https://theses.gla.ac.uk/

Theses Digitisation:

https://www.gla.ac.uk/myglasgow/research/enlighten/theses/digitisation/

This is a digitised version of the original print thesis.

Copyright and moral rights for this work are retained by the author

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge

This work cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given

Enlighten: Theses
https://theses.gla.ac.uk/
research-enlighten@glasgow.ac.uk

A Parallel Implementation of the Newton's Method in Solving Steady State Navier-Stokes Equations for Hypersonic Viscous Flows

-- α-GMRES: A new Parallelisable Iterative Solver for Large Sparse Non-symmetric Linear Systems

by

Xiao Xu

This thesis is submitted for the degree of
Doctor of Philosophy in
the University of Glasgow

© May, 1993 by Xiao Xu

ProQuest Number: 10992287

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

ProQuest 10992287

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by the Author.

All rights reserved.
This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Thesis
9494
copy 1

GLASGOW
UNIVERSITY
LIBRARY

Abstract

The motivation for this thesis is to develop a parallelizable fully implicit numerical Navier-Stokes solver for hypersonic viscous flows. The existence of strong shock waves, thin shear layers and strong flow interactions in hypersonic viscous flows requires the use of a high order high resolution scheme for the discretisation of the Navier-Stokes equations in order to achieve an accurate numerical simulation. However, high order high resolution schemes usually involve a more complicated formulation and thus longer computation time as compared to the simpler central differencing scheme. Therefore, the acceleration of the convergence of high order high resolution schemes becomes an increasingly important issue.

For steady state solutions of the Navier-Stokes equations a time dependent approach is usually followed using the unsteady governing equations, which can be discretised in time by an explicit or an implicit method. Using an implicit method, unconditional stability can be achieved and as the time step approaches infinity the method approaches the Newton's method, which is equivalent to directly applying the Newton's method for solving the N-dimensional non-linear *algebraic* system arising from the spatial discretisation of the steady governing equations in the global flowfield. The quadratic convergence may be achieved by using the Newton's method. However one main drawback of the Newton's method is that it is memory intensive, since the Jacobian matrix of the non-linear *algebraic* system generally needs to be stored. Therefore it is necessary to use a parallel computing environment in order to tackle substantial problems.

In the thesis the hypersonic laminar flow over a sharp cone at high angle of attack provides test cases. The flow is adequately modelled by the steady state locally conical Navier-Stokes (LCNS) equations. A structured grid is used since otherwise there are difficulties in generating the unstructured Jacobian matrix. A conservative cell centred finite volume formulation is used for the spatial discretisation. The schemes used for evaluating the fluxes on the cell boundaries are Osher's flux difference splitting scheme, which has continuous first partial derivatives, together with the third order MUSCL (Monotone Upwind Schemes for Conservation Law) scheme for the convective fluxes and the second order central difference scheme for the diffusive fluxes.

In developing the Newton's method a simplified approximate procedure has been proposed for the generation of the numerically approximate Jacobian matrix that speeds up the computation and reduces the extent of cells in which the discretised *physical* state variables need to be used in generating the matrix element. For solving the large sparse non-symmetric linear system in each Newton's iterative step the α-GMRES linear solver has been developed, which is a robust and efficient scheme in sequential computation. Since the linear solver is designed for generality it is hoped to apply the method for solving similar

large sparse non-symmetric linear systems that may occur in other research areas. Writing code for this linear solver is also found to be easy.

The parallel computation assigns the computational task of the global domain to multiple processors. It is based on a new decomposition method for the Nth order Jacobian matrix, in which each processor stores the non-zero elements in a certain number of columns of the matrix. The data is stored without overlap and it provides the main storage of the present algorithm. Corresponding to the matrix decomposition method any N-dimensional vector decomposition can be carried out. From the parallel computation point of view, the new procedure for the generation of the numerically approximate Jacobian matrix decreases the memory required in each processor. The α-GMRES linear solver is also parallelizable without any sequential bottle-neck, and has a high parallel efficiency. This linear solver plays a key role in the parallelization of an implicit numerical algorithm.

The overall numerical algorithm has been implemented in both sequential and parallel computers using both the sequential algorithm version and its parallel counterpart respectively. Since the parallel numerical algorithm is on the global domain and does not change any solution procedure compared with its sequential counterpart, the convergence and the accuracy are maintained compared with the implementation on a single sequential computer.

The computers used are IBM RISC system/6000 320H workstation and a Meiko Computer Surface, composed of T800 transputers.

To Xiaoju, Xing, my parents
and my motherland.

Acknowledgements

The financial support of the scholarship from University of Glasgow during study, and the ORS award of the CVCP (Committee of Vice-Chancellors and Principals) in the later two years are gratefully acknowledged. Acknowledgements are due to Dr. Ning Qin, Weiyu Chen, Ping Qian, Mr. Gaozhi Pan, Qiang Zhou, and his parents for their generous support to the author, by which his study in University of Glasgow could finally become true.

Acknowledgements are due to the University of Glasgow in which the author has used its resources during his study, and especially to Dr. Derek Higgins of the Computer Service Centre for his help in using Meiko Computing Surface. The author thanks Dr. Chang Shu, Dachun Jiang, Zhijian Wang, Ken Badcock and other members of staff in Department of Aerospace Engineering for their help during his research. Thanks are also given to Dr. Xiaoshi Jin for his worthful discussion in writing this thesis.

The author would like to express his gratitude to Prof. Zuosheng Yang, former supervisor of author, for encouraging the author's GMRES method applications, to Prof. Bryan E. Richards who leads CFD team in University of Glasgow in which all the pioneering researches are founded, and special thanks to Dr. Ning Qin who provided his explicit code and an original code for the generation of the numerically approximate Jacobian matrix, which enabled the author to start the research on linear solvers and parallel algorithms immediately.

The author is especially indebted to Prof. Bryan E. Richards for his deliberate and patient guidance, without whom he couldn't have completed his work. Finally, the author would like to acknowledge his wife, Xiaoju Shi, who has always supported him in his professional pursuits.

Declaration

I declare that the whole work of the thesis is carried out solely by myself.

CONTENTS

Chapter One

Introduction

Computational fluid dynamics (CFD) is a discipline that seeks the approximate numerical solutions of fluid flows, in which it is assumed that the basic equations describing their behaviour are known theoretically but for which no analytical solutions exist. It is one of the key areas that needs a very large number of numerical computations to be performed. The desire to solve increasingly complex physical problems and to use high resolution schemes for even more accurate numerical simulations has always been running ahead of the capabilities of the time, and has provided a driving force for the development of faster computing machines with larger memory. Currently, developments in parallel computing systems have offered the potential for the scalability to large numbers of processors that is required, and have drastically increased the amount of memory available for numerical simulations, which result in a speedup of computations of existing methods relative to those done by a single processor (standard, regular or scalar) computer. Therefore in CFD it is becoming a necessity to use parallel computing environments. However the achievement of this potential relies on efficient and portable software which takes advantage of all that the hardware can offer.

1.1 Hypersonic flows and numerical simulations

With the development of space technology, the design of re-usable hypersonic vehicles becomes one of the most important factors, and the enabling of reentry has become the most challenging problem. In the hypersonic regime a major design driver is the accurate prediction of peak aerodynamic forces and peak aerodynamic heating rates. However hypersonic flows around flight vehicles often involve strong flow interactions such as shock-shock, shock-boundary layer, shock-vortex and other viscous/inviscid interactions. Numerical solutions of Euler equations or even simpler inviscid modelling can sometimes provide useful data on aerodynamic forces and can be coupled with boundary layer codes to predict such important parameters as skin friction and heat transfer rates. However this approach is not applicable when strong viscous/inviscid interactions occur. The boundary layer approximation is no longer valid in such regions including those on the lee-side of the vehicle at high angles of attack, near the nose of the body and around the leading edge of the wing at high Mach number. Therefore, for strongly interactive flows, a numerical solution of the Navier-Stokes equations is required to predict hypersonic aerodynamic characteristics accurately.

The definition of a computational approach involves several steps leading from an initial mathematical model to a final numerical solution. The first step is the selection of an appropriate mathematical model in order to describe the physical problem researched. The second step is the choice of the discretisation method of the mathematical formulation and involves two components, the space discretisation and the equation discretisation. The space discretisation consists of setting up a grid by which the continuum of flow field is replaced by a finite number of cells, in which the numerical values of the discretised *physical* state variables will have to be determined either in the cells themselves or at their nodes. There are two different types of grid: structured grid and unstructured grid. Once a grid has been defined the equations can be discretised, leading to the transformation of the differential or integral equations to discrete *algebraic* operations involving the values of the unknown variables in the grid cells or at their nodes. Finite difference (FD), finite element (FE), and finite volume (FV) methods are available in this transformation, and many numerical schemes for evaluating numerical flux have also been developed according to the physical properties of the flow equations. The third step is to solve the resulting non-linear or linear *algebraic* system.

As pointed out above the Navier-Stokes equations are required for describing hypersonic viscous flows. In the Navier-Stokes equations there are convective and diffusive terms. The most general flow configuration for a non-viscous, non-heat-conducting fluid is described by the set of Euler equations, obtained from the Navier-Stokes equations by neglecting all shear stresses and heat conduction terms, i.e., the diffusive terms. After applying assumptions for some flow properties the mathematical system of Euler equations is a first order quasi-linear hyperbolic system and is associated with the propagation of waves. There are many numerical schemes developed for solving the Euler equations, since following Prandtl's boundary layer concept these provide a valid approximation for flows at high Reynolds numbers outside viscous regions developing in the vicinity of solid surfaces for well behaved flows. Therefore we in this work combine numerical discretisation methods for the Euler equations with the numerical formulations for the viscous and heat-conduction terms.

The existence of flow interaction phenomena in hypersonic viscous flows also requires the use of a high order high resolution scheme in the discretisation of the Navier-Stokes equations for an accurate numerical simulation. The so-called high resolution scheme is directed towards the introduction of physical properties of the flow equations into the discretised formulation [1]. The central space discretisation is suitable for the diffusive terms, but for the Euler equations the schemes, based on the central space discretisation, have a symmetry with respect to a change in sign of the Jacobian eigenvalues which does not distinguish upstream from downstream influences. Hence the physical propagation of perturbations along characteristics is not considered in the definition of the numerical model.

However, flux-split schemes have had a significant impact on CFD. These schemes have the often stated property (desirable to many) that additional numerical dissipation does not have to be added to stabilize them. They are directed towards an introduction of the physical properties of the flow equations into the discretised formulation and have led to the family of techniques known as upwinding, covering a variety of approaches, such as flux vector splitting, flux difference splitting and various 'flux controlling' methods.

The first level introduces only information on the sign of the eigenvalues, whereby the flux terms are split and discretised directionally according to the sign of the associated propagation speeds. This leads to the flux vector splitting methods [2,3].

A higher level of introduction of physical properties into the definition of the scheme can however, be defined, following the very remarkable scheme of Godunov [4]. In Godunov's method, the conservative variables are considered as piecewise constant over the cells at each time step and the time evolution is determined by the exact solution of the Riemann (shock tube) problem at the interface of cells. Hence, properties derived from the exact local solution of the Euler equations are introduced in the discretisation. This approach has been extended to higher orders, as well as to variants, whereby the local Riemann problem is only approximately solved through approximate Riemann solvers. They are referred to sometimes as flux difference splitting methods [5,6].

Both flux-vector split schemes and flux-difference schemes capture shock waves well, but flux-difference split schemes perform noticeably better on contact discontinuities. It is this ability of flux-difference split schemes to capture contact discontinuities that evidently makes the schemes so attractive for viscous flows [7].

Since first order accuracy is limited for practical problems [8] accuracy has to be improved. The straightforward replacement of the first order upwind space differences by appropriate second order accurate formulas leads to deficiencies similar to those encountered with central schemes, namely the generation of oscillations around discontinuities. This is somehow disappointing since one of the motivations behind upwind schemes is the hope that the introduction of physical propagation properties in the discretisation will prevent the generation of oscillations in the numerical solutions. This is only partly fulfilled in the sense that for non-linear equations, such as the Euler equations, oscillation-free results can be obtained for weak stationary discontinuities. However, this is not a general property, since it can be shown theoretically that linear second order upwind schemes always generate oscillations [9]. A deeper analysis is therefore necessary to achieve the goals of oscillation-free, second order schemes able to represent accurately shock as well as contact discontinuities. A systematic analysis of the conditions required by a scheme to satisfy these properties has been developed, initiated by Godunov [4] who introduced the important concept of monotonicity. For non-linear equations the concept of bounded total variation of the solution is more general and has been introduced by Harten [10] as a criterion to ensure

that unwanted oscillations are not generated by a numerical scheme. General families of schemes satisfying these conditions can be defined [10,11,12] but it is shown that these schemes can only be first order accurate. The only way to overcome this limitation, while satisfying the required conditions, is to introduce non-linear components. Non-linear discretisations imply that the schemes will be non-linear even when applied to linear equations. This important concept was introduced initially by Van Leer [13,14] and Boris and Book [15,16] under the form of 'limiters', which control the gradients of the computed solution such as to prevent the appearance of over- or undershoots.

In the numerical solution of Euler and Navier-Stokes equations, there are two major classes of problems, steady and unsteady. For steady state solutions, a time dependent approach is usually followed using the unsteady governing equations. There are two advantages of doing so. Firstly, the starting of the solution is robust in the sense that non-physical states can easily be avoided as long as the initial flow field is physically defined and the time step is small enough so that a physical path can be followed during the process of the solution. Secondly, the same code can be used for both steady and unsteady problems if accuracy is maintained. However, this approach also brings out some problems. As an iterative procedure for steady state solution, the physical path is not necessarily a fast convergence path. Acceleration techniques based on the time dependent approach such as local time stepping, multigrid and the use of approximate implicit operators destroy the time accuracy and, therefore, the second advantage cannot normally be achieved.

In the time dependent approach, the unsteady governing equations can be discretised in time by an explicit or an implicit method. Using an explicit method, the convergence for a steady state problem can be extremely slow due to the stability restrictions on time steps even if some acceleration techniques were employed. Using an implicit method, unconditional stability can be achieved and as the time step approaches infinity the method approaches the Newton's method for the solution of the non-linear *algebraic* system corresponding to the steady state problem. However it is generally not easy (1) to obtain the real Jacobian of the non-linear system and (2) to solve the resulting large sparse non-symmetric linear system .

Previous researchers in CFD, on one hand, have tried to avoid these two difficulties in the following ways respectively: (1) to construct simplified implicit operators [17,18]; (2) to use approximate factorization for the multidimensional implicit operator so that the resulting linear system can be solved easily. Both of these naturally negate the advantages of the implicit scheme. The time step size for a simplified implicit method is still limited due to the inconsistency of the implicit operator and the right hand side (the non-linear system) and the factorization error which increases with the time step. Simplified implicit methods will thus obviously not approach a Newton's method as the time step approaches infinity. On the other hand, instead of avoiding the difficulties for a fully implicit method, Qin and Richards [19,20] tried to tackle the problem directly in order to achieve fast convergence for the steady

state solution. The sparse quasi-Newton method (SQN) [21] and the sparse finite difference Newton method (SFDN) [22] were used so that the difficulty in getting the Jacobian of the non-linear system was tackled. After the linearization of the non-linear system is achieved, a large sparse non-symmetric linear system results. For one dimensional problems, a block pentadiagonal matrix solver was devised to obtain a direct solution of the resulting linear system. For multidimensional problems, the block line Gauss-Seidel iterative method was used. As pointed out by Qin and Richards [20], the convergence of the method for the linear system is still not satisfactory if higher than first order spatial discretisation is used. A similar problem resulting from the use of high order schemes was also found by Hemker and his colleagues [23,24,8] to achieve an effective application of the multigrid method. They introduced a defect correction technique to tackle the problem. From the research by Venkatakrishnan [18], Whitfield et. al. [25], and Orkwis et. al. [26,27,28], who generate the exact Jacobian matrix using the symbolic manipulation expert system MACSYMA, we can see that the conjugate gradient (CG) type methods and the generalized minimal residual (GMRES) technique are efficient methods for solving non-sysmmetric systems when used with a very efficient preconditioner for solving transonic and/or supersonic flow problems. Mallet et. al. [29] and Wigton et. al. [30] also use the GMRES technique to accelerate convergence. A family of efficient and widely used preconditioners is the incomplete lower-upper (ILU) factorization method. However this type of preconditioner causes the main obstacle to the design of a parallel algorithm since it includes forward and backward substitutions and thus introduces the sequential bottle-neck. Radicati di Brozolo and Robert proposed two ways to execute the ILU factorization in parallel computation [31]. Since in the parallel calculation the ILU factorization scheme is not corresponding to that in the sequential calculation case the efficiency decreases to very close to that obtained with diagonal preconditioning. Venkatakrishnan et. al. [32] use ILU in each subdomains. Other disadvantages of using ILU factorization as the preconditioner are that the lower and the upper matrixes take additional memory space at least equal to that for the original matrix, and the generation procedure for the lower and upper matrixes is dependent on that for the original matrix. Thus the method is complex and time consuming.

Since one main drawback of the Newton's method is its memory intensive nature, in which the Jacobian matrix of the non-linear *algebraic* system generally needs be stored, the Newton's method is limited by the capabilities of the computer in practical application. It is anticipated that this problem can hopefully be finally solved by using parallel computer systems. Therefore in this thesis from the point of view of numerical solutions we are facing four main tasks in using the Newton's method to CFD problems. (1) Evaluation of the Jacobian of the non-linear system for a high order high resolution scheme for viscous flows (it is almost impossible to generate the analytical Jacobian if turbulence and/or chemical reactions are involved). (2) Efficient decomposition of the storage of the Jacobian matrix in

the parallel computer. (3) Efficient solution of the resulting large sparse non-symmetric linear system when using the high order high resolution scheme for complicated fluid flows. (4) Efficient parallelization of the linear solver without any sequential bottle-necks.

1.2 Parallel features of the computer systems and parallel algorithm design

Because of the very large requirements for both speed of computation and computer memory, parallel computing systems are being developed through the ideal of performing as many operations as possible simultaneously, in parallel, instead of sequentially. They have been designed with a single purpose in mind; communication between processors to be reliable and predictable. Special purpose parallel architectures have been designed with a particular problem in mind. They result in parallel computing systems well suited for solving that particular problem, but which cannot in general be used for any other purpose. Parallel computing systems fall into a number of categories:

(a) Computer networks which link autonomous computers via a communication network.

(b) Massively parallel systems with thousands of processing elements, where each element has a dedicated memory module. These hold the greatest promise for significantly extending the range of practically solvable computational problems, e.g., the Thinking Machine's CM-2, which has pushed the number of processors up to 64 K, and holds performance records for several applications that fit its particular structure and constraints.

(c) There are two type of multiprocessor systems. One has few processors, which use a global shared memory that can be accessed by all processors. Examples are the IBM 3090 and the Cray 2. The alternative includes processors each with its own dedicated memory, i.e., it has a distributed memory. The processors are then loosely coupled [33] via a high-speed communication link, and they are called message-passing architecture processors. Examples are the Meiko Computer Surface and the Intel iPSC/860.

According to Flynn's definition [34,35] four broad classifications emerge based on the way the machine relates its instructions to the data being processed.

(a) SISD single instruction stream single data stream. This is the conventional serial von Neumann computer.

(b) SIMD single instruction stream multiple data stream. Some examples are the Cray 1 and the ILLIAC IV.

(c) MISD multiple instruction stream single data stream. No examples.

(d) MIMD multiple instruction stream multiple data stream. The examples are the IBM 3090, the Cray 2, the Alliant FX/8, the Meiko Computer Surface, and the iPSC/860.

In a message-passing architecture, processors communicate by sending and receiving messages. The processors in such systems normally operate asynchronously, and so the transfer of information requires the sending and receiving processes to synchronise the process. In a message-passing system the link between cooperating processes exists in the form of a naming convention within the Send_Message and Receive_Message operations, and then two alternatives are possible. An obvious naming convention would be for each message-passing operation to name explicitly the partner process (and/or the processor on which it resides) for the operation. For example, assuming that processes P1 and P2 exist, a message could be sent from P1 to P2 by the execution of the following code.

```
Processor 1                    Processor 2
    :                              :
    Send(P2, message)              Receive(P1, message)
    :                              :
```

An alternative naming convention can be implemented by directing messages through named channels. In this case, for two processes to communicate, they must both quote the same channel identifier in their respective message-passing operations as follows.

```
Processor 1                    Processor 2
    :                              :
    Send(chan_X, message)          Receive(chan_X, message)
    :                              :
```

For the second naming convention, there are two typically different message-passing operations in the receive statement, one includes the operation that determines the message to be received, the other includes the operation that only describes the message being received. For the latter case the computation may be terminated by receiving the disordered message. For example, assuming that processes P1, P2, and P3 exist, two messages need to be sent from P1 and P2 to P3 by the execution of the following code.

```
Processor 1          Processor 3          Processor 2
  :                    :                    :
  Send(chan_X, message1)  :                 Send(chan_X, message2)
  :          Receive(chan_X, messageX)      :
  :                    :                    :
  :          Receive(chan_X, messageY)      :
  :                    :                    :
```

In practical calculations we cannot know that messageX is message1 or message2 at the programming stage. If these two receive statements in the same subroutine we need a method to treat this problem according to the information received concerning the message, but if these two receive statements are in different subroutines we can do nothing at the code writing stage. The Meiko Computer Surface is one of the parallel computers which allows this kind of receive statement.

The software environment supported for parallel execution in the message-passing architecture parallel computer is that using standard sequential programming languages Fortran/C with message passing tools. Parallel programming languages are in the development stage.

There are different strategies in the design of parallel algorithms, which play a key role in the use of a parallel computing system. One approach is either to parallelise an existing sequential algorithm, perhaps after modifications, or to develop a new algorithm easier to parallelise, without being too specific about the implementation in particular types of machines. Here the parallel algorithm may maintain the same convergence procedure as its sequential counterpart, or one might be concerned with the algorithm's convergence and rate of convergence (in either a synchronous or an asynchronous computing environment), and with the algorithm's potential for substantial speedup over its sequential counterpart. A second approach is to focus on the details of implementation on a particular type of machine. The issues in this case are algorithmic correctness, as well as time and communication complexity of the implementation. In yet another approach, the choice of the algorithm and the parallel machine are interdependent to the point where the design of one has a strong influence on the design of the other. A typical example is when a VLSI chip is designed to execute efficiently a special type of parallel algorithm.

Domain decomposition is one general method for distributing the computational task in parallel computation in CFD. One parallel algorithm is used to distribute the flow problem to each subdomain and solve each part of the flow problem in an individual subdomain as a sequential case with boundary conditions around the subdomain. Since on internal boundaries, boundary conditions are unknown, global domain flow problem needs to be solved by communicating data between subdomains and constraining the values on the internal boundaries. The convergence of such a parallel algorithm is normally not equal to its sequential counterpart. Another method is to perform a part of the computational task in each subdomain, without imposing an internal boundary condition. Then there is no additional computation compared with the sequential case. The flow problem remains on a global domain similar to the sequential case. By using this method the convergence of the parallel algorithm become equal to its sequential counterpart. An alternative decomposition method, which corresponds to the second method of domain decomposition, can be constructed by

decomposing the Jacobian matrix and/or the vector of discretised *physical* state variables in the global domain in the Newton's method. This will be discussed in chapter 5.

1.3 About this thesis

This work is based on the studies of the CFD team at Glasgow University on the adaption of current state of the art CFD techniques towards predicting hypersonic viscous flows. The objective of this work is to contribute to the developments of (1) an efficient Newton's method for solving the steady state Navier-Stokes equations with high order high resolution spatial discretisation scheme, (2) an efficient parallel implementation of the above algorithm.

Motivated by the use of parallel computers, a linear solver for solving the large sparse non-symmetric linear system was proposed [36], which is robust for solving the linear system arising from the high order high resolution spatial discretisation scheme for complicated fluid flows and also can be thought as a general numerical algebraic method. In parallel computation the linear solver has high efficiency and does not have a sequential bottle-neck [37]. A simplified procedure was proposed in the generation of the numerically approximate Jacobian matrix, which speeds up the computation and reduces the extent of the cell in which the discretised *physical* state variables need to be used in generating column elements of the Jacobian matrix.

In this work, flows around a cone at high angle of attack are chosen as the test flow cases. In this case the governing equations can be simplified to the locally conical Navier-Stokes equations by using locally conical approximation. The flow includes a strong bow shock wave on the windward side and a separated shear layer on the leeward side. A conservative cell centred finite volume method is used for the space discretisation. In the finite volume method numerical approximations are stored inside the volumes, and the fluxes are calculated at the cell boundaries. For the convective terms at each cell boundary the flux is computed by approximately solving a local one-dimensional Riemann problem. The approximate Riemann solver used is the Osher's scheme [12,5,38]. The high order scheme is achieved by using the so called MUSCL approach as proposed by Van Leer [14].

Because the algorithm was originally designed for the sequential case a relatively simple strategy for parallel computation can be made. The computation algorithm allocated in each processors is required to be maintained as close as possible to the sequential version but with some communications between processors. The division of the computational task can be made by decomposing the Jacobian matrix of the non-linear *algebraic* system of the spatial discretisation. From the discussion in chapter 5, the parallel algorithm designed is suitable for a parallel computing system composed of a number of powerful processors. The parallel computer used is the Meiko Computer Surface in Glasgow University.

In chapter 2, we will describe the general equations and schemes for numerical discretisation of the Navier-Stokes equations, which includes conservative discretisation and the finite volume method, Osher's and Roe's flux difference splitting upwind schemes, high order variable interpolation, and evaluation of diffusive flux. In chapter 3, a general curvilinear coordinate transformation is performed for the Navier-Stokes equations and then the LCNS equations are derived through the spherical coordinate transformation and by applying the locally conical approximation. For the flows around a sharp cone, which is governed by the LCNS equations, the detailed discretisation steps are performed in this chapter. The main contributions of this doctoral work are in chapters 4 and 5. Chapter 4 includes the new simplified numerical Jacobian matrix generation and the new linear solver construction, and chapter 5 includes the parallel implementation of the algorithm.

Chapter Two

High order upwind schemes for Navier-Stokes equations

2.1 Introduction

Solving the full system of Navier-Stokes equations is the ultimate goal of a numerical flow simulation. In the Navier-Stokes equations there are the convective and diffusive terms, which describe different physical flow phenomena respectively. According to Prandtl's boundary layer analysis the Euler equations are a valid approximation for describing the flows at high Reynolds numbers outside viscous regions. There exist a considerable amount of numerical solutions of Euler equations. Therefore in this chapter the numerical scheme for the Navier-Stokes equations can be developed by combining the numerical schemes for Euler equations with the numerical formulations for the diffusive terms. Since the hypersonic viscous flows studied in this work include strong flow interaction phenomena we should choose the high order upwind scheme for the Euler equations, which has the ability for capturing both shock waves and contact discontinuities and further is suitable for hypersonic viscous flows calculation.

Chapter 2.2 describes the basic conservative form of the Navier-Stokes equations as the beginning of the discussion. Chapter 2.3 describes the conservative discretisations and finite volume method for the spatial discretisations of the Navier-Stokes equations. Chapter 2.4 focuses on the Euler equations, and describes the basic properties which is useful in developing the numerical schemes. Chapter 2.5 describes the detailed formulations in 3-dimensional space of the two well used Godunov-type approximate Riemann solvers, the Osher's scheme and the Roe's scheme. They have the ability to capture contact discontinuities. Chapter 2.6 gives a κ-parameter family of higher order schemes. Chapter 2.7 describes the method for evaluation of the diffusive flux.

2.2 Conservative laws and Navier-Stokes equations

General speaking we have the *scalar conservation law* and *vector conservative law* in fluid dynamics. From the derivations of the conservations of mass, momentum, and energy we have the integral compact form of the Navier-Stokes equations

$$\frac{\partial}{\partial t}\int_{\Omega} \mathbf{Q}\, d\Omega + \oint_{s} \overline{\mathbf{F}} \cdot d\vec{s} = \int_{\Omega} \mathbf{S}\, d\Omega \qquad (2.2.1)$$

where Ω is an arbitrary volume, fixed in flowfield space, bounded by a closed surface s, and

$$\mathbf{Q} = \begin{bmatrix} \rho \\ \rho \vec{v} \\ \rho E \end{bmatrix} \tag{2.2.2}$$

$$\overline{\mathbf{F}} = \begin{bmatrix} \rho \vec{v} \\ \rho \vec{v} \otimes \vec{v} + p \bar{\bar{I}} - \bar{\bar{\tau}} \\ \rho \vec{v} H - \bar{\bar{\tau}} \cdot \vec{v} - k \nabla T \end{bmatrix} \tag{2.2.3}$$

$$\mathbf{S} = \begin{bmatrix} 0 \\ \rho \vec{f}_e \\ \rho \vec{f}_e \cdot \vec{v} + q_H \end{bmatrix} \tag{2.2.4}$$

In formulations (2.2.2-4), the symbols ρ, p, T, $\vec{v}$, E, H, $\bar{\bar{\tau}}$, $\bar{\bar{I}}$, k, $\vec{f}_e$ and q_H represent the density, pressure, temperature, velocity, total energy per unit mass, total enthalpy per unit mass, stress tensor, 3×3 unit matrix, thermal conductivity coefficient, volume forces, and heat sources respectively.

Because Ω is an arbitrary volume we have the differential compact form of the Navier-Stokes equations following from the use of the Gauss formula

$$\frac{\partial \mathbf{Q}}{\partial t} + \nabla \cdot \overline{\mathbf{F}} = \mathbf{S} \tag{2.2.5}$$

In Cartesian coordinates x_1, x_2, x_3, the velocity vector has components v_1, v_2, v_3 and the flux vector-tensor $\overline{\mathbf{F}}$ has components $\mathbf{E}_i$-$\mathbf{E}_v$, $\mathbf{F}_i$-$\mathbf{F}_v$ $\mathbf{G}_i$-$\mathbf{G}_v$, i.e., we obtain

$$\mathbf{Q} = \begin{bmatrix} \rho \\ \rho v_1 \\ \rho v_2 \\ \rho v_3 \\ \rho E \end{bmatrix} \tag{2.2.6}$$

$$\mathbf{E}_i = \begin{bmatrix} \rho v_1 \\ \rho v_1^2 + p \\ \rho v_1 v_2 \\ \rho v_1 v_3 \\ \rho v_1 H \end{bmatrix} \qquad \mathbf{E}_v = \begin{bmatrix} 0 \\ \tau_{11} \\ \tau_{12} \\ \tau_{13} \\ \tau_{11} v_1 + \tau_{12} v_2 + \tau_{13} v_3 + q_1 \end{bmatrix} \tag{2.2.7a}$$

$$\mathbf{F}_i = \begin{bmatrix} \rho v_2 \\ \rho v_1 v_2 \\ \rho v_2^2 + p \\ \rho v_2 v_3 \\ \rho v_2 H \end{bmatrix} \qquad \mathbf{F}_v = \begin{bmatrix} 0 \\ \tau_{21} \\ \tau_{22} \\ \tau_{23} \\ \tau_{21} v_1 + \tau_{22} v_2 + \tau_{23} v_3 + q_2 \end{bmatrix} \tag{2.2.7b}$$

$$\mathbf{G}_i = \begin{bmatrix} \rho v_3 \\ \rho v_1 v_3 \\ \rho v_2 v_3 \\ \rho v_3^2 + p \\ \rho v_3 H \end{bmatrix} \qquad \mathbf{G}_v = \begin{bmatrix} 0 \\ \tau_{31} \\ \tau_{32} \\ \tau_{33} \\ \tau_{31} v_1 + \tau_{32} v_2 + \tau_{33} v_3 + q_3 \end{bmatrix} \tag{2.2.7c}$$

$$\mathbf{S} = \begin{bmatrix} 0 \\ \rho f_{ex_1} \\ \rho f_{ex_2} \\ \rho f_{ex_3} \\ \rho \vec{f}_e \cdot \vec{v} + q_H \end{bmatrix} \tag{2.2.8}$$

The conservative form of Navier-Stokes equations (2.2.1) is used in this paper since it can treat flow discontinuities automatically.

