residual level that low would almost never be necessary, which would mean the convergence times would be much closer to each other. This in turn leads to a higher level of fill in ILUT being more useful with the tendency of a low level of fill preconditioner to lead to less dependable results kept in mind. Still, a lower level of fill can be useful if the problem is very simple. Before going into conclusions, the last piece of comparative information should be included in the form of second order discretization in the following section.

5.1.3 Comparison of Second Order Discretized Solutions

In addition to the first order discretization done in chapter 2, it was stated in this thesis that second order discretization simply slows the algorithm by a large amount, thus making its comparison with its first order counterpart somewhat misguiding. That being taken into account, one still can draw some conclusions from its analysis. In this section, we will submit the results obtained from that approach and make a few remarks on their comparison. The discretization itself is done using MUSCL (Monotonic Upstream Centered Scheme Conservation Law) scheme interpolation by Van Leer. The flow variables on cell faces are found using the flow variables at cell centers of the four neighboring cells. The scheme and its application will be included here in summarized form instead of being in chapter 2 to avoid any misconceptions while reading all the previous chapters. As an alternative to equation (14) at page 11, the following is valid here:

$$\hat{Q}^{L}_{i+\frac{1}{2}} = \hat{Q}_i + \frac{1}{4}\{\Phi(r)[(1-K)\nabla + (1+K)\Delta]\}$$

$$\hat{Q}^{R}_{i+\frac{1}{2}} = \hat{Q}_{i+1} - \frac{1}{4}\{\Phi(r)[(1+K)\nabla + (1-K)\Delta]\}_{i+1} \qquad (42)$$

Where

$$r_i = \frac{\Delta_i}{\nabla_i} \quad , \quad \Delta_i = \hat{Q}_{i+1} - \hat{Q}_i \quad , \quad \nabla_i = \hat{Q}_i - \hat{Q}_{i-1}$$

Here, *K* defines the order of the differencing. It is taken 0 to be used with Van Albada limiter function. The limiter function we mentioned would be $\Phi(r)$, used to limit oscillations and prevent artificial solutions where high gradients are found.

This transforms (42) into (43):

$$\hat{Q}^{L}_{i+\frac{1}{2}} = \hat{Q}_i + \delta^{L}_{i+\frac{1}{2}}$$

$$\hat{Q}^{R}_{i+\frac{1}{2}} = \hat{Q}_{i+1} - \delta^{R}_{i+\frac{1}{2}} \qquad (43)$$

Then, with $K = 0$,

$$\delta = \frac{(a^2+\varepsilon)\mathrm{b}_i + (b^2+\varepsilon)a}{a^2+b^2+2\varepsilon}$$

$$a_L = \Delta_i \quad , \quad b_L = \nabla_i \quad , \quad a_R = \nabla_{i+1} \quad , \quad b_R = \Delta_{i+1}$$

As the limiter is used to prevent oscillations and artificial solutions at high gradient areas, a small value of $\varepsilon = 0.0008$ is added to the δ formula. Formulation done with, we can move on to graphical and numerical comparisons.

5.1.3.1 Graphical Comparisons for ηk = 0.4

To start with, it must be stated that second order discretizations have a tendency to diverge at lower η_k values. This makes it impossible to *reliably* compare the results of unpreconditioned Newton-GMRES with first order discretization at those values for η_k = 0.4 level. The preconditioned algorithm performs better at that level, enabling some comparisons. As a change of pace to enable easier inspection, the comparisons will be done one by one instead of all-in-one, as in ILU(0) preconditioned for first order discretization being compared to the same for second order discretization only and so on. Without further ado, the results are as follows from Figure 5.44 through 5.52 in terms of CPU times:

