A Third Order Accurate Semi-Implicit Runge-Kutta method for the Compressible Navier-Stokes Equations

Safdar Abbas

September 2007
## Contents

1 Introduction 2

2 Governing Equations And Spatial Discretization 5
   2.1 Control Volume .................................................. 5
   2.2 Navier Stokes Equations ........................................ 6
   2.3 Spatial Discretization ........................................... 8
      2.3.1 Central Difference Scheme ................................. 9
      2.3.2 Upwind Scheme ........................................... 10

3 Time Stepping for Steady Cases 12
   3.1 Explicit Time Stepping .......................................... 12
   3.2 Implicit Time Stepping .......................................... 13
      3.2.1 Lower-Upper Symmetric Gauss-Seidel (LU-SGS) ........ 15
      3.2.2 Rossow-operator with SGS Iterations .................... 17
   3.3 Numerical Examples ............................................ 19
      3.3.1 Steady Flow over 2D RAE 2822 case 9 Configuration  
       (Viscous Flow) ............................................... 20

4 Time Stepping for Unsteady Cases 36
   4.1 Dual Time Stepping .............................................. 36
   4.2 Unconstrained Implicit Runge-Kutta (UIRK) method .......... 37
   4.3 Numerical Examples ............................................. 39
      4.3.1 An Expansion Fan in a Shock tube on a Structured  
       Grid (Inviscid Flow) ........................................ 40
      4.3.2 Separating Vortices From a Blunt Body (Viscous Flow) 44

5 Conclusions 52
List of Figures

3.1 Computational Mesh for the 2D RAE 2822 case 9 Airfoil . . . 21
3.2 Solution obtained after residual reduction of $10^{-5}$ for the 2D
RAE 2822 case 9 Airfoil . . . . . . . . . . . . . . . . . . . . . . 21
3.3 Residual reduction for the most efficient schemes in terms of
the CPU time . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
3.4 Convergence of the lift coefficient . . . . . . . . . . . . . . . 23
3.5 Convergence of the drag coefficient . . . . . . . . . . . . . . 24
3.6 Effect of number of SGS iterations on CPU time . . . . . . . 27
3.7 Effect of number of SGS iterations on residual reduction for
BE-Rossow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
3.8 Effect of number of SGS iterations on residual reduction for
RK-Rossow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
3.9 Effect of fine grid CFL number on CPU time . . . . . . . . . 30
3.10 Effect of number of fine grid CFL number on residual reduction
for BE-Rossow . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
3.11 Effect of number of fine grid CFL number on residual reduction
for RK-Rossow . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
3.12 Effect of number of multigrid levels on CPU time . . . . . . . 33
3.13 Effect of number of multigrid levels on residual reduction for
BE-Rossow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
3.14 Effect of number of multigrid levels on residual reduction for
RK-Rossow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
4.1 Part of the structured shock tube grid. Cell aspect ratio
reaches 1 : 100 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
4.2 Normalized pressure at $time = 0.003sec$ . . . . . . . . . . . . 41
4.3 Error in the solution verses time-step . . . . . . . . . . . . . . 42
4.4 Error in the solution verses CPU-time . . . . . . . . . . . . . . 43
4.5 Part of the far-field mesh for the blunt body . . . . . . . . . . 45
4.6 Solution obtained at $time = 3.0sec$ . . . . . . . . . . . . . . 45
LIST OF FIGURES

4.7 Oscillations of the lift coefficient $time = 3.0sec$ . . . . . . . . . 46
4.8 Amplitude of the lift coefficient versus time-steps $\delta t$ . . . . . . 47
4.9 Frequency of the lift coefficient versus time-steps $\delta t$ . . . . . . 48
4.10 Amplitude of the lift coefficient versus CPU-time . . . . . . . . 49
4.11 Frequency of the lift coefficient versus CPU-time . . . . . . . . 50
Abstract

For the computation of unsteady flows, the efficiency and stability of time-stepping methods have the utmost importance. Efficiency is measured in terms of the CPU time taken to achieve a given accuracy and stability is measured in terms of the largest absolute time-step for which the scheme is stable. In this piece of work, efficiency and the stability of different operators for a semi-implicit Runge-Kutta scheme are investigated and compared to the efficiency and stability of an explicit Runge-Kutta method and the dual time method for unsteady flow. Implicit operators which are compared include the LU-SGS operator involving an approximation of the flux-Jacobian (LU-SGS operator) and an operator wherein the flux-Jacobian is stored in primitive variables and the flux is based on the Roe-averaged variables (Rossow operator). Before applying these operators to unsteady cases, efficiency and stability is investigated for a steady case by applying the operators as the linear system solver with the backward Euler method. Stability in the steady case is measured in terms of the maximum stable fine grid CFL number. The results have shown that the Rossow operator has a clear advantage in both the steady and the unsteady cases over all other schemes examined.
Acknowledgement

The task of writing this thesis at the Institute of Aerodynamics and Flow Technology was a challenging as well as exciting opportunity for me. I would like to thank Prof. Dr. Norbert Kroll and my supervisor Dr. Richard Dwight for first of all proposing this topic and then for their continued support during the course of this thesis. Specially Dr. Dwight has been a very friendly and his way of explanation has been quite impressive. This would have been impossible without some of the most interesting lectures delivered by Prof. Rossow at TU Braunschweig which created and enhanced my interest in CFD. His time to time guidance was also crucial to this work.

All of this would not be possible without the continued support of my wife Sara without whose support I would not have completed the challenging course work of CSE and to whom I owe so much of happiness during the last two years. And I would like to dedicate all of this to my mother and father who created the lust of knowledge in my heart and who have always been praying for me.
Chapter 1

Introduction

In the field of Computational Fluid Dynamics the requirement for increased efficiency and stability of numerical algorithms is ever increasing. Recent efforts for that purpose have concentrated on the steady flow computations using the techniques such as Full Approximate Storage (FAS) Multigrid and approximate Newton solvers.

For the computation of the time accurate unsteady flows, efficiency and stability are even more critical. For explicit schemes the maximum stable time-step is very small due to the CFL condition thus requiring a lot more time for the computations. For fully implicit methods the work required to solve the linear system of equations at each integration step may offset any advantage achieved by unconditional stability of the method. The most successful implicit technique up to now is the dual time algorithm wherein an implicit \textit{Backward Difference Formula} (BDF) is solved by writing the solution at each time-step as a steady state problem, and solving this by using existing time-inaccurate method [3] for stationary problems. Thus two loops for the time-stepping algorithm are required, an outer loop where the actual time-stepping is done and the inner loop where the steady state problem is solved by using pseudo time-steps within each real time-step. Here it must be ensured that sufficient convergence is achieved within each real time-step by the inner iterations. This results in an order-of-magnitude higher cost for unsteady computations over steady computations. Thus some new methods for unsteady computation which are more efficient than and as stable as the dual time algorithm are required.

One of the candidates in this regard are the semi implicit Runge-Kutta methods presented by Nikitin [16]. He applied this method to compute the incompressible flow inside a driven cavity. This scheme is based upon a specially chosen three stage explicit Runge-Kutta method with some special
properties that ensure that addition of an arbitrary implicit term at each stage of the method does not disturb the order of the method. Thus the implicit term may be added based upon the stability requirements alone without regard for the time accuracy. Thus the most critical choice in the efficient implementation of the semi-implicit Runge-Kutta method is the choice of the implicit operator. All types of approximate Newton solvers available for the steady flow solver can be used as the implicit operator in the semi implicit Runge-Kutta scheme.

In this work two specific operators are investigated. The first one is a highly optimized version of the Lower-Upper Symmetric Gauss-Seidel (LU-SGS) algorithm [11]. This involves a first order approximation of the Jacobian, constructed on-the-fly, whereby all terms in the spatial discretization, including boundary conditions are present. One sweep of symmetric Gauss-Seidel method is applied to give a very approximate solution of the linear system. The use of this operator in SIRK results in a method more stable than explicit methods and cheaper per time-step than the dual time method. Stability of the method is not as good as for dual time as it is shown in the viscous cases the method is stable for the maximum fine grid CFL number of 5.