2.3 Conservative discretisations and finite volume method

Because the conservative equations can treat flow discontinuities automatically, we should attempt to keep this property in the numerical schemes. This is named conservative discretisation. In the following we use integral form conservative discretisation.

The conservation law is

$$\frac{\partial}{\partial t}\int_{\Omega} \mathbf{Q}\, d\Omega + \oint_s \overline{\mathbf{F}} \cdot d\vec{s} = \int_{\Omega} \mathbf{S}\, d\Omega \tag{2.3.1}$$

where Ω is an arbitrary volume, fixed in space, bounded by a closed surface s. Since Ω is an arbitrary volume, for an arbitrary subvolume of the volume Ω we also can write the conservation law (2.3.1).

For an arbitrary subdivision of the volume Ω into, say, three subvolumes we can write the conservation law for each subvolume as follows

$$\begin{aligned}
&\frac{\partial}{\partial t}\int_{\Omega_1} \mathbf{Q}\, d\Omega + \oint_{s_1 \cup (\Omega_1 \cap \Omega_2) \cup (\Omega_1 \cap \Omega_3)} \overline{\mathbf{F}} \cdot d\vec{s} = \int_{\Omega_1} \mathbf{S}\, d\Omega \\
&\frac{\partial}{\partial t}\int_{\Omega_2} \mathbf{Q}\, d\Omega + \oint_{s_2 \cup (\Omega_2 \cap \Omega_1) \cup (\Omega_2 \cap \Omega_3)} \overline{\mathbf{F}} \cdot d\vec{s} = \int_{\Omega_2} \mathbf{S}\, d\Omega \\
&\frac{\partial}{\partial t}\int_{\Omega_3} \mathbf{Q}\, d\Omega + \oint_{s_3 \cup (\Omega_3 \cap \Omega_1) \cup (\Omega_3 \cap \Omega_2)} \overline{\mathbf{F}} \cdot d\vec{s} = \int_{\Omega_3} \mathbf{S}\, d\Omega
\end{aligned} \tag{2.3.2}$$

where $\Omega = \Omega_1 \cup \Omega_2 \cup \Omega_3$, adjacent Ω_j may overlap if each internal surface is common to two subvolumes, $s = s_1 \cup s_2 \cup s_3$. Notice that the essential significance of these formulations lies

in the presence of the surface integral and the fact that the time variation of **Q** inside the volume only depends on the surface values of the fluxes. We do the calculation by adding up the three subvolume conservation laws. For volume Ω_1 we have a contribution of the fluxes

$$\int_{\Omega_1 \cap \Omega_2} \overline{\mathbf{F}} \cdot d\vec{s}$$

while for Ω_2 we have an integral on the internal surface $\Omega_2 \cap \Omega_1$. Since the internal surface $\Omega_1 \cap \Omega_2$ and $\Omega_2 \cap \Omega_1$ have opposite outward normal we obtain

$$\int_{\Omega_2 \cap \Omega_1} \overline{\mathbf{F}} \cdot d\vec{s} = - \int_{\Omega_1 \cap \Omega_2} \overline{\mathbf{F}} \cdot d\vec{s}$$

where we assume that the surface integral formulations are the same in subvolumes Ω_1 and Ω_2. So after adding up the first two conservation laws we can cancel the internal surface integrals between Ω_1 and Ω_2. Therefore we obtain the global conservation law after adding up the three subvolume conservation laws. This is the essential property that has to be satisfied by the numerical discretisation of the flux contributions in order for a scheme to be conservative.

From the above discussion we know that to construct an integral form conservative scheme we need the requirements for the geometry subvolumes, which can be called the control volumes or cells, and the calculations of the discretised fluxes are carried out as follows:

(1) The sum of the control volumes should cover the whole domain Ω;

(2) Adjacent control volumes may overlap if each internal surface is common to two control volumes;

(3) Fluxes along a control volume surface have to be computed by formulas independent of the control volume in which they are considered.

The finite volume method is a conservative scheme with the surface integral replaced by the sum of integrals over the faces of the control volume, which are further replaced by the product of the fluxes on the faces and the area of the faces, and all spatial integrals replaced by the product of the spatial quantity and the average value of the integrand. The method takes full advantage of an arbitrary mesh, where a large number of options are open for the definition of the control volumes around which the conservation laws are expressed. Modifying the shape and location of the control volumes associated with a given mesh point, as well as varying the rules and accuracy for the evaluation of the fluxes through the control surfaces, gives considerable flexibility to the finite volume method.

A cell centred finite volume method, which is employed in this paper, is defined so that the discretised *physical* state variables are associated with the control volume, i.e., in 3-dimensional space we can define that $\mathbf{Q}_{ijk}$ is the average value of $\mathbf{Q}$ in the control volume so that

$$\int_{\Omega_{ijk}} \mathbf{Q}\, d\Omega = \mathbf{Q}_{ijk}\, \Omega_{ijk} \tag{2.3.3}$$

In order to calculate a surface flux it is convenient to think of $\mathbf{Q}_{ijk}$ as the value of $\mathbf{Q}$ at some average point in the cell, with the = sign replaced by the $\approx$ sign. A characteristic of the finite volume method is that the precise location of this average point is not required during the calculation. Only in the output of the solution is the location of this point desired.

The discretised equations in a cell are given as follows

$$\frac{\partial}{\partial t}(\mathbf{Q}_{ijk}\Omega_{ijk}) + \sum_{\text{surfaces}} \overline{\mathbf{F}} \cdot \Delta\vec{s} = \mathbf{S}_{ijk}\Omega_{ijk} \tag{2.3.4}$$

Eq.(2.3.4) is a compact form and includes five sub-equations. Then the major task in the finite volume method is to evaluate the fluxes through the control surfaces. In the following we will evaluate inviscid and viscous fluxes respectively.

2.4 Some general properties of Euler equations

The reason for discussing the properties of Euler equations is to develop formulations necessary for deriving the high resolution discretisation method for evaluating the inviscid flux on a cell interface.

We have (1) the differential form Euler equations in Cartesian coordinates:

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{E}_i}{\partial x_1} + \frac{\partial \mathbf{F}_i}{\partial x_2} + \frac{\partial \mathbf{G}_i}{\partial x_3} = \mathbf{S} \tag{2.4.1}$$

and (2) the integral form Euler equations:

$$\frac{\partial}{\partial t}\int_{\Omega} \mathbf{Q}\, d\Omega + \oint_{s} \overline{\mathbf{F}}_i \cdot d\vec{s} = \int_{\Omega} \mathbf{S}\, d\Omega \tag{2.4.2}$$

It is assumed the fluid satisfy the relation

$$p = \rho f(e) \tag{2.4.3}$$

where e is the internal energy. Then we have

$$\mathbf{E}_i = A\mathbf{Q} = \frac{\partial \mathbf{E}_i}{\partial \mathbf{Q}}\mathbf{Q}\ , \quad \mathbf{F}_i = B\mathbf{Q} = \frac{\partial \mathbf{F}_i}{\partial \mathbf{Q}}\mathbf{Q}\ , \quad \mathbf{G}_i = C\mathbf{Q} = \frac{\partial \mathbf{G}_i}{\partial \mathbf{Q}}\mathbf{Q} \tag{2.4.4}$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ -v_1^2+\frac{\gamma-1}{2}\vec{v}^2 & (3-\gamma)v_1 & -(\gamma-1)v_2 & -(\gamma-1)v_3 & \gamma-1 \\ -v_1v_2 & v_2 & v_1 & 0 & 0 \\ -v_1v_3 & v_3 & 0 & v_1 & 0 \\ -v_1(\gamma E-(\gamma-1)\vec{v}^2) & \gamma E-\frac{\gamma-1}{2}(\vec{v}^2+2v_1^2) & -(\gamma-1)v_1v_2 & -(\gamma-1)v_1v_3 & \gamma v_1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ -v_1v_2 & v_2 & v_1 & 0 & 0 \\ -v_2^2+\frac{\gamma-1}{2}\vec{v}^2 & -(\gamma-1)v_1 & (3-\gamma)v_2 & -(\gamma-1)v_3 & \gamma-1 \\ -v_2v_3 & 0 & v_3 & v_2 & 0 \\ -v_2(\gamma E-(\gamma-1)\vec{v}^2) & -(\gamma-1)v_1v_2 & \gamma E-\frac{\gamma-1}{2}(\vec{v}^2+2v_2^2) & -(\gamma-1)v_2v_3 & \gamma v_2 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ -v_1v_3 & v_3 & 0 & v_1 & 0 \\ -v_2v_3 & 0 & v_3 & v_2 & 0 \\ -v_3^2+\frac{\gamma-1}{2}\vec{v}^2 & -(\gamma-1)v_1 & -(\gamma-1)v_2 & (3-\gamma)v_3 & \gamma-1 \\ -v_3(\gamma E-(\gamma-1)\vec{v}^2) & -(\gamma-1)v_1v_3 & -(\gamma-1)v_2v_3 & \gamma E-\frac{\gamma-1}{2}(\vec{v}^2+2v_3^2) & \gamma v_3 \end{bmatrix}$$

This results in the quasi-linear Euler equations

$$\frac{\partial \mathbf{Q}}{\partial t} + A\frac{\partial \mathbf{Q}}{\partial x_1} + B\frac{\partial \mathbf{Q}}{\partial x_2} + C\frac{\partial \mathbf{Q}}{\partial x_3} = \mathbf{S} \tag{2.4.5}$$

and

$$\frac{\partial}{\partial t}\int_\Omega \mathbf{Q}\, d\Omega + \oint_s \overline{A}\cdot d\vec{s}\,\mathbf{Q} = \int_\Omega \mathbf{S}\, d\Omega \tag{2.4.6}$$

where $\overline{A} = (A,B,C)$.

For any unit vector

$$\vec{k} = (k_{x_1}, k_{x_2}, k_{x_3}) \tag{2.4.7}$$

we can calculate the eigenvalues λ_i and their corresponding right column eigenvectors R_i of matrix $(Ak_{x_1}+Bk_{x_2}+Ck_{x_3})$, i = 1,2,...,5. Assume the λ_i have been labelled in increasing order, i.e., $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_5$, and matrix P is composed of the eigenvectors R_i , i.e., $P = [R_1, R_2, ..., R_5]$ we have

$$P^{-1}(\overline{A}\cdot\vec{k})P = \Lambda \tag{2.4.8}$$

where

$$P = \begin{bmatrix} \frac{\rho}{2c} & k_{x_1} & k_{x_2} & k_{x_3} & \frac{\rho}{2c} \\ \frac{\rho}{2c}(v_1 - k_{x_1}c) & v_1k_{x_1} & v_1k_{x_2}-\rho k_{x_3} & v_1k_{x_3}+\rho k_{x_2} & \frac{\rho}{2c}(v_1+k_{x_1}c) \\ \frac{\rho}{2c}(v_2 - k_{x_2}c) & v_2k_{x_1}+\rho k_{x_3} & v_2k_{x_2} & v_2k_{x_3}-\rho k_{x_1} & \frac{\rho}{2c}(v_2+k_{x_2}c) \\ \frac{\rho}{2c}(v_3 - k_{x_3}c) & v_3k_{x_1}-\rho k_{x_2} & v_3k_{x_2}+\rho k_{x_1} & v_3k_{x_3} & \frac{\rho}{2c}(v_3+k_{x_3}c) \\ \frac{\rho}{2c}(H - c\vec{v}\cdot\vec{k}) & \vec{b}\cdot\vec{1}_{x_1} & \vec{b}\cdot\vec{1}_{x_2} & \vec{b}\cdot\vec{1}_{x_3} & \frac{\rho}{2c}(H+c\vec{v}\cdot\vec{k}) \end{bmatrix}$$

$$P^{-1} = \begin{bmatrix} \frac{c}{\rho}(\frac{\gamma-1}{2}\frac{\vec{v}^2}{c^2}+\frac{\vec{v}\cdot\vec{k}}{c}) & \vec{C}_-\cdot\vec{1}_{x_1} & \vec{C}_-\cdot\vec{1}_{x_2} & \vec{C}_-\cdot\vec{1}_{x_3} & \frac{\gamma-1}{\rho c} \\ \vec{B}_0\cdot\vec{1}_{x_1} & (\gamma-1)\frac{v_1}{c^2}k_{x_1} & (\gamma-1)\frac{v_2}{c^2}k_{x_1}+\frac{k_{x_3}}{\rho} & (\gamma-1)\frac{v_3}{c^2}k_{x_1}-\frac{k_{x_2}}{\rho} & \frac{\gamma-1}{\rho^2}k_{x_1} \\ \vec{B}_0\cdot\vec{1}_{x_2} & (\gamma-1)\frac{v_1}{c^2}k_{x_2}-\frac{k_{x_3}}{\rho} & (\gamma-1)\frac{v_2}{c^2}k_{x_2} & (\gamma-1)\frac{v_3}{c^2}k_{x_2}+\frac{k_{x_1}}{\rho} & \frac{\gamma-1}{\rho^2}k_{x_2} \\ \vec{B}_0\cdot\vec{1}_{x_3} & (\gamma-1)\frac{v_1}{c^2}k_{x_3}+\frac{k_{x_2}}{\rho} & (\gamma-1)\frac{v_2}{c^2}k_{x_3}-\frac{k_{x_1}}{\rho} & (\gamma-1)\frac{v_3}{c^2}k_{x_3} & \frac{\gamma-1}{\rho^2}k_{x_3} \\ \frac{c}{\rho}(\frac{\gamma-1}{2}\frac{\vec{v}^2}{c^2}-\frac{\vec{v}\cdot\vec{k}}{c}) & \vec{C}_+\cdot\vec{1}_{x_1} & \vec{C}_+\cdot\vec{1}_{x_2} & \vec{C}_+\cdot\vec{1}_{x_3} & \frac{\gamma-1}{\rho c} \end{bmatrix}$$

and

$$\Lambda = \begin{bmatrix} \vec{v}\cdot\vec{k} - ck & & & & \\ & \vec{v}\cdot\vec{k} & & & \\ & & \vec{v}\cdot\vec{k} & & \\ & & & \vec{v}\cdot\vec{k} & \\ & & & & \vec{v}\cdot\vec{k} + ck \end{bmatrix}$$

$$\vec{b} = \frac{\vec{v}^2}{2}\vec{k} + \rho\,(\vec{v}\times\vec{k})$$

$$\vec{B}_0 = \left(1 - \frac{\gamma-1}{2}\frac{\vec{v}^2}{c^2}\right)\vec{k} - \frac{1}{\rho}(\vec{v}\times\vec{k})$$

$$\vec{C}_\pm = \pm\frac{\vec{k}}{\rho} - \frac{\gamma-1}{\rho c}\vec{v}$$

$$\vec{1}_{x_1} = (1, 0, 0), \qquad \vec{1}_{x_2} = (0, 1, 0), \qquad \vec{1}_{x_3} = (0, 0, 1)$$

Let primitive variables be defined as

$$\mathbf{V} = \begin{bmatrix} \rho \\ v_1 \\ v_2 \\ v_3 \\ p \end{bmatrix}. \tag{2.4.9}$$

Let $M = \delta Q/\delta V$, where δ presents an arbitrary variation (either ∂_t, or ∇). Then we have

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ v_1 & \rho & 0 & 0 & 0 \\ v_2 & 0 & \rho & 0 & 0 \\ v_3 & 0 & 0 & \rho & 0 \\ \frac{\vec{v}^2}{2} & \rho v_1 & \rho v_2 & \rho v_3 & \frac{1}{\gamma-1} \end{bmatrix}$$

$$M^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{-v_1}{\rho} & \frac{1}{\rho} & 0 & 0 & 0 \\ \frac{-v_2}{\rho} & 0 & \frac{1}{\rho} & 0 & 0 \\ \frac{-v_3}{\rho} & 0 & 0 & \frac{1}{\rho} & 0 \\ (\gamma-1)\frac{\vec{v}^2}{2} & -(\gamma-1)v_1 & -(\gamma-1)v_2 & -(\gamma-1)v_3 & \gamma-1 \end{bmatrix}$$

Let $\delta \mathbf{W} = P^{-1}\,\delta \mathbf{Q}$, where $\mathbf{W}$ is the characteristic variable vector. Then we have $\delta \mathbf{W} = P^{-1} M\,\delta \mathbf{V} = L^{-1}\,\delta \mathbf{V}$, where

$$L = \begin{bmatrix} \frac{\rho}{2c} & k_{x_1} & k_{x_2} & k_{x_3} & \frac{\rho}{2c} \\ \frac{-k_{x_1}}{2} & 0 & -k_{x_3} & k_{x_2} & \frac{k_{x_1}}{2} \\ \frac{-k_{x_2}}{2} & k_{x_3} & 0 & -k_{x_1} & \frac{k_{x_2}}{2} \\ \frac{-k_{x_3}}{2} & -k_{x_2} & k_{x_1} & 0 & \frac{k_{x_3}}{2} \\ \frac{\rho c}{2} & 0 & 0 & 0 & \frac{\rho c}{2} \end{bmatrix}$$

$$L^{-1} = \begin{bmatrix} 0 & -k_{x_1} & -k_{x_2} & -k_{x_3} & \frac{1}{\rho c} \\ k_{x_1} & 0 & k_{x_3} & -k_{x_2} & \frac{-k_{x_1}}{c^2} \\ k_{x_2} & -k_{x_3} & 0 & k_{x_1} & \frac{-k_{x_2}}{c^2} \\ k_{x_3} & k_{x_2} & -k_{x_1} & 0 & \frac{-k_{x_3}}{c^2} \\ 0 & k_{x_1} & k_{x_2} & k_{x_3} & \frac{1}{\rho c} \end{bmatrix}.$$

Therefore we have the relationship between the variation of the characteristic variables and the variation of the primitive variables as follows

$$\begin{bmatrix} \delta w_1 \\ \delta w_2 \\ \delta w_3 \\ \delta w_4 \\ \delta w_5 \end{bmatrix} = \begin{bmatrix} -(k_{x_1}\delta v_1 + k_{x_2}\delta v_2 + k_{x_3}\delta v_3) + \frac{\delta p}{\rho c} \\ k_{x_1}\delta\rho + k_{x_3}\delta v_2 - k_{x_2}\delta v_3 - k_{x_1}\frac{\delta p}{c^2} \\ k_{x_2}\delta\rho - k_{x_3}\delta v_1 + k_{x_1}\delta v_3 - k_{x_2}\frac{\delta p}{c^2} \\ k_{x_3}\delta\rho - k_{x_2}\delta v_1 - k_{x_1}\delta v_2 - k_{x_3}\frac{\delta p}{c^2} \\ k_{x_1}\delta v_1 + k_{x_2}\delta v_2 + k_{x_3}\delta v_3 + \frac{\delta p}{\rho c} \end{bmatrix} \qquad (2.4.10)$$

Not to lose generality, we can assume $k_{x_3} \neq 0$, and the Riemann invariants corresponding to the eigenvector R_1 are

$$\alpha_1 = \vec{v} \cdot \vec{k} + \frac{2}{r-1} ck\,, \quad \alpha_2 = v_2 k_{x_3} - v_3 k_{x_2}\,,$$
$$\alpha_3 = v_1 k_{x_3} - v_3 k_{x_1}\,, \quad \alpha_4 = s\,, \tag{2.4.11}$$

those corresponding with the eigenvector R_2, R_3, R_4 are

$$\beta_1 = \vec{v} \cdot \vec{k}\,, \quad \beta_2 = p \tag{2.4.12}$$

and those corresponding with the eigenvector R_5 are

$$\gamma_1 = \vec{v} \cdot \vec{k} - \frac{2}{r-1} ck\,, \quad \gamma_2 = v_2 k_{x_3} - v_3 k_{x_2}\,,$$
$$\gamma_3 = v_1 k_{x_3} - v_3 k_{x_1}\,, \quad \gamma_4 = s\,, \tag{2.4.13}$$

where $s = c_v \ln (p / \rho^{\gamma})$.

2.5 Upwind schemes for Euler equations

For the Euler equations we have to solve exactly the Riemann problem. This requires the solution of a non-linear equation at each control volume interface which is quite time consuming. Since the exact solution is averaged over the control volume, we can also consider approximate Riemann solutions which would require less computational work. Osher and Roe derived the two most useful approximate Riemann solvers. They are referred to sometimes as flux difference splitting methods and we will refer to the family of methods which call on exact or approximate local properties of basic solutions to the Euler equations as Godunov-type methods.

Consider the following equation

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} = 0 \tag{2.5.1}$$

with the initial conditions

$$\mathbf{Q}(x,0) = \begin{cases} \mathbf{Q}^L & x<0 \\ \mathbf{Q}^R & x>0 \end{cases}$$

and $d\mathbf{F} / d\mathbf{Q} = A(\mathbf{Q})$.

2.5.1 Osher's approximate Riemann solver

Suppose there exist functions $\mathbf{F}^+(\mathbf{Q})$ and $\mathbf{F}^-(\mathbf{Q})$ such that $\mathbf{F}(\mathbf{Q}) = \mathbf{F}^+(\mathbf{Q}) + \mathbf{F}^-(\mathbf{Q})$, $d\mathbf{F}^+ / d\mathbf{Q} = A^+(\mathbf{Q})$ and $d\mathbf{F}^- / d\mathbf{Q} = A^-(\mathbf{Q})$. Osher defined the approximate Riemann solver as

$$\mathbf{F}^{(A)}(\mathbf{Q}^L,\mathbf{Q}^R) = \mathbf{F}^+(\mathbf{Q}^L) + \mathbf{F}^-(\mathbf{Q}^R) \tag{2.5.2}$$

then

$$\mathbf{F}^{(A)}(\mathbf{Q}^L,\mathbf{Q}^R) = \mathbf{F}(\mathbf{Q}^L) - \mathbf{F}^-(\mathbf{Q}^L) + \mathbf{F}^-(\mathbf{Q}^R) = \mathbf{F}(\mathbf{Q}^L) + \int_{Q^L}^{Q^R} A^-(\mathbf{Q})\, d\mathbf{Q}$$

$$= \mathbf{F}^+(\mathbf{Q}^L) + \mathbf{F}(\mathbf{Q}^R) - \mathbf{F}^+(\mathbf{Q}^R) = \mathbf{F}(\mathbf{Q}^R) - \int_{Q^L}^{Q^R} A^+(\mathbf{Q})\, d\mathbf{Q}$$

$$= \frac{1}{2}\left[\mathbf{F}(\mathbf{Q}^L) + \mathbf{F}(\mathbf{Q}^R) - \int_{Q^L}^{Q^R} |A(\mathbf{Q})|\, d\mathbf{Q}\right]$$

where the phase space integrals are independent of the integration path.

Unfortunately, in the general non-linear case no function $\mathbf{F}^+(\mathbf{Q})$ and $\mathbf{F}^-(\mathbf{Q})$ exist; this is equivalent to saying that the phase space integrals

$$\int_{Q^L}^{Q^R} A^-(\mathbf{Q})\, d\mathbf{Q} \quad \text{and} \quad \int_{Q^L}^{Q^R} A^+(\mathbf{Q})\, d\mathbf{Q}$$

depend on the integration path.

For the Euler equations the Osher's scheme can be described as when the integration path in phase space from $\mathbf{Q}^L$ to $\mathbf{Q}^R$ is split over all simple wave solutions, associated with the eigenvalue and the the right eigenvector.

We will present the 3-dimensional Osher's scheme in the following. Suppose there is a small surface s in the 3-dimension space with the normal

$$\vec{k} = (k_{x_1}, k_{x_2}, k_{x_3}) \tag{2.5.3}$$

we assume $k_{x_3} \neq 0$ and $k = |\vec{k}|$, then we have

$$\begin{aligned}\overline{\mathbf{F}}_i \cdot \vec{k} &= \mathbf{E}_i k_{x_1} + \mathbf{F}_i k_{x_2} + \mathbf{G}_i k_{x_3} \\ &= (\overline{A} \cdot \vec{k})\,\mathbf{Q} = \frac{d(\overline{\mathbf{F}}_i \cdot \vec{k})}{d\mathbf{Q}}\mathbf{Q}\end{aligned} \tag{2.5.4}$$

Let $\mathbf{Q}^L$ be the value of $\mathbf{Q}$ in the negative direction of s and $\mathbf{Q}^R$ be the value of the $\mathbf{Q}$ in the positive direction of s.

The Osher's approximate Riemann solver on surface s is:

$$\begin{aligned}\left(\overline{\mathbf{F}}_i \cdot \vec{k}\right)_s &= \overline{\mathbf{F}}_i(\mathbf{Q}^L) \cdot \vec{k} + \int_{Q^L}^{Q^R} (\overline{A}^- \cdot \vec{k})\, d\mathbf{Q} \\ &= \overline{\mathbf{F}}_i(\mathbf{Q}^R) \cdot \vec{k} - \int_{Q^L}^{Q^R} (\overline{A}^+ \cdot \vec{k})\, d\mathbf{Q} \\ &= \frac{1}{2}\left[\left(\overline{\mathbf{F}}_i(\mathbf{Q}^L) + \overline{\mathbf{F}}_i(\mathbf{Q}^R)\right) \cdot \vec{k} - \int_{Q^L}^{Q^R} \left|(\overline{A} \cdot \vec{k})\right| d\mathbf{Q}\right]\end{aligned} \tag{2.5.5}$$

Similar to the discussion of Spekreijse [8] we have the following results. Suppose that the states $\mathbf{Q}^L$ and $\mathbf{Q}^R$ can be connected with each other by an integral path Γ_k which is tangential to the eigenvector R_k, i.e.,

$$\begin{cases} \dfrac{d\mathbf{Q}}{d\xi}(\xi) = R_k(\mathbf{Q}(\xi)) \\ \mathbf{Q}(0) = \mathbf{Q}^L; \quad \mathbf{Q}(\xi_R) = \mathbf{Q}^R \end{cases} \tag{2.5.6}$$

Then, we see that

$$\begin{aligned}\int_{Q^L}^{Q^R} (\overline{A}^- \cdot \vec{k})\, d\mathbf{Q} &= \int_0^{\xi_R} (\overline{A}^- \cdot \vec{k}) \frac{d\mathbf{Q}}{d\xi} d\xi \\ = \int_0^{\xi_R} (\overline{A}^- \cdot \vec{k})\, R_k(\mathbf{Q}(\xi))\, d\xi &= \int_0^{\xi_R} \lambda_k^-(\mathbf{Q}(\xi))\, R_k(\mathbf{Q}(\xi))\, d\xi\end{aligned} \tag{2.5.7}$$

Then we consider two eventualities

A $\lambda_k(\mathbf{Q}(\xi))$ does not change sign along the integral path.

If $\lambda_k(\mathbf{Q}(\xi)) \geq 0 \;\; \forall \xi \in (0, \xi_R)$ then

$$\int_{Q^L}^{Q^R} (\bar{A}^- \cdot \vec{k})\, dQ = 0 \tag{2.5.8}$$

If $\lambda_k(Q(\xi)) \le 0 \ \forall \xi \in (0, \xi_R)$ then

$$\int_{Q^L}^{Q^R} (\bar{A}^- \cdot \vec{k})\, dQ = \int_0^{\xi_R} \lambda_k^-(Q(\xi))\, R_k(Q(\xi))\, d\xi$$

$$= \int_0^{\xi_R} \lambda_k(Q(\xi))\, R_k(Q(\xi))\, d\xi = \int_0^{\xi_R} (\bar{A} \cdot \vec{k})\, R_k(Q(\xi))\, d\xi \tag{2.5.9}$$

$$= \int_0^{\xi_R} \frac{d(\bar{F}_i \cdot \vec{k})}{dQ} \frac{dQ}{d\xi}\, d\xi = \bar{F}_i(Q^R) \cdot \vec{k} - \bar{F}_i(Q^L) \cdot \vec{k}$$

B $\lambda_k(Q(\xi))$ changes sign along the integral path.

Suppose $\lambda_k(Q(\xi))$ changes sign only once at $\xi = \xi_s$, $0<\xi_s<\xi_R$. Define $Q^s = Q(\xi_s)$.

If $\lambda_k(Q(\xi)) \ge 0 \ \forall \xi \in (0, \xi_s)$ and $\lambda_k(Q(\xi)) \le 0 \ \forall \xi \in (\xi_s, \xi_R)$ then

$$\int_{Q^L}^{Q^R} (\bar{A}^- \cdot \vec{k})\, dQ = \int_0^{\xi_R} \lambda_k^-(Q(\xi))\, R_k(Q(\xi))\, d\xi$$

$$= \int_{\xi_s}^{\xi_R} \lambda_k(Q(\xi))\, R_k(Q(\xi))\, d\xi = \bar{F}_i(Q^R) \cdot \vec{k} - \bar{F}_i(Q^s) \cdot \vec{k} \tag{2.5.10}$$

If $\lambda_k(Q(\xi)) \le 0 \ \forall \xi \in (0, \xi_s)$ and $\lambda_k(Q(\xi)) \ge 0 \ \forall \xi \in (\xi_s, \xi_R)$ then

$$\int_{Q^L}^{Q^R} (\bar{A}^- \cdot \vec{k})\, dQ = \int_0^{\xi_R} \lambda_k^-(Q(\xi))\, R_k(Q(\xi))\, d\xi$$

$$= \int_0^{\xi_s} \lambda_k(Q(\xi))\, R_k(Q(\xi))\, d\xi = \bar{F}_i(Q^s) \cdot \vec{k} - \bar{F}_i(Q^L) \cdot \vec{k} \tag{2.5.11}$$

Because at this point the pair (Q^L, Q^R) can be connected by a continuous integral path Γ which is decomposed into 5 subcurves Γ_k:

$$\Gamma = \bigcup_{k=1}^{5} \Gamma_k \tag{2.5.12}$$

where each subcurve Γ_k is tangential to the eigenvector R_k. The subcurve Γ_1 starts in $\mathbf{Q}^L \equiv \mathbf{Q}^0$ and the subcurve Γ_5 ends in $\mathbf{Q}^R \equiv \mathbf{Q}^1$. Defining the 4 points of intersection $\mathbf{Q}^{k/5}$, k=1,...,4 by

$$\mathbf{Q}^{k/5} = \Gamma_k \cap \Gamma_{k+1}\,, \tag{2.5.13}$$

the intersection points are then easily found with the use of Riemann invariants.