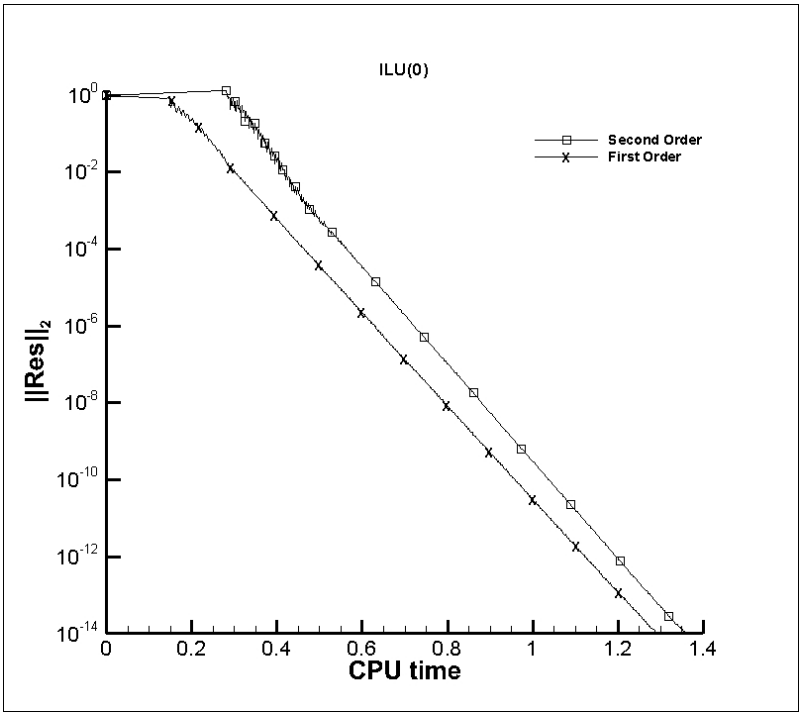

Figure 5.44 CPU Time Comparison for ILU(0) for Coarse Grid

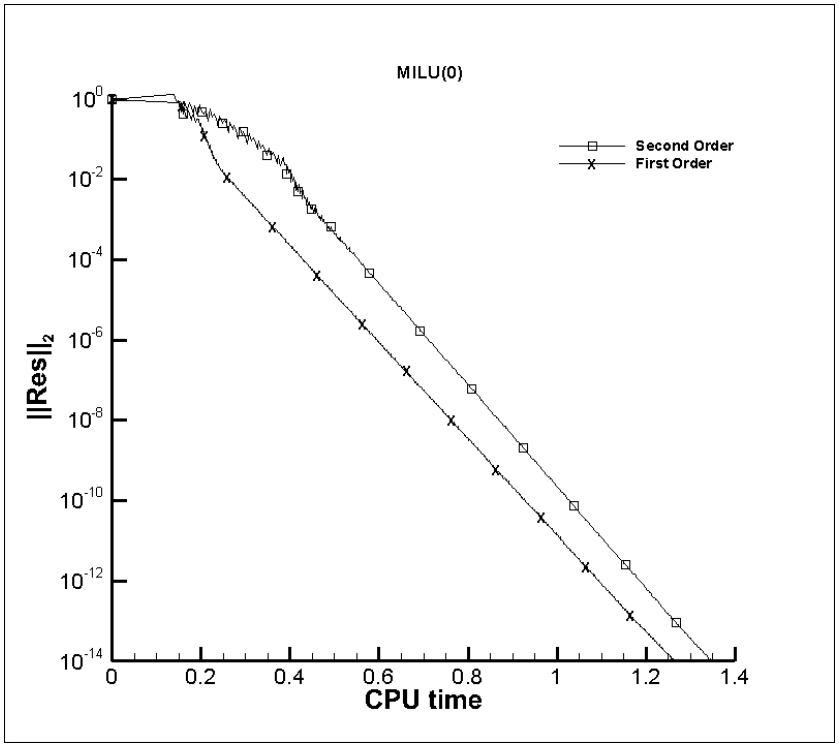

Figure 5.45 CPU Time Comparison for MILU(0) for Coarse Grid

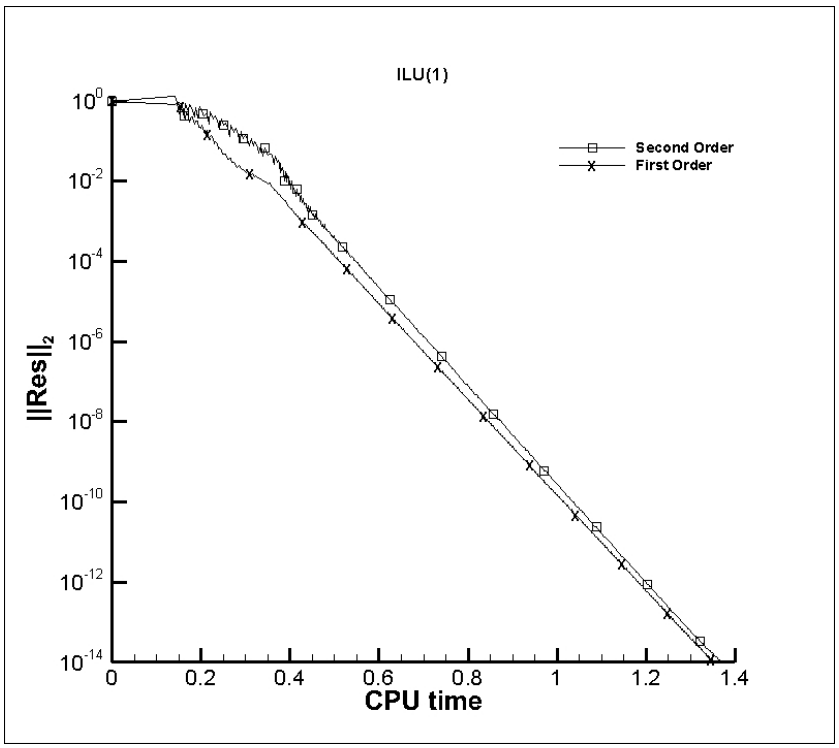

Figure 5.46 CPU Time Comparison for ILU(1) for Coarse Grid

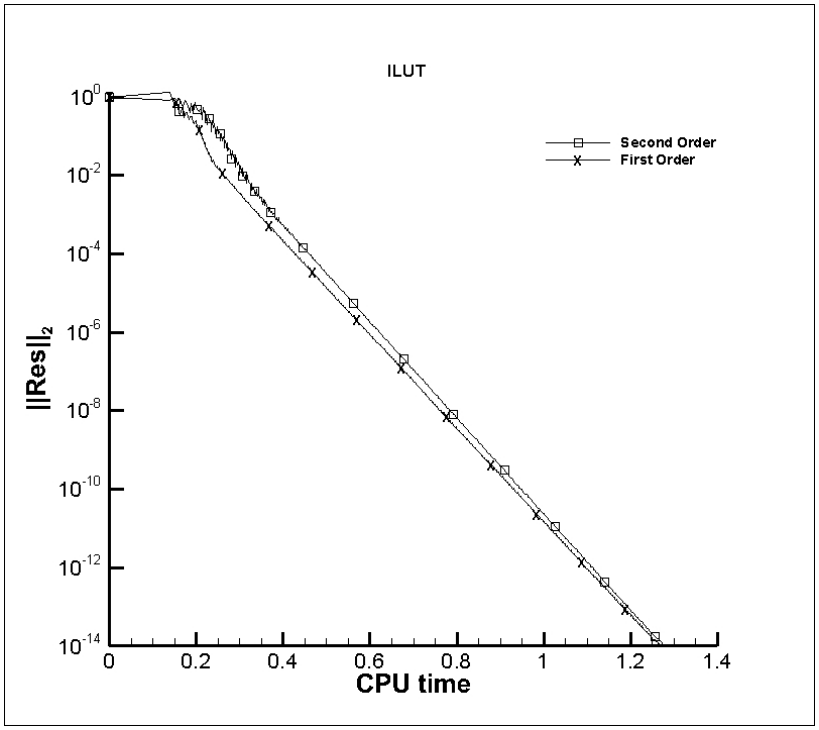

Figure 5.47 CPU Time Comparison for ILUT for Coarse Grid

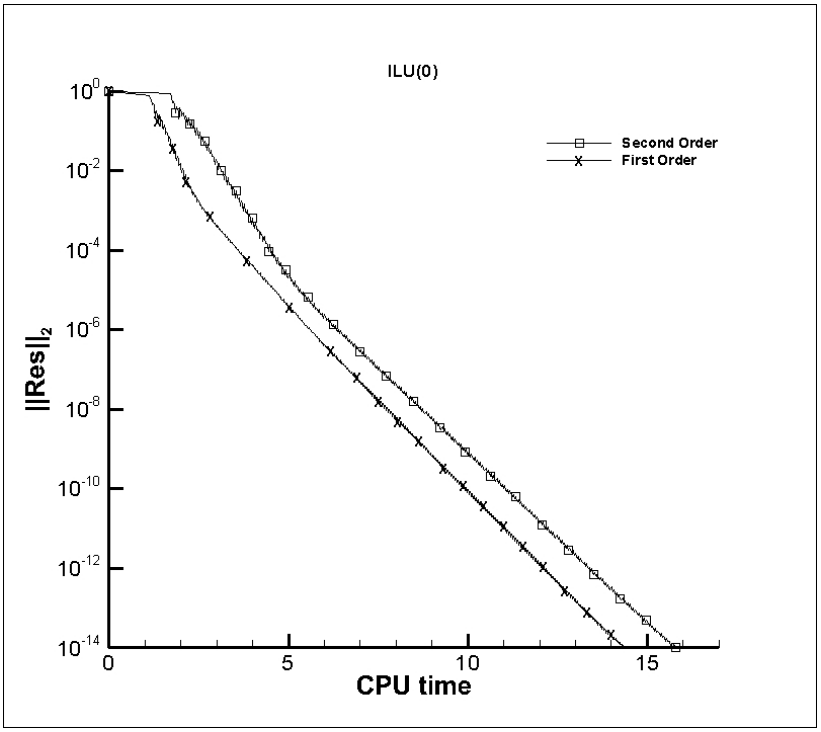

Figure 5.48 CPU Time Comparison for ILU(0) for Medium Grid

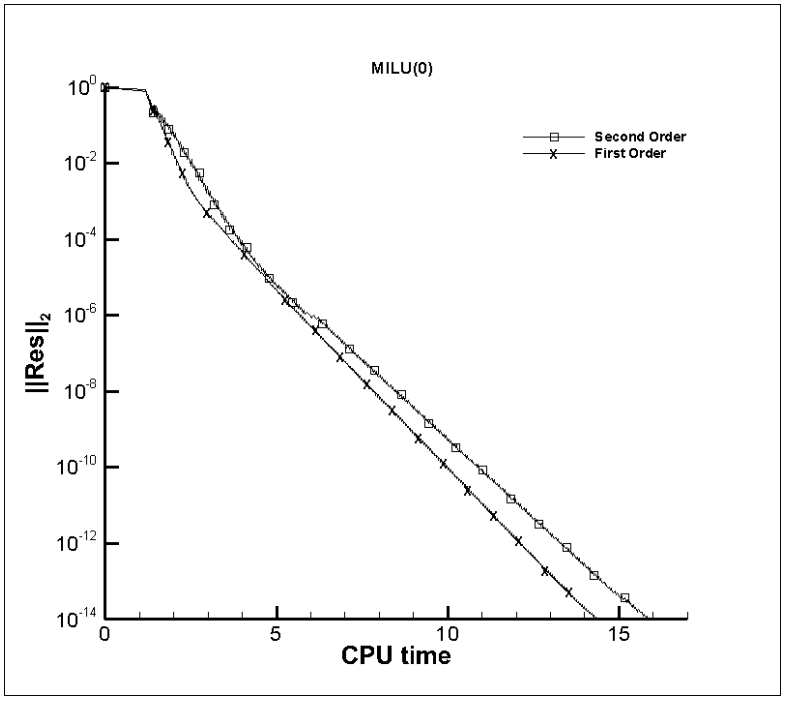

Figure 5.49 CPU Time Comparison for MILU(0) for Medium Grid

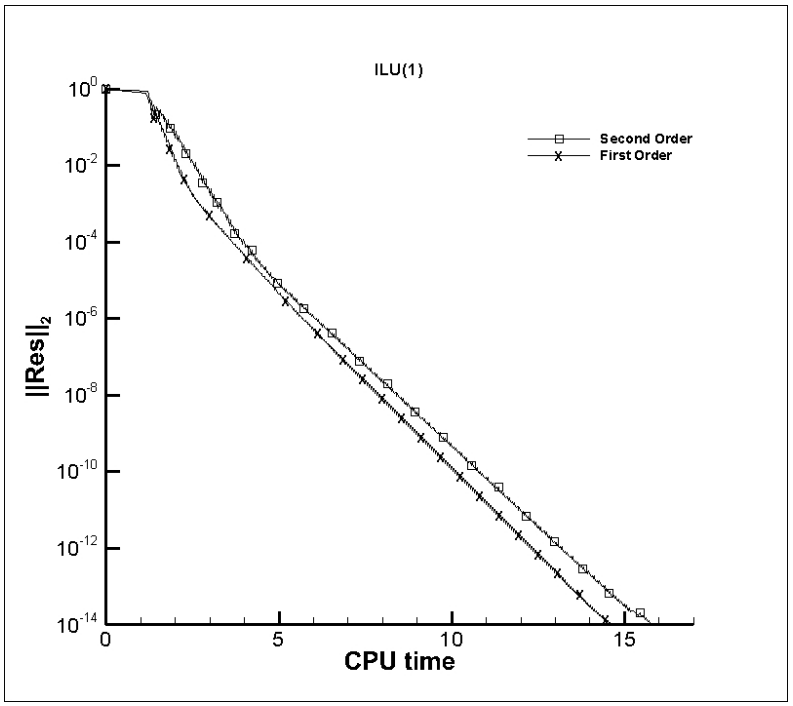

Figure 5.50 CPU Time Comparison for ILU(1) for Medium Grid

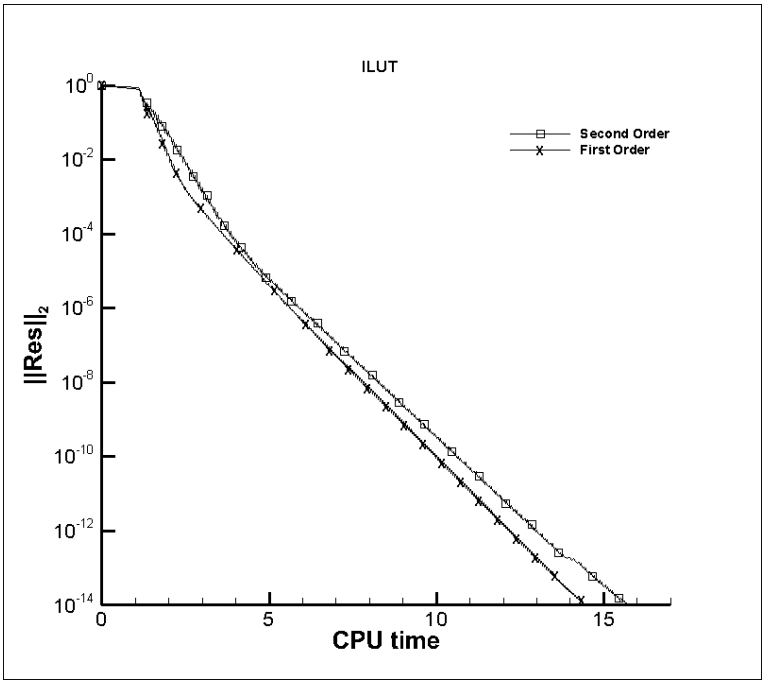

Figure 5.51 CPU Time Comparison for ILUT for Medium Grid