The other operator used is that presented by Rossow [9] for the convergence acceleration of steady state problems. It involves splitting the flux-Jacobian into a positive and a negative part. The splitting is achieved by adding and subtracting $|J_n|$ from $J_n$. Where $|J_n|$ is a flux-Jacobian calculated from the Roe-averaged values of the flow variables on the cell-face and $J_n$ is the flux-Jacobian calculated from the reconstructed values of the flow variables on the cell-face. The flux-Jacobians are stored in a memory efficient manner by converting them to primitive variables and expressing the dissipative part of the fluxes as a function of the cell interface Mach number. Only the Roe-averaged velocities and the cell interface Mach number are precomputed and stored for each cell face; all other components can be recomputed efficiently whenever necessary. Thus memory requirements are reduced considerably without approximation of the flux-Jacobian. This operator is referred to as the Rossow operator in this report as it uses the Roe-averaged values to compute the Jacobian.

First of all the performance of aforementioned implicit operators was compared for a steady case where the operators were used as the approximate Newton solvers for the steady cases with different multigrid levels and different number of SGS iterations. The results were compared to those obtained by the explicit Runge-Kutta method and explicit Runge-Kutta method with the residual smoothing using the Rossow operator. This comparison showed that the greatest efficiency can be achieved by using the Rossow operator as
CHAPTER 1. INTRODUCTION

a residual smoother.

Then the two operators were applied for unsteady computations using the third order accurate semi-implicit Runge-Kutta method and the results were compared to the time-stepping by dual time method. The results show that dual time has an edge in terms of stability over the LU-SGS operator while the Rossow operator with the semi-implicit scheme was as stable as the dual time algorithm. For efficiency in terms of the CPU time, dual time is more time expensive than the third order accurate semi-implicit Runge-Kutta method with both the operators. The Rossow operator out performs the LU-SGS operator by a factor of 10 to achieve a given accuracy.

This document is organized as follows: the governing equations are described in Chapter 2, wherein the finite volume spatial discretization is described in Section 2.3 including the central difference schemes and the upwind schemes. Then time stepping methods for steady flows are described in Chapter 3 wherein the explicit and the implicit time-stepping is explained with the approximate Newton solvers such as LU-SGS in Section 3.2.1 and the Rossow operator in Section 3.2.2 followed by the numerical examples in Section 3.3. After that the time-stepping schemes for the unsteady cases are described in Chapter 4 with the description of the dual time in Section 4.1 and the Unconstrained semi implicit Runge-Kutta method in Section 4.2 followed by the numerical examples in Section 4.3. That is followed by the conclusions in Chapter 5.
Chapter 2

Governing Equations And Spatial Discretization

The governing principle of the Navier-Stokes Equations is the conservation of the following three quantities:

- Conservation of mass.
- Conservation of momentum.
- Conservation of energy.

To apply these principles the notion of a control volume is used.

2.1 Control Volume

A control volume $V$ is an imaginary volume drawn in a finite region of the flow [14]. Control surface $S$ is the surface that bounds the control volume. The control volume can be fixed in space or can be moving with the flow. In the latter case the contents of the control volume are constant while in the previous case the contents can change while the flow is moving through the control volume. Fluid flow equations can be obtained by applying the physical principles to the fluid inside the control volume. So instead of looking at the whole flow field we concentrate on the finite control volume. Applying the conservation principles to the fluid inside the control volume we get the integral form of the equations [14]. If the control volume is fixed in space then the form of the equations are called the Eulerian form of the equations.
If the control volume is moving then this is the Lagrangian form of the equations. Here we shall be dealing with only the conservation form of the governing equations.

### 2.2 Navier Stokes Equations

Applying the three conservation principles on the fluid inside a control volume that is fixed in time and space results in the integral equations called the *Navier-Stokes equations*. The general form of balancing a physical quantity is given as following:

\[
\iint_V \frac{\partial \rho z}{\partial t} \, dV + \iint_S \rho z (\mathbf{q} \cdot \mathbf{n}) \, dS = \iiint_V \rho z F \, dV + \iiint_V \rho z P \, dV.
\] (2.1)

Where \( z \) is a mass specific quantity, \( \rho \) is the density, \( \mathbf{q} \) is the velocity vector and \( \mathbf{n} \) is the vector normal to the surface \( S \). So \( (\mathbf{q} \cdot \mathbf{n}) \) represents the component of the velocity vector \( \mathbf{q} \) normal to the surface \( S \). \( z F \) represents the effect of the field forces on the quantity \( z \) e.g. gravity and electromagnetic effects and \( z P \) represents the internal production rate of the quantity \( z \) due to internal processes e.g. production of a substance due to a chemical reaction. The first volume integral on the left hand side represents the temporal change of the quantity \( z \) inside the control volume \( V \) and the second surface integral represents the flow of the quantity \( z \) through the control surface \( S \). In the case of Aerodynamic flows the effect of \( z F \) and \( z P \) can be neglected. So (2.1) can be written as:

\[
- \iint_S \rho z \mathbf{q} \cdot \mathbf{n} \, dS = \iiint_V \frac{\partial \rho z}{\partial t} \, dV.
\] (2.2)

So the basic principle of balancing a physical quantity according to (2.2) is following:

\[
\text{Net flow of the quantity } z \text{ out of the control volume through surface } S = \text{ time rate of decrease the quantity } z \text{ inside the control volume } V.\]

When the physical quantity is the mass then (2.1) can be written as:

\[
\iiint_V \frac{\partial \rho}{\partial t} \, dV + \iint_S \rho (\mathbf{q} \cdot \mathbf{n}) \, dS = 0,
\] (2.3)

and it is called the *balance of mass equation*. 
When the physical quantity is the momentum then (2.1) can be written as:
\[
\int\int\int_V \frac{\partial \rho q}{\partial t} \, dV + \int_S \rho q \cdot (q \cdot n) \, dS + \int_S p \, n \, dS = \int_S \vec{\tau} \cdot n \, dS, \tag{2.4}
\]
and it is called the balance of momentum equation. Where \( p \) is pressure and \( \vec{\tau} \) is the viscous stress tensor. Volume integrals with these two terms represent the effect of pressure and the viscous stresses on the surface flux of momentum.

When the physical quantity is the energy then (2.1) can be written as:
\[
\int\int\int_V \frac{\partial \rho E}{\partial t} \, dV + \int_S \rho H \cdot (q \cdot n) \, dS = \int_S (\vec{\tau} \cdot n) \cdot q \, dS + \int_S k \cdot n \, dS, \tag{2.5}
\]
and it is called the balance of energy equation. Where \( E \) is the mass specific total energy, \( H \) is the total specific enthalpy and \( k \) is the vector of diffusive heat flux. The second surface integral on the right hand side represents the heat energy flow through the diffusive process. Pressure is calculated through the equation of state:
\[
p = (\gamma - 1)\rho \left( E - \frac{q_x^2 + q_y^2 + q_z^2}{2} \right), \tag{2.6}
\]

(2.3), (2.4) and (2.5) can be combined in the following form:
\[
\int\int\int_V \frac{\partial W}{\partial t} \, dV + \int_S \vec{F} \cdot n \, dS = 0, \tag{2.7}
\]
where \( W = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix} \) is the Vector of conservative variables. (2.8)

The flux density tensor \( \vec{F} \) is composed of the fluxes in the three coordinate directions i.e. \( \vec{F}_x, \vec{F}_y \) and \( \vec{F}_z \). Each of these flux density vectors are divided in viscous and inviscid parts which are shown with the superscript \( v \) and \( i \) respectively.

\[
\begin{align*}
\vec{F}_x^i &= \begin{pmatrix} \rho u \\ \rho u^2 + p_x \\ \rho w \\ \rho w \end{pmatrix} \\
\begin{pmatrix} \rho u \\ \rho u^2 + p_x \\ \rho w \\ \rho H u \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
\vec{F}_x^v &= \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ \tau_{yy} + \nu \tau_{xy} + \omega \tau_{xz} + \kappa \frac{\partial T}{\partial x} \end{pmatrix}
\end{align*}
\]
CHAPTER 2. GOVERNING EQUATIONS AND SPATIAL DISCRETIZATION

\[
F_y^i = \begin{pmatrix}
\rho v \\
\rho vu \\
\rho v^2 + p_y \\
\rho Hv
\end{pmatrix}
\quad \text{And} \quad
F_y^v = \begin{pmatrix}
0 \\
\tau_{yx} \\
\tau_{yy} \\
u\tau_{yx} + v\tau_{yy} + w\tau_{yz} + \kappa_l \frac{\partial T}{\partial y}
\end{pmatrix}
\quad (2.9)
\]

\[
F_z^i = \begin{pmatrix}
\rho w \\
\rho wu \\
\rho wv \\
\rho w^2 + p_z \\
\rho Hw
\end{pmatrix}
\quad \text{And} \quad
F_z^v = \begin{pmatrix}
0 \\
\tau_{zx} \\
\tau_{zy} \\
u\tau_{zx} + v\tau_{zy} + w\tau_{zz} + \kappa_l \frac{\partial T}{\partial z}
\end{pmatrix}
\]