Because we have

$$\frac{d}{d\xi}\lambda_k(\mathbf{Q}(\xi)) = \nabla\lambda_k(\mathbf{Q}(\xi))\frac{d\mathbf{Q}(\xi)}{d\xi}(\xi) = \nabla\lambda_k(\mathbf{Q}(\xi))\mathbf{R}_k(\mathbf{Q}(\xi)) \tag{2.5.14}$$

then $\lambda_2 = \lambda_3 = \lambda_4 = \vec{v} \cdot \vec{k}$ makes (2.5.14) equal to zero, and we have

$$\begin{aligned} &\int_{\Gamma_2\cup\Gamma_3\cup\Gamma_4} (\bar{A}^- \cdot \vec{k})\, d\mathbf{Q} \\ &= \int_{\Gamma_2} (\bar{A}^- \cdot \vec{k})\, d\mathbf{Q} + \int_{\Gamma_3} (\bar{A}^- \cdot \vec{k})\, d\mathbf{Q} + \int_{\Gamma_4} (\bar{A}^- \cdot \vec{k})\, d\mathbf{Q} \\ &= \begin{cases} \bar{\mathbf{F}}_i(\mathbf{Q}^{4/5}) \cdot \vec{k} - \bar{\mathbf{F}}_i(\mathbf{Q}^{1/5}) \cdot \vec{k} & \text{if } \vec{v} \cdot \vec{k} < 0 \\ 0 & \text{if } \vec{v} \cdot \vec{k} > 0\,. \end{cases} \end{aligned} \tag{2.5.15}$$

So we need only calculate $\mathbf{Q}^{1/3} = \mathbf{Q}^{1/5}$, $\mathbf{Q}^{2/3} = \mathbf{Q}^{4/5}$.

Let $z = \ln(p / \rho^\gamma)$, then following the discussion of Spekreijse [8] we have: $u^0 = \vec{v}^0 \cdot \vec{k}$, $u^{1/3} = \vec{v}^{1/3} \cdot \vec{k}$, $u^{2/3} = \vec{v}^{2/3} \cdot \vec{k}$, $u^1 = \vec{v}^1 \cdot \vec{k}$, and

$$\begin{aligned} u^0 + \frac{2}{\gamma-1} c^0 k &= u^0 + \frac{2}{\gamma-1} c^0 k \equiv \psi^0 \\ v_2^0 k_{x_3} - v_3^0 k_{x_2} &= v_2^{1/3} k_{x_3} - v_3^{1/3} k_{x_2} \\ v_1^0 k_{x_3} - v_3^0 k_{x_1} &= v_1^{1/3} k_{x_3} - v_3^{1/3} k_{x_1} \\ z^0 &= z^{1/3} \end{aligned} \tag{2.5.16}$$

$$\begin{aligned} u^{1/3} = u^{2/3} &\equiv u^H \\ p^{1/3} &= p^{2/3} \end{aligned} \tag{2.5.17}$$

$$
\begin{aligned}
u^{2/3} - \frac{2}{\gamma-1} c^{2/3} k &= u^1 - \frac{2}{\gamma-1} c^1 k \equiv \psi^1 \\
v_2^{2/3} k_{x_3} - v_3^{2/3} k_{x_2} &= v_2^1 k_{x_3} - v_3^1 k_{x_2} \\
v_1^{2/3} k_{x_3} - v_3^{2/3} k_{x_1} &= v_1^1 k_{x_3} - v_3^1 k_{x_1} \\
z^{2/3} &= z^1
\end{aligned}
\tag{2.5.18}
$$

Because $\lambda_1 = \vec{v} \cdot \vec{k} - c\,k$, and $\lambda_5 = \vec{v} \cdot \vec{k} + c\,k$ make (2.5.14) non-zero, then a sonic point $\mathbf{Q}^0{}_s$ exists along Γ_1 when

$$
\lambda_1(\mathbf{Q}^0)\,\lambda_1(\mathbf{Q}^{1/3}) = (u^0 - c^0)\,(u^H - c^{1/3}) < 0 \tag{2.5.19}
$$

which can be found from

$$
\begin{aligned}
u^0 + \frac{2}{\gamma-1} c^0 k &= u^0{}_s + \frac{2}{\gamma-1} c^0{}_s k \\
v_2^0 k_{x_3} - v_3^0 k_{x_2} &= v_{2\,s}^0 k_{x_3} - v_{3\,s}^0 k_{x_2} \\
v_1^0 k_{x_3} - v_3^0 k_{x_1} &= v_{1\,s}^0 k_{x_3} - v_{3\,s}^0 k_{x_1} \\
z^0 &= z^0{}_s \\
u^0{}_s - c^0{}_s &= 0 .
\end{aligned}
\tag{2.5.20}
$$

Furthermore a sonic point $\mathbf{Q}^1{}_s$ exists along Γ_5 when

$$
\lambda_5(\mathbf{Q}^{2/3})\,\lambda_5(\mathbf{Q}^1) = (u^H + c^{2/3})\,(u^1 + c^1) < 0 \tag{2.5.21}
$$

which can be found from

$$
\begin{aligned}
u^1{}_s - \frac{2}{\gamma-1} c^1{}_s k &= u^1 - \frac{2}{\gamma-1} c^1 k \\
v_{2\,s}^1 k_{x_3} - v_{3\,s}^1 k_{x_2} &= v_2^1 k_{x_3} - v_3^1 k_{x_2} \\
v_{1\,s}^1 k_{x_3} - v_{3\,s}^1 k_{x_1} &= v_1^1 k_{x_3} - v_3^1 k_{x_1} \\
z^1{}_s &= z^1 \\
u^1{}_s + c^1{}_s &= 0 .
\end{aligned}
\tag{2.5.22}
$$

Let $a = c^2 / \gamma$, then $\rho = \exp((\ln(a) - z) / (\gamma - 1))$, $p = a\,\rho$, $\rho E = 1/2\,\rho\,v^2 + p / (\gamma-1)$. Let $\alpha = c_{2/3} / c_{1/3}$, then $\alpha = \exp((z_1 - z_0) / 2\gamma)$, and

$$
c_{1/3} = \frac{\gamma-1}{2} \frac{\psi_1 - \psi_0}{1+\alpha},\quad c_{2/3} = \alpha\, c_{1/3},\ \text{and}\ u_H = \frac{\psi_1 + \alpha\,\psi_0}{1+\alpha}.
$$

In this way we can obtain all the variables at all points.

If we let $\mathbf{F}(\mathbf{Q}) = \overline{\mathbf{F}}_i(\mathbf{Q}) \cdot \vec{k}$, then we have the Osher's approximate Riemann solver (2.5.5) described in the table below:

	$u_0<c_0$, $u_1>-c_1$	$u_0>c_0$, $u_1>-c_1$	$u_0<c_0$, $u_1<-c_1$	$u_0>c_0$, $u_1<-c_1$
$c_{1/3}<u_H$	$F(Q^0{}_s)$	$F(Q^0)$	$F(Q^0{}_s)-F(Q^1{}_s)$ $+F(Q^1)$	$F(Q^0)-F(Q^1{}_s)+F(Q^1)$
$0<u_H<c_{1/3}$	$F(Q^{1/3})$	$F(Q^0)-F(Q^0{}_s)$ $+F(Q^{1/3})$	$F(Q^{1/3})-F(Q^1{}_s)$ $+F(Q^1)$	$F(Q^0)-F(Q^0{}_s)+F(Q^{1/3})$ $-F(Q^1{}_s)+F(Q^1)$
$-c_{2/3}<u_H<0$	$F(Q^{2/3})$	$F(Q^0)-F(Q^0{}_s)$ $+F(Q^{2/3})$	$F(Q^{2/3})-F(Q^1{}_s)$ $+F(Q^1)$	$F(Q^0)-F(Q^0{}_s)+F(Q^{2/3})$ $-F(Q^1{}_s)+F(Q^1)$
$u_H<-c_{2/3}$	$F(Q^1{}_s)$	$F(Q^0)-F(Q^0{}_s)$ $+F(Q^1{}_s)$	$F(Q^1)$	$F(Q^0)-F(Q^0{}_s)+F(Q^1)$

2.5.2 Roe's approximate Riemann solver

As for the linear system a first-order upwind scheme can be written for equation (2.5.1)

$$\begin{aligned} \mathbf{F}^{(A)}(\mathbf{Q}^L,\mathbf{Q}^R) &= \mathbf{F}^{+L} + \mathbf{F}^{-R} \\ &= \frac{1}{2}\left[\mathbf{F}(\mathbf{Q}^L) + \mathbf{F}(\mathbf{Q}^R) - \left|\overline{\overline{A}}(\mathbf{Q}^L,\mathbf{Q}^R)\right|(\mathbf{Q}^R - \mathbf{Q}^L)\right] \end{aligned} \tag{2.5.23}$$

Considering the transformation from conservative to characteristic variables $\delta\mathbf{Q} = P\,\delta\mathbf{W}$, $\delta\mathbf{Q} = \mathbf{Q}^R - \mathbf{Q}^L$ can be expressed as a sum of simple wave contributions. The conservative property of the wave decomposition requires that the sum still reduces to a flux difference as in the linear case, i.e., we should have

$$\mathbf{F}(\mathbf{Q}^R) - \mathbf{F}(\mathbf{Q}^L) = \overline{\overline{A}}(\mathbf{Q}^L,\mathbf{Q}^R)(\mathbf{Q}^R - \mathbf{Q}^L) \tag{2.5.24}$$

In the general case a Jacobian matrix $\overline{\overline{A}}(\mathbf{Q}^L,\mathbf{Q}^R)$ should be defined with the following properties:

(1) For any pair $\mathbf{Q}^L$, $\mathbf{Q}^R$ one should have exactly the property (2.5.24);
(2) For $\mathbf{Q}^L = \mathbf{Q}^R = \mathbf{Q}$ the matrix $\overline{\overline{A}}(\mathbf{Q},\mathbf{Q}) = A(\mathbf{Q}) \equiv \partial\mathbf{F}/\partial\mathbf{Q}$;
(3) $\overline{\overline{A}}$ has real eigenvalues with linearly independent eigenvectors.

Once such a matrix is defined, the above wave decompositions can be written without any change. The eigenvalues of this matrix can be considered as the wave speeds of the approximate Riemann problem and the right eigenvectors as the associated waves.

Independently of the particular form of the $\overline{\overline{A}}$ matrix, its definition indicates the nature of the Riemann problem approximation it provides. Its eigenvalues C satisfy the relations

$$\mathbf{F}(\mathbf{Q}^R) - \mathbf{F}(\mathbf{Q}^L) = C(\mathbf{Q}^R - \mathbf{Q}^L) \tag{2.5.25}$$

which are identical to the Rankine-Hugoniot relations for a discontinuity of speed C between the states $\mathbf{Q}^L$ and $\mathbf{Q}^R$. The projection of the corresponding eigenvector represents the intensity of the jump over this discontinuity. Hence the approximate Riemann solver contained in the definition recognizes only, and exactly, discontinuities. A consequence of this fact is that the method will not be able to identify properly an expansion fan containing a sonic point and in particular a stationary expansion for which $\mathbf{F}^L = \mathbf{F}^R$ and $\mathbf{Q}^L \neq \mathbf{Q}^R$ will appear as an expansion shock.

We will now outline the 3-dimensional Roe's scheme. As in Osher's scheme suppose there is a small suface s in the 3-dimension space with the normal $\vec{k}$ on it. We have

$$\begin{aligned}(\overline{\mathbf{F}}_i \cdot \vec{k})_s &= \overline{\mathbf{F}}_i(\mathbf{Q}^L) \cdot \vec{k} + (\overline{\overline{A}}^- \cdot \vec{k})(\mathbf{Q}^R - \mathbf{Q}^L) \\ &= \overline{\mathbf{F}}_i(\mathbf{Q}^R) \cdot \vec{k} - (\overline{\overline{A}}^+ \cdot \vec{k})(\mathbf{Q}^R - \mathbf{Q}^L) \\ &= \frac{1}{2}\left[\left(\overline{\mathbf{F}}_i(\mathbf{Q}^L) + \overline{\mathbf{F}}_i(\mathbf{Q}^R)\right) \cdot \vec{k} - \left|(\overline{\overline{A}} \cdot \vec{k})\right| (\mathbf{Q}^R - \mathbf{Q}^L)\right].\end{aligned} \tag{2.5.26}$$

Now the main problem is to find the matrix $\overline{\overline{A}}$. Roe observes that the column vectors $\mathbf{Q}$ and $\mathbf{E}_i$, $\mathbf{F}_i$, $\mathbf{G}_i$ can be expressed as quadratic functions of the variable $\mathbf{Z}$ defined by

$$\mathbf{Z} = \sqrt{\rho}\begin{bmatrix}1\\ v_1\\ v_2\\ v_3\\ H\end{bmatrix}$$

$$\mathbf{Q} = \begin{bmatrix} z_1^2 \\ z_1 z_2 \\ z_1 z_3 \\ z_1 z_4 \\ z_1 z_5/\gamma + (\gamma-1)(z_2^2+z_3^2+z_4^2)/2\gamma \end{bmatrix}$$

$$\mathbf{E}_i = \begin{bmatrix} z_1 z_2 \\ (\gamma-1) z_1 z_5/\gamma + (\gamma+1) z_2^2/2\gamma - (\gamma-1)(z_3^2+z_4^2)/2\gamma \\ z_2 z_3 \\ z_2 z_4 \\ z_2 z_5 \end{bmatrix}$$

$$\mathbf{F}_i = \begin{bmatrix} z_1 z_3 \\ z_3 z_2 \\ (\gamma-1) z_1 z_5/\gamma + (\gamma+1) z_3^2/2\gamma - (\gamma-1)(z_2^2+z_4^2)/2\gamma \\ z_3 z_4 \\ z_3 z_5 \end{bmatrix}$$

$$\mathbf{G}_i = \begin{bmatrix} z_1 z_4 \\ z_4 z_2 \\ z_4 z_3 \\ (\gamma-1) z_1 z_5/\gamma + (\gamma+1) z_4^2/2\gamma - (\gamma-1)(z_2^2+z_3^2)/2\gamma \\ z_4 z_5 \end{bmatrix}$$

Hence one can apply the following identity for quadratic functions valid for arbitrary variations $\delta a_{i+1/2} = a_{i+1} - a_i$, where the overbar indicates an arithmetic average

$$\bar{a} = (a_{i+1} + a_i) / 2 \equiv a_{i+1/2}$$

$$\delta(ab)_{i+1/2} = \bar{a}\, \delta b_{i+1/2} + \bar{b}\, \delta a_{i+1/2} .$$

When applied to $\mathbf{Q}$ as given by equation above, we have identically

$$\mathbf{Q}^R - \mathbf{Q}^L \equiv \bar{\bar{B}}(\mathbf{Z}^R - \mathbf{Z}^L)$$

with

$$\bar{\bar{B}} = \begin{bmatrix} 2\bar{z}_1 & 0 & 0 & 0 & 0 \\ \bar{z}_2 & \bar{z}_1 & 0 & 0 & 0 \\ \bar{z}_3 & 0 & \bar{z}_1 & 0 & 0 \\ \bar{z}_4 & 0 & 0 & \bar{z}_1 & 0 \\ \frac{1}{\gamma}\bar{z}_5 & \frac{\gamma-1}{\gamma}\bar{z}_2 & \frac{\gamma-1}{\gamma}\bar{z}_3 & \frac{\gamma-1}{\gamma}\bar{z}_4 & \frac{1}{\gamma}\bar{z}_1 \end{bmatrix}.$$

An analogous elementary calculation gives for the flux difference the identity

$$\mathbf{E}_i^R - \mathbf{E}_i^L \equiv \bar{\bar{C}}_1(\mathbf{Z}^R - \mathbf{Z}^L)$$
$$\mathbf{F}_i^R - \mathbf{F}_i^L \equiv \bar{\bar{C}}_2(\mathbf{Z}^R - \mathbf{Z}^L)$$
$$\mathbf{G}_i^R - \mathbf{G}_i^L \equiv \bar{\bar{C}}_3(\mathbf{Z}^R - \mathbf{Z}^L)$$

with

$$\overline{\overline{C}}_1 = \begin{bmatrix} \bar{z}_2 & \bar{z}_1 & 0 & 0 & 0 \\ \frac{\gamma-1}{\gamma}\bar{z}_5 & \frac{\gamma+1}{\gamma}\bar{z}_2 & \frac{\gamma-1}{\gamma}\bar{z}_3 & \frac{\gamma-1}{\gamma}\bar{z}_4 & \frac{\gamma-1}{\gamma}\bar{z}_1 \\ 0 & \bar{z}_3 & \bar{z}_2 & 0 & 0 \\ 0 & \bar{z}_4 & 0 & \bar{z}_2 & 0 \\ 0 & \bar{z}_5 & 0 & 0 & \bar{z}_2 \end{bmatrix}$$

$$\overline{\overline{C}}_2 = \begin{bmatrix} \bar{z}_3 & 0 & \bar{z}_1 & 0 & 0 \\ 0 & \bar{z}_3 & \bar{z}_2 & 0 & 0 \\ \frac{\gamma-1}{\gamma}\bar{z}_5 & \frac{\gamma-1}{\gamma}\bar{z}_2 & \frac{\gamma+1}{\gamma}\bar{z}_3 & \frac{\gamma-1}{\gamma}\bar{z}_4 & \frac{\gamma-1}{\gamma}\bar{z}_1 \\ 0 & 0 & \bar{z}_4 & \bar{z}_3 & 0 \\ 0 & 0 & \bar{z}_5 & 0 & \bar{z}_3 \end{bmatrix}$$

$$\overline{\overline{C}}_3 = \begin{bmatrix} \bar{z}_4 & 0 & 0 & \bar{z}_1 & 0 \\ 0 & \bar{z}_4 & 0 & \bar{z}_2 & 0 \\ 0 & 0 & \bar{z}_4 & \bar{z}_3 & 0 \\ \frac{\gamma-1}{\gamma}\bar{z}_5 & \frac{\gamma-1}{\gamma}\bar{z}_2 & \frac{\gamma-1}{\gamma}\bar{z}_3 & \frac{\gamma+1}{\gamma}\bar{z}_4 & \frac{\gamma-1}{\gamma}\bar{z}_1 \\ 0 & 0 & 0 & \bar{z}_5 & \bar{z}_4 \end{bmatrix}$$

$$\overline{\overline{A}} = (\overline{\overline{C}}_1 k_{x_1} + \overline{\overline{C}}_2 k_{x_2} + \overline{\overline{C}}_3 k_{x_3})\, \overline{\overline{B}}^{-1} \qquad (2.5.27)$$

$$\overline{\overline{B}}^{-1} = \begin{bmatrix} \frac{1}{2\bar{z}_1} & 0 & 0 & 0 & 0 \\ \frac{\bar{z}_2}{2\bar{z}_1^2} & \frac{1}{\bar{z}_1} & 0 & 0 & 0 \\ \frac{\bar{z}_3}{2\bar{z}_1^2} & 0 & \frac{1}{\bar{z}_1} & 0 & 0 \\ \frac{\bar{z}_4}{2\bar{z}_1^2} & 0 & 0 & \frac{1}{\bar{z}_1} & 0 \\ b_{51}^{-1} & -(\gamma-1)\frac{\bar{z}_2}{\bar{z}_1^2} & -(\gamma-1)\frac{\bar{z}_3}{\bar{z}_1^2} & -(\gamma-1)\frac{\bar{z}_4}{\bar{z}_1^2} & \frac{\gamma}{\bar{z}_1} \end{bmatrix}$$

$$b_{51}^{-1} = \frac{\bar{z}_5 - (\gamma-1)(\bar{z}_2^2+\bar{z}_3^2+\bar{z}_4^2)/\bar{z}_1}{2\bar{z}_1^2} .$$

A straightforward calculation shows a very remarkable result: matrix $\overline{\overline{A}}$ is identical to the local Jacobian given in 2.4, when expressed as a function of the variables v_1, v_2, v_3, and H, if these variables are replaced by an average weighted by the square root of the densities.

These particular averages are defined by setting $R_{1/2} = \sqrt{\rho_R/\rho_L}$:

$$\begin{aligned} \overline{\overline{\rho}} &= \sqrt{\rho_R \rho_L} \equiv R_{1/2}\,\rho_L \\ \overline{\overline{v_i}} = \frac{\bar{z}_{i+1}}{\bar{z}_1} &= \frac{(v_i\sqrt{\rho})_R + (v_i\sqrt{\rho})_L}{(\sqrt{\rho})_R + (\sqrt{\rho})_L} \equiv \frac{R_{1/2}\,(v_i)_R + (v_i)_L}{R_{1/2}+1} \\ \overline{\overline{H}} = \frac{\bar{z}_5}{\bar{z}_1} &= \frac{(H\sqrt{\rho})_R + (H\sqrt{\rho})_L}{(\sqrt{\rho})_R + (\sqrt{\rho})_L} \equiv \frac{R_{1/2}\,(H)_R + (H)_L}{R_{1/2}+1} \end{aligned} \qquad (2.5.28)$$

where i = 1,2,3.

The eigenvectors and eigenvalues of the linearized matrix $\overline{\overline{A}}$ can now be obtained without further calculations.

Roe's scheme is therefore completely defined and can be summarized as follows:

(1) For each boundary between cell L and R, calculate the above averaged values as well as the associated averaged speed of sound by

$$\overline{\overline{c}}^2 = (\gamma-1)\left(\overline{\overline{H}} - \frac{\vec{\overline{\overline{v}}}^2}{2}\right) ;$$

(2) Calculate the eigenvalues

$$\overline{\overline{\lambda}}_1 = \vec{\overline{\overline{v}}} \cdot \vec{k} - \overline{\overline{c}}\,k \qquad \overline{\overline{\lambda}}_2 = \overline{\overline{\lambda}}_3 = \overline{\overline{\lambda}}_4 = \vec{\overline{\overline{v}}} \cdot \vec{k} \qquad \overline{\overline{\lambda}}_5 = \vec{\overline{\overline{v}}} \cdot \vec{k} + \overline{\overline{c}}\,k$$

with the eigenvectors $\overline{\overline{R}}_1, \overline{\overline{R}}_2, \ldots, \overline{\overline{R}}_5$. Within the $\overline{\overline{R}}_i$ all physical variables are replaced by the average values, e.g.,

$$\overline{\overline{R}}_1 = \frac{\overline{\overline{\rho}}}{2\overline{\overline{c}}}\begin{bmatrix} 1 \\ \overline{\overline{v}}_1 - k_{x_1}\overline{\overline{c}} \\ \overline{\overline{v}}_2 - k_{x_2}\overline{\overline{c}} \\ \overline{\overline{v}}_3 - k_{x_3}\overline{\overline{c}} \\ \overline{\overline{H}} - c\overline{\overline{\vec{v}}}\cdot\vec{1}_k \end{bmatrix} \quad \overline{\overline{R}}_2 = \begin{bmatrix} k_{x_1} \\ \overline{\overline{v}}_1 k_{x_1} \\ \overline{\overline{v}}_2 k_{x_1} + \overline{\overline{\rho}} k_{x_3} \\ \overline{\overline{v}}_3 k_{x_1} - \overline{\overline{\rho}} k_{x_2} \\ \overline{\overline{\vec{b}}}\cdot\vec{1}_{x_1} \end{bmatrix} \quad \overline{\overline{R}}_5 = \frac{\overline{\overline{\rho}}}{2\overline{\overline{c}}}\begin{bmatrix} 1 \\ \overline{\overline{v}}_1 + k_{x_1}\overline{\overline{c}} \\ \overline{\overline{v}}_2 + k_{x_2}\overline{\overline{c}} \\ \overline{\overline{v}}_3 + k_{x_3}\overline{\overline{c}} \\ \overline{\overline{H}} + c\overline{\overline{\vec{v}}}\cdot\vec{1}_k \end{bmatrix};$$

(3) Calculate the wave amplitudes, all quantities are evaluated at the boundary

$$\begin{bmatrix} \delta w_1 \\ \delta w_2 \\ \delta w_3 \\ \delta w_4 \\ \delta w_5 \end{bmatrix} = \begin{bmatrix} -(k_{x_1}\delta\overline{\overline{v}}_1 + k_{x_2}\delta\overline{\overline{v}}_2 + k_{x_3}\delta\overline{\overline{v}}_3) + \dfrac{\delta\overline{\overline{p}}}{\overline{\overline{\rho}}\,\overline{\overline{c}}} \\ k_{x_1}\delta\overline{\overline{\rho}} + k_{x_3}\delta\overline{\overline{v}}_2 - k_{x_2}\delta\overline{\overline{v}}_3 - k_{x_1}\dfrac{\delta\overline{\overline{p}}}{\overline{\overline{c}}^2} \\ k_{x_2}\delta\overline{\overline{\rho}} - k_{x_3}\delta\overline{\overline{v}}_1 + k_{x_1}\delta\overline{\overline{v}}_3 - k_{x_2}\dfrac{\delta\overline{\overline{p}}}{\overline{\overline{c}}^2} \\ k_{x_3}\delta\overline{\overline{\rho}} - k_{x_2}\delta\overline{\overline{v}}_1 - k_{x_1}\delta\overline{\overline{v}}_2 - k_{x_3}\dfrac{\delta\overline{\overline{p}}}{\overline{\overline{c}}^2} \\ k_{x_1}\delta\overline{\overline{v}}_1 + k_{x_2}\delta\overline{\overline{v}}_2 + k_{x_3}\delta\overline{\overline{v}}_3 + \dfrac{\delta\overline{\overline{p}}}{\overline{\overline{\rho}}\,\overline{\overline{c}}} \end{bmatrix}$$

$$\delta\overline{\overline{v}}_1 = v_1^R - v_1^L \qquad \delta\overline{\overline{v}}_2 = v_2^R - v_2^L \qquad \delta\overline{\overline{v}}_3 = v_3^R - v_3^L$$
$$\delta\overline{\overline{\rho}} = \rho^R - \rho^L \qquad \delta\overline{\overline{p}} = p^R - p^L \quad ;$$

(4) Evaluate the numerical flux of Roe's schemes by any of the following formulas:

$$\begin{aligned} \left(\overline{\overline{F}}_i\cdot\vec{k}\right)_s &= \overline{\overline{F}}_i(Q^L)\cdot\vec{k} + \sum_{j=1}^{5}\overline{\overline{\lambda}}_j^{-}\,\delta w_j\,\overline{\overline{R}}_j \\ &= \overline{\overline{F}}_i(Q^R)\cdot\vec{k} - \sum_{j=1}^{5}\overline{\overline{\lambda}}_j^{+}\,\delta w_j\,\overline{\overline{R}}_j \\ &= \frac{1}{2}\left[\left(\overline{\overline{F}}_i(Q^L) + \overline{\overline{F}}_i(Q^R)\right)\cdot\vec{k} - \sum_{j=1}^{5}\left|\overline{\overline{\lambda}}_j\right|\delta w_j\,\overline{\overline{R}}_j\right] \end{aligned} \qquad (2.5.29)$$

where the ± sign on the eigenvalues represents positive and negative values respectively.

2.6 High order schemes

The variable interpolations determine the resulting accuracy of the scheme. A κ-parameter family of higher-order schemes can be written as

$$\begin{aligned} Q^L &= Q_{s-1/2} + \left\{\left(\frac{b}{4}\right)[(1-\kappa b)\Delta_- + (1+\kappa b)\Delta_+]Q\right\}_{s-1/2} \\ Q^R &= Q_{s+1/2} - \left\{\left(\frac{b}{4}\right)[(1+\kappa b)\Delta_- + (1-\kappa b)\Delta_+]Q\right\}_{s+1/2} \end{aligned} \tag{2.6.1}$$

where

$$(\Delta_+)_{s-1/2} = Q_{s+1/2} - Q_{s-1/2}, \qquad (\Delta_-)_{s-1/2} = Q_{s-1/2} - Q_{s-3/2} \tag{2.6.2}$$

denote forward and backward difference operators, respectively. The parameter κ determines the spatial accuracy of the difference approximation: $\kappa = -1$ corresponds to a fully-upwind second order scheme; $\kappa = 1$ to a central difference scheme; and $\kappa = 1/3$ to a third order upwind-biased scheme. The parameter b serves to limit higher-order terms in the interpolation in order to avoid oscillations at discontinuities such as shock waves in the solutions. According to Anderson et al. [39], the limiting is implemented by locally modifying the difference values in the interpolation to ensure monotone interpolation as

$$b = \frac{2\Delta_+\Delta_- + \varepsilon}{(\Delta_+)^2 + (\Delta_-)^2 + \varepsilon} \tag{2.6.3}$$

and ε is a small number preventing division by zero in regions of null gradients.

From the above variable interpolation formulations we infer that for the fully-upwind second order scheme and the third order upwind-biased scheme all the variables in the four volumes in one direction need to be used in the calculation of one interface inviscid flux.

2.7 Evaluation of diffusive flux

In the Navier-Stokes equations when we have to calculate the viscous and heat conduction flux components we first need to calculate the gradients of velocity and temperature. This means that we have to estimate appropriate values of these gradients on the cell interfaces. A general procedure, valid for an arbitrary control volume, can be derived by application of the divergence theorem. For a scalar U defined in volume Ω, which is bounded by a closed surface s, we have

$$\int_{\Omega} \nabla U \, d\Omega = \oint_{s} U \, d\vec{s} \,. \tag{2.7.1}$$

In the Cartesian coordinates x_i, i = 1,2,3, the vector formulation given above has three components formulated as follows:

$$\int_{\Omega} \frac{\partial U}{\partial x_i} d\Omega = \oint_{s} U \, \vec{1}_{x_i} \cdot d\vec{s} \tag{2.7.2}$$

where $\vec{1}_{x_i}$ is a unit vector in the x_i direction. Thus we can define the averaged gradients as

$$\overline{\left(\frac{\partial U}{\partial x_i}\right)}_{\Omega} \equiv \frac{1}{\Omega} \int_{\Omega} \frac{\partial U}{\partial x_i} d\Omega = \frac{1}{\Omega} \oint_{s} U \, \vec{1}_{x_i} \cdot d\vec{s} \tag{2.7.3}$$

From the above discussion it is seen that in order to calculate the gradient of the scalar on the cell interface we can choose an appropriate volume, which includes the cell interface, and then calculate alternatively the scalar integral on the surface.

2.8 Results

From the above discussion in a cell the discretised Navier-Stokes equations can be carried out from Eq.(2.3.4) for high order high resolution scheme as follows:

$$\frac{\partial}{\partial t} \begin{pmatrix} v^c_1 \\ v^c_2 \\ v^c_3 \\ v^c_4 \\ v^c_5 \end{pmatrix} + \begin{pmatrix} R^c_1(V) \\ R^c_2(V) \\ R^c_3(V) \\ R^c_4(V) \\ R^c_5(V) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{2.8.1}$$

where $v_{cell} = (v^c_1, v^c_2, v^c_3, v^c_4, v^c_5)^T$ are five *conservative* state variables in the cell, $R_{cell} = (R^c_1(V), R^c_2(V), R^c_3(V), R^c_4(V), R^c_5(V))^T$ are five residuals, and $V = (v_1, v_2, ..., v_N)^T$ are all *conservative* state variables in all cells, where N is the number of cells by five. The calculation of R_{cell} can be carried out by using the high order high resolution scheme described above. However in this procedure we do not need to use all v_i in the vector V but only those in the cell and in its neighbouring cells.