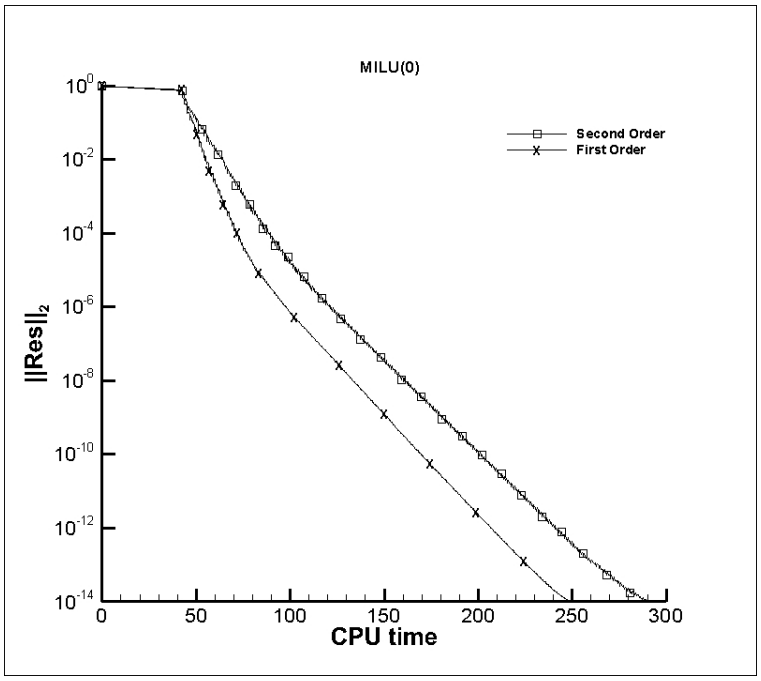

Figure 5.52 CPU Time Comparison for MILU(0) for Fine Grid

As can be seen from Figures 5.44 through 5.52, the second order results are always slower than the first order results. In addition to that, none of the algorithms except for MILU(0) preconditioned algorithm can reliably converge at this level for the fine grid

(that is, not have inconsistent CPU times). The explanation for that is the discarded element conservation property of MILU(0) algorithm. The problem solution obtained with second order discretization are the same with Newton's method and first order discretization.

5.1.3.2 Numerical Comparisons for ηk = 0.4

Here, the CPU times can be inspected more closely in comparison to their first order counterparts in Table 5.8:

Table 5.8 CPU Time Comparisons for 1st and 2nd Order Discretization at $\eta_k = 0.4$

Method	Coarse		Medium		Fine	
	First Order	Second Order	First Order	Second Order	First Order	Second Order
Newton - GMRES	1,76	-	21,51	-	450,12	-
ILU(0) Preconditioned Newton- GMRES	1,31	1,36	14,45	15,83	245,37	-
MILU(0) Preconditioned Newton- GMRES	1,26	1,35	14,4	15,94	246,07	291,01
ILU(1) Preconditioned Newton- GMRES	1,36	1,37	14,65	15,83	239,89	-
ILUT(0.0001,5) Preconditioned Newton- GMRES	1,27	1,28	14,46	15,73	253,01	-

The results are as stated before, the algorithm is always slower when using second order discretization. ILUT generally seems to show the best performance with this method when smaller grids are used; but one should take a look at other forcing factors to draw a more solid conclusion.