Considering \( W \) be constant inside the control volume and the control volume \( V \) is not moving then from (2.7) the temporal change of the conservative variables can be derived as:

\[
\frac{\partial}{\partial t} W = - \oint_S \vec{F} \cdot \vec{n} dS \oint_V dV.
\quad (2.10)
\]

This equation means that the change of the flow variable in a control volume \( V \) is equal to the flux over the control volume surface \( S \) relative to the size of the control volume. If the control volume is fixed in space and time then (2.10) can be written as:

\[
\frac{d}{dt} W = - \frac{1}{V} \cdot Q^F,
\quad (2.11)
\]

where \( Q^F \) represents the fluxes over the boundaries of the control volume. If the boundary consists of \( n \) number of faces then \( Q^F \) can be written as:

\[
Q^F = \sum_{i=1}^{n} Q_i^F = \sum_{i=1}^{n} (Q_i^{F,c} - D_i),
\]

where \( Q_i^{F,c} \) represents the inviscid fluxes over the respective face and \( D_i \) represents the artificial dissipation terms. For upwind schemes the dissipation terms are zero as the dissipation is incorporated in the scheme.

2.3 Spatial Discretization

The spatial discretization of convective fluxes can be achieved by using two different types of schemes namely:
2.3.1 Central Difference Scheme

The values of the conservative variables are calculated at the cell center. Considering $W_i$ and $W_j$ represents the cell center values in the neighboring cells $i$ and $j$. The average flux is given as:

$$F_{ij}^c = \frac{1}{2}(F_i^c + F_j^c). \quad (2.12)$$

This is well known that this scheme is unstable [11]. In order to fix this problem some artificial dissipative terms (artificial viscosity) are added to the flux. This artificial dissipation term is represented by the 2nd or 4th undivided differences. Where the term undivided difference means a standard finite difference approximation of a partial derivative without the denominator. The 2nd order undivided differences of the cell centered conservative variables is represented as follows:

$$W_{i+1} - 2W_i + W_{i-1} \approx \Delta x^2 \frac{\partial^2 W}{\partial x^2}. \quad (2.13)$$

Adding the artificial viscosity in 2.12 results in the following form of the equation:

$$F_{ij}^c = \frac{1}{2}(F_i^c + F_j^c) - \frac{1}{2} \tilde{\alpha} \left[ (\epsilon(2)W_j - W_i) - (\epsilon(4)L_j(W) - L_i(W)) \right] \quad (2.14)$$

where $\tilde{\alpha}$ describes the scaling of dissipation either scalar dissipation according to [10, Mavriplis] or matrix dissipation according to [12, Turkel]. The coefficients $\epsilon(2)$ and $\epsilon(4)$ contain shock switches for the second and fourth order dissipation, and control the amount of dissipation to be added. The differences $L(W)$ are calculated as:

$$L_i(W) = \sum_{j \in N(i)} W_j - W_i, \quad (2.15)$$

where $N(i)$ represents all the neighboring cells of the cell $i$. On shocks only the first order scheme is monotone so only the second differences are activated near shocks. When a shock is nearing, strong pressure gradients are experienced which are sensed by a pressure sensor and the second differences are activated making the scheme overall of first order. Otherwise fourth differences are activated making the scheme overall of 2nd order.
2.3.2 Upwind Scheme

An alternative for calculating the inviscid fluxes over a face of a control volume, is to solve a Riemann problem for that face. The exact solution of the Riemann problem is too expensive to be used for practical purposes. For that reason approximate Riemann solvers are used. There are two different types of approximate Riemann solvers.

- Flux Difference Splitting.
- Flux Vector Splitting.

In Flux Difference Splitting schemes the flux on the cell face is divided into an average part plus a dissipative term. Whereas in Flux Vector Splitting, the flux vector is split into the positive and negative fluxes at the cell-face (Van Leer [6]).

In the current work the Roe [17] type Flux Difference splitting is used. The Roe method computes the inviscid flux as:

\[ Q^{F,c} = \frac{1}{2}(F^c_L + F^c_R) - \frac{1}{2} \Delta F \]  \hspace{1cm} (2.16)

where \( \Delta F \) represents the Flux Difference and is computed as:

\[
\Delta F = \Delta F^+ + \Delta F^-,
\]

\[
\Delta F^+ = \sum_{n=1}^{5} \lambda^+_n r_n \delta w_n, \hspace{1cm} (2.17)
\]

\[
\Delta F^- = \sum_{n=1}^{5} \lambda^-_n r_n \delta w_n
\]

The \( \lambda_n \) are the eigenvalues with (+) indicating the positive and (−) indicating the negative eigenvalues. The \( r_n \) are the right eigenvectors representing the direction of propagation of the waves. \( \delta w_n \) are the Riemann invariants representing the strength of the waves,

\[
\delta w_1 = \delta \rho - \frac{\delta p}{c^2},
\]

\[
\delta w_2 = n_1 \delta w - n_3 \delta u,
\]

\[
\delta w_3 = n_2 \delta u - n_1 \delta v,
\]

\[
\delta w_4 = \frac{\delta p}{\rho c} + n_1 \delta u + n_2 \delta v + n_3 \delta w,
\]

\[
\delta w_5 = \frac{\delta p}{\rho c} - n_1 \delta u + n_2 \delta v + n_3 \delta w,
\]
where the differences are calculated as:

\[ \delta \rho = \rho_R - \rho_L, \]
\[ \delta p = p_R - p_L, \]
\[ \delta u = u_R - u_L, \]
\[ \delta v = v_R - v_L, \]
\[ \delta w = w_R - w_L. \]

Flow properties at the face are calculated using \textit{Roe-averaging} as follows:

\[ \rho^2 = \rho_R \rho_L, \]
\[ u = \frac{\sqrt{\rho_L} u_L + \sqrt{\rho_R} u_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}. \]

Similarly \( v, w, p \) and \( H \) are calculated.
Chapter 3

Time Stepping for Steady Cases

The temporal variation of the flow quantities can be written in general form for a point \( i \) from (2.11):

\[
\frac{d}{dt} W_i + \frac{1}{V} \sum_{\text{all faces}} F_n S = 0, \tag{3.1}
\]

where \( F_n \) corresponds to the flux density vector normal to the cell face, \( S \) is the area of the cell face and \( V \) is the volume of the computational cell. Equation (3.1) can be written in the following form:

\[
\frac{dW_i}{dt} = -R_i(W), \tag{3.2}
\]

where \( R_i(W) = \frac{1}{V} \sum_{\text{all faces}} F_n S \) represents the conservative residuals of continuity, momentum and energy equations.

3.1 Explicit Time Stepping

The time integration of (3.1) can be performed using an explicit Runge-Kutta scheme, as described by Jameson[2]. Let \( W^n \) be the value after \( n \) time-steps. A general \( m \)-stage scheme to advance a time-step \( \Delta t \) can be written as:
\[
\begin{align*}
W^{(0)} &= W^n \\
W^{(1)} &= W^{(0)} - \alpha_1 \delta t R^{(0)} \\
&\vdots \\
W^{(m-1)} &= W^{(0)} - \alpha_{m-1} \delta t R^{(m-2)} \\
W^{(m)} &= W^{(0)} - \alpha_m \delta t R^{(m-1)} \\
W^{n+1} &= W^{(m)}
\end{align*}
\] (3.3)

where \( \alpha \) is the Runge-Kutta stage coefficient and residual in each stage is computed from the variables of the previous stage e.g. residual in stage \( m-1 \) is \( R^{(m-2)} = R(W^{(m-2)}) \). The residual is updated after each stage using the latest values of the conservative variables. The time step of an explicit method is limited by the stability criterion set by the CFL condition [4]. To increase the robustness of such schemes, various acceleration techniques are used to accelerate the convergence of the solution toward the steady state.

e.g.