Consolidating all the discretised Navier-Stokes equations in every cell we have a N-dimensional non-linear *algebraic* system as follows:

$$\frac{\partial V}{\partial t} + R(V) = 0 \tag{2.8.2}$$

where $R = (R_1(V), R_2(V), ..., R_N(V))^T$ is a N-dimensional residual vector, V is a N-dimensional vector of discretised *conservative* state variables.

Chapter Three

Numerical discretisation of the locally conical Navier-Stokes equations

3.1 Introduction

Examination of many experimental studies [40,41,42,43,44] of supersonic or hypersonic laminar flows around conical shapes revealed that these flows exhibit a locally conical behaviour downstream of the nose region even though relatively large viscous regions exist. Based on this observation, McRae [45] introduced a locally conical approximation to the full Navier-Stokes equations for the solution of supersonic/hypersonic viscous flows around cones. This approximation has also been used for numerical solutions of supersonic/ hypersonic viscous flows around other conical shapes [46,47,48]. The validity of this approximation has been well established through these experiments and computations and the comparison between them.

3.2 General curvilinear coordinates translation

From chapter 2 we re-state the 3-dimensional Navier-Stokes equations as

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial (\mathbf{E_i}-\mathbf{E_v})}{\partial x_1} + \frac{\partial (\mathbf{F_i}-\mathbf{F_v})}{\partial x_2} + \frac{\partial (\mathbf{G_i}-\mathbf{G_v})}{\partial x_3} = \mathbf{S} \qquad (3.2.1)$$

A translation to general curvilinear coordinates can be carried out through:

$$x_1' = x_1'(x_1, x_2, x_3)\,,\quad x_2' = x_2'(x_1, x_2, x_3)\,,\quad x_3' = x_3'(x_1, x_2, x_3) \qquad (3.2.2)$$

The Jacobians of the translation are

$$J = \left|\frac{\partial(x_1', x_2', x_3')}{\partial(x_1, x_2, x_3)}\right| = \begin{vmatrix} x_{1x_1}' & x_{1x_2}' & x_{1x_3}' \\ x_{2x_1}' & x_{2x_2}' & x_{2x_3}' \\ x_{3x_1}' & x_{3x_2}' & x_{3x_3}' \end{vmatrix} \qquad (3.2.3)$$

and

$$\hat{J} = \left|\frac{\partial(x_1, x_2, x_3)}{\partial(x_1', x_2', x_3')}\right| = \begin{vmatrix} x_{1x_1'} & x_{1x_2'} & x_{1x_3'} \\ x_{2x_1'} & x_{2x_2'} & x_{2x_3'} \\ x_{3x_1'} & x_{3x_2'} & x_{3x_3'} \end{vmatrix} \qquad (3.2.4)$$

where $\hat{J} = 1/J$.

After the translation of the coordinates we obtain the following properties:

1) the scale will not change, i.e., $a = a$;

2) the vector becomes $a_i' = x_{ix_j'} a_j$;

3) the tensor becomes $a_{ij}' = x_{ix_m'} x_{jx_n'} a_{mn}$;

4) $$\begin{bmatrix} dx_1 \\ dx_2 \\ dx_3 \end{bmatrix} = \begin{bmatrix} x_{1x_1'} & x_{1x_2'} & x_{1x_3'} \\ x_{2x_1'} & x_{2x_2'} & x_{2x_3'} \\ x_{3x_1'} & x_{3x_2'} & x_{3x_3'} \end{bmatrix} \begin{bmatrix} dx_1' \\ dx_2' \\ dx_3' \end{bmatrix};$$

5) and $$\frac{\partial a}{\partial x_i} = \frac{\partial a}{\partial x_1'} x_{1x_i}' + \frac{\partial a}{\partial x_2'} x_{2x_i}' + \frac{\partial a}{\partial x_3'} x_{3x_i}' .$$

Now we will derive the Navier-Stokes equations in the new curvilinear coordinate system. Because we have

$$dx_1' = \frac{\begin{vmatrix} dx_1 & x_{1x_2'} & x_{1x_3'} \\ dx_2 & x_{2x_2'} & x_{2x_3'} \\ dx_3 & x_{3x_2'} & x_{3x_3'} \end{vmatrix}}{\hat{J}} \quad , \quad dx_2' = \frac{\begin{vmatrix} x_{1x_1'} & dx_1 & x_{1x_3'} \\ x_{2x_1'} & dx_2 & x_{2x_3'} \\ x_{3x_1'} & dx_3 & x_{3x_3'} \end{vmatrix}}{\hat{J}} \quad , \quad dx_3' = \frac{\begin{vmatrix} x_{1x_1'} & x_{1x_2'} & dx_1 \\ x_{2x_1'} & x_{2x_2'} & dx_2 \\ x_{3x_1'} & x_{3x_2'} & dx_3 \end{vmatrix}}{\hat{J}}$$

therefore $$\hat{J}\, x_{1x_1}' = \begin{vmatrix} x_{2x_2'} & x_{2x_3'} \\ x_{3x_2'} & x_{3x_3'} \end{vmatrix} \quad , \quad \hat{J}\, x_{2x_1}' = \begin{vmatrix} x_{2x_3'} & x_{2x_1'} \\ x_{3x_3'} & x_{3x_1'} \end{vmatrix} \quad , \quad \hat{J}\, x_{3x_1}' = \begin{vmatrix} x_{2x_1'} & x_{2x_2'} \\ x_{3x_1'} & x_{3x_2'} \end{vmatrix}$$

and

$$\frac{\partial}{\partial x_1'}\left(\frac{x_{1x_1}'}{J}\right) + \frac{\partial}{\partial x_2'}\left(\frac{x_{2x_1}'}{J}\right) + \frac{\partial}{\partial x_3'}\left(\frac{x_{3x_1}'}{J}\right) = \frac{\partial(\hat{J}\, x_{1x_1}')}{\partial x_1'} + \frac{\partial(\hat{J}\, x_{2x_1}')}{\partial x_2'} + \frac{\partial(\hat{J}\, x_{3x_1}')}{\partial x_3'}$$

$$= x_{2x_1'x_2'}x_{3x_3'} + x_{2x_2'}x_{3x_1'x_3'} - x_{2x_1'x_3'}x_{3x_2'} - x_{2x_3'}x_{3x_1'x_2'}$$

$$+ x_{2x_2'x_3'}x_{3x_1'} + x_{2x_3'}x_{3x_1'x_2'} - x_{2x_1'x_2'}x_{3x_3'} - x_{2x_1'}x_{3x_2'x_3'}$$

$$+ x_{2x_1'x_3'}x_{3x_2'} + x_{2x_1'}x_{3x_2'x_3'} - x_{2x_2'x_3'}x_{3x_1'} - x_{2x_2'}x_{3x_1'x_3'}$$

$$= 0 \quad .$$

Similarly we also have

$$\frac{\partial}{\partial x_1'}\left(\frac{x_{1x_2}'}{J}\right) + \frac{\partial}{\partial x_2'}\left(\frac{x_{2x_2}'}{J}\right) + \frac{\partial}{\partial x_3'}\left(\frac{x_{3x_2}'}{J}\right) = \frac{\partial}{\partial x_1'}\left(\frac{x_{1x_3}'}{J}\right) + \frac{\partial}{\partial x_2'}\left(\frac{x_{2x_3}'}{J}\right) + \frac{\partial}{\partial x_3'}\left(\frac{x_{3x_3}'}{J}\right) = 0 .$$

We now can set the new fluxes as follows

$$\hat{E}_i = \frac{x'_{1x_1}}{J}E_i + \frac{x'_{1x_2}}{J}F_i + \frac{x'_{1x_3}}{J}G_i \qquad \hat{E}_v = \frac{x'_{1x_1}}{J}E_v + \frac{x'_{1x_2}}{J}F_v + \frac{x'_{1x_3}}{J}G_v \qquad (3.2.6a)$$

$$\hat{F}_i = \frac{x'_{2x_1}}{J}E_i + \frac{x'_{2x_2}}{J}F_i + \frac{x'_{2x_3}}{J}G_i \qquad \hat{F}_v = \frac{x'_{2x_1}}{J}E_v + \frac{x'_{2x_2}}{J}F_v + \frac{x'_{2x_3}}{J}G_v \qquad (3.2.6b)$$

$$\hat{G}_i = \frac{x'_{3x_1}}{J}E_i + \frac{x'_{3x_2}}{J}F_i + \frac{x'_{3x_3}}{J}G_i \qquad \hat{G}_v = \frac{x'_{3x_1}}{J}E_v + \frac{x'_{3x_2}}{J}F_v + \frac{x'_{3x_3}}{J}G_v \qquad (3.2.6c)$$

Using property 5) and the above formulations we then derive expressions for the new fluxes about the new curvilinear coordinates variables resulting in:

$$\begin{aligned}
&\frac{\partial \hat{E}_i}{\partial x'_1} + \frac{\partial \hat{F}_i}{\partial x'_2} + \frac{\partial \hat{G}_i}{\partial x'_3} = \frac{\partial}{\partial x'_1}\left(\frac{x'_{1x_1}}{J}E_i + \frac{x'_{1x_2}}{J}F_i + \frac{x'_{1x_3}}{J}G_i\right) \\
&+ \frac{\partial}{\partial x'_2}\left(\frac{x'_{2x_1}}{J}E_i + \frac{x'_{2x_2}}{J}F_i + \frac{x'_{2x_3}}{J}G_i\right) + \frac{\partial}{\partial x'_3}\left(\frac{x'_{3x_1}}{J}E_i + \frac{x'_{3x_2}}{J}F_i + \frac{x'_{3x_3}}{J}G_i\right) \\
&= \frac{\partial}{\partial x'_1}\left(\frac{x'_{1x_1}}{J}E_i\right) + \frac{\partial}{\partial x'_2}\left(\frac{x'_{2x_1}}{J}E_i\right) + \frac{\partial}{\partial x'_3}\left(\frac{x'_{3x_1}}{J}E_i\right) \\
&+ \frac{\partial}{\partial x'_1}\left(\frac{x'_{1x_2}}{J}F_i\right) + \frac{\partial}{\partial x'_2}\left(\frac{x'_{2x_2}}{J}F_i\right) + \frac{\partial}{\partial x'_3}\left(\frac{x'_{3x_2}}{J}F_i\right) \\
&+ \frac{\partial}{\partial x'_1}\left(\frac{x'_{1x_3}}{J}G_i\right) + \frac{\partial}{\partial x'_2}\left(\frac{x'_{2x_3}}{J}G_i\right) + \frac{\partial}{\partial x'_3}\left(\frac{x'_{3x_3}}{J}G_i\right) \\
&= \frac{x'_{1x_1}}{J}\frac{\partial E_i}{\partial x'_1} + \frac{x'_{2x_1}}{J}\frac{\partial E_i}{\partial x'_2} + \frac{x'_{3x_1}}{J}\frac{\partial E_i}{\partial x'_3} + \frac{x'_{1x_2}}{J}\frac{\partial F_i}{\partial x'_1} + \frac{x'_{2x_2}}{J}\frac{\partial F_i}{\partial x'_2} + \frac{x'_{3x_2}}{J}\frac{\partial F_i}{\partial x'_3} \\
&+ \frac{x'_{1x_3}}{J}\frac{\partial G_i}{\partial x'_1} + \frac{x'_{2x_3}}{J}\frac{\partial G_i}{\partial x'_2} + \frac{x'_{3x_3}}{J}\frac{\partial G_i}{\partial x'_3} = \frac{1}{J}\left(\frac{\partial E_i}{\partial x_1} + \frac{\partial F_i}{\partial x_2} + \frac{\partial G_i}{\partial x_3}\right).
\end{aligned}$$

The same derivation can be used for the viscous fluxes, resulting in

$$\frac{\partial \hat{Q}}{\partial t} + \frac{\partial(\hat{E}_i - \hat{E}_v)}{\partial x'_1} + \frac{\partial(\hat{F}_i - \hat{F}_v)}{\partial x'_2} + \frac{\partial(\hat{G}_i - \hat{G}_v)}{\partial x'_3} = \hat{S} \qquad (3.2.7)$$

where

$$\hat{Q} = \frac{Q}{J} \qquad \hat{S} = \frac{S}{J}. \qquad (3.2.8)$$

From property 3) of the translation of coordinates we have the stress tensor formulation resulting from the translation as

$$\begin{bmatrix} \tau'_{11} & \tau'_{12} & \tau'_{13} \\ \tau'_{21} & \tau'_{22} & \tau'_{23} \\ \tau'_{31} & \tau'_{32} & \tau'_{33} \end{bmatrix} = \begin{bmatrix} x_{1x'_1} & x_{1x'_2} & x_{1x'_3} \\ x_{2x'_1} & x_{2x'_2} & x_{2x'_3} \\ x_{3x'_1} & x_{3x'_2} & x_{3x'_3} \end{bmatrix} \begin{bmatrix} \tau_{11} & \tau_{12} & \tau_{13} \\ \tau_{21} & \tau_{22} & \tau_{23} \\ \tau_{31} & \tau_{32} & \tau_{33} \end{bmatrix} \begin{bmatrix} x_{1x'_1} & x_{1x'_2} & x_{1x'_3} \\ x_{2x'_1} & x_{2x'_2} & x_{2x'_3} \\ x_{3x'_1} & x_{3x'_2} & x_{3x'_3} \end{bmatrix} \tag{3.2.9}$$

3.3 The locally conical Navier-Stokes equations

The locally conical Navier-Stokes equations can be derived through the general coordinate transformation

$$\begin{aligned} x_1 &= x_1(\xi,\eta,\zeta) = r(\xi)\sin\theta(\eta,\zeta)\cos\varphi(\eta,\zeta) \\ x_2 &= x_2(\xi,\eta,\zeta) = r(\xi)\sin\theta(\eta,\zeta)\sin\varphi(\eta,\zeta) \\ x_3 &= x_3(\xi,\eta,\zeta) = r(\xi)\cos\theta(\eta,\zeta) \end{aligned} \tag{3.3.1}$$

where $\xi = x'_1$, $\eta = x'_2$, $\zeta = x'_3$ are the new curvilinear coordinates.

Here $r(\xi)$ is the transformation to the radial coordinate. The parameters $\theta(\eta,\zeta)$ and $\varphi(\eta,\zeta)$ are the general two dimensional transformations to fit different conical shapes and control the clustering of grid points.

Neglecting the volume sources and the heat sources from equations (3.2.7), and following the above derivation we obtain

$$\frac{\partial\hat{\mathbf{Q}}}{\partial t} + \frac{\partial(\hat{\mathbf{E}}_i - \hat{\mathbf{E}}_v)}{\partial\xi} + \frac{\partial(\hat{\mathbf{F}}_i - \hat{\mathbf{F}}_v)}{\partial\eta} + \frac{\partial(\hat{\mathbf{G}}_i - \hat{\mathbf{G}}_v)}{\partial\zeta} = 0 \tag{3.3.2}$$

with

$$\hat{\mathbf{E}}_i = \frac{\xi_{x_1}}{J}\mathbf{E}_i + \frac{\xi_{x_2}}{J}\mathbf{F}_i + \frac{\xi_{x_3}}{J}\mathbf{G}_i \qquad \hat{\mathbf{E}}_v = \frac{\xi_{x_1}}{J}\mathbf{E}_v + \frac{\xi_{x_2}}{J}\mathbf{F}_v + \frac{\xi_{x_3}}{J}\mathbf{G}_v \tag{3.3.3a}$$

$$\hat{\mathbf{F}}_i = \frac{\eta_{x_1}}{J}\mathbf{E}_i + \frac{\eta_{x_2}}{J}\mathbf{F}_i + \frac{\eta_{x_3}}{J}\mathbf{G}_i \qquad \hat{\mathbf{F}}_v = \frac{\eta_{x_1}}{J}\mathbf{E}_v + \frac{\eta_{x_2}}{J}\mathbf{F}_v + \frac{\eta_{x_3}}{J}\mathbf{G}_v \tag{3.3.3b}$$

$$\hat{\mathbf{G}}_i = \frac{\zeta_{x_1}}{J}\mathbf{E}_i + \frac{\zeta_{x_2}}{J}\mathbf{F}_i + \frac{\zeta_{x_3}}{J}\mathbf{G}_i \qquad \hat{\mathbf{G}}_v = \frac{\zeta_{x_1}}{J}\mathbf{E}_v + \frac{\zeta_{x_2}}{J}\mathbf{F}_v + \frac{\zeta_{x_3}}{J}\mathbf{G}_v \tag{3.3.3c}$$

Now we discuss the derivation of terms for ξ. Generally we have the following properties: (1) We state here the formulation without providing detail of the derivation

$$\hat{J}\,\xi_{x_1} = \begin{bmatrix} x_{2\eta} & x_{2\zeta} \\ x_{3\eta} & x_{3\zeta} \end{bmatrix} = r^2(\xi)\,f_1(\eta,\zeta) \tag{3.3.4a}$$

$$\hat{J}\,\xi_{x_2} = \begin{bmatrix} x_{3\eta} & x_{3\zeta} \\ x_{1\eta} & x_{1\zeta} \end{bmatrix} = r^2(\xi)\,f_2(\eta,\zeta) \tag{3.3.4b}$$

$$\hat{J}\,\xi_{x_3} = \begin{bmatrix} x_{1\eta} & x_{1\zeta} \\ x_{2\eta} & x_{2\zeta} \end{bmatrix} = r^2(\xi)\, f_3(\eta,\zeta) \tag{3.3.4c}$$

where $f_i(\eta,\zeta)$ are only the functions of η and ζ but not ξ, i = 1,2,3. (2) Using the locally conical approximation, in other words we assume that the derivatives of flow properties to r are neglected, we obtain

$$\frac{\partial \mathbf{E}_i}{\partial \xi} = 0,\ \frac{\partial \mathbf{F}_i}{\partial \xi} = 0,\ \frac{\partial \mathbf{G}_i}{\partial \xi} = 0 \tag{3.3.5}$$

(3) Because we have

$$\tau_{ij} = \frac{2\mu}{Re_L}\left(\frac{1}{2}\left(\frac{\partial v_i}{\partial x_j'} + \frac{\partial v_j}{\partial x_i'}\right) - \frac{1}{3}\,\mathrm{div}\,(v_k)\,\delta_{ij}\right) = f_{ij} / r(\xi) \tag{3.3.6}$$

$$q_i = \frac{\mu}{(\gamma-1)M_\infty^2 Re_L Pr}\frac{\partial T}{\partial x_i'} = f^i / r(\xi) \tag{3.3.7}$$

where f_{ij} and f^i are composed by flow properties, we obtain:

$$\frac{\partial \mathbf{E}_v}{\partial \xi} = -\frac{r_\xi(\xi)}{r(\xi)}\mathbf{E}_v,\ \frac{\partial \mathbf{F}_v}{\partial \xi} = -\frac{r_\xi(\xi)}{r(\xi)}\mathbf{F}_v,\ \frac{\partial \mathbf{G}_v}{\partial \xi} = -\frac{r_\xi(\xi)}{r(\xi)}\mathbf{G}_v \tag{3.3.8}$$

Therefore we have

$$\frac{\partial \hat{\mathbf{E}}_i}{\partial \xi} = \frac{\partial(\hat{J}\,\xi_{x_1})}{\partial \xi}\mathbf{E}_i + \frac{\partial(\hat{J}\,\xi_{x_2})}{\partial \xi}\mathbf{F}_i + \frac{\partial(\hat{J}\,\xi_{x_3})}{\partial \xi}\mathbf{G}_i + \hat{J}\,\xi_{x_1}\frac{\partial \mathbf{E}_i}{\partial \xi} + \hat{J}\,\xi_{x_2}\frac{\partial \mathbf{F}_i}{\partial \xi} + \hat{J}\,\xi_{x_3}\frac{\partial \mathbf{G}_i}{\partial \xi}$$

$$= \frac{\partial(r^2(\xi)\, f_1(\eta,\zeta))}{\partial \xi}\mathbf{E}_i + \frac{\partial(r^2(\xi)\, f_2(\eta,\zeta))}{\partial \xi}\mathbf{F}_i + \frac{\partial(r^2(\xi)\, f_3(\eta,\zeta))}{\partial \xi}\mathbf{G}_i$$

$$= 2\, r(\xi)\, r_\xi(\xi)\left(f_1(\eta,\zeta)\,\mathbf{E}_i + f_2(\eta,\zeta)\,\mathbf{F}_i + f_3(\eta,\zeta)\,\mathbf{G}_i\right)$$

$$= \frac{2\, r_\xi(\xi)}{r(\xi)}\left(r^2(\xi)\, f_1(\eta,\zeta)\,\mathbf{E}_i + r^2(\xi)\, f_2(\eta,\zeta)\,\mathbf{F}_i + r^2(\xi)\, f_3(\eta,\zeta)\,\mathbf{G}_i\right) = \frac{2\, r_\xi(\xi)}{r(\xi)}\hat{\mathbf{E}}_i$$

$$\frac{\partial \hat{\mathbf{E}}_v}{\partial \xi} = \frac{r_\xi(\xi)}{r(\xi)}\hat{\mathbf{E}}_v$$

The general locally conical Navier-Stokes equations can now be written as:

$$\frac{\partial \hat{\mathbf{Q}}}{\partial t}+\frac{\partial \hat{\mathbf{F}}}{\partial \eta}+\frac{\partial \hat{\mathbf{G}}}{\partial \zeta}+\hat{\mathbf{H}}=0 \tag{3.3.9}$$

where

$$\hat{\mathbf{H}}=\frac{r_{\xi}(\xi)}{r(\xi)}\left(2\hat{\mathbf{E}}_i-\hat{\mathbf{E}}_v\right), \quad \text{and } \hat{\mathbf{Q}}=\frac{\mathbf{Q}}{J}. \tag{3.3.10}$$

When we choose $r(\xi)=\xi$, then

$$\hat{\mathbf{H}}=\left(2\hat{\mathbf{E}}_i-\hat{\mathbf{E}}_v\right)/\xi . \tag{3.3.11}$$

For equations (3.3.9) we can process an integral over a surface $s(\eta,\zeta)$, in which ξ = constant, i.e.,

$$\frac{\partial}{\partial t}\int_{s(\eta,\zeta)}\hat{\mathbf{Q}}\,ds+\int_{s(\eta,\zeta)}\left(\frac{\partial \hat{\mathbf{F}}}{\partial \eta}+\frac{\partial \hat{\mathbf{G}}}{\partial \zeta}\right)ds+\int_{s(\eta,\zeta)}\hat{\mathbf{H}}\,ds=0 \tag{3.3.12}$$

By using Green's theorem we have

$$\frac{\partial}{\partial t}\int_{s(\eta,\zeta)}\hat{\mathbf{Q}}\,ds+\oint_{L}\left(\hat{\mathbf{F}},\hat{\mathbf{G}}\right)\cdot d\vec{l}+\int_{s(\eta,\zeta)}\hat{\mathbf{H}}\,ds=0 \tag{3.3.13a}$$

$$\frac{\partial}{\partial t}\int_{s(\eta,\zeta)}\hat{\mathbf{Q}}\,ds+\oint_{L}\left(\hat{\mathbf{F}}\,d\zeta+\hat{\mathbf{G}}\,d\eta\right)+\int_{s(\eta,\zeta)}\hat{\mathbf{H}}\,ds=0 \tag{3.3.13b}$$

where L is the bounding line of the area $s(\eta,\zeta)$. In the equations (3.3.13) the last term operates in a similar fashion to the source term.

In order to discretise the above equations by using the cell centred finite volume method, the 3-dimensional cell described in chapter 2 will be degenerated to a 2-dimensional cell. The cell used here also needs to satisfy the definition in chapter 2.3, and we then have

$$\frac{\partial}{\partial t}\left(\mathbf{Q}_{ij}S_{ij}\right)+\sum_{\text{lines}}\left(\hat{\mathbf{F}},\hat{\mathbf{G}}\right)\cdot\Delta\vec{l}+\hat{\mathbf{H}}S_{ij}=0 \tag{3.3.14}$$

Notice that now Eq.(3.3.14) is also a compact form and includes five sub-equations.

3.4 Calculation of flux

We will now present the formulation which will enable the calculation of the eigenvalues, eigenvectors, and Riemann invariants for the locally conical Navier-Stokes equations. Now the inviscid fluxes are

$$\hat{\mathbf{F}}_i = \frac{\eta_{x_1}}{J}\mathbf{E}_i + \frac{\eta_{x_2}}{J}\mathbf{F}_i + \frac{\eta_{x_3}}{J}\mathbf{G}_i \equiv \overline{\mathbf{F}}_i \cdot \vec{\eta}_J \tag{3.4.1a}$$

$$\hat{\mathbf{G}}_i = \frac{\zeta_{x_1}}{J}\mathbf{E}_i + \frac{\zeta_{x_2}}{J}\mathbf{F}_i + \frac{\zeta_{x_3}}{J}\mathbf{G}_i \equiv \overline{\mathbf{F}}_i \cdot \vec{\zeta}_J \tag{3.4.1b}$$

where we define two 3-dimensional vectors as:

$$\vec{\eta}_J = \left(\frac{\eta_{x_1}}{J}, \frac{\eta_{x_2}}{J}, \frac{\eta_{x_3}}{J}\right) \qquad \vec{\zeta}_J = \left(\frac{\zeta_{x_1}}{J}, \frac{\zeta_{x_2}}{J}, \frac{\zeta_{x_3}}{J}\right) \tag{3.4.2}$$

For an arbitrary 2-dimensional surface normal $\vec{k} = (k_\eta, k_\zeta)$ in the ξ = constant domain we have

$$\begin{aligned}(\hat{\mathbf{F}}_i, \hat{\mathbf{G}}_i) \cdot \vec{k} &= (\overline{\mathbf{F}}_i \cdot \vec{\eta}_J, \overline{\mathbf{F}}_i \cdot \vec{\zeta}_J) \cdot (k_\eta, k_\zeta) \\ = k_\eta \overline{\mathbf{F}}_i \cdot \vec{\eta}_J + k_\zeta \overline{\mathbf{F}}_i \cdot \vec{\zeta}_J &= \overline{\mathbf{F}}_i \cdot (k_\eta \vec{\eta}_J + k_\zeta \vec{\zeta}_J) \equiv \overline{\mathbf{F}}_i \cdot \vec{k}_J\end{aligned} \tag{3.4.3}$$

where

$$\vec{k}_J = k_\eta \vec{\eta}_J + k_\zeta \vec{\zeta}_J \tag{3.4.4}$$

is a 3-dimensional vector. Thus Eq.(3.4.3) shows that the inner product of inviscid flux and a surface normal in locally conical coordinate is equal to a inner product of a flux and a vector in 3-dimensional space. For viscous flux we can have the same conclusion.

Therefore we have formulations to calculate the eigenvalue, eigenvector, and the Riemann invariant of (3.4.3) in the same form as in the 3-dimensional case given in chapter 2. The 3-dimensional vector (3.4.4) is generated by the 2-dimensional (k_η, k_ζ) and the coordinate translation formulations.

The discretised LCNS equations will have the same form as Eq.(2.8.1). In the following we will give the detailed formulation for specific test problems, which includes a description of the physical problems studied, the structured grid used, boundary conditions, and the evaluation of inviscid and viscous fluxes.

3.5 Numerical discretisation

3.5.1 The physical problems

We consider a compressible Mach 7.95 laminar flow around a sharp cone of half angle 10^o with a cold wall (T_w =309.8K) and with angles of attack 12^o and 24^o. The Reynolds number is 4.1×10^6 and the flow temperature is 55.4K. This case produces a flow which has a large separated flow region with embedded shock waves on the leeward side of the cone and strong gradients in the thin boundary layer on the windward side. We solve these problems at each ξ = constant 2-dimensional domain, in which the flow still can be described effectively by the locally conical Navier-Stokes equations.

3.5.2 The structured grid and control volume

The sharp cone and the grid at $r(\xi) = \xi$ = constant are illustrated in Fig.3.5.1. The 2-dimensional structured grid generation method is described in Appendix 2. Because the yaw angle is 0^o we consider just the half side of the flow, then the grid overlaps by one point the line of symmetry (Fig.3.5.2). We call this type of grid the primary grid, which has I+2 points in the η direction and J+2 points in the ζ direction.

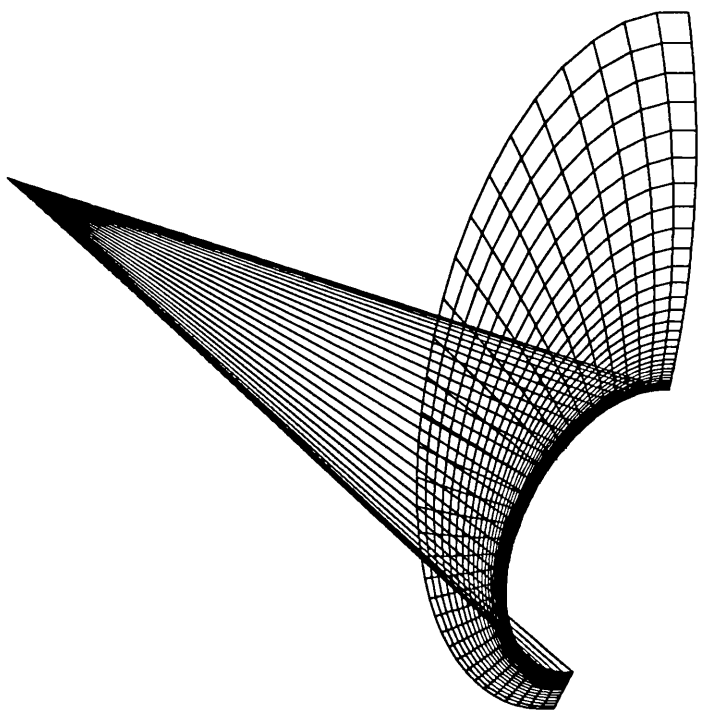


Fig.3.5.1 The sharp cone and location of the grid

A secondary grid can be obtained by determining the centres of the primary cells and connecting them across cell faces. This has I cells in the η direction and J cells in the ζ

direction. We will choose the cells in the secondary grid as the control volumes in the cell centred finite volume method as illustrated in Fig.3.5.3. Each cell contains the state variables similar to the 3-dimensional flow, i.e., 5 conservative components ρ, ρv_1, ρv_2, ρv_3, ρE or primitive components ρ, v_1, v_2, v_3, p. The unknown variables are set in all I×J cells in the secondary grid and we have in all (I+2)×(J+2) grid nodes.