5.1.3.3 Graphical Comparisons for ηk = 0.6

To do away with repetitive clutter, only the graphical result for fine grid will be included here. The results for medium and coarse grids mimic the fine grid and the numerical data for all will be included in the next part. At this level of η_k, the algorithm converges for all methods applied.

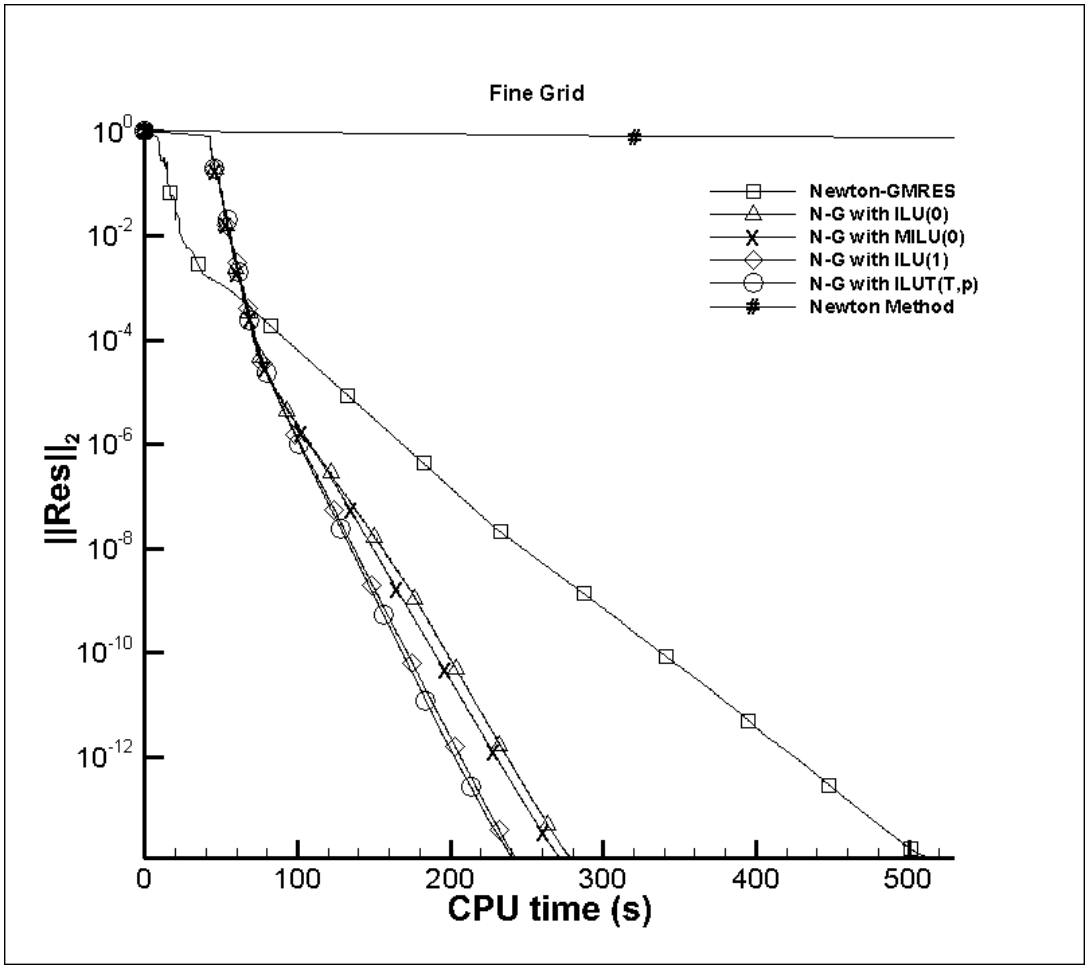

Figure 5.53 CPU Time Comparison of all 2nd Order Discretized Methods for Fine Grid

In Figure 5.53, one can easily see the performance difference preconditioned methods make. Apart from that, ILUT preconditioning approach is seen to be the fastest among its counterparts. The Newton method takes far longer than is appropriate to show on the plot as was with the previous results. To further visually demonstrate the first and second order difference, we can use the ILUT preconditioned algorithm result as follows in the next plot.