- Local Time Stepping [5]
- Residual Smoothing [15]
- Multigrid [1]

### 3.2 Implicit Time Stepping

The change in the conservative variables \( \delta W \) from time \( n \) to the time \( n + 1 \) can be expressed as:

\[
\frac{dW_i}{dt} = \frac{W_i^{n+1} - W_i^n}{\delta t}.
\] (3.4)

Implicit formulation of (3.2) can be written as:

\[
\frac{dW_i}{dt} = -R^{n+1}(W_i),
\] (3.5)

Equating (3.4) and (3.5) gives:

\[
W_i^{n+1} = W_i^n - \delta t R^{n+1}(W_i).
\] (3.6)

This gives an implicit scheme for the time-stepping called the backward Euler method. In order to find the value of \( W_i^{n+1} \) from (3.5), an iterative solution
method must be used e.g. Gauss-Seidel or Jacobi iterations. The implicit formulation of (3.1) can be written as:

$$\frac{d}{dt} W_i + \frac{1}{V} \sum_{\text{all faces}} F_{n+1}^{n+1} S = 0,$$

(3.7)

where $F_{n+1}^{n+1}$ is evaluated at the new time level $n + 1$. Linearizing the $F_{n}^{n+1}$ about the current time level $n$, one obtains

$$F_{n}^{n+1} = F_{n}^{n} + \left( \frac{\partial F_{n}}{\partial W} \right) \delta W = F_{n}^{n} + J_{n} \delta W,$$

(3.8)

where $\delta W$ is the time difference of the conservative variables and $J_{n}$ is the flux-Jacobian in the direction normal to the cell face $S$. Putting (3.8) into (3.7) one obtains

$$\frac{\delta W}{\delta t} + \frac{1}{V} \sum_{\text{all faces}} J_{n} \delta W S = -\frac{1}{V} \sum_{\text{all faces}} F_{n}^{n} S.$$

(3.9)

After rearranging which leads to

$$\left( I + \frac{\delta t}{V} \sum_{\text{all faces}} J_{n} S \right) \delta W = -\frac{\delta t}{V} \sum_{\text{all faces}} F_{n}^{n} S = \delta t R^{n}$$

(3.10)

$$\left( A \right) \delta W = \delta t R^{n}$$

(3.11)

this encapsulates the implicit scheme and $A$ represents the implicit system matrix. This scheme consists of constructing the left hand side which includes constructing the Jacobian $J$. The right hand side is obtained by the spatial discretization. Then the equation system is solved to find the change in the conservative variables $W$.

If one constructs and stores the exact Jacobian then it is desirable to solve the resulting system of equations exactly using Newton method. Otherwise the effort to construct the exact Jacobian will be wasted if large errors are incurred by solving the equations approximately [11]. In order to construct a scheme of low memory requirements one has to resort to several approximations of the Jacobian. Thus there is no point in solving an approximate Jacobian by using very exact methods. So the low powered linear solvers like Symmetric Gauss-Seidel (SGS) and Lower Upper Symmetric Gauss-Seidel (LU-SGS) (explained in Section 3.2.1) methods are the candidates for solving such system of equations.
3.2.1 Lower-Upper Symmetric Gauss-Seidel (LU-SGS)

The Gauss-Seidel method can be explained as follows: suppose a non-singular linear system \( A \cdot x = b \), where \( A \) is implicit system matrix. \( A \) can be decomposed like follows:

\[
A \cdot x = (L + D + U) \cdot x = b,  \tag{3.12}
\]

where \( L, U \) and \( D \) are the strictly lower triangular, upper triangular and diagonal matrices. In (4.20)-(4.23) the term \((I + \gamma L \delta t / V)\) represents the implicit system matrix \( A \), the unknown vectors of conservative variables like \( W' \) represents \( x \) and the right hand side represents \( b \). The two possible Gauss-Seidel iterations with unknown \( x^n \) are

\[
(D + L)x^{n+1} = b - U \cdot x^n, \quad \text{and} \quad (D + U)x^{n+1} = b - L \cdot x^n,  \tag{3.13}
\]

if either of these iterations converge then the solution of the linear system is \( x^{n+1} = x^n = x \). The iteration where \( x^{n+1}_i \) is a function of all \( x^n \) as well as \( x^{n+1}_j \) for \( j < i \) is the forward sweep and the iteration where \( x^{n+1}_i \) is a function of all \( x^n \) as well as \( x^{n+1}_j \) for \( j > i \) is the backward sweep. In order to automatically satisfy the CFL condition a composite of these two iterations can be written as

\[
(D + L)x^* = b - U \cdot x^n, \\
(D + U)x^{n+1} = b - L \cdot x^*,  \tag{3.14}
\]

This is called the Symmetric Gauss-Seidel Sweep (SGS). By restricting the starting value of SGS iterations as \( x^0 = 0 \) one gets the following single iteration scheme

\[
(D + L)x^* = b, \\
(D + U)x^1 = D \cdot x^*,  \tag{3.15}
\]

which may be written as

\[
(D + L) \cdot D^{-1} \cdot (D + U)x^1 = b  \tag{3.16}
\]

which is so called \( LU-SGS \) iteration. Equation (3.16) specifies an alternate interpretation of the method, as the system \( A \cdot x = b \) have been replaced by \( \bar{A} \cdot x = b \). The two systems are related by

\[
A = (L + D + U) \\
= (D + L) \cdot D^{-1} \cdot (D + U) - L \cdot D^{-1} \cdot U \\
= \bar{A} - L \cdot D^{-1} \cdot U,  \tag{3.17}
\]
i.e. approximate factorization of $A$ with an error of $L \cdot D^{-1} \cdot U$. This form suggests that the flux-Jacobian must be so chosen that the diagonal blocks contains only elements on its diagonal. As the diagonal blocks are the only blocks that needed to be inverted thus the memory requirements and the CPU time is significantly reduced. In the next section the method for the approximation of the flux-Jacobian is described.

**Inviscid flux-Jacobian**

Let the values to the left and right sides of the face of a control volume $W_L$ and $W_R$ be approximated by piecewise linear reconstruction from the neighboring cells, so that $W_L = W_i$ and $W_R = W_j$ then

$$R_i = \sum_{j \in N(i)} F(W_L, W_R; n_{ij}) + \sum_{m \in B(i)} F_b(W_L; n_m),$$

(3.20)

where $F$ is the numerical flux and $F_b$ is the numerical boundary flux. $N(i)$ are the all immediate neighboring points of the point $i$ and $B(i)$ are all neighboring boundary faces. Writing the flux in dissipation form as:

$$F(W_L, W_R; n_{ij}) = \frac{1}{2}(F^c(W_L) + F^c(W_R)) \cdot n_{ij} - \frac{1}{2} D(W_L, W_R; n_{ij}),$$

(3.21)

where $F^c$ is the exact convective flux as shown in (2.2)-(2.2). Equation (3.20) can be written as

$$R_i = \frac{1}{2} F^c(W_i) \cdot \sum_{j \in N(i)} n_{ij} + \frac{1}{2} \sum_{j \in N(i)} F^c(W_j) \cdot n_{ij}$$

$$- \frac{1}{2} \sum_{j \in N(i)} D(W_i, W_j; n_{ij}) + \sum_{m \in B(i)} F_b(W_i; n_m).$$

(3.22)

For a closed control volume not touching any boundaries

$$\sum_{j \in N(i)} n_{ij} = 0.$$  

(3.23)

To find the diagonal of the flux-Jacobian, the $R_i$ must be differentiated with respect to $W_i$. Putting (3.23) in (3.22) and differentiating with respect to $W_i$ for a non-boundary control volume one gets:

$$\frac{\partial R_i}{\partial W_i} = \frac{1}{2} \sum_{j \in N(i)} \frac{\partial D(W_i, W_j; n_{ij})}{\partial W_i}.$$  

(3.24)
(3.24) shows that the flux-Jacobian is only dependent on the dissipative part. Thus choosing a simple dissipation means choosing a simpler Jacobian matrix diagonal. For example choosing the first order Lax-Friedrich’s numerical flux

\[ F_{LF}(W_L, W_R; n) = \frac{1}{2}(F(W_L; n) + F(W_R; n)) - \frac{1}{2}|\lambda|(W_R - W_L) \]  

(3.25)

where \( \lambda \) is treated as constant in differentiation. When the above flux is differentiated with respect to \( W \) for a non-boundary control volume we get the following

\[ \frac{\partial D_{LF}}{\partial W_i} = \frac{1}{2}|\lambda|I. \]

(3.26)

So the derivative of the dissipation and the diagonal of the Jacobian matrix is a positive scalar multiple of the identity matrix. Thus it results in a very simple diagonal of the flux-Jacobian to store and invert. The off-diagonal entries in the flux-Jacobian are computed explicitly on-the-fly during the computations. The complete Jacobian is given in the reference [11].