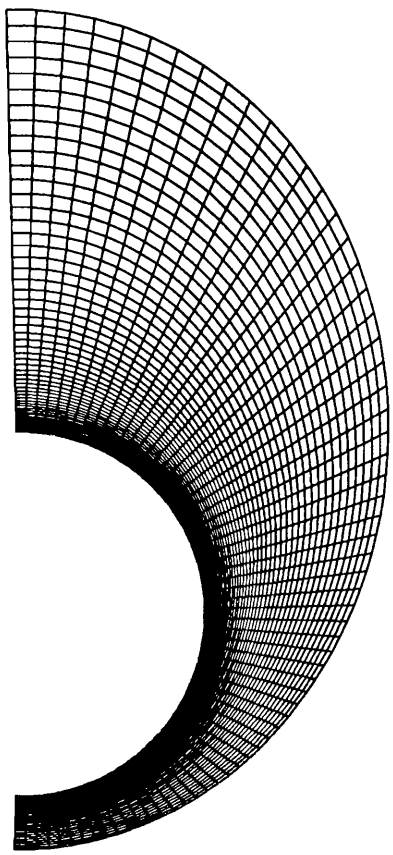
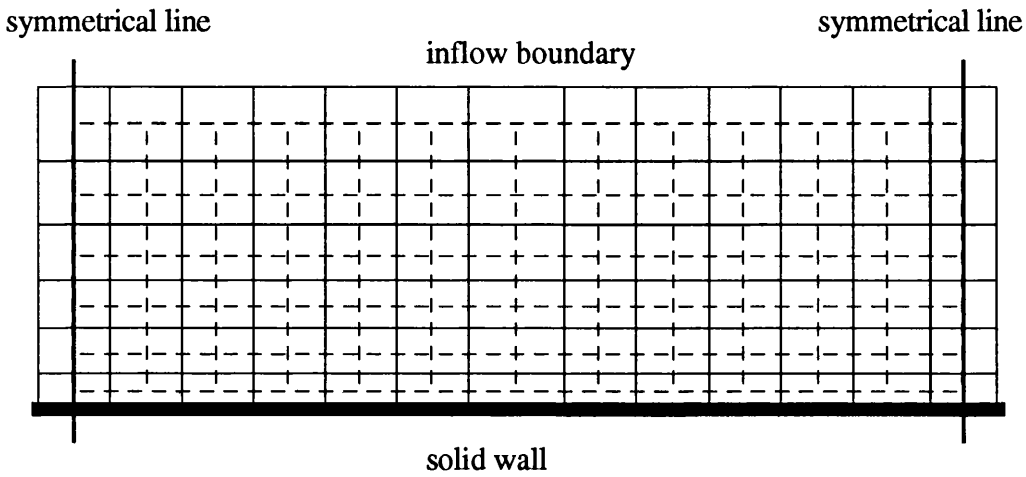


Fig.3.5.2 The 2-dimensional grid

From the thin shear layer approximation [49] we have

$$\frac{\partial p}{\partial n} = 0 \tag{3.5.1}$$

on the solid wall. This relation can be a numerical boundary condition for the Navier-Stokes solution if we choose that (1) near the solid wall there exist very fine grid lines along the orthogonal direction and (2) each grid line drawn from the solid wall are normal to the wall. Therefore these requirements give the constraints for the grid generation.



The plain line is the primary grid. The dashed line is the secondary grid

Fig.3.5.3 The primary and secondary grids

3.5.3 Fluxes evaluation

We will now calculate the numerical fluxes on an interface, such as the inviscid flux $FI_{i+1/2,j}$ and viscous flux $FV_{i+1/2,j}$ on the interface between cells (i,j) and (i+1,j), which have areas $s_{i,j}$ and $s_{i+1,j}$ respectively. First we discuss the inviscid flux $FI_{i+1/2,j}$. In the first order Osher's scheme we just use the variables in the cell (i,j) and (i+1,j), i.e., $\mathbf{Q}_{i,j}$ and $\mathbf{Q}_{i+1,j}$, which are the $\mathbf{Q}^L$ and $\mathbf{Q}^R$ in the formulation (2.5.5) respectively (Fig.3.5.4a). When the high order scheme is used the $\mathbf{Q}^L$ are interpolated by $\mathbf{Q}_{i-1,j}$, $\mathbf{Q}_{i,j}$,and $\mathbf{Q}_{i+1,j}$, and $\mathbf{Q}^R$ are interpolated by $\mathbf{Q}_{i,j}$, $\mathbf{Q}_{i+1,j}$,and $\mathbf{Q}_{i+2,j}$, in the formulation (2.6.1) (Fig.3.5.4b). It is found [7] that the use of primitive variables **V** in the interpolation is more robust than the use of conserved variables **Q** in the sense that non-physical states from the interpolation, such as, negative pressure, are easier to avoid. Therefore in the calculation of interface inviscid fluxes we need to use variables in 4 cells in one coordinate direction.

For the calculation of viscous flux $FV_{i+1/2,j}$ we need construct a new cell $c_{i+1/2,j}$ and $s_{i+1/2,j} = 1/2(s_{i,j}+s_{i+1,j})$ as in Fig.3.5.4c. Then for an arbitrary scalar U we have

$$\int_{s_{i+1/2,j}} \nabla U \, ds = \oint_L U \, d\vec{l} \tag{3.5.2}$$

$$\overline{\left(\frac{\partial U}{\partial x_i'}\right)}_{s_{i+1/2,j}} \equiv \frac{1}{s_{i+1/2,j}} \int_{s_{i+1/2,j}} \frac{\partial U}{\partial x_i'} \, ds = \frac{1}{s_{i+1/2,j}} \oint_L U \, \vec{1}_{x_i'} \cdot d\vec{l} \tag{3.5.3}$$

Therefore we have

$$\overline{\left(\frac{\partial U}{\partial x_i'}\right)}_{s_{i+1/2,j}} = \frac{1}{s_{i+1/2,j}}\left(U_1\int_{L_1}\vec{1}_{x_i'}\cdot d\vec{l} + U_2\int_{L_2}\vec{1}_{x_i'}\cdot d\vec{l} + U_3\int_{L_3}\vec{1}_{x_i'}\cdot d\vec{l} + U_4\int_{L_4}\vec{1}_{x_i'}\cdot d\vec{l}\right) \tag{3.5.4}$$

where $U_1 = U_{i,j}$, $U_2 = 1/4(U_{i,j} + U_{i,j-1} + U_{i+1,j-1} + U_{i+1,j})$, $U_3 = U_{i+1,j}$, and $U_4 = 1/4(U_{i,j} + U_{i,j+1} + U_{i+1,j+1} + U_{i+1,j})$, $x_i' = \eta$ or ζ.

For an arbitrary U we also have

$$\int_{s_{i,j}} U\, ds \equiv \frac{1}{s_{i,j}}\overline{(U)}_{s_{i,j}} \tag{3.5.5}$$

which is used for the discretisation of the first and last terms of equations (3.3.13). Fig.3.5.4d shows all the cells that are involved in the calculation of the fluxes on the interface between (i,j) and (i+1,j). In the same way we can calculate the fluxes in the ζ direction.

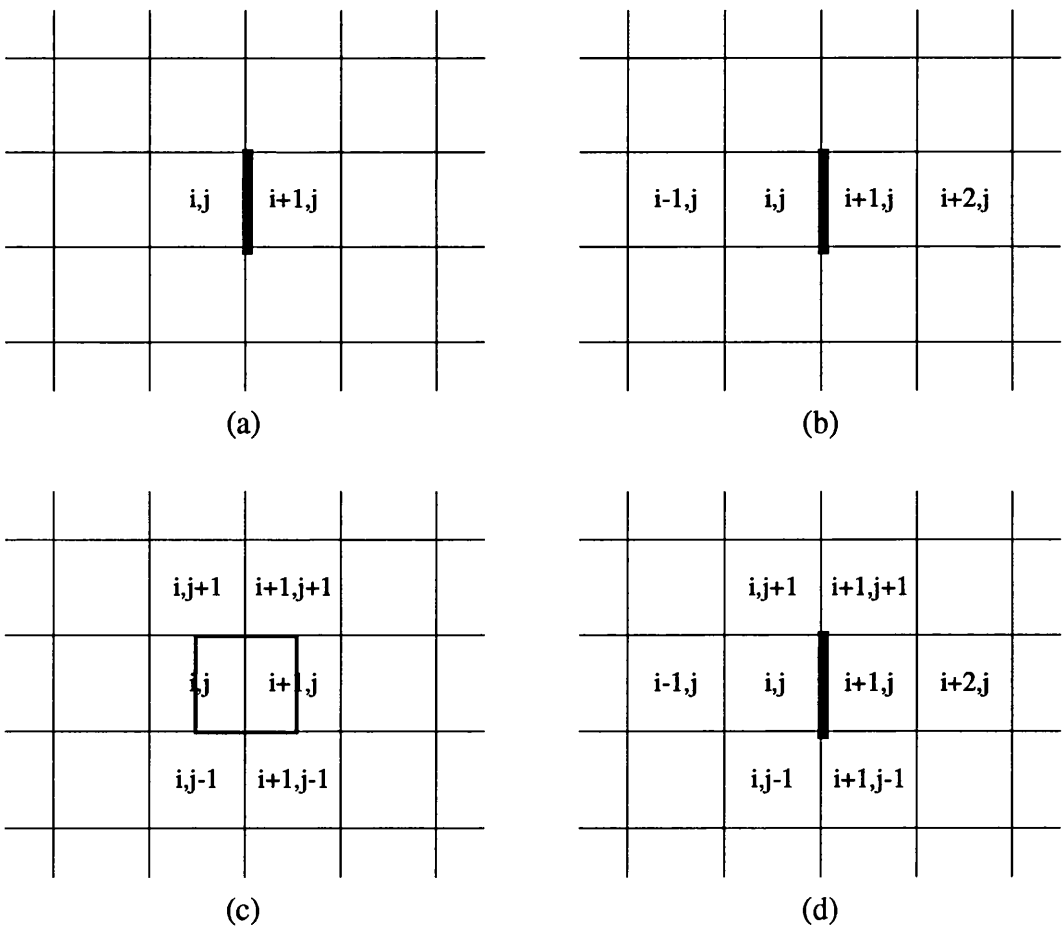


Fig.3.5.4 Calculation of the flux between cell (i,j) and (i+1,j)

3.5.4 The discretised equations

When calculating the residuals in cell (i,j), we need to calculate the fluxes on the four interfaces between the cell pairs (i-1,j), (i,j); (i,j), (i+1,j); (i,j-1), (i,j); and (i,j), (i,j+1) respectively, i.e., we have a formulation as follows:

$$R_{cell} = \frac{1}{A_{cell}}\{FI_{i+1/2,j} - FI_{i-1/2,j} + FI_{i,j+1/2} - FI_{i,j-1/2} + FV_{i+1/2,j} - FV_{i-1/2,j} + FV_{i,j+1/2} - FV_{i,j-1/2}\} + H_{i,j} \tag{3.5.6}$$

where R_{cell} is a residual vector in the cell, A_{cell} is the area of cell, and $H_{i,j}$ are the source terms. Formulation (3.5.6) is a compact form and includes five components.

Therefore we have the discretised equations in a cell as follows:

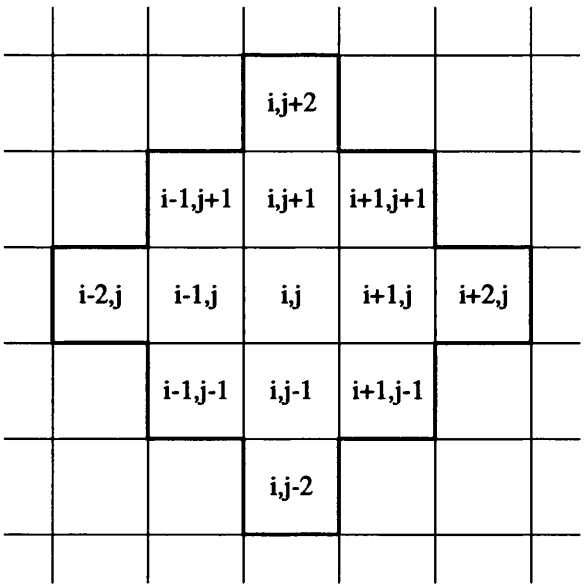
$$\frac{\partial}{\partial t}\begin{pmatrix} v^c_1 \\ v^c_2 \\ v^c_3 \\ v^c_4 \\ v^c_5 \end{pmatrix} + \begin{pmatrix} R^c_1(V) \\ R^c_2(V) \\ R^c_3(V) \\ R^c_4(V) \\ R^c_5(V) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.5.7}$$

where $(v^c_1, v^c_2, v^c_3, v^c_4, v^c_5)^T = v_{cell}$ are five *conservative* state variables in the cell, $(R^c_1(V), R^c_2(V), R^c_3(V), R^c_4(V), R^c_5(V))^T = R_{cell}$ are five residuals, and $V = (v_1, v_2, ..., v_N)^T$ are all *conservative* state variables in all cells, where N is the number of cells by five. The calculation of R_{cell} can be carried out by (3.5.6). However in this procedure we do not need to use all elements in vector V but only these in cell (i,j) and its neighbouring cells. The stencils of the discretised *physical* state variables used in the calculations of these residuals in the cell (i,j) using the cell centred finite volume method are illustrated in Fig.3.5.5.

Consolidating all the discretised Navier-Stokes equations in every cell we have a N-dimensional non-linear *algebraic* system as follows:

$$\frac{\partial V}{\partial t} + R(V) = 0 \tag{3.5.8}$$

where $N = I \times J \times 5$ is the number of all unknown variables, $R^T = (R_1, R_2, ..., R_N)$ is the residual vector and $V^T = (v_1, v_2, ..., v_N)$ is the discretised *conservative* state variables vectors. We note that $\{v_n, v_{n+1}, v_{n+2}, v_{n+3}, v_{n+4}\} \in V$ are the discretised *conservative* state variables in the cell (i,j), and $\{R_n, R_{n+1}, R_{n+2}, R_{n+3}, R_{n+4}\} \in R$ are the residuals in the same cell, where $n = 5\times((i-1)\times J+j-1)+1$.



The residuals calculated for the cell (i,j) and
the discretised *physical* state variables used lie in the 13 cells within the bold line.

Fig.3.5.5 13 point stencils

3.5.5 Explicit methods

A simple Euler explicit scheme for the time dependent system (3.5.8) is

$$V^{n+1} = V^n - \Delta t\, R\,(V^n) \tag{3.5.9}$$

In this scheme techniques such as local time stepping, residual smoothing, and multigrid, can be used since time accuracy is not required for solving the steady state problem.

A four-step Runge-Kutta integration with time [50] can also be applied to the time dependent system (3.5.8)

$$\begin{aligned} V^0 &= V^n \\ V^1 &= V^n - \alpha_1 \Delta t\, R\,(V^0) \\ V^2 &= V^n - \alpha_2 \Delta t\, R\,(V^1) \\ V^3 &= V^n - \alpha_3 \Delta t\, R\,(V^2) \\ V^{n+1} &= V^n - \Delta t\, R\,(V^3) \end{aligned} \tag{3.5.10}$$

where $\alpha_1 = 1/4$, $\alpha_2 = 1/3$, $\alpha_3 = 1/2$. This is a fourth-order accurate scheme (in time), which is robust when starting the solution from free stream conditions but slow in convergence. In the next chapter this scheme will be used for providing the initial guess, and the convergence will be carried out using the implicit method.

Chapter Four

The Newton's method and linear solver

4.1 Introduction

The classical Newton's method used for solving the non-linear *algebraic* system has quadratic convergence properties, and is robust when starting from a 'good initial' guess. The main drawback of this method is that it is memory intensive. The Newton's method was once thought to be impractical for CFD applications. In recent years advances in computational hardware have allowed researchers to reexamine this previously inaccessible method in the search for improvements to the accuracy, efficiency and robustness of existing CFD algorithms. The recent generation of large memory computers and supercomputers, such as the IBM RISC System/6000 workstation and Cray 2 and Y-MP, enable us to test the Newton's method for relatively simple object flow problems. The computer memory limitation remains a problem when we try to deal with practical applications.

In this chapter we will describe the sequential implementations of the Newton's method for fast steady state Navier-Stokes solutions. In the approach for developing such numerical schemes (1) a simplified approximate procedure is proposed for generating the numerical Jacobian of the non-linear *algebraic* system, which is very fast and minimises the number of cells in which the discretised *physical* state variables need to be used for generating the elements of the Jacobian; (2) a new α-GMRES linear solver is also proposed for solving a large sparse non-symmetric linear system. This solver modifies the linear system by means of a simple block diagonal preconditioner and damping factor α, constructs a new iterative procedure, and at each iterative step solves a modified linear system by the GMRES scheme. The new linear solver can overcome the non-convergence phenomenon for the test flow problems which happen when using the GMRES linear solver with just a simple block diagonal preconditioner. However the new linear solver requires nearly the same memory as the GMRES linear solver. Bearing in mind the possibility of parallelization of the code, it is arranged that no sequential bottle-neck occurs in the new linear solver. The initial guess used here was provided by an explicit time dependent approach using the Runge-Kutta method with local time stepping described in chapter 3, which is robust when starting the solution from free stream conditions but slow in convergence. The test flow problems are the hypersonic laminar flows around a conical shape, governed by the locally conical Navier-Stokes equations. The computer used is an IBM RISC System/6000 320H workstation in the Department of Aerospace Engineering, University of Glasgow.

4.2 The Newton's method

After spatial discretisation a semi-discretised system of ordinary differential equations in time can be defined as:

$$\frac{\partial V}{\partial t} + R(V) = 0 \tag{4.2.1}$$

where $R(V)$ is a N-dimensional non-linear *algebraic* system in the global domain, and $N = I \times J \times 5$ is the number of all unknown variables. We have that $R^T = (R_1, R_2, ..., R_N)$ is the residual vector and $V^T = (v_1, v_2, ..., v_N)$ is the discretised *physical* state variables vector, the $(v_n, v_{n+1}, v_{n+2}, v_{n+3}, v_{n+4}) = v_{cell} \in V$ are in the cell (i,j), and the $(R_n, R_{n+1}, R_{n+2}, R_{n+3}, R_{n+4}) = R_{cell} \in R$ are in the same cell, where $n = 5 \times ((i-1) \times J + j - 1) + 1$. Assume $R(V)$ has being calculated and stored during the numerical computation procedure.

Using a fully implicit method, e.g., the backward Euler implicit method,

$$\left[\frac{1}{\Delta t} I + \left(\frac{\partial R}{\partial V}\right)^k\right] \Delta V^k = - R(V^k)$$
$$\Delta V^k = V^{k+1} - V^k \tag{4.2.2}$$

unconditional stability can be achieved and as the time step approaches infinity the method approaches the Newton's method

$$\left(\frac{\partial R}{\partial V}\right)^k \Delta V^k = - R(V^k)$$
$$\Delta V^k = V^{k+1} - V^k \tag{4.2.3}$$

for the solution of the non-linear system (4.2.1), the iteration being for k=1,2, ... Because our discussion is focussed on the steady state flow problems and time accuracy is not required, we can solve the linear system Eq.(4.2.3) to update the variables V.

The Newton's method (4.2.3) can also be derived by solving the non-linear system: $R(V) = 0$. Thus in (4.2.3) the discretised *physical* state variables V could be the conservative variables or primitive variables.

4.2.1 The Jacobian matrix

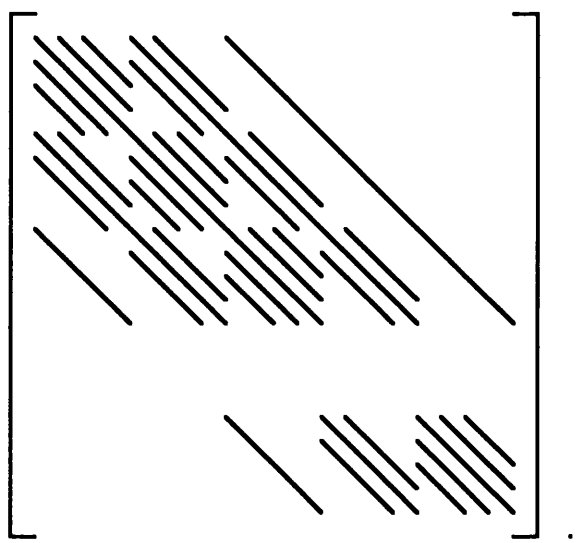
Since Eq.(4.2.1) is composed of equations (3.5.7) in each cell, we can write the Newton's formulation by using the cells as follows:

$$\left(\partial R_{cell}/\partial V\right)^k \Delta V^k = - R_{cell}(V^k), \qquad cell = (1,1), ..., (I,J) \tag{4.2.4}$$

For any v_{cell}, $\partial R_{cell}/\partial v_{cell}$ is a 5×5 matrix, where R_{cell} is in a cell (i,j) and v_{cell} is in a cell (l,m), where i, l =1, ..., I, and j, m =1, ..., J. From chapter 3 we know that $\partial R_{cell}/\partial v_{cell}$ is not equal to zero only for the cell (l,m) within the thirteen cells around cell (i,j) (Fig.3.5.5). Thus corresponding to each cell (i,j) the elements of the Jacobian matrix, in five rows, have the form of thirteen 5×5 submatrixes as follows:

$$\begin{aligned}
&\partial R_{cell(i,j)}/\partial v_{cell(i-2,j)},\ \partial R_{cell(i,j)}/\partial v_{cell(i-1,j-1)},\ \partial R_{cell(i,j)}/\partial v_{cell(i-1,j)},\\
&\partial R_{cell(i,j)}/\partial v_{cell(i-1,j+1)},\ \partial R_{cell(i,j)}/\partial v_{cell(i,j-2)},\ \partial R_{cell(i,j)}/\partial v_{cell(i,j-1)},\\
&\qquad\qquad \partial R_{cell(i,j)}/\partial v_{cell(i,j)},\\
&\partial R_{cell(i,j)}/\partial v_{cell(i,j+1)},\ \partial R_{cell(i,j)}/\partial v_{cell(i,j+2)},\ \partial R_{cell(i,j)}/\partial v_{cell(i+1,j-1)},\\
&\partial R_{cell(i,j)}/\partial v_{cell(i+1,j)},\ \partial R_{cell(i,j)}/\partial v_{cell(i+1,j+1)},\ \partial R_{cell(i,j)}/\partial v_{cell(i,j+2)}.
\end{aligned} \tag{4.2.5}$$

Therefore we obtain the Jacobian matrix of Eq.(4.2.3) which is a order N, block 13-point diagonal matrix and each block is a 5×5 submatrix, which can be denoted as



In this paper the approximate numerical Jacobian matrix of the non-linear system is used instead of the analytical one, i.e., for a Jacobian matrix of form

$$\mathcal{J} = \frac{\partial R}{\partial v} = \begin{bmatrix} \mathcal{J}_{11} & \mathcal{J}_{12} & \cdots & \mathcal{J}_{1N} \\ \mathcal{J}_{21} & \mathcal{J}_{22} & \cdots & \mathcal{J}_{2N} \\ \vdots & \vdots & & \vdots \\ \mathcal{J}_{N1} & \mathcal{J}_{N2} & \cdots & \mathcal{J}_{NN} \end{bmatrix} \tag{4.2.6}$$

where

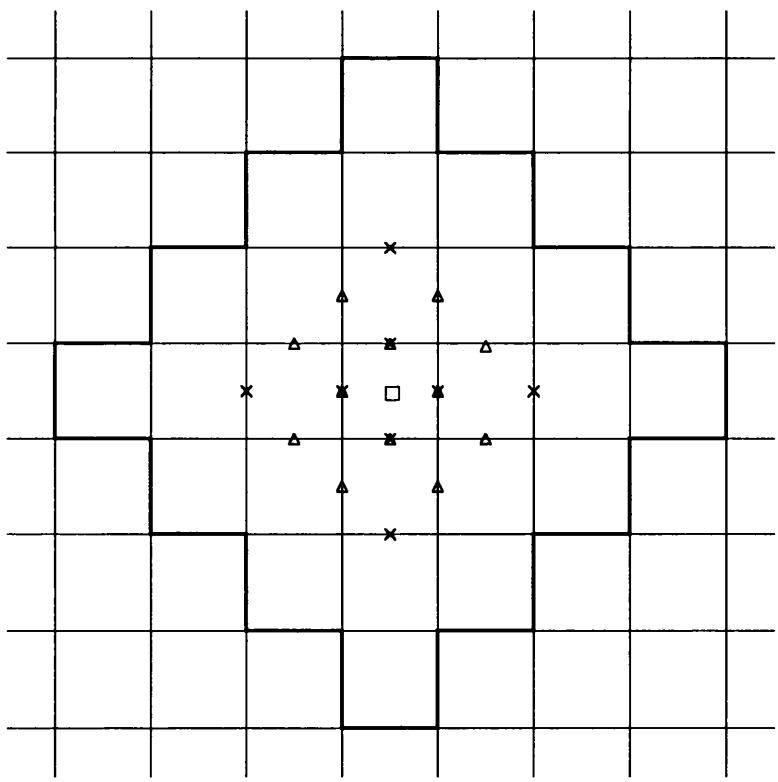
$$\mathcal{J}_{n_1 n_2} = \frac{\partial R_{n_1}}{\partial v_{n_2}} \tag{4.2.7}$$

we replace (4.2.7) with a difference quotient of the form:

$$\mathcal{J}_{n_1 n_2} = \frac{R_{n_1}(V + \Delta v_{n_2}) - R_{n_1}(V)}{\Delta v_{n_2}} \tag{4.2.8}$$

Various selections of Δv_{n_2} have been suggested in the literature of numerical analysis. When choosing $\Delta v_{n_2} = h\, e_{n_2}$, where e_{n_2} is the n_2th unit vector, Dennis and Schnabel [51] pointed out that if a sequence $\{h_S\}$ is used for the step size h, and if this sequence is properly chosen, then the quadratic convergence property of Newton's method is retained and Newton's method using finite differences is 'virtually indistinguishable' from Newton's method using analytic derivatives. In this paper h is chosen as $\varepsilon \times v_{n_2}$, where $\varepsilon \approx \sqrt{[\text{machine epsilon}]}$.

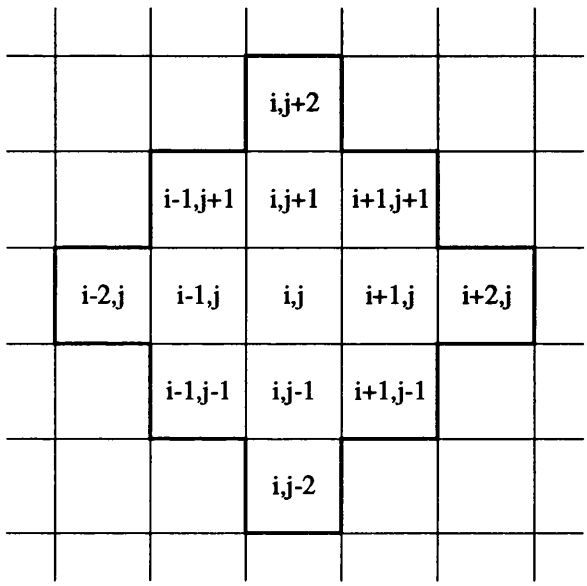
We now consider that one *physical* state variable perturbation occurs at a cell (i,j), which will affect the computational results of the inviscid **fluxes** on 8 interfaces and the viscous **fluxes** on 12 interfaces (Fig.4.2.1). However in order to calculate the above fluxes we need to use the *physical* state variables in 25 cells, which is within the bold line in the figure.



where ×: inviscid flux needed, Δ: viscous flux needed,
square: the perturbation position, *physical* state variables used lie within the bold line.

Fig.4.2.1 The fluxes should be calculated

On the other hand, this *physical* state variables perturbation will affect the computational results of **residuals** in 13 cells, which is illustrated in Fig.4.2.2. According to the requirements for the *physical* state variables in calculating residual in a cell in chapter 3, Fig.4.2.3 shows that the *physical* state variables, which are required for calculating all the residuals in 13 cells, are in 41 cells. In Fig.4.2.3 we also see that 36 inviscid fluxes and 36 viscous fluxes need to be calculated for evaluating the residuals in the 13 cells. It is obvious that the above conclusions are true for any variable perturbation at the cell (i,j).



For a perturbation occurring in cell (i,j), the residual calculations affected by the perturbation are at all the 13 cells within bold line. Because the residual vector at a cell have 5 components we have overall 13×5 residual calculations affected by the perturbation.

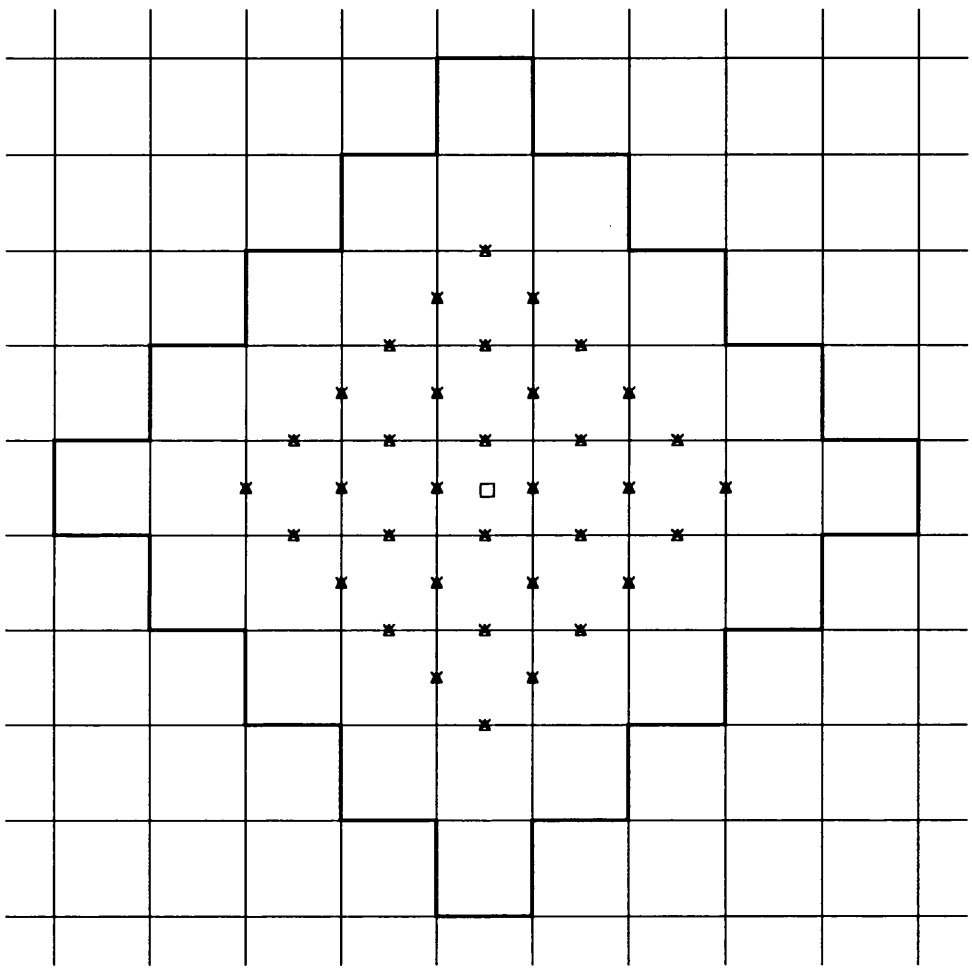
Fig.4.2.2 13 point stencils

In the numerical computation procedure we can use the column by column method to generate the Jacobian matrix, i.e., each time we generate a column of elements of the matrix, as we proceed from column 1 to N. From the formulations (4.2.6) and (4.2.7) we know that each time, we fix n_2 and generate all the n_1th rows elements $\mathcal{J}_{n_1 n_2}$ of the Jacobian matrix. Therefore to enable formulation (4.2.8) we provide a perturbation at the n_2th component of vector V and calculate the residual component $R_{n_1}(V+\Delta v_{n_2})$ at different n_1, which is changed because of the perturbation, and then generate the n_1th row elements $\mathcal{J}_{n_1 n_2}$ of Jacobian matrix. When the residual component $R_{n_1}(V+\Delta v_{n_2})$ is not changed at n_1 we have $\mathcal{J}_{n_1 n_2} = 0$.