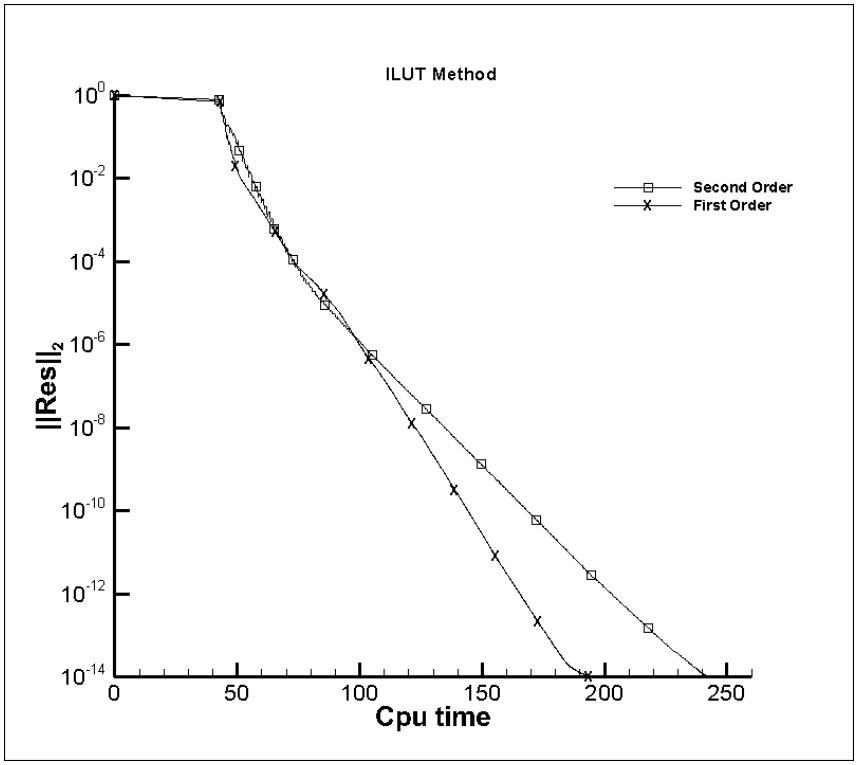

Figure 5.54 CPU Time Comparison of ILUT Methods on Fine Grid for 1st and 2nd Order Discretizations

The CPU time difference can easily be seen in Figure 5.54. A more detailed analysis of the numerical values can be observed in section 5.1.3.4.

5.1.3.4 Numerical Comparisons for ηk = 0.6

Table 5.9 will summarize all differences found between first and second order discretizations.

Table 5.9 CPU Time Comparisons for 1st and 2nd Order Discretization at $\eta_k = 0.6$

Method	Coarse		Medium		Fine	
	First Order	Second Order	First Order	Second Order	First Order	Second Order
Newton - GMRES	1,32	1,76	12,8	21,46	259,11	518,36
ILU(0) Preconditioned Newton- GMRES	1,14	1,37	10,45	14,45	191,51	279,96
MILU(0) Preconditioned Newton- GMRES	1,16	1,27	10,35	14,57	195,61	274,01
ILU(1) Preconditioned Newton- GMRES	1,24	1,28	10,41	15,01	191,49	244,89
ILUT(0.0001,5) Preconditioned Newton- GMRES	1,46	1,32	10,29	14,59	193,31	242,74

In Table 5.9, it can be seen that MILU(0) keeps the trend of being the fastest method in the coarse grid. ILU(0) performs best in the medium grid and ILUT(0.0001,5) gives the best results for the fine grid as opposed to how the preconditioners behaved at $\eta_k = 0.4$. Putting internal comparison aside, one can easily see that first order discretization performs better under all conditions when the residual level is kept the same and the same solutions for the problem are obtained.

This leads into the final section of this thesis, which consists of all the tables of comparison for wall clocks at all forcing factors and orders.

5.1.3.5 Numerical Comparisons for all conditions

In this last section of this chapter, the average Wall Clock times are displayed. The averages are calculated using between 3 and 10 trials (3 for Coarse mesh, 5 for Medium and 10 for Fine mesh cases as the least amount of difference is seen in Coarse and highest amount of difference is seen in Fine meshes). In second order results, the averages for fine mesh at the lowest forcing term are less reliable due to a wider range of results obtained for that case. All results are showcased in the following tables. Best results for each case is grayed.

Table 5.10 Wall Clock Comparisons for All Conditions, Coarse Mesh

Method	Coarse							
	First Order				Second Order			
	$\eta_k = 0.4$	$\eta_k = 0.5$	$\eta_k = 0.6$	$\eta_k = 0.7$	$\eta_k = 0.4$	$\eta_k = 0.5$	$\eta_k = 0.6$	$\eta_k = 0.7$
Newton - GMRES	1,724	1,721	1,752	1,734	219,655	2,313	1,96	2,187
ILU(0) Preconditioned Newton- GMRES	1,306	1,223	1,346	1,471	1,569	1,569	1,921	1,498
MILU(0) Preconditioned Newton- GMRES	1,222	1,25	1,603	1,39	1,539	1,545	1,471	1,75
ILU(1) Preconditioned Newton- GMRES	1,263	1,503	1,442	1,774	1,513	1,657	1,663	1,386
ILUT(0.0001,5) Preconditioned Newton- GMRES	1,398	1,433	1,607	1,482	1,495	1,53	1,674	1,6

Table 5.10 shows results consistent with the expectation that all preconditioned methods should be faster than unpreconditioned versions. It also shows that ILU(0) and MILU(0) are simpler and as such, more efficient methods when applied to simpler problems in comparison to somewhat more complex ILUT and ILU(1) methods. Still, it can be said that the difference is not extreme in most cases. It can be seen here that the best case scenario for unpreconditioned Newton-GMRES for a Coarse mesh is 1,721 seconds at $\eta_k = 0.5$ and first order while the best case scenario for a preconditioned approach is 1,222 seconds with MILU(0) with $\eta_k = 0.4$ and first order. The decrease in time is 0,5 seconds, which is significant enough to legitimize the usage of a preconditioner.