**Viscous flux-Jacobian**

The viscous flux-Jacobian is first simplified such that the stencil for the computation of \( R_i \) consists of only the immediate neighbors of the point \( i \). This is achieved by replacing the full Green-Gauss gradient based flux with the Thin Shear Layer (TSL) approximation i.e. neglecting the flux-Jacobian tangential to the face (as described in the reference [11]). The off-diagonal entries are computed explicitly like the inviscid flux but the diagonal block of the viscous flux is not diagonal itself thus damaging the memory requirements and simplicity of the LU-SGS. This problem is fixed by taking the largest eigenvalue of the operator and approximating the viscous Jacobian of the diagonal block with the largest eigenvalue of the operator. The complete Jacobian is given in the reference [11].

### 3.2.2 Rossow-operator with SGS Iterations

An alternative Jacobian is obtained by splitting the Jacobian \( J_n \) into a positive \( J_n^+ \) and a negative \( J_n^- \) part according to their eigenvalues.

\[ J_n = J_n^+ + J_n^- \]

(3.27)

where \( J_n^+ = \frac{1}{2}(J_n + |J_n|) \) , \( J_n^- = \frac{1}{2}(J_n - |J_n|) \),

(3.28)
where $J_n$ is the Jacobian calculated from the reconstructed values of the flow variables on the cell-face and $|J_n|$ is the flux-Jacobian calculated from the Roe-averaged values of the flow variables on the cell-face. The implicit system (3.10) may be rewritten as:

$$
\left( I + \frac{\delta t}{V} \sum_{\text{all faces}} J_n^+ S \right) \delta \mathbf{W}_i = \delta t R_n^i - \frac{\delta t}{V} \sum_{\text{all faces}} J_n^- \delta \mathbf{W}_j S \quad (3.29)
$$

where $i$ denotes the current cell and $j$ denotes all the direct neighbors. In order to use the changes in the conservative variables as the residual update in an explicit Runge-Kutta scheme, the $\delta \mathbf{W}$ is denoted as the new residual $\delta \tilde{R}_i$

$$
\left( I + \frac{\delta t}{V} \sum_{\text{all faces}} J_n^+ S \right) \delta \tilde{R}_i = \delta t R_n^{(m-1)} - \frac{\delta t}{V} \sum_{\text{all faces}} J_n^- \delta \tilde{R}_j S. \quad (3.30)
$$

The memory requirements for storing the split Jacobian is still too high for the practical purposes. A practical solution for that can be achieved by converting the implicit system in primitive variables (Rossow [8]) and storing the Jacobian in terms of the local Mach number as presented in the reference [7]. A scheme so formulated considerably reduces the memory requirements without compromising on the accuracy. Equation (3.30) can be written in the form of primitive variables like follows:

$$
\left( I + \frac{\delta t}{V} \sum_{\text{all faces}} P_n^+ S \right) \delta \tilde{Q}_i = \delta t Q_i^{(m-1)} - \frac{\delta t}{V} \sum_{\text{all faces}} P_n^- \delta \tilde{Q}_j S. \quad (3.31)
$$

where $Q$ is the residual in the primitive variables. $P^+$ and $P^-$ are the Jacobians expressed in terms of primitive variables. The residuals in the primitive variables can be computed from the residuals in the conservative variables by

$$
Q_i^{(m-1)} = \frac{\partial U}{\partial \mathbf{W}} R_i^{(m-1)}. \quad (3.32)
$$

The advantage gained by transforming the residuals and the Jacobians from conservative to the primitive variables is more obvious by storing the product of Jacobian matrix and vector of primitive variables, thus storing only a vector instead of storing a vector and a matrix. The product $P_n^+ \cdot \tilde{Q}$ and $P_n^- \cdot \tilde{Q}$ are shown below.

$$
P_n^+ \cdot \tilde{Q} = \\
\begin{bmatrix}
(q_n + |q_n|) \tilde{Q}(p) + \frac{1}{c}(1 - |M_0|) \tilde{Q}(p) + \rho(1 + M_0) \tilde{Q}(q_n) \\
(q_n + |q_n|) \tilde{Q}(p) + (\gamma - 1) \frac{c}{\rho}(1 - |M_0|) \tilde{Q}(p) + \gamma \rho(1 + M_0) \tilde{Q}(q_n) \\
(q_n + |q_n|) \tilde{Q}(u) + n_x(1 + M_0) \tilde{Q}(p) + n_x c(1 - |M_0|) \tilde{Q}(q_n) \\
(q_n + |q_n|) \tilde{Q}(v) + n_y(1 + M_0) \tilde{Q}(p) + n_y c(1 - |M_0|) \tilde{Q}(q_n) \\
(q_n + |q_n|) \tilde{Q}(w) + n_z(1 + M_0) \tilde{Q}(p) + n_z c(1 - |M_0|) \tilde{Q}(q_n)
\end{bmatrix}
$$
CHAPTER 3. TIME STEPPING FOR STEADY CASES

\[
P_n^\top \tilde{Q} = \begin{bmatrix}
(q_n + |q_n|) \hat{Q}(p) - \frac{1}{c}(1 - |M_0|) \hat{Q}(p) + \rho(1 - M_0) \hat{Q}(q_n) \\
(q_n + |q_n|) \hat{Q}(p) - (\gamma - 1) \frac{T}{c}(1 - |M_0|) \hat{Q}(p) + \gamma p(1 - M_0) \hat{Q}(q_n) \\
(q_n + |q_n|) \hat{Q}(u) + n_x(1 - M_0) \hat{Q}(p) - n_x c(1 - |M_0|) \hat{Q}(q_n) \\
(q_n + |q_n|) \hat{Q}(v) + n_y(1 - M_0) \hat{Q}(p) - n_x c(1 - |M_0|) \hat{Q}(q_n) \\
(q_n + |q_n|) \hat{Q}(w) + n_z(1 - M_0) \hat{Q}(p) - n_x c(1 - |M_0|) \hat{Q}(q_n)
\end{bmatrix},
\]

(3.33)

where \( \tilde{Q} = (\tilde{Q}(\rho), \tilde{Q}(p), \tilde{Q}(u), \tilde{Q}(v), \tilde{Q}(w)) \) represents the components of the updated vector of primitive variables and \( \hat{Q}(q_n) \) represents the component of the velocity residual normal to the cell face

\[
\hat{Q}(q_n) = n_x \hat{Q}(u) + n_y \hat{Q}(v) + n_z \hat{Q}(w).
\]

(3.34)

\( \gamma, h \) and \( c \) are the specific heat ratio, specific enthalpy and the speed of sound. All the variables are computed at the cell face by using the \textit{Roe-averaging} as described in Section 2.3.2. The contributions of the viscous flux-Jacobians in primitive variables are incorporated as suggested by Abarbanel [18]. The solution of (3.31) is achieved using multiple Symmetric Gauss-Seidel (SGS) sweeps. After computing the updated primitive residuals from (3.31), the variables are converted to the conservative variables by:

\[
\tilde{R}_i^{(m-1)} = \frac{\partial W}{\partial U} \hat{Q}_i^{(m-1)}.
\]

(3.35)

For steady flows this method combined with an explicit Runge-Kutta method gives a very stable scheme where the implicit operator is used as a residual smoother and the updated values of the residuals are used in each Runge-Kutta stage. This scheme is shown to be stable for the CFL numbers of \( O(100) \) to \( O(1000) \) [9].

3.3 Numerical Examples

Convergence acceleration properties of the following four types of schemes were investigated for the steady cases:

- Three stage explicit Runge-Kutta method (RK-Exp).
- Backward Euler method with LU-SGS as the linear solver and approximation of the flux-Jacobian as described in sections 3.2.1 and 3.2.1 (BE-LUSGS).
- Backward Euler method with SGS iterations with the flux-Jacobian stored in an efficient memory storage way as explained in Section 3.2.2 (BE-Rossow).
• explicit Runge-Kutta method with the implicit operator by Rossow [9] used as the residual smoother in each Runge-Kutta stage (RK-Rossow).