The above procedure can be done for each cell (i,j), i.e., after the perturbations of all five variables respectively in the cell (i,j) we can obtain thirteen 5×5 matrixes by the numerically approximate implementation of the following derivations:

$$\begin{aligned}
&\partial R_{cell(i-2,j)}/\partial v_{cell(i,j)},\ \partial R_{cell(i-1,j-1)}/\partial v_{cell(i,j)},\ \partial R_{cell(i-1,j)}/\partial v_{cell(i,j)},\\
&\partial R_{cell(i-1,j+1)}/\partial v_{cell(i,j)},\ \partial R_{cell(i,j-2)}/\partial v_{cell(i,j)},\ \partial R_{cell(i,j-1)}/\partial v_{cell(i,j)},\\
&\partial R_{cell(i,j)}/\partial v_{cell(i,j)},\\
&\partial R_{cell(i,j+1)}/\partial v_{cell(i,j)},\ \partial R_{cell(i,j+2)}/\partial v_{cell(i,j)},\ \partial R_{cell(i+1,j-1)}/\partial v_{cell(i,j)},\\
&\partial R_{cell(i+1,j)}/\partial v_{cell(i,j)},\ \partial R_{cell(i+1,j+1)}/\partial v_{cell(i,j)},\ \partial R_{cell(i,j+2)}/\partial v_{cell(i,j)}.
\end{aligned} \tag{4.2.9}$$

Therefore we obtain thirteen 5×5 submatrixes of the Jacobian matrix in five columns. After the cell proceed from (1,1) to (I,J) we can obtain the whole Jacobian matrix.



where ×: inviscid flux needed, Δ: viscous flux needed,
square: the perturbation position. *Physical* state variables used within bold line.

Fig.4.2.3 The fluxes should be calculated

Because the Jacobian matrix elements $\mathcal{J}_{n_1n_2}$ and $\mathcal{J}_{n_2n_1}$ are generated from the perturbations at the n_2th component and the n_1th component of vector V respectively, they are normally not equal and, therefore the Jacobian matrix is non-symmetric.

We store the Jacobian matrix using the following thirteen 4-dimensional arrays:

PW2(I,J), PSW(I,J), PW1(I,J), PNW(I,J), PS2(I,J), PS1(I,J),
PC(I,J), PN1(I,J), PN2(I,J), PSE(I,J), PE1(I,J), PNE(I,J), PE2(I,J)

For a cell (i,j) the above arrays are thirteen 5×5 matrixes, within five rows of the Jacobian matrix, and correspond to the derivations in (4.2.5) consecutively.

Using a column by column method to generate the Jacobian matrix results in thirteen 5×5 matrixes in five columns of the matrix. These 5×5 matrixes are obtained due to the variable perturbations in cell (i,j) and are:

PW2(i+2,j), PSW(i+1,j+1), PW1(i+1,j), PNW(i+1,j-1),
PS2(i,j+2), PS1(i,j+1), PC(i,j), PN1(i,j-1), PN2(i,j-2),
PSE(i-1,j-1), PE1(i-1,j), PNE(i-1,j-1), PE2(i-2,j).

When using the formulation (4.2.8) to calculate the elements of the Jacobian matrix, we need to calculate the residuals in all 13 cells after a perturbation. This includes the calculations of 36 inviscid fluxes and 36 viscous fluxes (Fig.4.2.3) for calculating residuals R_{n1} $(V+\Delta v_{n2})$, and then generating the elements of the Jacobian matrix.

4.2.2 A simplified procedure for generating Jacobian matrix

Since the calculation of the residual in each cell involves linearly combining the inviscid and viscous numerical fluxes on the cell interfaces (3.5.6) we can re-write the formulation (4.2.8) by using fluxes directly instead of using residuals. We have

$$\partial R_{cell(i,j)}/\partial v_{cell} = \Big\{ \big[\big(\{ FI_{i+1/2,j} - FI_{i-1/2,j} + FI_{i,j+1/2} - FI_{i,j-1/2} + FV_{i+1/2,j} - FV_{i-1/2,j} + FV_{i,j+1/2} - FV_{i,j-1/2} \} + \overline{H}_{i,j} \big) (V+\Delta v_{cell}) - \big(\{ FI_{i+1/2,j} - FI_{i-1/2,j} + FI_{i,j+1/2} - FI_{i,j-1/2} + FV_{i+1/2,j} - FV_{i-1/2,j} + FV_{i,j+1/2} - FV_{i,j-1/2} \} + \overline{H}_{i,j} \big) (V) \big] \Big\} / \Delta v_{cell}$$

where $\overline{H}_{i,j} = A_{cell} H_{i,j}$.

Therefore a new simplified procedure can be derived. In the above modified formulation we can cancel terms, such that we only need to calculate the flux affected by the variable perturbation. The simplified procedure then has the potential to decrease computation time. From Fig.4.2.1 we know that only 8 inviscid fluxes and 12 viscous fluxes vectors need to be calculated five times for generating all the 13 5×5 matrixes. They are

$FI_{i-3/2,j}$, $FI_{i-1/2,j}$, $FI_{i+1/2,j}$, $FI_{i+3/2,j}$, $FI_{i,j-3/2}$, $FI_{i,j-1/2}$, $FI_{i,j+1/2}$, $FI_{i,j+3/2}$

$$FV_{i-1,j-1/2},\ FV_{i-1,j+1/2},\ FV_{i,j-1/2},\ FV_{i,j+1/2},\ FV_{i+1,j-1/2},\ FV_{i+1,j+1/2}$$
$$FV_{i-1/2,j-1},\ FV_{i+1/2,j-1},\ FV_{i-1/2,j},\ FV_{i+1/2,j},\ FV_{i-1/2,j+1},\ FV_{i+1/2,j+1}.$$

As we saw in the original formulation (4.2.8) 36 inviscid fluxes and 36 viscous fluxes need to be calculated five times for the generation of the 13 5×5 matrixes. Therefore the new simplified procedure takes only 8/36 and 12/36 of the computation time of the original procedure for inviscid and viscous fluxes respectively. Since the locally conical Navier-Stokes equations include the 'source terms', the simplified procedure only takes 1/13 of the computation time of the original procedure for the 'source terms'. It is noted that by using the new procedure we need to store the all fluxes FI(V) and FV(V) in the procedure for evaluating the non-linear system R (V) from the physical state variable, and these fluxes do not need to be calculated in the procedure for generating the element of Jacobian matrix.

We now present some detailed examples of fluxes that need to be calculated in order to generate the Jacobian matrix elements by using the column by column method:

(1) calculating PS2(i,j+2) includes the calculation of

inviscid flux between points (i,j+1) and (i,j+2) (Fig.4.2.4a), i.e., we have

$$PS2(i,j+2) = \left[FI_{i+3/2,j}\ (V+\Delta v_{cell}) - FI_{i+3/2,j}\ (V)\right] / \Delta v_{cell}$$

(2) calculating PS1(i,j+1) includes the calculation of

inviscid flux between points (i,j) and (i,j+1);

inviscid flux and between points (i,j+1) and (i,j+2);

viscous flux between points (i,j) and (i,j+1);

viscous flux between points (i-1,j+1) and (i,j+1);

viscous flux between points (i,j+1) and (i+1,j+1) (Fig.4.2.4b), i.e., we have

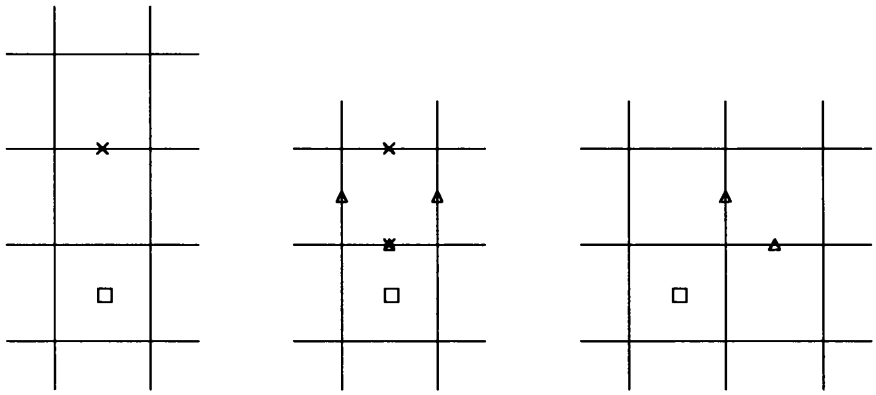
$$PS1(i,j+1) = \begin{cases} [\langle\{FI_{i+1/2,j+1} + FI_{i,j+3/2} \\ + FV_{i+1/2,j+1} - FV_{i-1/2,j+1} - FV_{i,j+1/2}\rangle\}\ (V+\Delta v_{cell}) \\ -\langle\{FI_{i+1/2,j+1} + FI_{i,j+3/2} \\ + FV_{i+1/2,j+1} - FV_{i-1/2,j+1} - FV_{i,j+1/2}\rangle\}\ (V)] \\ / \Delta v_{cell} \end{cases}$$

(3) calculating PSW(i+1,j+1) includes the calculation of

viscous flux between points (i+1,j) and (i+1,j+1);

viscous flux between points (i,j+1) and (i+1,j+1) (Fig.4.2.4c), i.e., we have

$$PSW(i+1,j+1) =$$
$$\left[(FV_{i+1,j+1/2} - FV_{i+1/2,j+1})(V+\Delta v_{cell}) - (FV_{i+1,j+1/2} - FV_{i+1/2,j+1})(V)\right] / \Delta v_{cell}\ .$$



where ×: inviscid flux needed, Δ: viscous flux needed,

square: the position of the perturbation.

Fig.4.2.4 The fluxes to be calculated in the simplified procedure

Another benefit of this simplified procedure is that it reduces the number of cells to the minimal value, in which the physical state variables need to be used in the procedure for generation the Jacobian matrix. This result can be shown by comparing Figs.4.2.1 and 4.2.3.

Table 4.2.1 gives the comparison of the cpu time in seconds needed for generation of the Jacobian matrix for the original and simplified procedures using one step of an explicit iteration as the scale.

Table 4.2.1 Comparison of CPU time for the generation of Jacobian matrix

Grids	Original procedure	Simplified procedure
34 × 34	27.3	5.4
66 × 34	29.5	5.9
66 × 66	31.2	5.6
66 × 130	36.1	6.2

4.3 The linear solver

We now need to solve the non-symmetric linear system (4.2.3). There exists considerable literature in the area of iterative solutions of large linear systems. The conjugate gradient (CG) method for the solution of a symmetric positive-definite system is well established. For solving non-sysmmetric systems some CG type methods have been developed such as the bi-conjugate gradient (BiCG) method, the conjugate gradient squared (CGS) method and the CGSTAB method, which is similar to CGS and has the stability properties. On the other hand, the generalized minimal residual (GMRES) technique is an efficient method for solving non-sysmmetric systems [52] and has been used in a wide range

of applications. In the foregoing discussion the non-symmetric linear system (4.2.3) can be denoted as

$$\mathcal{A}x = b . \tag{4.3.1}$$

4.3.1 The CGS linear solver

The CGS method was introduced by Sonneveld et al. [53]. The algorithm is derived from BiCG proposed by Fletcher [54]. As in BiCG, the exact arithmetic CGS converges to the correct solution in at most N iterations provided that it does not break down, where N is the order of the linear system. CGS has several advantages over BiCG: the transpose of the coefficient matrix is not needed, the algorithm is theoretically assured to converge whenever BiCG does, and it generally converges more rapidly [53]. Assume x_0 is an initial guess, and $r_0 = b - \mathcal{A}x_0$, we describe below the CGS algorithm. The symbol $\mathcal{K}$ represents a suitable preconditioning matrix ($\mathcal{K} \approx \mathcal{A}$):

Step 1: Initially, we set

$$\tilde{r}_0, \text{ an arbitrary vector, such that } (\tilde{r}_0 , r_0) \neq 0,$$

$$\rho_0 = (\tilde{r}_0 , r_0), \quad \beta_0 = \rho_0, \quad p_{-1} = q_0 = 0;$$

Step 2: set i = 1,

$$u_i = r_i + \beta_0 q_i, \quad p_i = u_i + \beta_i(q_i + \beta_i p_{i-1}),$$

$$\text{solve } \tilde{p} \text{ from } \mathcal{K}\tilde{p} = p_i,$$

$$\tilde{v} = \mathcal{A}\tilde{p}, \quad q_{i+1} = u_i - \alpha_i \tilde{v}, \quad \alpha_i = \rho_i/(\tilde{r}_0 , \tilde{v}),$$

$$\text{solve } \tilde{u} \text{ from } \mathcal{K}\tilde{u} = u_i + q_{i+1},$$

$$x_{i+1} = x_i + \alpha_i \tilde{u},$$

if x_{i+1} is accurate enough then the iteration is ended, else

Step 3: set

$$r_{i+1} = r_i - \alpha_i \mathcal{A}\tilde{u}, \quad \rho_{i+1} = (\tilde{r}_0 , r_{i+1}),$$

if $\rho_{i+1} = 0$ then the method fails to converge, else

Step 4: set

$$\beta_{i+1} = \rho_{i+1}/\rho_i,$$

and let i = i+1 and go to step 2.

4.3.2 The GMRES linear solver

The GMRES algorithm was proposed by Saad and Schultz [52]. It seeks a solution x under the form $x = x_0 + z$ where x_0 is the initial guess and z belongs to the Krylov subspace $K=<r_0, \mathcal{A}r_0, ..., \mathcal{A}^{k-1}r_0>$ ($r_0=b-\mathcal{A}x_0$). The solution x is chosen such that $\|b-\mathcal{A}x\|$ is the minimum.

First we find an orthonormal basis of space K via Gramm-Schmidt orthonormalization. In this process, a (k+1)×k Hessenberg matrix $\mathcal{H}_k$ is formed. The following calculations are performed.

Initially, we set

$$v_1 = r_0, \quad \hat{v}_1 = \frac{v_1}{\|v_1\|},$$

and for i=1 to k

$$v_{i+1} = \mathcal{A}\hat{v}_i - \sum_{j=1}^{i} \beta_{i+1,j}\hat{v}_j, \quad \text{where } \beta_{i+1,j} = (\mathcal{A}\hat{v}_i, \hat{v}_j)$$

$$\hat{v}_{i+1} = \frac{v_{i+1}}{\|v_{i+1}\|}.$$

After k steps, the Hessenberg matrix is formed as

$$\mathcal{H}_k = \begin{pmatrix} \beta_{2,1} & \beta_{3,1} & \cdots & \beta_{k+1,1} \\ \|v_2\| & \beta_{3,2} & \cdots & \beta_{k+1,2} \\ 0 & \|v_3\| & \ddots & \vdots \\ \vdots & \vdots & \ddots & \beta_{k+1,k} \\ 0 & 0 & \cdots & \|v_{k+1}\| \end{pmatrix}_{(k+1),\,k} \tag{4.3.2}$$

If we denote $V_k = (\hat{v}_1, \hat{v}_2, ..., \hat{v}_k)$, the N×k matrix whose columns are the first k basis vectors, we have

$$
\begin{aligned}
V_{k+1}\,\mathcal{H}_k &= [\hat{v}_1, \hat{v}_2, \cdots, \hat{v}_k, \hat{v}_{k+1}]\,\mathcal{H}_k \\
&= [\hat{v}_1(\mathcal{A}\hat{v}_1, \hat{v}_1) + \hat{v}_2\,\|v_2\|, \\
&\quad \hat{v}_1(\mathcal{A}\hat{v}_2, \hat{v}_1) + \hat{v}_2(\mathcal{A}\hat{v}_2, \hat{v}_2) + \hat{v}_3\,\|v_3\|, \\
&\quad \vdots \\
&\quad \hat{v}_1(\mathcal{A}\hat{v}_{k-1}, \hat{v}_1) + \hat{v}_2(\mathcal{A}\hat{v}_{k-1}, \hat{v}_2) + \cdots + \hat{v}_k\,\|v_k\|, \\
&\quad \hat{v}_1(\mathcal{A}\hat{v}_k, \hat{v}_1) + \hat{v}_2(\mathcal{A}\hat{v}_k, \hat{v}_2) + \cdots + \hat{v}_{k+1}\,\|v_{k+1}\|] \\
&= [\mathcal{A}\hat{v}_1, \mathcal{A}\hat{v}_2, \cdots, \mathcal{A}\hat{v}_{k-1}, \mathcal{A}\hat{v}_k] = \mathcal{A}\,V_k
\end{aligned}
\tag{4.3.3}
$$

Having generated the orthonormal basis V_k we proceed to find $x = x_0 + z = x_0 + \sum_{l=1}^{k} y_l \hat{v}_l$, where y_l are real. Let us define $y = (y_1, y_2, ..., y_k)^T{}_k$, $e_1 = (1, 0, ..., 0)^T{}_{k+1}$ and $\delta = \|r_0\|$. We have

$$
\| b - \mathcal{A}x \| = \| b - \mathcal{A}(x_0 + \sum_{l=1}^{k} y_l \hat{v}_l) \| = \| r_0 - \mathcal{A}V_k y \| \tag{4.3.4}
$$

and

$$
\| b - \mathcal{A}x \| = \| V_{k+1}(\delta e_1 - \mathcal{H}_k y) \| = \| \delta e_1 - \mathcal{H}_k y \| \tag{4.3.5}
$$

Therefore

$$
\min_{z \in K} \| b - \mathcal{A}x \| = \min_{y \in R^k} \| \delta e_1 - \mathcal{H}_k y \| \tag{4.3.6}
$$

The problem is now reduced to the solution of a smaller least squares problem. Due to the special structure of the Hessenberg matrix $\mathcal{H}_k$, a Q-R factorization algorithm can easily be applied for the system $\mathcal{H}_k y = \delta e_1$ as follows:

We use H_{ij} to present the element of the matrix $\mathcal{H}_k$, for i = 1,2, ...,k, then

$$
\begin{aligned}
R_{ii} &= \sqrt{\sum_{j=1}^{i+1} H_{ji}^2} \\
\vec{q}_i^{\,T} &= (H_{1i}, \cdots, H_{i+1i}) / R_{ii} \\
c_i &= \delta\, q_{1i} \\
&\begin{cases}
R_{ii+1} = \sum_{j=1}^{i+1} q_{ji}\, H_{ji+1} \\
\quad \vdots \\
R_{ik} = \sum_{j=1}^{i+1} q_{ji}\, H_{jk}
\end{cases}
\end{aligned}
$$

$$\begin{cases} H_{i+1} = H_{i+1} - R_{ii+1}\,\vec{q}_i \\ \vdots \\ H_k = H_{i+1} - R_{ik}\,\vec{q}_i \end{cases}$$

We obtain

$$\begin{pmatrix} R_{11} & R_{12} & \cdots & R_{1k} \\ 0 & R_{22} & \cdots & R_{2k} \\ & & \ddots & \vdots \\ 0 & 0 & \cdots & R_{kk} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_k \end{pmatrix} \tag{4.3.7}$$

and

$$y_k = c_k / R_{kk}$$

and for i = k-1,...,2,1, we obtain

$$y_i = (c_i - \sum_{j=i+1}^{k} R_{ij}\, y_j) / R_{ii} .$$

For an efficient practical calculation, the dimension of the Krylov subspace, k, is very small compared to the order of the matrix $\mathcal{A}$ because storing all the previous directions is very costly. In application, the algorithm is restarted every k steps until the required accuracy is achieved.

Generally speaking, the CGS and GMRES algorithms have the property of super-linear convergence, and in practice the above two linear solvers need to be used with a preconditioner. One method is to add the preconditioner by applying the schemes explicitly to the systems

$$\mathcal{P}\mathcal{A}\,x = \mathcal{P}\,b \tag{4.3.8a}$$

or

$$\mathcal{A}\,Q(Q^{-1}\,x) = b \tag{4.3.8b}$$

or other mixed forms, where $\mathcal{P}$ and Q are referred to, respectively, as left preconditioner and right preconditioner. A second method is to add the preconditioner in the iterative scheme as described in the CGS approach above. For the GMRES scheme we can use the first method, but for the CGS scheme both methods of preconditioner are available.

A family of efficient preconditioners arises out of the incomplete lower-upper (LU) factorization of $\mathcal{A}$ and is referred to as ILU(k). Here k represents the level of fill-in. The expression k = 0 implies no fill-in beyond the original non-zero pattern. If k = 1, the fill-in caused by the original non-zero pattern is allowed, but no further fill-in caused by these recently filled-in elements is permitted, and so on. As k increases, the preconditioning improves whilst also becoming more expensive to run. After the LU preconditioner is generated we can: choose $\mathcal{K}$= LU in the preconditioned CGS scheme; choose $\mathcal{P}$=$\mathcal{Q}$= $(LU)^{-1}$ to change the system; or change the system as

$$L^{-1} \mathcal{A} x = L^{-1} b \tag{4.3.9}$$

and let

$$y = U x \tag{4.3.10a}$$

i.e., we have to solve the new system as

$$L^{-1} \mathcal{A} U^{-1} y = L^{-1} b . \tag{4.3.10b}$$

Because in the solution of the lower or upper system there are involved backwards and forwards substitution, which imposes a sequential bottle-neck, the ILU(k) preconditioner appears not to be suitable for parallel computation. Another drawback is that we need memory to store the lower and the upper matrixes, i.e., equivalently when k = 0 we need to store another Jacobian matrix.

In the current research we need then to develop a new linear solver, which is robust with relatively less requirement for memory and without the sequential bottle-neck. At first we tested a simple preconditioner, i.e., the block diagonal matrix $\mathcal{D}$, then we solved the system as

$$\mathcal{D}^{-1} \mathcal{A} x = \mathcal{D}^{-1} b . \tag{4.3.11}$$

This diagonal preconditioning has the following advantages: (1) it is simple to programme; (2) the operation is localised so that parallelization can be implemented effectively. However, it has been found from numerical tests of the current LCNS problems with a 34×34 grid, that this simple preconditioning alone is not able to overcome the non-convergence when using the GMRES method as illustrated in Fig.4.3.1.

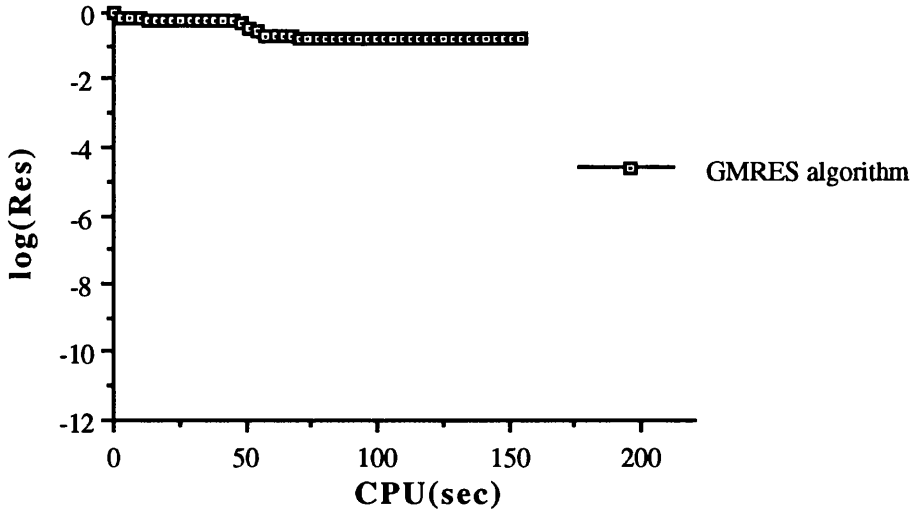


Fig.4.3.1 Convergence of GMRES algorithm for $(\mathcal{D}^{-1}\mathcal{A})\mathbf{x}=\mathcal{D}^{-1}\mathbf{b}$

We then introduced a damping factor α into Eq.(4.3.11)

$$(\alpha I + \mathcal{D}^{-1}\mathcal{A})\,\mathbf{x} = \mathcal{D}^{-1}\mathbf{b}\,,\ (\alpha > 0) \tag{4.3.12}$$

For this new linear system numerical tests were carried out using both GMRES and CGS methods with a 34×34 grid. Fig.4.3.2 shows the norm of the residual vector (Res) change during the iterative procedure of the GMRES method as applied to Eq.(4.3.12) with different values of the damping factor α. Fig.4.3.3 (a)-(d) show the comparison of the convergence of Res with the GMRES and CGS methods as applied to Eq.(4.3.12) with different values of the damping factor α.

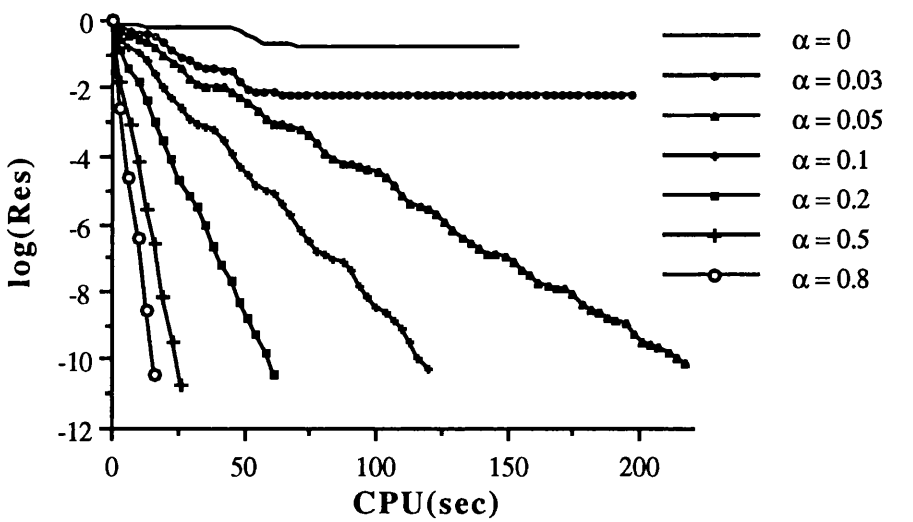
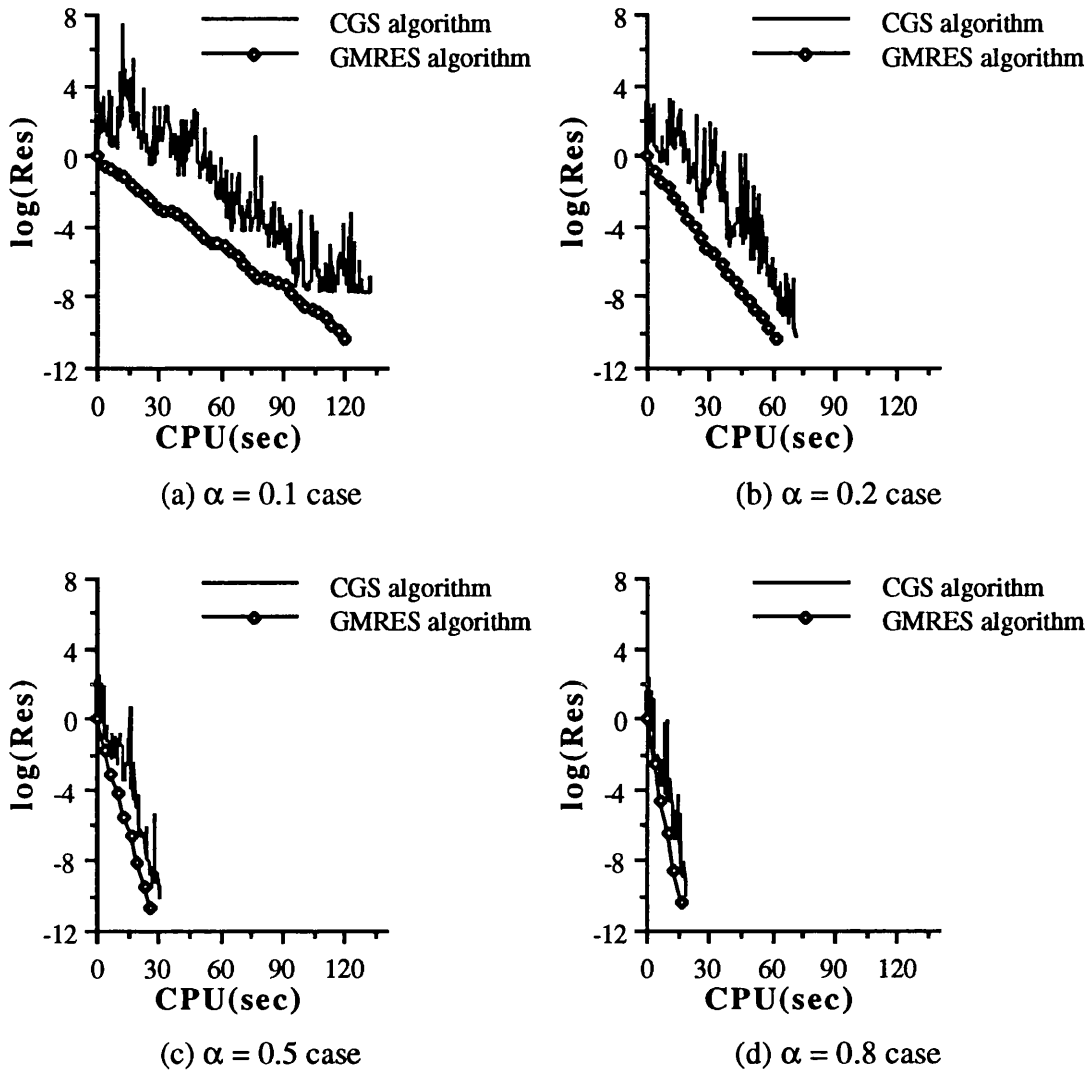


Fig.4.3.2 Convergence of GMRES algorithm for $(\alpha I+\mathcal{D}^{-1}\mathcal{A})\mathbf{x}=\mathcal{D}^{-1}\mathbf{b}$



(a) $\alpha = 0.1$ case (b) $\alpha = 0.2$ case

(c) $\alpha = 0.5$ case (d) $\alpha = 0.8$ case

Fig.4.3.3 GMRES and CGS algorithms for $(\alpha I+\mathcal{D}^{-1}\mathcal{A})\mathbf{x}=\mathcal{D}^{-1}\mathbf{b}$

The figures illustrate that (1) the larger the value of α the faster the convergence for both methods, (2) with a very small α the lack of convergence mentioned earlier still appears, (3) the Res decreases monotonically for the GMRES method but not for the CGS method, and (4) the convergence of the GMRES method is faster than that of the CGS method. Observation (3) is more important regarding the choice of the GMRES method for the linear solver in this research. Because the solution of the linear system is only the inner iterative loop of Newton's method we only require the Res of the linear system reduced by a few orders of magnitude. However the oscillation observed will delay the convergence.