Table 5.11 Wall Clock Comparisons for All Conditions, Medium Mesh

Method	Medium							
	First Order				Second Order			
	$\eta_k = 0.4$	$\eta_k = 0.5$	$\eta_k = 0.6$	$\eta_k = 0.7$	$\eta_k = 0.4$	$\eta_k = 0.5$	$\eta_k = 0.6$	$\eta_k = 0.7$
Newton - GMRES	16,403	15,766	16,279	17,378	754,228	22,938	24,486	22,098
ILU(0) Preconditioned Newton- GMRES	14,177	12,628	12,161	13,047	17,910	15,745	17,172	17,042
MILU(0) Preconditioned Newton- GMRES	13,222	12,509	11,984	13,706	19,736	16,041	15,227	15,737
ILU(1) Preconditioned Newton- GMRES	12,809	11,801	11,536	11,828	16,697	16,911	16,197	16,022
ILUT(0.0001,5) Preconditioned Newton-GMRES	12,406	12,818	12,305	12,894	16,843	15,341	15,113	14,629

Table 5.11 shows the wall clock comparisons for all tested conditions for the medium mesh. As an example, the CPU time for ILUT preconditioned Newton-GMRES for first order and $\eta_k = 0.4$ is 12,221 seconds, which makes the difference between CPU time and wall clock negligible. The preconditioned algorithm is faster than its unpreconditioned counterparts as expected again. A trend can be seen in terms of the faster preconditioning method with ILU(1) and ILUT preconditioners being the fastest with the medium mesh.The fastest convergence achieved with unpreconditioned Newton-GMRES is at $\eta_k = 0.5$ with 15,766 seconds while the fastest convergence with a preconditioned method is at $\eta_k = 0.6$ with 11,536 seconds. The improvement in performance at a level of 33% when the best convergences are compared is satisfactory. Also, one can say that the more complex ILU(1) and ILUT methods are performing better at somewhat larger mesh sizes, but if that conclusion can hold up to further testing for larger meshes is up to debate.

Table 5.12 Wall Clock Comparisons for All Conditions, Fine Mesh

Method	Fine							
	First Order				Second Order			
	$\eta_k = 0.4$	$\eta_k = 0.5$	$\eta_k = 0.6$	$\eta_k = 0.7$	$\eta_k = 0.4$	$\eta_k = 0.5$	$\eta_k = 0.6$	$\eta_k = 0.7$
Newton - GMRES	328,296	440,908	453,77	343,038	738,619	696,399	554,129	463,801
ILU(0) Preconditioned Newton- GMRES	332,763	174,616	167,049	170,183	754,317	275,395	250,051	265,826
MILU(0) Preconditioned Newton- GMRES	176,762	168,392	165,925	173,772	391,593	291,111	271,382	330,077
ILU(1) Preconditioned Newton- GMRES	175,313	167,505	166,733	171,497	893,925	289,566	274,674	427,655
ILUT(0.0001,5) Preconditioned Newton-GMRES	174,091	166,944	166,268	170,568	-	350,389	354,157	279,004

Table 5.12 shows the performances of all methods in terms of wall clocks for fine mesh. Again for comparison, the average CPU time for ILUT preconditioned Newton-GMRES at $\eta_k = 0.4$ for first order is 173,997 seconds, making CPU time and wall clocks be almost equal. Here, the previous assumption that ILU(1) and ILUT works better for more complex meshes can be nullified, as in fact, the best results are obtained using ILU(0) and MILU(0). This brings forward the need to further study different problems to gain a better understanding. Here, the improvement in efficiency is far better, which is to be expected as preconditioning is always better when applied to larger problems in general. The best convergence for unpreconditioned Newton-GMRES is obtained at $\eta_k = 0.4$ for first order with 328,296 seconds while the best convergence for a preconditioned method is at $\eta_k = 0.6$ for first order with MILU(0), which is 165,925 seconds. The improvement in performance is almost 100% with 157 seconds reduction in average wall clock time.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 Conclusion

Throughout this thesis, Newton, Newton-GMRES and preconditioned Newton-GMRES methods have been implemented in the solution of a sample problem of 3-D Euler equations on a supersonic nozzle geometry. The Newton's method made use of Van Leer Upwind scheme and a second order discretization. The calculation of the full Jacobian for the Newton's method was done analytically. UMFPACK was used to solve the system resulting from Newton's method.

The Newton-GMRES method was used next for the solution of the same problem, to see if faster convergence was possible without losing accuracy. Here, satisfactory results with respect to Newton's method were obtained. These results were compared to Newton's method. Different approaches to Newton-GMRES algorithm were tried in terms of the forcing term η_k, non-dimensionalization of the algorithm and ε. The effect of η_k on different approaches is showcased.

The main concern of this thesis was next inspected. The necessity, cost, usage, types and choice of a preconditioner for Newton-GMRES algorithm were explained. A variety of ILU based preconditioners were chosen and inspected. ILU(0), ILU(1), MILU(0) and ILUT(ρ,τ) preconditioners were compared and the level of fill and threshold values were determined for the last preconditioner. The mentioned preconditioners were applied to the same problem as before under the same conditions as Newton's and unpreconditioned Newton-GMRES methods. The results were found to be satisfactory in the speeding of the algorithm, saving CPU time. No accuracy loss was detected with the desired parameters. The effect of the level of fill was determined to be critical in the usage of ILUT(ρ,τ) preconditioner. The preconditioning matrix

used was created only from the block-diagonal terms of the first order Jacobian. The preconditioning matrix M was frozen at the first iteration and used all through the process. A full Jacobian was created to be used as the preconditioning matrix but proved extremely costly with little gain in CPU time, so the idea was discarded. The preconditioned Newton-GMRES turned out to be superior to both Newton's and Newton-GMRES methods in all aspects but one; as the grid size grows larger than those investigated here, the increasing number of variables can cause a memory segmentation fault which could lead to constant undesirable debugging.