Unless otherwise specified the following defaults are used in the computations:

• Reynolds number = 6500000
• Prandtl number = 0.72
• Sutherland constant = 110.4
• Sutherland reference viscosity = 1.716e-05
• Sutherland reference temperature = 273
• Maximum number of SGS stages = 3
• Fluxes = 2nd order Upwind Roe flux
• Number of SGS stages for the solution of Rossow-operator = 2
• Minimum Residual = $10^{-5}$
• Number of stages of explicit Runge-Kutta Scheme = 3
• Explicit Runge-Kutta stage co-efficients = [0.15, 0.5, 1]

All the CPU times were noted on the Intel Pentium IV Workstation using 3.00 GHz processor.

3.3.1 Steady Flow over 2D RAE 2822 case 9 Configuration (Viscous Flow)

In order to compare the convergence behavior of different implicit operators, computations were carried out for the steady flow around 2D RAE 2822 case 9 airfoil at a Mach number of 0.73. The computational mesh is an unstructured mesh around the airfoil. Near field is a structured mesh as shown in Figure 3.1. Upwind Roe discretization with the Green-Gauss reconstruction is used for the spatial discretization. Flow is initiated as a uniform flow and the airfoil is placed in the flow thus disturbing the flow. After some time flow is steady. The flow solver is iterated to reduce the residual enough to achieve the stationary flow. Cut-off criteria for computations was set as the minimum
CHAPTER 3. TIME STEPPING FOR STEADY CASES

Figure 3.1: Computational Mesh for the 2D RAE 2822 case 9 Airfoil

Figure 3.2: Solution obtained after residual reduction of $10^{-5}$ for the 2D RAE 2822 case 9 Airfoil
Figure 3.3: Residual reduction for the most efficient schemes in terms of the CPU time
Figure 3.4: Convergence of the lift coefficient
Figure 3.5: Convergence of the drag coefficient
residual of $10^{-5}$ or 4000 iterations, whichever comes first. Efficiency of different schemes was measured in terms of the time taken to achieve the minimum residual. Some computations diverged before achieving the targeted residual reduction herein those computations are called unstable. Computations were carried out using different combinations of the following parameters:

- Multigrid levels: single grid (sg), 2-Level V cycle (2v), 3-Level V cycle (3v) and 4-Level W cycle (4w).
- Fine grid CFL numbers: 5, 10, 100 and 1000 for BE-LUSGS, BE-Rossow & RK-Rossow. And 1, 1.5 and 2 for RKExp.
- Coarse grid CFL numbers: 2, 5, 10.
- Number of SGS iterations in the Rossow operator 1, 2 and 4.
- Number of Runge-Kutta stages 1, 2, 3 and 5 in RK-Rossow.

The most efficient result in terms of the CPU time to achieve $10^{-5}$ residuals was achieved by the following combination:

<table>
<thead>
<tr>
<th>Method</th>
<th>Multigrid Level</th>
<th>Fine Grid CFL Number</th>
<th>Coarse Grid CFL Number</th>
<th>Number of SGS Stages</th>
<th>Number of Runge-Kutta Stages</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>BE-Rossow</td>
<td>4w</td>
<td>100</td>
<td>5</td>
<td>1</td>
<td>-</td>
<td>834.09 sec</td>
</tr>
</tbody>
</table>

RK-Rossow method seems to be better than the BE-Rossow as it was found out in this case that BE-Rossow was unstable for the uniform flow initial conditions for the CFL number of $O(100)$ but RK-Rossow method with 3 stages of Runge Kutta was found to be stable even for the uniform flow initial conditions. But RK-Rossow computations done with two Runge-Kutta stages, fine grid CFL number of 1000 and 4 SGS iterations diverged. Due to that reason the safe number of Runge-Kutta stages is proposed to be 3. With three number of Runge-Kutta stages the most efficient combination in terms of the CPU time is following:
The most efficient combination in terms of the CPU time for the existing schemes BE-LUSGS in the Tau code was found out to be as follows:

Based on these comparisons it is recommended to used the 3-stage Runge-Kutta method with the implicit Rossow operator as the residual smoother with a single SGS iteration. This combination is stable for the uniform flow initial conditions as well as for a fine grid CFL number of O(1000). Convergence history for the lift and drag coefficients is shown in the Figures 3.4 and 3.5 respectively. Residual reduction for the most efficient schemes in terms of the CPU time is shown in the Figure 3.3. In order to investigate the effects of the number of SGS iterations, fine grid CFL number and various multigrid levels on efficiency of the method, all of these criterion are plotted against the number of iterations and the CPU time. Effect of number of SGS iterations on CPU time is plotted in Figure 3.6 for BE-Rossow and RK-Rossow schemes. The CPU time increases by increasing the number of SGS iterations. But increasing the number of SGS iterations decrease the number of iterations required to achieve the targeted residual reduction in the case of RK-Rossow as shown in Figure 3.8. A better result is obtained for the implicit system equation by increasing the number of SGS iterations but increased effort due to the increased number of SGS iterations offset any advantage gained by the reduced number of iterations. That is why the CPU time required to achieve the targeted residual reduction increases when increasing the number of SGS iteration. Here it is proposed that one SGS iteration will be optimal in terms of the CPU time and stability. Similarly the effect of fine grid CFL number and various multigrid levels is shown in the Figures 3.9 and 3.12 respectively. Showing that the CPU time decreases by

<table>
<thead>
<tr>
<th>Method</th>
<th>Multigrid</th>
<th>Fine grid CFL number</th>
<th>Coarse grid CFL number</th>
<th>Number of SGS stages</th>
<th>Number of Runge-Kutta stages</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RK-Rossow</td>
<td>4w</td>
<td>1000</td>
<td>10</td>
<td>1</td>
<td>3</td>
<td>1341.85 sec</td>
</tr>
<tr>
<td>BE-LUSGS</td>
<td>3v</td>
<td>100</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>1593.51 sec</td>
</tr>
</tbody>
</table>
Figure 3.6: Effect of number of SGS iterations on CPU time
Figure 3.7: Effect of number of SGS iterations on residual reduction for BE-Rossow
Figure 3.8: Effect of number of SGS iterations on residual reduction for RK-Rossow
Figure 3.9: Effect of fine grid CFL number on CPU time
Figure 3.10: Effect of number of fine grid CFL number on residual reduction for BE-Rossow
Figure 3.11: Effect of number of fine grid CFL number on residual reduction for RK-Rossow
increasing the CFL number up to CFL number $O(100)$ but further increasing the CFL number increases the CPU time required to achieve the targeted residual reduction due to the increased number of iterations required to achieve the targeted residual as is evident from Figures 3.10 and 3.11. By increasing the multigrid level from 2v to 4w CPU time required to achieve the targeted residual reduction decreases due to the decreased number of iterations required to achieve the targeted residual reduction as evident from Figures 3.13 and 3.14.

Figure 3.12: Effect of number of multigrid levels on CPU time
Figure 3.13: Effect of number of multigrid levels on residual reduction for BE-Rossow
Figure 3.14: Effect of number of multigrid levels on residual reduction for RK-Rossow
Chapter 4

Time Stepping for Unsteady Cases

Unsteady flow problems require time-accurate computations. If an explicit method is used to compute the time-accurate flow then the maximum stable time-step is too small to be practical [3]. Implicit schemes allow larger time-steps but the work required in each time-step may become excessively large, especially in three dimensional calculations [3]. Two different methods are discussed here for the efficient computations of the unsteady flows.

- Dual Time Stepping.
- Unconstrained implicit Runge-Kutta (UIRK) method.

4.1 Dual Time Stepping

As proposed by Jameson [3] dual time is a time-accurate stepping method solving the implicit Backward Difference Formula (BDF) by writing the solution at each time-step as a steady state problem, and solving this by using a known time-inaccurate method. Equation (3.1) is approximated as:

\[ D_t [W_{i}^{n+1}] = -R^{n+1}, \]  

(4.1)

where \( D_t \) is the \( k \)th order accurate backward difference operator of the form:

\[ D_t = \frac{1}{\delta t} \sum_{q=1}^{k} \frac{1}{q} [\Delta^{-}]^q \]  

(4.2)
where \( \triangle^- W^{n+1} = W^{n+1} - W^n \).