The discovery that the modified linear system (4.3.12) can be solved by GMRES and CGS methods plays a key role for us to construct new linear solver.

4.3.3 The α-GMRES linear solver

It is clear that Eq.(4.3.12) is not equivalent to Eq.(4.3.1). To solve Eq.(4.3.1), an outer loop has to be introduced. This is done through a multi-level iterative scheme written as

$$(\alpha I + \mathcal{D}^{-1}\mathcal{A})\, x^{m+1} = \mathcal{D}^{-1} b + \alpha x^{m}. \qquad (4.3.13)$$

Given x^m, the above equation is solved for x^{m+1} using the GMRES method. This procedure is continued until the sequence x^m is converged, and the convergent vector is the solution of the original linear system. This procedure could be thought of as an inner iterative loop of the GMRES algorithm combined with the outer iterative loop of the α-GMRES algorithm. It is obvious that only the outer iterative loop has the property of linear convergence. We have proved the following convergence theorem for the iterative procedure (4.3.13) as follows:

Theorem:

(1) If x^m converges to x^*, x^* will be the solution of Eq.(4.3.1).

(2) There exists a positive number $\beta > 0$ such that if $0 < \alpha < \beta$, the iterative procedure (4.3.13) converges.

Proof:

(1) This is an obvious result of Eq.(4.3.13).

(2) From Eq.(4.3.13), we have

$$\begin{aligned} x^{m+1} - x^{m} &= (\alpha I + \mathcal{D}^{-1}\mathcal{A})^{-1}[(\mathcal{D}^{-1}b + \alpha x^{m}) - (\mathcal{D}^{-1}b + \alpha x^{m-1})] \\ &= \alpha (\alpha I + \mathcal{D}^{-1}\mathcal{A})^{-1}(x^{m} - x^{m-1}) \\ &= \alpha^{m} (\alpha I + \mathcal{D}^{-1}\mathcal{A})^{-m}(x^{1} - x^{0}). \end{aligned}$$

Thus

$$\| x^{m+1} - x^{m} \| \le \alpha^{m} \| (\alpha I + \mathcal{D}^{-1}\mathcal{A})^{-m} \| \, \| x^{1} - x^{0} \| \le [\alpha \| (\alpha I + \mathcal{D}^{-1}\mathcal{A})^{-1} \|]^{m} \| x^{1} - x^{0} \|.$$

Let us define a positive function f: $f(\alpha) = \| (\alpha I + \mathcal{D}^{-1}\mathcal{A})^{-1} \|$. The function f is obviously a continuous function of α and $f(0) = \| \mathcal{D}^{-1}\mathcal{A} \|$ is a constant. From the continuity of f, given a constant $c > 0$, we can find $\alpha_1 > 0$ such that when $0 < \alpha < \alpha_1$, we have

$$0 < f(\alpha) < f(0) + c.$$

On the other hand, for a given constant ε: $0 < \varepsilon < 1$, we can find $\alpha_2 > 0$ such that

$$\alpha_2 [f(0) + c] < 1 - \varepsilon.$$

Let $\beta = \min\{\alpha_1, \alpha_2\}$ and choose α: $0 < \alpha < \beta$, then we have

$$\alpha \| (\alpha I + \mathcal{D}^{-1}\mathcal{A})^{-1} \| = \alpha f(\alpha) < \alpha_2[f(0) + c] < 1-\varepsilon.$$

Thus

$$\| x^{m+1} - x^m \| < (1-\varepsilon)^m \| x^1 - x^0 \|.$$

Therefore we have proved the convergence of the iterative procedure (4.3.13).

In practical application, a value of α has to be selected to balance the convergence of the outer iterative procedure (4.3.13) and that of the inner GMRES algorithm. Two parameters are used in solving the linear system to give the convergence criterion, i.e., ε_1, the convergence criterion of the inner GMRES algorithm and ε_2 the convergence criterion of the outer loop of the α-GMRES algorithm.

Fig.4.3.4 shows the convergent results of the α-GMRES method for the LCNS equations in the present test case, where $\varepsilon_1 = 10^{-1}$, and $\varepsilon_2 = 10^{-10}$.

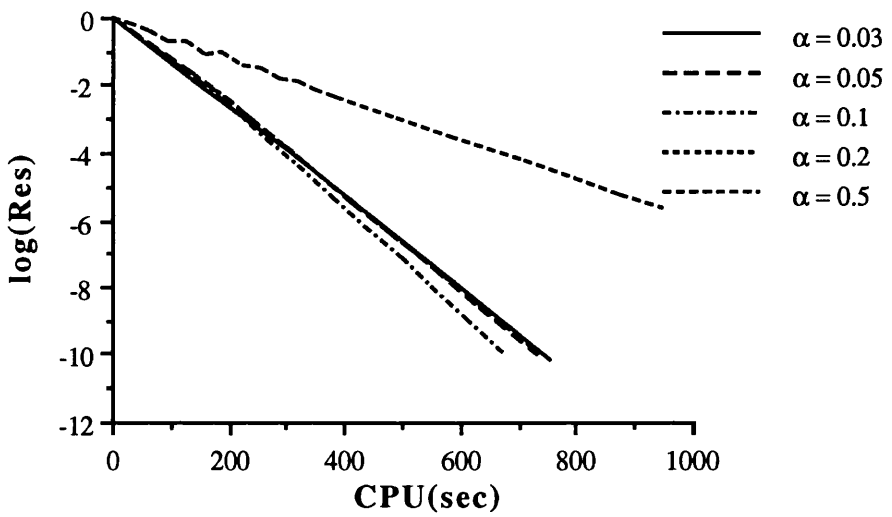


Fig.4.3.4 Convergence of α-GMRES algorithm for $\mathcal{A}x=b$

Table 4.3.1 shows the details of the calculation for different α. It should be noted that for the case of $\alpha = 0.03$ the GMRES algorithm cannot converge to machine zero but this does not influence the convergence of the α-GMRES algorithm because the full convergence of the inner iteration is not required. From this table we can also see that the performance of the multi-iterative method is not sensitive to the choice of α tested for α around 0.1.

Table 4.3.1 Iterative number for different α

α	ITERATIVE NUMBER of OUTER LOOP of α-GMRES ALGORITHM	TOTAL RESTART NUMBER of GMRES ALGORITHM
0.03	48	235
0.05	64	226
0.10	114	210
0.20	214	224

Generally speaking, the factors to effect the convergence of the linear solver are (1) the character of the Jacobian matrix; (2) the dimension of the matrix; (3) the dimension of the Krylov subspace; (4) the parameter α; (5) the choice of the convergence criteria.

4.4 Computational tests for LCNS equations

In this section we will give the numerical test results for the LCNS equations for the different cases. The flow problems involve the Mach 7.95 laminar flow around a sharp cone with half angle 10^o with a cold wall (T_w =309.8K) at a high angle of attack of 24^o. The Reynolds number is 4.1×10^6 and the flow temperature is 55.4K. Let ε_3 be the convergence criterion of the whole algorithm, and we require the relative Res $R/R_b < \varepsilon_3$, where R_b is the Res of the starting step. In this paper we always set $\varepsilon_3 = 10^{-10}$. Solutions include different levels of grid spacing.

The first test case is for the 34×34 grid. The unknown variables amount to 32×32×5 and the initial guess is set by using 1000 explicit Runge-Kutta iterations which result in the relative Res R/R_b = 0.4715e-3.

Table 4.4.1-3 provides the comparisons of different choices of convergence criterion ε_1 for the inner GMRES algorithm, the damping factor α, and the dimensions k of the Krylov subspace in GMRES algorithm, respectively.

From these tables we see that when we choose the convergence criterion of the outer loop of α-GMRES algorithm as $\varepsilon_2 = 10^{-2}$, all test cases are convergent in six steps. It is found that the choice of the convergence criterion of the outer loop of α-GMRES algorithm will determine the whole algorithm convergence property. We do not need to solve the linear systems using the GMRES or α-GMRES methods to a high accuracy as long as a reasonable convergence in the non-linear iteration can be achieved

Table 4.4.1 Effects of different ε_1

	k=40,α=.08 ε1=.08 ε2=.01		k=40,α=.08 ε1=.1 ε2=.01		k=40,α=.08 ε1=.12 ε2=.01	
	Res	Time	Res	Time	Res	Time
1	.2370e-2	173	.2367e-2	166	.2365e-2	161
2	.2147e-2	281	.2121e-2	266	.2017e-2	261
3	.6221e-5	381	.7098e-5	365	.9625e-5	348
4	.1663e-5	616	.9382e-6	577	.8301e-6	553
5	.3024e-8	726	.2673e-8	680	.2070e-8	708
6	.1210e-10	951	.1767e-10	897	.1381e-10	912

Table 4.4.2 Effects of different α

	k=40,α=.06 ε1=.1 ε2=.01		k=40,α=.08 ε1=.1 ε2=.01		k=40,α=.1 ε1=.1 ε2=.01	
	Res	Time	Res	Time	Res	Time
1	.2396e-2	163	.2367e-2	166	.2363e-2	169
2	.2098e-2	270	.2121e-2	266	.2002e-2	269
3	.7487e-5	373	.7098e-5	365	.9995e-5	369
4	.8237e-6	594	.9382e-6	577	.7712e-6	561
5	.2300e-8	752	.2673e-8	680	.1925e-8	731
6	.1515e-10	950	.1767e-10	897	.1688e-10	940

Table 4.4.3 Effects of different k

	k=30,α=.08 ε1=.1 ε2=.01		k=40,α=.08 ε1=.1 ε2=.01		k=50,α=.08 ε1=.1 ε2=.01	
	Res	Time	Res	Time	Res	Time
1	.2365e-2	170	.2367e-2	166	.2368e-2	176
2	.2049e-2	301	.2121e-2	266	.2061e-2	284
3	.8697e-5	395	.7098e-5	365	.8427e-5	379
4	.9436e-6	634	.9382e-6	577	.1097e-5	573
5	.2908e-8	771	.2673e-8	680	.3457e-8	705
6	.2496e-10	993	.1767e-10	897	.2705e-10	888

Fig.4.4.1 shows the results using different convergence criteria for the iterative linear solver with the damping factor $\alpha = 0.1$ and the dimensions of the Krylov subspace $k = 40$. In this figure we can see that the smaller the value of ε_2, the more the present

Newton's method approaches the quadratic convergence property sought. This does not mean necessarily that less CPU time is used. As can be seen, the convergence for the explicit scheme is slow even though local time stepping has already been employed for efficiency.

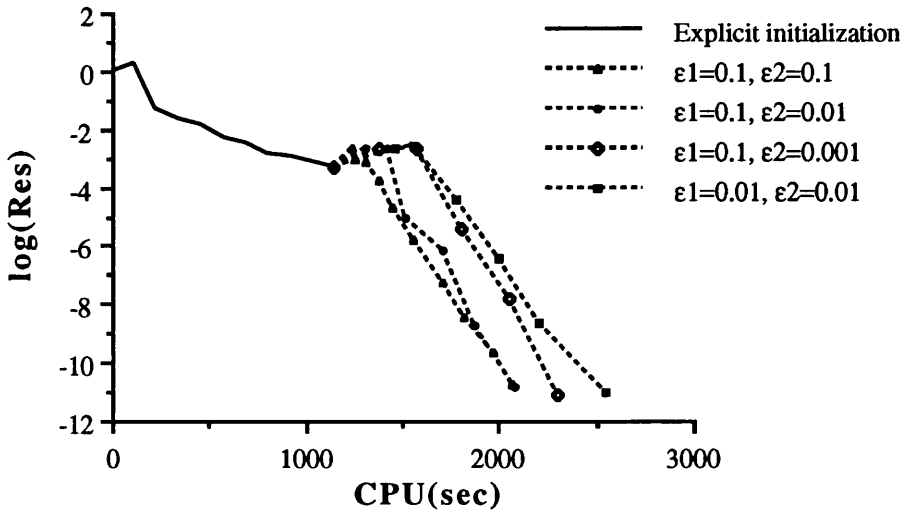


Fig.4.4.1 Parameter tests for the Newton's method (grid 34×34)

Fig.4.4.2 shows the results for the dimensions of the Krylov subspace k = 40, and the convergence criterion of the outer loop of α-GMRES algorithm $\varepsilon_2 = 10^{-2}$. In this figure we can see that the damping factor α around 0.1 is the best choice.

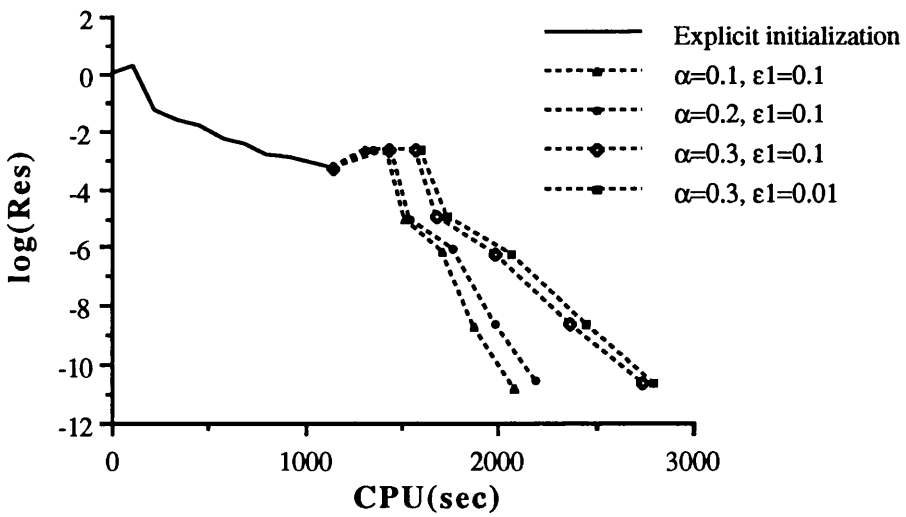


Fig.4.4.2 Parameter tests for the Newton's method (grid 34×34)

The second test case is for the 66×34 grid. The number of unknown variables is then equal to 64×32×5, and the initial guess is set by using explicit Runge-Kutta iterations. In Fig.4.4.3 we will compare the results by using different switch points of 1000, 2000, and 3000 steps of explicit iterations respectively, and we choose the damping factor α = 0.1, the

dimensions of the Krylov subspace k = 40, the convergence criterion of the outer loop of α-GMRES algorithm $\varepsilon_2 = 10^{-1}$, and the convergence criterion of the inner GMRES algorithm $\varepsilon_1 = 10^{-1}$. After 2000 and 3000 steps of explicit iterations the relative Res are R/R_b = 0.3735e-2 and R/R_b = 0.1998e-3 respectively. The figure shows that when the initial guess chosen is not sufficiently converged the present Newton's method may produce oscillations in its convergence procedure or is even not convergent (in this case the switch point is at 1000 steps).

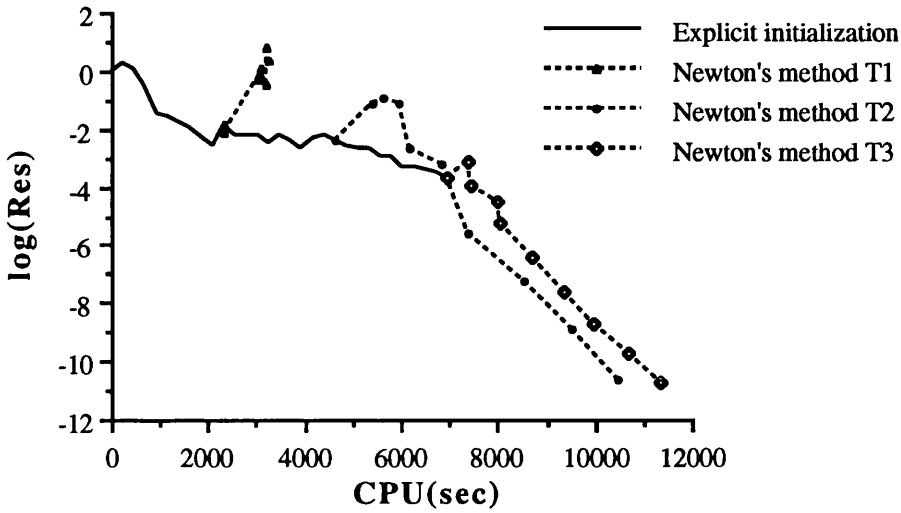


Fig.4.4.3 Parameter tests for the Newton's method (grid 66×34)

The third test case is for the 66×66 grid, for which the number of unknown variables is equal to 64×64×5, and the initial guess is again set by using explicit Runge-Kutta iterations. Fig.4.4.4 illustrates the results by using different switch points at 2000 and 3000 explicit iterations respectively. We choose the damping factor α = 0.1, the dimensions of the Krylov subspace k = 50, the convergence criterion of the outer loop of α-GMRES algorithm $\varepsilon_2 = 10^{-2}$, and the convergence criterion of the inner GMRES algorithm $\varepsilon_1 = 10^{-1}$. After 2000 and 3000 explicit iterations the relative Res are R/R_b = 0.2256e-2 and R/R_b = 0.1735e-3 respectively.

Fig.4.4.5 shows the results for the same grid size with damping factor α = 0.1, the dimensions of the Krylov subspace k = 50 case, and the initial guess set by using 2000 steps of explicit Runge-Kutta iterations resulting in the relative Res R/R_b = 0.2256e-2. When choosing 1500 steps as the switch point, the Newton's method failed to converge.

The fourth test case is for the 66×130 grid, in which the number of unknown variables is equal to 64×128×5, and the initial guess is set by using 3000 steps of explicit Runge-Kutta iterations result with the relative Res R/R_b = 0.1537e-3. Fig.4.4.6 shows the results for different dimensions of Krylov subspace k = 50 and k = 70. The other parameters

are: damping factor $\alpha = 0.1$; the convergence criterion of the inner GMRES algorithm $\varepsilon_1 = 10^{-1}$; and the convergence criterion of the outer loop of α-GMRES algorithm $\varepsilon_2 = 10^{-2}$.

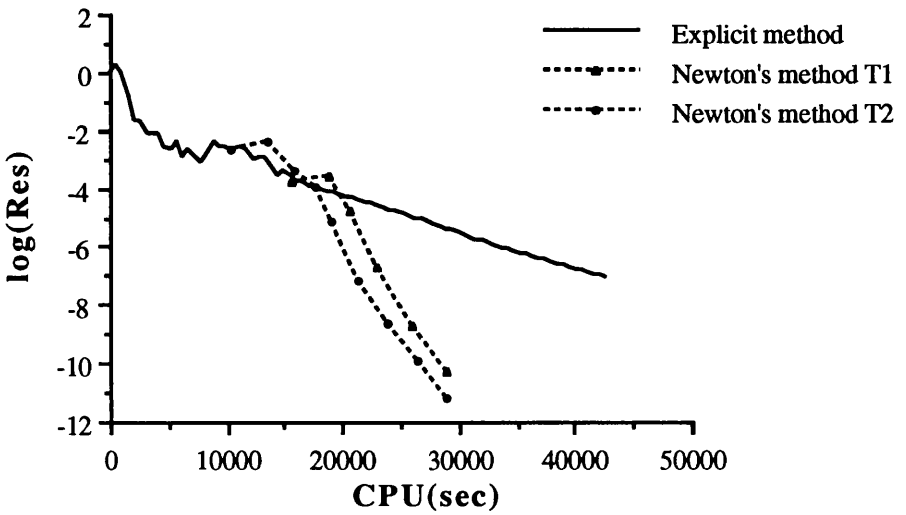


Fig.4.4.4 Parameter tests for the Newton's method (grid 66×66)

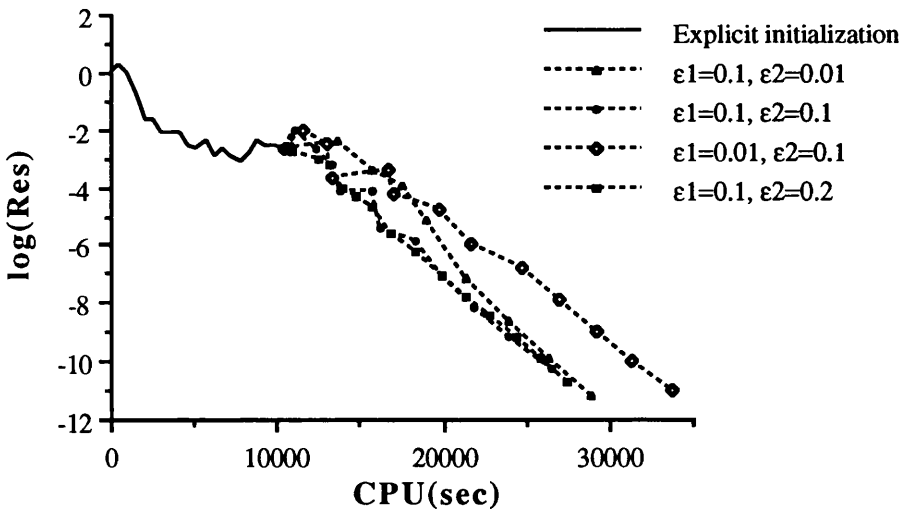


Fig.4.4.5 Parameter tests for the Newton's method (grid 66×66)

Fig.4.4.7 plots the convergence against computing time for calculations using the Newton's method and quasi-Newton's method [55] on a 33×33 grid. After switching to the implicit method, the solutions converge quadratically or superlinearly respectively and the Res reduced to machine zero in 4 or 8 iterations.

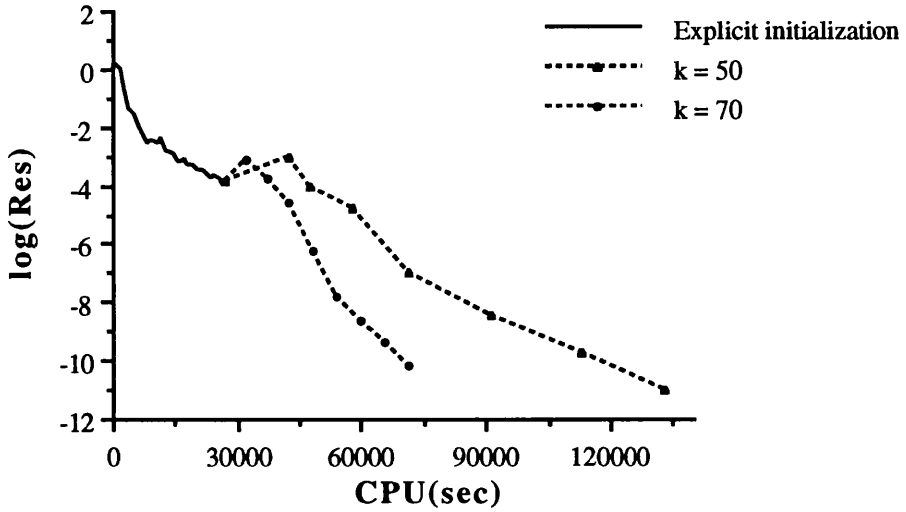


Fig.4.4.6 Parameter tests for the Newton's method (grid 66×130)

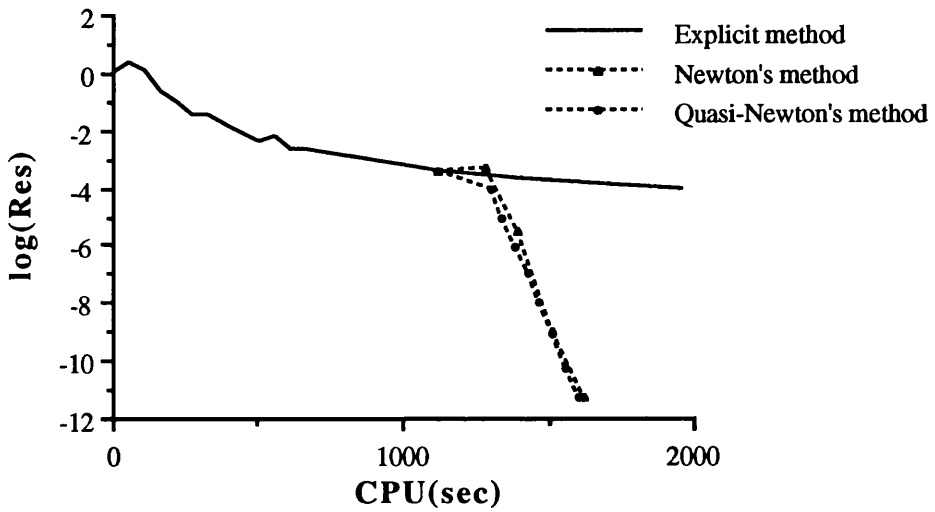


Fig.4.4.7 Convergence of the Newton's and Quasi-Newton's methods as compared with the Runge-Kutta explicit method (grid 33×33)

Another test carried out was for the flow over the cone at 12° angle of attack. For this test case we used a 34×34 grid, and the parameters used are: damping factor $\alpha = 0.1$; convergence criterion of the inner GMRES algorithm $\varepsilon_1 = 10^{-1}$; and convergence criterion of the outer loop of α-GMRES algorithm $\varepsilon_2 = 10^{-2}$. Fig.4.4.8 shows the results for the different dimension k of Krylov subspace chosen. As can be seen, when k increases from 30 to 50 the CPU time used decreases, however when k increases from 50 to 60 the CPU time used increases. Generally when k increases the convergence speed of solving the linear systems can increase, but we need to use more memory and at each iterative step the amount

of the calculation increases, such that overall more CPU time is used.

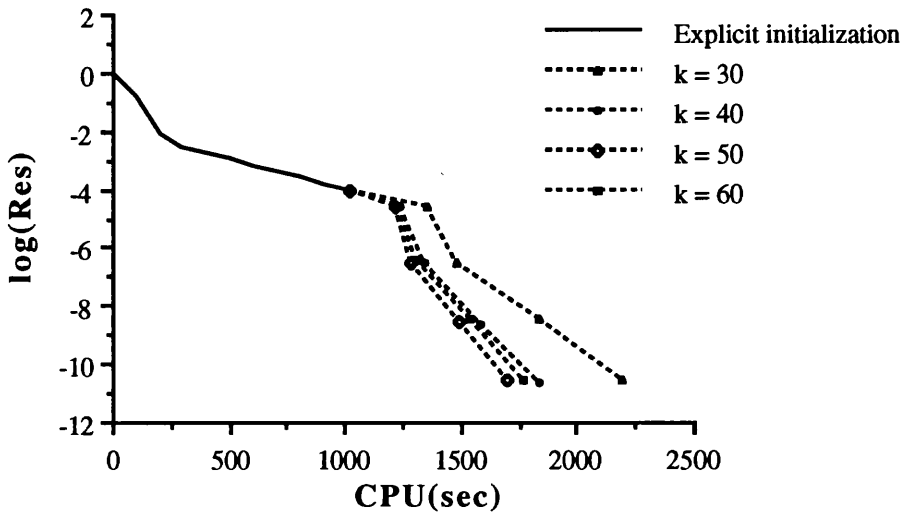


Fig.4.4.8 Parameter tests for the Newton's method for the cone at 12° angle of attack

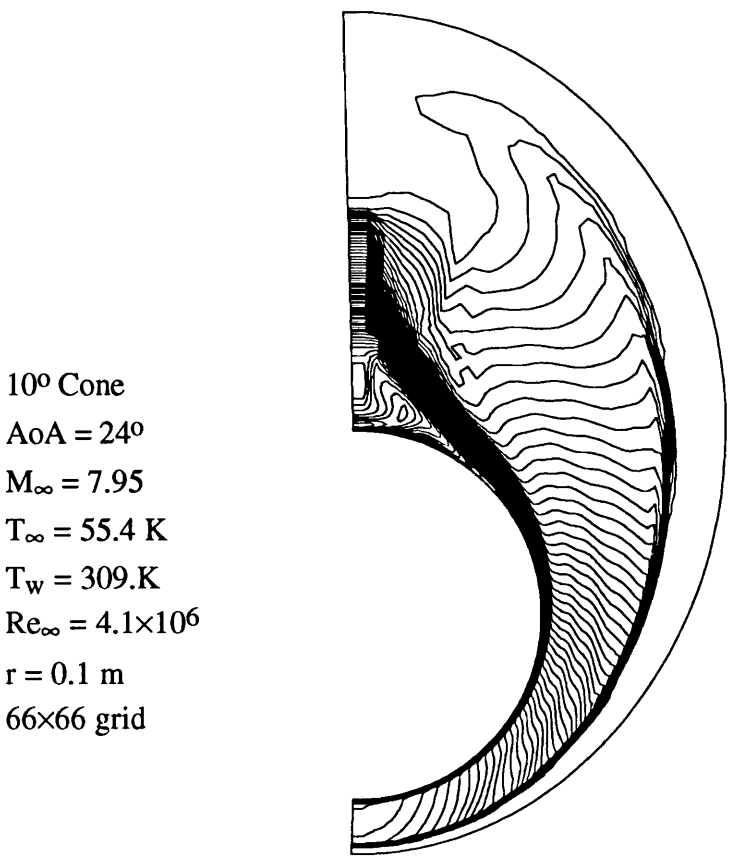


Fig.4.4.9 Flow conditions and cross flow temperature contours

Fig.4.4.9-10 illustrates the flow conditions and the cross-sectional view of temperature and pressure of the solved flowfield for 66×66 grid for the high angle of attack case, in which the strong bow shock wave on the windward side and the separated shear layer on the leeward side can clearly been seen.

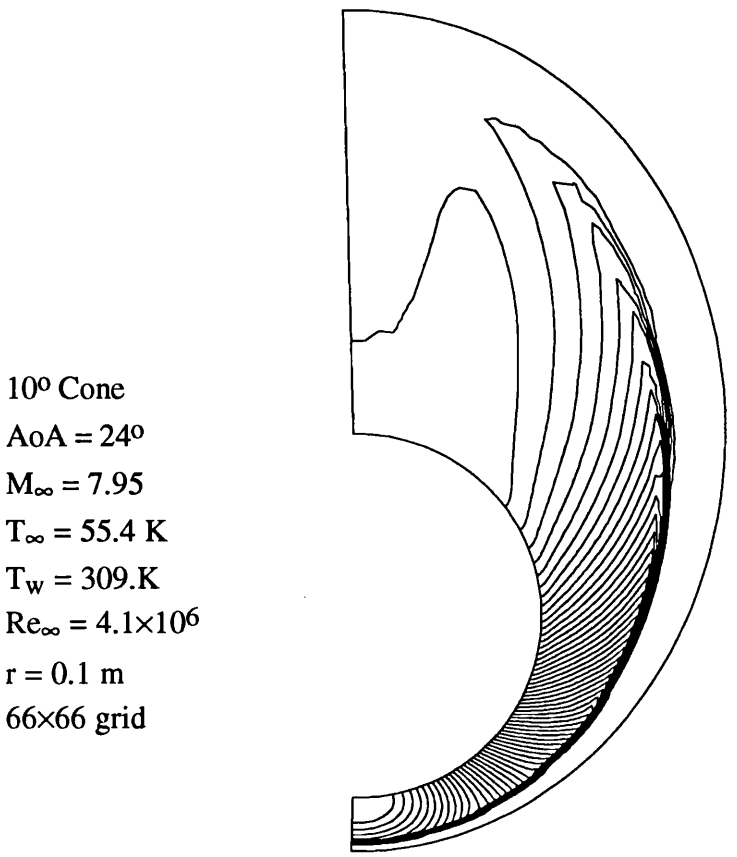


Fig.4.4.10 Cross flow pressure contours

4.5 Conclusions

The Newton's method has been developed as a fully implicit method for solving the steady state locally conical Navier-Stokes equations. The flow field involves complex physical phenomenon and a high order high resolution scheme was used in the discretisation of the equations. For all the numerical test cases the relative Res can decrease by ten orders of magnitude and, therefore the robustness and efficiency of this method have been proved. In the procedure for developing the numerical algorithm, the generation of the numerically approximate Jacobian matrix and the solution of the large sparse non-symmetric linear systems play key roles. The new linear solver is not dependent on the physical problems discussed and, therefore, it might be suitable for solving general large sparse non-symmetric linear systems.