In the solution of the problem, all methods have been applied on coarse, medium and fine grids. The residuals of Newton-GMRES and preconditioned Newton-GMRES methods have been converged to 10^{-17} just to prove they can do that while the residual of Newton's Method was converged to 10^{-14} due to the unwieldiness of further convergence (it takes a *large* amount of time). These results have of course been compared at identical convergence points, that point being 10^{-14}. The effect of changing level of fill and the threshold in the preconditioner has been discussed. No samples for the threshold change was given, as the matrix that is being formed only has values of similar orders of magnitude and tightening the threshold for dropping values affects the preconditioning process extremely adversely. Yet, if the same algorithm is applied to a different, wilder problem, this threshold will gain higher importance again. First and second order discretizations were compared for perspective gain. It was found that using different preconditioners for different grid sizes is necessary to obtain best results for each case. It was seen that simpler methods such as ILU(0) and MILU(0) have a general tendency to get the best results. Overall, the results were as expected and satisfactory.

6.2 Future Works

Some of what has been in mind to be implemented in the first few months of this study had to be discarded mostly due to time concerns. The parallelization of the whole solver is one useful idea that can be achieved with much more time. The application of the solver to a variety of different problems is the most important future work that can be done as it enables other researchers to obtain their results faster and further

validates the results of this thesis. While routines like the simple Jacobi have been found wanting in the general analysis of preconditioners, sparse approximate inverse matrix preconditioners could not be tried and is an area of high interest. The update of the preconditioner through a variety of means to further improve it is also a great idea, for which Broyden's "Good" method, restarting GMRES or simply the recalculation of the Jacobian using the previous Jacobian as some sort of basis at every fixed amount of steps can be helpful. Using Navier-Stokes equations instead of Euler equations can lead to more realistic results; but this will also lead to a high amount of processing power necessity. If parallelization efforts are successful, Navier-Stokes equation usage will be more approachable.

An absolutely matrix-free preconditioner would be the end purpose, but that would be art as much as science.

REFERENCES

[1]. Eckert, M. (2006). "The Dawn of Fluid Dynamics: A Discipline Between Science and Technology." Wiley. p. ix. ISBN 3-527-40513-5.

[2]. Arfken, G. (1985) "Mathematical Methods for Physicists", *3rd ed.* Orlando, FL: Academic Press, pp. 963-964

[3]. Knoll, D.A. and Keyes, D.E. (2004) "Jacobian-free Newton-Krylov Methods: A Survey of Approaches and Applications" J. Computat Phys, Vol.193 pp. 357-397

[4]. Saad, Y., and Schultz, M. H. (1986) "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," SIAM Journal on Scientific and Statistical Computing, Vol. 7, No. 3, 1986, pp.856-869.

[5]. Van Leer, B. (1982, September) "Flux Vector Splitting for the Euler Equations", ICASE Report 82-30

[6]. Davis, T. A. (2003) UMFPACK Version 4.1 User Manual, University of Florida, Florida

[7]. Schenk, O. and Gärtner, K. (2004) "Solving Unsymmetric Sparse Systems of Linear Equations with PARDISO", Journal of Future Generation Computer Systems, 20(3):475—487

[8]. Eyi, S., and Muslubaş, Y.E. (2015) "Performances of Newton and Preconditioned Newton-GMRES Methods in Hypersonic Flow Solutions", AIAA Paper

[9]. Yildizlar, B. (2014) "Performance Comparison of Newton and Newton-GMRES Methods in 3-D Flow Analysis" Master thesis, Middle East Technical University

[10]. Venkatakrishnan, V. (July, 1989) "Newton Solution of Inviscid and Viscous Problems", *AIAA Journal*, Vol. 27, pp. 885-891.

[11]. Saad, Y. (1996) "Iterative Methods for Sparse Linear Systems, 1st Edition"

[12]. Saad, Y. and Schultz, M.H. (1986) "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems", *SIAM J. Sci. Stat. Comput.*, 7:856-869, doi:10.1137/0907058

[13]. Greenbaum, A., Ptak, V. and Strakos, Z. (1996) Any nonincreasing convergence curve is possible for GMRES, SIAM J. Matrix Anal. Appl., v 17, pp. 465–469

[14]. Choquet, R. (June, 1995) "A matrix-free preconditioner applied to CFD", Rappport de recherché, Institut National de Recherche en Informatique et en Automatique

[15]. Chen, Y. and Shen, C. (August, 2006) "A Jacobian-Free Newton-GMRES(m) Method with Adaptive Preconditioner and Its Application for Power Flow Calculations", IEEE Transactions on power systems, Vol. 21, No. 3

[16]. Barth, T. J. and Linton, S. W. (1995) "An Unstructured Mesh Newton Solver for Compressible Fluid Flow and Its Parallel Implementation", AIAA Conference Paper 95 0221

[17]. Blanco, M and Zingg, D. W. (1997) "A Fast Solver for the Euler Equations on Unstructured Grids Using a Newton-GMRES Method", AIAA Conference Paper 97-0331

[18]. Delanaye, M., Geuzaine Ph. And Essers, J.A. (1997) "Compressible Flows on Unstructured Adaptive Grids", AIAA Conference Paper, 97-2120

[19]. Householder, A. S. (1953) "Principles of Numerical Analysis" New York: McGraw-Hill, pp. 135-138

[20]. Trefethen, L. N. and Bau, D. (1997) III, "Numerical Linear Algebra" Society for Industrial and Applied Mathematics

[21]. Saad, Y. (1996) "Iterative Methods for Sparse Linear Systems, 1st Edition", p. 297

[22]. Dongarra, J. et al. (1994) "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", pp. 35-45

[23]. Saad, Y., (1994) "SPARSKIT: a basic tool kit for sparse matrix computations," Tech. rep., http://www.cs.umn.edu/ Research/ arpa/ SPARSKIT/ sparskit.html [last accessed on: 10.09.2015]

[24]. Gatsis, J., (2013) "Preconditioning Techniques for a Newton-Krylov Algorithm for the Compressible Navier-Stokes Equations", Doctoral Thesis