Using the second order backward difference operator (4.1) is written as:

\[
\frac{3}{2\delta t} [VW^{n+1}] - \frac{2}{\delta t} [VW^n] + \frac{1}{2\delta t} [VW^{n-1}] + R^{n+1} = 0.
\]

(4.4)

Treating (4.4) as modified steady state problem with variable time steps in fictitious time \( t^* \) as follows:

\[
\frac{\partial W^*}{\partial t^*} + R^*(W^*) = 0,
\]

(4.5)

where \( R^*(W^*) \) is the modified residual given as:

\[
R^*(W^*) = \frac{3}{2\delta t} [W^*] + \frac{1}{V} [R(W^*) - S [W^n, W^{n-1}]]
\]

(4.6)

with the fixed source term:

\[
S [W^n, W^{n-1}] = \frac{2}{\delta t} [VW^n] - \frac{1}{2\delta t} [VW^{n-1}]
\]

(4.7)

Equation (4.4) can be solved by introducing a pseudo unsteady time, \( W^* \) is iterated during the inner iterations. At a steady state \( \frac{\partial W^*}{\partial t^*} = 0 \) and hence \( W^* = W^{n+1} \).

### 4.2 Unconstrained Implicit Runge-Kutta (UIRK) method

Another candidate for the time-accurate computations of the Navier-Stokes equations is the semi implicit Third Order Accurate Runge-Kutta method presented by Nikitin [16]. This method is based upon following third order accurate three stage explicit Runge-Kutta method:

\[
\frac{W' - W^n}{\Delta t} = -\frac{2}{3} R(W^n),
\]

(4.8)

\[
\frac{W'' - W^n}{\delta t} = -\frac{1}{3} R(W^n) - \frac{1}{3} R(W'),
\]

(4.9)

\[
\frac{W^{n+1} - W^n}{\delta t} = -\frac{1}{4} R(W^n) - \frac{3}{4} R(W'').
\]

(4.10)

The construction of the semi-implicit methods relies upon two important characteristics of this scheme. First, the last stage of the method does not
use $R(W')$. Second, the $R(W')$ and $R(W'')$ are evaluated at the same time-
step i.e. $(t + \frac{2}{3} \delta t)$. The conservative variables calculated from the above
method can be seen as following:

\[
W' = W(t^n + \frac{2}{3} \delta t) + O(\delta t^2), \quad (4.11)
\]

\[
W'' = W(t^n + \frac{2}{3} \delta t) + O(\delta t^3), \quad (4.12)
\]

\[
W^{n+1} = W(t^{n+1}) + O(\delta t^4). \quad (4.13)
\]

The explicit scheme can be perturbed by perturbing the right hand sides of
(4.8)-(4.10) as follows:

\[
\frac{W' - W^n}{\delta t} = -\frac{2}{3} R(W'') - \frac{\gamma L}{V} (W' - W''), \quad (4.14)
\]

\[
\frac{W'' - W^n}{\delta t} = -\frac{1}{3} R(W'') - \frac{1}{3} R(W') - \frac{\gamma L}{V} (W'' - W'), \quad (4.15)
\]

\[
\frac{W^{n+1} - W^n}{\delta t} = -\frac{1}{4} R(W'') - \frac{3}{4} R(W'') - \frac{\gamma L}{V} (W^{n+1} - \tilde{W}^{n+1}), \quad (4.16)
\]

here the linear operator $L$ is some approximation of the flux-Jacobian $\partial R/\partial W$
and $\gamma$ is a positive coefficient that controls the strength of implicit terms. The
right hand side (RHS) of (4.14) is an $O(\delta t)$ perturbation of the RHS of (4.8),
thus resulting in the $O(\delta t^2)$ perturbation of $W'$ and $R(W')$, thus preserving
the order of accuracy of $W'$. Similarly the right hand side of (4.15) is an
$O(\delta t^2)$ perturbation of the RHS of (4.9), thus resulting in an $O(\delta t^3)$ pertur-
bation of $W''$. Assuming that $\tilde{W}^{n+1} = W(t^{n+1}) + O(\delta t^3)$ then the right hand
side of (4.16) is an $O(\delta t^3)$ perturbation of the RHS of (4.10), thus resulting in
the $O(\delta t^4)$ perturbation of $W^{n+1}$. Thus the local error of the scheme
$W^{n+1} - W(t^{n+1})$ remains $O(\delta t^4)$ which means that the scheme retains its
third order of accuracy. This is only possible if the the last stage of the
method does not use $R(W')[16]$.

$\tilde{W}^{n+1}$, a second order approximation to the full time increment may be
written as :

\[
\frac{\tilde{W}^{n+1} - W^n}{\delta t} = -\frac{1}{4} R(W'') - \frac{3}{4} R(W') - \frac{\gamma L}{V} (\tilde{W}^{n+1} - \tilde{W}^{n+1}), \quad (4.17)
\]

where $\tilde{W}^{n+1}$ is an $O(\delta t)$ approximation of $W(t^{n+1})$, which is chosen as

\[
\tilde{W}^{n+1} = \frac{3}{2}(\alpha W' + (1 - \alpha)W'') - \frac{1}{2} W^n. \quad (4.18)
\]

Thus the choice of operator $L$ does not effect the order of accuracy of the
method. But the addition of an arbitrary operator can negatively effect the
order of the scheme. So it is advisable to use as small a value of $\gamma$ as possible. But the method is more implicit for larger values of $\gamma$ and hence more stable. So a compromise has to be made between stability and accuracy in the choice of $\gamma$. Due to the absence of any restrictions on the operator $L$ this method is called *Unconstrained implicit Runge-Kutta (UIRK) method*. This is to differentiate this from more commonly used Adam-Bashford-Crank-Nicolson (ABCN) type semi-implicit methods for the convection-diffusion equation where the formulation of the implicit part must be exact in order for the scheme to be of desired order. One example of such a scheme is that of Yoh and Zhong [13].

The UIRK scheme progresses as follows:

\begin{equation}
W' + \frac{\delta t}{V} \gamma L W' = W^n - \frac{2}{3} \delta t R(W^n) + \frac{\delta t}{V} \gamma L W^n, (4.19)
\end{equation}

which gives

\begin{equation}
\left( I + \frac{\delta t}{V} \gamma L \right) W' = W^n - \frac{2}{3} \delta t R(W^n) + \frac{\delta t}{V} \gamma L W^n, (4.20)
\end{equation}

where $I$ is the identity matrix. Similarly from (4.15), (4.16) and (4.17) one has:

\begin{equation}
\left( I + \frac{\delta t}{V} \gamma L \right) W'' = W^n - \frac{1}{3} R(W^n) - \frac{1}{3} R(W') + \frac{\delta t}{V} \gamma L W', \quad (4.21)
\end{equation}

\begin{equation}
\left( I + \frac{\delta t}{V} \gamma L \right) W^{n+1} = W^n - \frac{1}{4} R(W^n) - \frac{3}{4} R(W'') + \frac{\delta t}{V} \gamma L \tilde{W}^{n+1}, \quad (4.22)
\end{equation}

\begin{equation}
\left( I + \frac{\delta t}{V} \gamma L \right) \tilde{W}^{n+1} = W^n - \frac{1}{4} R(W^n) - \frac{3}{4} R(W') + \frac{\delta t}{V} \gamma L \tilde{W}^{n+1}, \quad (4.23)
\end{equation}

respectively. First of all $W'$ is calculated solving (4.20) and then the value of $W''$ is calculated using (4.21). After that the value of $\tilde{W}^{n+1}$ is calculated from (4.18) and used in (4.23) to calculate the $\tilde{W}^{n+1}$. At the end the value for the next time-step is calculated using (4.22). In solving (4.20)-(4.23) different linear solvers as explained in Section 3.2.1 can be used. Flux-Jacobians can be approximated as given in the Section 3.2.1 or stored as the primitive variables as explained in the Section 3.2.2.

### 4.3 Numerical Examples

Numerical stability and the efficiency of the following four types of schemes were investigated for *unsteady cases*:

- Third order accurate explicit Runge-Kutta method (Exp3).
- Third order accurate semi implicit Runge-Kutta method with LU-SGS as the linear solver and approximation of the flux-Jacobian as described in sections 3.2.1 and 3.2.1 (SI3-LUSGS).

- Third order accurate semi implicit Runge-Kutta method with SGS iterations with the flux-Jacobian stored in an efficient memory storage way as explained in Section 3.2.2 (SI3-Rossow).

- Fully implicit Second order accurate Dual-Time method (DT2) (whereby it was ensured that the dual-time inner iterations were fully converged at each time-step to eliminate any error due to non-zero residuals).

Unless otherwise specified the following defaults are used in the computations:

- Maximum number of SGS stages = 3
- Fluxes = 2nd order Upwind Roe flux
- Number of SGS stages for the solution of Rossow-operator = 2

For the viscous case (Section 4.3.2) following defaults are used:

- Reynolds number = 1000
- Prandtl number = 0.72
- Sutherland constant = 110.4
- Sutherland reference viscosity = 1.716e-05
- Sutherland reference temperature = 273

All the CPU times were noted on the Intel Pentium IV Workstation using 3.00 GHz processor.

### 4.3.1 An Expansion Fan in a Shock tube on a Structured Grid (Inviscid Flow)

As the first test case a Riemann problem of an expansion fan in a shock tube is simulated for an *inviscid flow*. The computational grid is shown in Figure 4.1. Cell aspect ratios vary from 1:1 at the outer wall to 1:100 near the inner wall. This is an attempt to emulate a typical viscous computational grid where the aspect ratios vary significantly over the computational domain. The
Figure 4.1: Part of the structured shock tube grid. Cell aspect ratio reaches 1:100

Figure 4.2: Normalized pressure at $t = 0.003$ sec

Figure 4.2: Normalized pressure at time $t = 0.003$ sec.
Figure 4.3: Error in the solution verses time-step
Figure 4.4: Error in the solution verses CPU-time
problem is initiated as a discontinuity. The presence of shock was deliberately avoided due to spatial discretization effects such as limiter switching. The upwind Roe discretization with the Green-gauss reconstruction is used for the spatial discretization. Normalized pressure after 0.003 seconds is shown in the Figure 4.2.

In Figure 4.3 the $L_2$ norm of the solution for each time step is logarithmically plotted against the time-step size. The $L_2$ norm of the solution for each time step is calculated with respect to a reasonably converged solution obtained by a very small time-step. The slope of each graph represents order of the method. As it is evident from the plots that the third order semi-implicit method with LU-SGS operator (SI3-LUSGS) and with the Rossow operator (SI3-Rossow) are stable for the large time-steps but the 3rd order of convergence is only achieved at the time-steps where the 3rd order explicit Runge-Kutta scheme is already stable. In contrast to that the dual time scheme exhibits the 2nd order of convergence right from the start and it is also unconditionally stable. Thus the dual time algorithm has an edge over all the other schemes in terms of stability and order of convergence.

Figure 4.4 shows the $L_2$ norm of the solution for each time step is logarithmically plotted against the CPU time taken for the computations. All the CPU times are measured on the same PC with no other programs running. For a given accuracy, semi-implicit method with LU-SGS operator (SI3-LUSGS) is computationally more expensive than the semi-implicit method with the Rossow operator (SI3-Rossow). But both of these methods are computationally inexpensive per step as compared to the dual time method. For a very high accuracy, 3rd order explicit Runge-Kutta scheme is computationally cheapest.

4.3.2 Separating Vortices From a Blunt Body (Viscous Flow)

As a second test case separating vortices from a blunt body are simulated for a viscous flow. The computational mesh is an unstructured mesh shown in Figure 4.5. Flow is initiated as a uniform initial flow and the blunt body is placed in that flow making it unstable with the vortices separating from the body. Upwind Roe discretization with the Green-Gauss reconstruction is used for the spatial discretization. Figure 4.6 shows a typical solution 3 seconds after initializing the flow.

A vortex is shed from the top and the bottom of the blunt body periodically thus an oscillating plot for the lift coefficient is obtained as shown in Figure 4.7. In this case, convergence is shown as the plots of amplitude
CHAPTER 4. TIME STEPPING FOR UNSTEADY CASES

Figure 4.5: Part of the far-field mesh for the blunt body

Figure 4.6: Solution obtained at time = 3.0sec
CHAPTER 4. TIME STEPPING FOR UNSTEADY CASES

Figure 4.7: Oscillations of the lift coefficient time = 3.0sec

and frequency of the lift coefficients against the logarithmic time-step and logarithmic CPU time. It can be seen from Figures 4.8 and 4.9 that the semi-implicit method with the Rossow operator (SI3-Rossow) and dual time (DT2) schemes are the most stable schemes for the larger time-steps. The largest stable time-step for the semi-implicit method with the LU-SGS operator (SI3-LUSGS) is $10^{-3}$ sec in this case. This is smaller than the most practically relevant time-steps in the range of period/25 to period/100. This shows a contrasting behavior as compared to the inviscid shocktube test case where both SI3-LUSGS and SI3-Rossow were stable for the larger time steps. For a given time-step SI3-Rossow and DT2 give better results as compared the SI3-LUSGS. This difference in the accuracy of the semi-implicit method is the consequence of using different approximations of the flux-Jacobian. As the Jacobian in the Rossow operator is based on the Roe fluxes it is a better approximation as compared to the approximation of the LU-SGS operator where only simple dissipative fluxes contribute to the diagonal of the flux-Jacobian and the requirement that the each diagonal block must be diagonal itself reduces the accuracy of the method. The efficiency of the schemes is measured in terms of the CPU time. Figures 4.10 and 4.11 show the plots of amplitude and frequency of the lift coefficient against the CPU time. For a given accuracy of amplitude of the lift coefficient, SI3-Rossow is cheaper than both SI3-LUSGS and DT2. For a given accuracy of frequency of the lift coefficient, SI3-Rossow is cheaper than both SI3-LUSGS and DT2.
Figure 4.8: Amplitude of the lift coefficient versus time-steps $\delta t$
Figure 4.9: Frequency of the lift coefficient versus time-steps $\delta t$
Figure 4.10: Amplitude of the lift coefficient versus CPU-time
Figure 4.11: Frequency of the lift coefficient versus CPU-time
coefficient, SI3-Rossow is cheaper than DT2 but a little bit more expensive than SI3-LUSGS. Thus it is always computationally cheaper to use UIRK method with the Rossow operator in the practically relevant time-steps in the range of period/25 to period/100 but for more accurate results dual time algorithm can be used at the expense of the CPU time of the order of 100.
Chapter 5

Conclusions

Two different operators were chosen and tested in this work as the residual 
smoothers in the steady case and as the implicit operator in the semi-implicit 
Runge-Kutta method for the unsteady case. One with an approximation of 
the flux-Jacobian (LU-SGS) and the other with an efficient storage mech-
anism for an approximate flux Jacobian where the Jacobian was computed 
and stored in primitive variables (Rossow). The former solved with the sin-
gle LU-SGS iteration while the later solved with different number of SGS 
iterations. In case of the steady state computations it was found out that 
the 3-stage Runge-Kutta method with the implicit Rossow operator was the 
best combination in terms of stability and efficiency. This combination was 
stable for the uniform flow initial conditions as well as for the fine grid CFL 
number of O(1000).

In the unsteady case both of these operators were used as an implicit 
operator on the right hand side of the semi-implicit Runge-Kutta scheme. 
Comparison of these schemes with the 2nd order dual time algorithm showed 
that both of the schemes were as stable as the dual time scheme for the larger 
time steps in the inviscid case while for the viscous case SI3-Rossow was stable 
for the larger time-steps as compared to SI3-LUSGS. In the viscous case only 
dual time (DT2) and the SI3-Rossow schemes were found to be stable for the 
practical range of the time steps i.e. period/25 − period/100 where period is 
the time-period of the lift coefficient.

In terms of efficiency for a given accuracy, SI3-Rossow scheme is most time 
efficient in both the viscous and the inviscid cases. UIRK method might be 
a potential successor to the dual time algorithm but further investigations 
into the stability of the scheme SI3-Rossow are needed in case of high Mach 
number flows and the flows with discontinuities like shocks. Additionally 
another better implicit operator may be investigated in place of the Rossow
operator in order to increase accuracy of the method for the larger time steps.
Bibliography