Table 4.5.1 shows the required CPU time for numerical computation for the test cases with the different incidences and different grid sizes. From this table, relative to grid 34×34 at an angle of incident 24°, we can see that the grids 66×34, 66×66, and 66×130 increase the size of problem by 2^1, 2^2, and 2^3 respectively, and the CPU time required only increases to $2.57\times2^1\times Tc_S$, $3.66\times2^2\times Tc_S$, and $4.60\times2^3\times Tc_S$ respectively, where Tc_S is the total CPU times required for the 34×34 grid, 24° angle of attack case. The trend in computing time can thus be approximated as

$$T = (1.6+n)\times2^n\times Tc_S$$

where n is the exponent representing the size of problem and T is the CPU time predicted. With this trend it is therefore anticipated that this Navier-Stokes solver can be used for solving very large grid and/or 3-dimensional flow problems.

Table 4.5.1 Performance of the N-S solver on IBM RS6000/320H

Inc. deg.	Grid size INxJN	No of gr. pts	Rel.	CPU-s Expl't Switch	Rel.	CPU-s Impl't only	Rel.	CPU-s Expl't+ Impl't	Rel.
12	34 34	1156	1.00	1100	0.92	500	0.56	1600	0.76
24	34 34	1156	1.00	1200	1.00	900	1.00	2100	1.00
24	66 34	2244	1.94	7000	5.83	3500	3.89	10500	5.00
24	66 66	4356	3.77	16000	13.33	13000	14.44	29000	13.81
24	66 66	4356	3.77	11000	9.17	17000	18.89	28000	13.33
24	66 130	8580	7.42	27000	22.50	45000	50.00	72000	34.29

Since the Newton's method needs a 'good initial' guess, a different solver needs to be used to provide this starting solution. In this work a four-step Runge-Kutta explicit method is used. Other methods could also be used, e.g., an Euler explicit method. From the test results we can see that the switch point, at which the solver from the explicit method is shifted to the Newton's method, needs to be chosen carefully to achieve the convergence.

Even though this Navier-Stokes solver did not deal with turbulent phenomenon, there is no reason to restrict it to solving the laminar flow problem only. Solving Reynolds' averaged Navier-Stokes equations for the turbulent flow problems is a research direction for the future.

Chapter Five

Parallel solution for Navier-Stokes equations

5.1 Introduction

In the previous chapter we have developed the Newton's method for solving the locally conical Navier-Stokes equations for the hypersonic laminar flow around a slender cone on a sequential computer. It is seen that using this method there are very large requirements for computer memory. One way to tackle the memory intensive problems is by using a parallel computer instead of using the sequential computer. Parallel processing also offers the potential for speedup of computations for existing methods. In developing a parallel algorithm there are various factors to consider. The most important factor is that the algorithm should not include sequential bottle-necks, such as backward and forward substitution in solving the lower or upper linear systems. The sequential bottle-neck will greatly decrease the efficiency of parallel algorithms. The other important factor is that we need to arrange the storage efficiently. This means attempting to avoid the data re-storage in different processors.

With parallel computation in mind the new linear solver proposed in the last chapter which has no sequential bottle-necks is particularly suitable for parallel implementation and this obviously plays a key role in the parallelization of the overall algorithm. In this chapter we will describe the parallel implementation of the Newton's method for solving the LCNS equations. During the process of developing the parallel numerical scheme, a very efficient data storage method has been proposed to store the non-zero elements of the Jacobian matrix of the non-linear systems, which causes no data overlap in different processors and then leads to a type of domain decomposition. Thus, in this research the parallelization uses data decomposition rather than direct domain decomposition. After data decomposition the new linear solver can be implemented in a parallel manner without any change of the algorithm. Since the parallel implementation does not change the original algorithm, every iterative step has its sequential counterparts on the global domain, and the convergence and the accuracy are maintained compared with the implementation on a single sequential computer. The simplified procedure proposed for generating the numerically approximate Jacobian matrix also contributes to decreasing the data re-storage in each processor.

The parallel computer employed in this work is the Meiko Computing Surface in the University of Glasgow, which is a coarse-grained and message-passing system composed of T800 transputers. The high level programming languages are Fortran and C. Fortran language is used in the test cases.

5.2 Parallel algorithm analysis

The algorithm developed in this chapter is suitable for the kind of parallel computers that have coarse-grained parallelism, MIMD, and distributed memory architecture.

5.2.1 Performance analysis

Assuming that there are P processors available, to be able to analyze them and to compare performance of algorithms we shall define the following:

T_S is the run time for one processor, which is the sequential case,
T_P is the run time for P processors.

and we can now define

$$\text{efficiency} = T_S / PT_P,$$
$$\text{and speedup} = T_S / T_P.$$

From the above definitions we obtain

$$\text{efficiency} = \text{speedup} / P$$

5.2.2 Communication time t_c

The cost of communication between neighbouring processors in a message-passing system can usually be modelled by the linear form:

$$t_c = L + kT$$

where L is the latency time to initialize the message, T is the transmission time of a byte and k is the number of bytes in the transmitted message.

Another method of estimating the efficiency of a parallel algorithm is through the cost of communication relative to calculation. If the parallel algorithm involves no additional calculation steps compared to the sequential algorithm we can estimate the cost of communication relative to calculation from the efficiency or speedup.

5.3 Parallel implementation for Navier-Stokes equations

Generally speaking, the algorithm developed for numerically solving the steady-state Navier-Stokes equations can be summarised by the following steps:

(1) Grid generation;
(2) Ordering of all the cells;
(3) Cell-centred finite volume method for discretization;
(4) High resolution upwind scheme for evaluating the convective flux;
(5) Second order central finite difference scheme for evaluating the diffusive flux;
(6) The non-linear system on the global domain formed by the residual vector;
(7) Iterative method for evaluation of the discretised *physical* state variables, which includes solving the linear systems.

Steps (1) to (6) are the same whether an explicit or implicit scheme is used, and up to step (6) we finish up with a N-dimensional non-linear *algebraic* system to be solved on the global domain as following:

$$R(V) = 0 \tag{5.3.1}$$

The different operations lie within step (7), and then we use different methods to solve Eq.(5.3.1). Generally speaking, we use the following iterative method:

$$\begin{aligned} \mathcal{M}^k \Delta V^k &= - R(V^k) \\ \Delta V^k &= V^{k+1} - V^k \end{aligned} \tag{5.3.2}$$

The different choices of matrix $\mathcal{M}^k$ control the iterative methods required, and we need to solve a linear system $\mathcal{M}^k \Delta V^k = - R(V^k)$ in each step of the iteration. If $\mathcal{M}^k$ is a diagonal matrix which is composed of $\Omega_c / \Delta t$, subscript c corresponding to the cell, the iterative method is Euler explicit. If

$$\mathcal{M}^k = \frac{1}{\Delta t} I + \left(\frac{\partial R}{\partial V}\right)^k$$

the iterative method is the backward Euler implicit method. And if

$$\mathcal{M}^k = \left(\frac{\partial R}{\partial V}\right)^k$$

the iterative method is the Newton's method. How to solve the linear system in parallel in step (7) plays the key role in the parallelization of the Navier-Stokes solver, since solving the linear system involves an inversion procedure which has global characteristics.

When a structured grid is used the first two steps are simple to implement. Notice that steps (3) to (6) are all implemented with local characteristics and therefore can involve parallel implementation by domain decomposition. Because in Newton's method in step (7)

the Jacobian matrix is also generated with the local characteristics we can handle the procedure in parallel using the same methods as in steps (3) to (6).

We will focus our discussion, then, on step (7) of the Newton's method for solving the laminar steady state approximate locally conical Navier-Stokes equations.

5.3.1 Data partition and corresponding domain decomposition

As described in chapter 3, we consider the numerical solution on a structured grid, the global control volumes in the cell centred finite volume method being given in the order {{ (i,j), j=1, J}, i=1, I}, where I is the number of control volumes in the coordinate η direction and J is the number of control volumes in the coordinate ζ direction. The unknown variables are set in the whole control volume (I,J). It is assumed that there are P processors available, and I = Int × P, where Int is an integer number.

As in the discussion of the sequential calculation in chapter 4, when the Newton's method is employed, we need to solve a N-dimensional large sparse non-symmetric linear *algebraic* system in the global domain in each iteration as follows

$$\mathcal{A}\, x = b$$

where $\mathcal{A}$ is a block 13-point diagonal matrix and N = I × J × 5.

In the Newton's method, the main memory is used in storing the Jacobian matrix, i.e., this matrix takes I×J×5×5×13 words of memory, however the memory for storing the discretised *physical* state variables, which has five elements in each cell, is I×J×5 words. We need, then, an efficient method to divide the storage into each processors. One method is through writing the Jacobian matrix $\mathcal{A}$ in columns as $\mathcal{A} = [\, \mathcal{A}^1, \mathcal{A}^2, ..., \mathcal{A}^P \,]$, and storing $\mathcal{A}^p$ in processor p, $\mathcal{A}^p$ is a N row, Int × J × 5 ≡ N_p column matrix, p=1,2,...,P. The reason for dividing the matrix in columns is that we will generate the matrix along columns. According to the method above we can divide any N-dimensional vector v as following

$$v = \begin{pmatrix} v^1 \\ v^2 \\ \vdots \\ v^P \end{pmatrix}$$

where v^p is the N_p-dimensional vector corresponding to $\mathcal{A}^p$. The vectors v^p are also stored in processor p, p=1,2,...,P.

Assume that v is the discretised *physical* state variables, then arising from the division process we know that each v^p corresponds to all the discretised *physical* state variables in a part of the global domain, and we call this type of part domain a subdomain. This type of subdomain is named the **first type subdomain**, and all first type subdomains contribute to the global domain, and have no overlap with each other. It can be said that we

now have a domain decomposition, and each subdomain includes Int × J cells (Fig.5.3.1). However we are required to generate the components of the residual vector and the elements of the Jacobian matrix in the cells in the first type subdomains and then solve the linear system. Since the calculation of a component of the residual vector in a cell needs to use the discretised *physical* state variables in the surrounding 13 cells (Fig.3.5.5), the calculation of the components of the residual vector in all cells in the first type subdomain needs to use the discretised *physical* state variables in the neighbouring subdomains. Based on the first type subdomain we can construct the **second type subdomain** (Fig.5.3.1) which is an extension of the first type subdomains. Similarly, since the generation of matrix elements needs to use the discretised *physical* state variables in the surrounding cells (Fig.4.2.3, Fig.4.2.1 for the original and simplified procedure respectively), the generation of the matrix elements in the first type subdomain needs to use the discretised *physical* state variables in the neighbouring subdomains. Based on the first type subdomain we can also construct the **third type subdomain** (Fig.5.3.1) which is again an extension of the first type subdomains. In Fig.5.3.1 we can see that different approximate procedures used for the Jacobian matrix generation will lead to different divisions (1) and (2) of the third type subdomain. Using the simplified procedure corresponds to type (1) and using the original procedure corresponds to type (2) respectively. It is obvious that the second or the third type subdomains have overlaps.

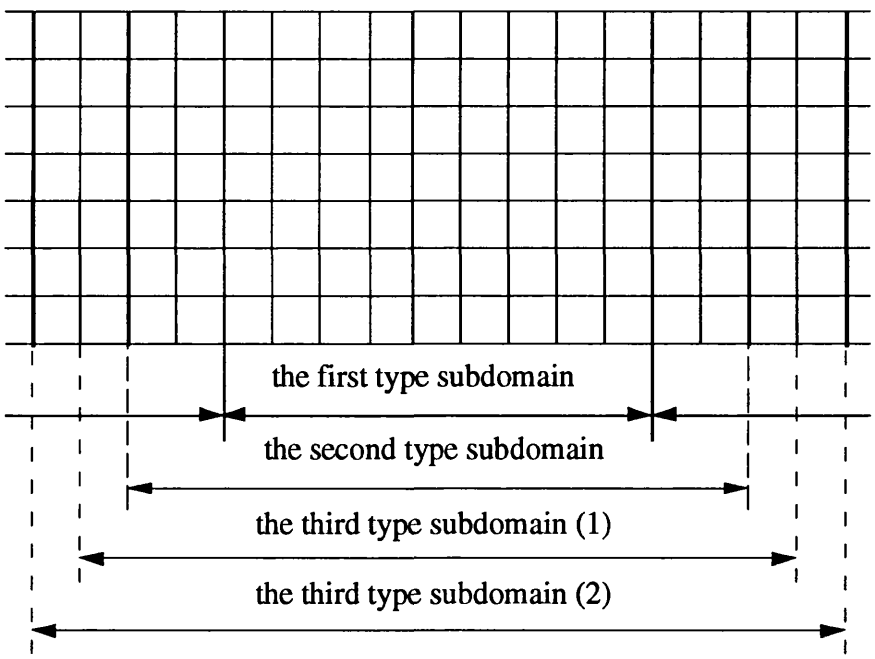


Fig.5.3.1 The pth subdomain of three type divisions of grid

5.3.2 Parallel generation of Jacobian matrix and residual vector

Because the third type subdomain include the second and the first type subdomains, then when storing the discretised *physical* state variables in the third type subdomain in each

processor, which results in the data storage overlaps, we can generate the components of the residual vector and the non-zero elements of the Jacobian matrix in all cells in the first type subdomain in each processor exactly the same as we did in the sequential case in the previous chapter. Here we can see also that the use of the simplified procedure for the Jacobian matrix generation decreases the storage in each processor.

From the next paragraph we will see that after solving the linear system in parallel, we obtain only the updated discretised *physical* state variables in the first type subdomain. Therefore an appropriate method of communication is needed for us to obtain the discretised *physical* state variables in the third type subdomain. If the Int is large enough, e.g., Int ≥ 3 or 4 for the simplified or original approximate procedure for generating the Jacobian matrix respectively, we find that every processor p needs only to transfer data with its neighbouring processors p-1 and p+1 only. This kind of communication can be called the **first type communication**. For the simplified approximate procedure to generate the Jacobian matrix, each processor needs to send the discretised *physical* state variables in 3×J cells to its neighbouring two processors respectively and also needs to receive the same amount of data from its neighbouring processors, i.e., the amount of words to be communicated between two neighbouring processors is 2×3×J×5, so that the total amount of communication is (P-1)×2×3×J×5. If we arrange the processors into a ring structure (Fig.5.3.2) we can transfer all the data in 4 steps, in each step each processor sends or receives 3×J×5 words (Fig.5.3.3). So the number of words transferred is equal just to 4×3×J×5.

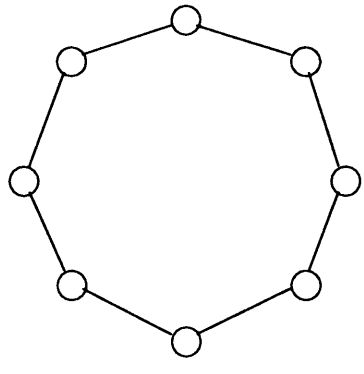


Fig.5.3.2 Ring architecture

If we use the original approximate procedure to generate the Jacobian matrix the total number of communications is (P-1)×2×4×J×5, and when using the above ring structured processor architecture the words transferred are equal to 4×4×J×5.

Incidentally, if the explicit scheme is used we need not generate the Jacobian matrix and, therefore we need only store the discretised *physical* state variables in the second type subdomain in each processor and calculate the components of the residual vector in the cells in the first type subdomain in each processor in exactly the same way as we did in the sequential case in the previous chapter. Then the updated discretised *physical* state variables

are in the first type subdomain. The first type communication is needed for us to obtain the discretised *physical* state variables in the second type subdomain. Each processor needs to send the discretised *physical* state variables in 2×J cells to its neighbouring two processors respectively and also needs to receive the same amount of data from its neighbouring processors. Then the amount of words to be communicated between two neighbouring processors is 2×2×J×5, so that the total amount of communication is (P-1)×2×2×J×5. When the processors have a ring structure we can transfer all the data in 4 steps with for each step, each processor sending 2×J×5 or receiving 2×J×5 words (Fig.5.3.3). Then the words transferred are equal to 4×2×J×5.

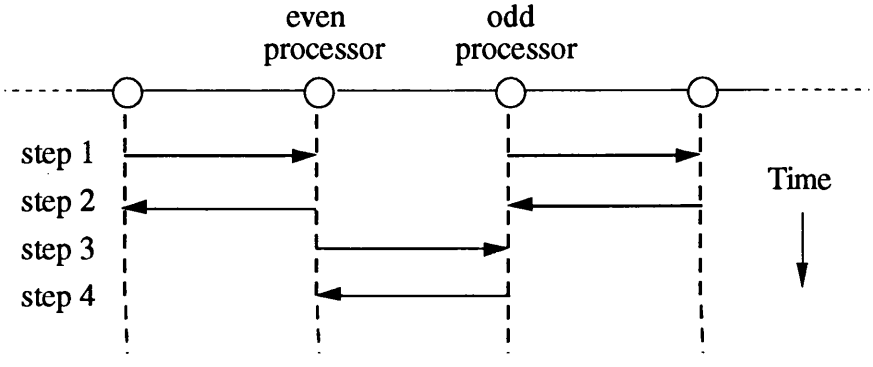


Fig.5.3.3 Communications in the matrix-vector multiplication

5.3.3 Parallel α-GMRES method

In the α-GMRES algorithm the matrix-vector multiplication is the main time consuming calculation, which can be implemented for a sparse matrix and a vector stored in different processors as follows:

$$\mathcal{A}v=(\mathcal{A}^1,\mathcal{A}^2,\cdots,\mathcal{A}^P)\left(\begin{pmatrix}v^1\\0\\\vdots\\0\end{pmatrix}+\begin{pmatrix}0\\v^2\\\vdots\\0\end{pmatrix}+\cdots+\begin{pmatrix}0\\0\\\vdots\\v^P\end{pmatrix}\right)=\mathcal{A}^1v^1+\mathcal{A}^2v^2+\cdots+\mathcal{A}^Pv^P$$

$$=\begin{pmatrix}*\\*\\\vdots\\*\end{pmatrix}+\begin{pmatrix}*\\*\\\vdots\\*\end{pmatrix}+\cdots+\begin{pmatrix}*\\*\\\vdots\\*\end{pmatrix}\Rightarrow\begin{pmatrix}\dot{v}^1\\0\\\vdots\\0\end{pmatrix}+\begin{pmatrix}0\\\dot{v}^2\\\vdots\\0\end{pmatrix}+\cdots+\begin{pmatrix}0\\0\\\vdots\\\dot{v}^P\end{pmatrix}$$

where "⇒" indicates the communication of data among different processors to form $\dot{v}^p$. In this way, the task of calculating $\mathcal{A}v$, for P processors, is divided by calculating $\mathcal{A}^p v^p$ on processor p. The resulting vector $\dot{v}^p$ is again distributed to the P processors. The only communication required in the calculation is in the formation of $\dot{v}^p$, and this is of the first type communication.

In this procedure the number of calculations is the same as in the sequential case, and it equals 2×(N-J)×6×5+N×5 multiplications and 2×(N-J)×6×5+N×5-N additions. The

number of words to be communicated between two neighbouring processors is 2×2×J×5, so that the total number of communications is (P-1)×2×2×J×5. As in last paragraph if we arrange the processors in a ring structure we can transfer all the data in 4 steps with at each step each processor sending 2×J×5 or receiving 2×J×5 words (Fig.5.3.3). So the number of words transferred are just equal to 4×2×J×5.

Another calculation in the α-GMRES algorithm is the inner product, which requires accumulating partial inner products carried out on each processor and broadcasting the summary to every processor. This type of communication can be called the **second type communication**. Suppose each processor could have any number of communication channels, i.e., when the ring structure is used for a P processor connection, each processor could also connect to any other of the processors, we could find that for the second type communication: (1) if there are 2 processors the communication needs 2 steps, i.e., one accumulation and one broadcast step; (2) if there are 4 processors the communication needs 4 steps, i.e., two accumulation and two broadcast steps; and (3) generally speaking, for P processors the communication needs $\log_2 P$ steps. However in practice with the transputer based system available we have only 4 communication channels for each processor. For the second type communication apart from the ring structure we need to design appropriate connections between processors. When P is less than 5 we still can achieve the above results, Fig.5.3.4 illustrates that 4 steps are needed for accumulation in the P=4 case. When P=5 and P=6 we need to increase this by 1 and 2 shift steps respectively. Fig.5.3.5 illustrates that 6 steps are needed for accumulation in the P=5 case. Other networks can achieve better results for $\log_2 P$ communications. One example is the Hypercube architecture parallel computer, which is illustrated in Fig.5.3.6. Because of the properties of the receiving message in the Meiko computer, when using it we need to use a master processor which carries out the accumulation and broadcast and therefore makes each processor synchronous. Incidentally, this type of communication is also required for us to obtain the norm of the residual vector.

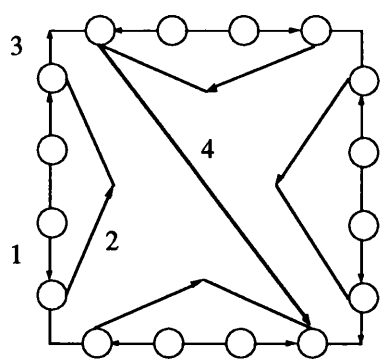
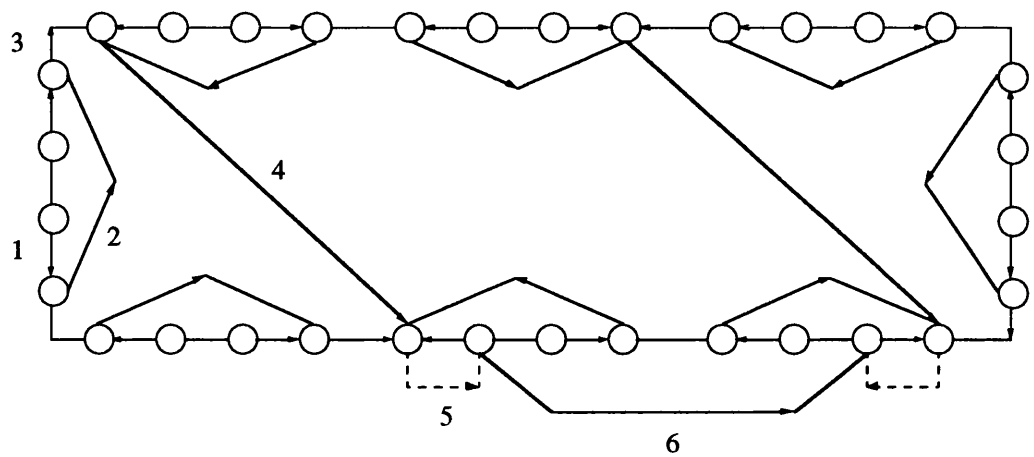


Fig.5.3.4 16 processors case

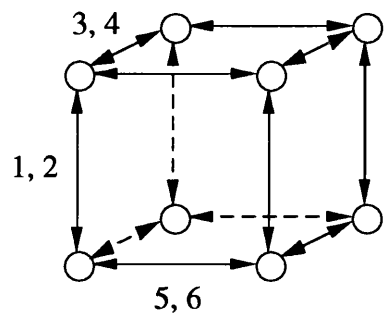
The calculation of the linear combination of vectors can be carried out by their

respective components distributed over each processor. Using this procedure no communication is needed.



The dashed line does not represent the actual connection between processors, this step is the transfer of the data to the neighbouring processor, and we call it the shift step.

Fig.5.3.5 32 processors case



For each step data is interchanged in one direction, and appropriate calculations are done. Thus after 6 steps we can obtain the inner products.

Fig.5.3.6 3-dimensional hypercube

The parallel α-GMRES algorithm

Let ε_1 be the convergence criterion of the inner GMRES algorithm and ε_2 be the convergence criterion of the outer loop of the α-GMRES algorithm. In processor p, we perform the following calculations and communications.

Step 1: Initialization

Set an initial guess $x^p{}_0$, we have

$$\mathcal{A}^p x^p{}_0 = \bar{r}^p{}_0 \quad \text{, (needs the first type communication)}$$

$$r^p{}_0 = b^p - \bar{r}^p{}_0 \ ,$$

and

$$\|r_0\| = \sqrt{\sum_{p=1}^{P} (r^p{}_0 \, , r^p{}_0)} \quad \text{, (needs the second type communication)}$$

Let $\delta = \|r_0\|$ and $\tilde{x}^p{}_0 = x^p{}_0$, set $\omega = 0$.

Step 2: Calculate $\mathcal{B} = (\alpha I + \mathcal{D}^{-1}\mathcal{A})$ and $\mathcal{D}^{-1}b$

We can write $\mathcal{B}$ in columns as $\mathcal{B} = [\mathcal{B}^1, \mathcal{B}^2, ..., \mathcal{B}^P]$, which has the same stencil as matrix $\mathcal{A}$. $\mathcal{D}^{-1}$ is formed by calculating the inverse of the 5×5 submatrix in each processor separately. Because $\mathcal{D}^{-1}\mathcal{A}$ involves a row transformation for $\mathcal{A}$, and $\mathcal{A}$ is divided in columns, $\mathcal{D}^{-1}\mathcal{A}$ can be performed in each processor provided that appropriate communications are arranged. This type of communication is named the **third type communication**. In a similar fashion to the above discussion we know that the amount of data communicated is equal to 4×2×J×5×5, and it needs only be done once in solving a linear system. Then the α is added in the diagonal elements in each processor so that we have $\mathcal{B}^p$ in each processor. $\mathcal{D}^{-1}b$ can be performed in each processor without any communication and we can use $\mathcal{D}^{-1}b^p$ to present the vector in processor p.

Step 3: Calculation

Let $\tilde{b}^p = \mathcal{D}^{-1} b^p + \alpha \tilde{x}^p{}_0$, we have

$$\mathcal{B}^p \tilde{x}^p{}_0 = \tilde{r}^p{}_0 \quad \text{, (needs the first type communication)}$$

$$\tilde{r}^p{}_0 = \tilde{b}^p - \tilde{r}^p{}_0 \quad ,$$

and then set

$$\tilde{v}^p{}_1 = \tilde{r}^p{}_0 \quad ,$$

so we have

$$\|\tilde{v}_1\| = \sqrt{\sum_{p=1}^{P} (\tilde{v}^p{}_1 \, , \tilde{v}^p{}_1)} \quad \text{, (needs the second type communication)}$$

$$\hat{v}^p{}_1 = \frac{\tilde{v}^p{}_1}{\|\tilde{v}_1\|} \quad .$$

Set i = 0. If $\omega = 0$ then let $\delta_1 = \|\tilde{v}_1\|$ and set $\omega = 1$.

Step 4: Set i = i + 1,

$$\mathcal{B}^p \hat{v}^p{}_i = \tilde{v}^p{}_i \quad \text{. (needs the first type communication)}$$

The elements of the Hessenberg matrix are calculated for j = 1 to i using

$$\beta_{i+1,j} = \sum_{p=1}^{P} (\bar{v}^p{}_i \, , \hat{v}^p{}_j) \quad .$$

We then calculate

$$\tilde{v}^p{}_{i+1} = \bar{v}^p{}_i - \sum_{j=1}^{i} \beta_{i+1,j} \hat{v}^p{}_j \quad ,$$

and

$$\| \tilde{v}_{i+1} \| = \sqrt{\sum_{p=1}^{P} (\tilde{v}^p{}_{i+1} \, , \tilde{v}^p{}_{i+1})} \quad , \text{(needs the second type communication)}$$

and normalise the base vector as follows

$$\hat{v}^p{}_{i+1} = \frac{\tilde{v}^p{}_{i+1}}{\| \tilde{v}_{i+1} \|} \quad .$$

If i < k go to Step 4, else we have the Hessenberg matrix

$$\mathcal{H}_k = \begin{pmatrix} \beta_{2,1} & \beta_{3,1} & \cdots & \beta_{k+1,1} \\ \| \tilde{v}_2 \| & \beta_{3,2} & \cdots & \beta_{k+1,2} \\ 0 & \| \tilde{v}_3 \| & \ddots & \vdots \\ \vdots & \vdots & \ddots & \beta_{k+1,k} \\ 0 & 0 & \cdots & \| \tilde{v}_{k+1} \| \end{pmatrix}_{(k+1) \times k} .$$

Step 5: Use a Q-R algorithm to find y such that

$$\| \delta_2 e_1 - \mathcal{H}_k y \| = \min_{y_0 \in R^k} \| \delta_2 e_1 - \mathcal{H}_k y_0 \| \quad ,$$

where $y = (y_1, y_2, ..., y_k)^T$, $e_1 = (1, 0_1, ..., 0_k)^T$.

So we have $\tilde{x}^p = \tilde{x}^p{}_0 + \sum_{i=1}^{k} y_i \hat{v}^p{}_i$.

Step 6: Calculation

$$\mathcal{B}^p \tilde{x}^p = \bar{r}^p{}_0 \quad , \text{(needs the first type communication)}$$

$$\tilde{r}^p{}_0 = \tilde{b}^p - \bar{r}^p{}_0 \ ,$$

and we have

$$\|\tilde{r}_0\| = \sqrt{\sum_{p=1}^{P} (\tilde{r}^p{}_0 \ , \tilde{r}^p{}_0)} \quad \text{. (needs the second type communication)}$$

If $\|\tilde{r}_0\| < \delta_1 \times \varepsilon_1$ then we go to next step, else set i = 0 and let $\tilde{x}^p{}_0 = \tilde{x}^p$, go to step 3.

Step 7: Calculation

$$\mathcal{A}^p \tilde{x}^p = \bar{r}^p{}_0 \quad \text{, (needs the first type communication)}$$

$$\tilde{r}^p{}_0 = b^p - \bar{r}^p{}_0 \ ,$$

and

$$\|\tilde{r}_0\| = \sqrt{\sum_{p=1}^{P} (\tilde{r}^p{}_0 \ , \tilde{r}^p{}_0)} \quad \text{. (needs the second type communication)}$$

If $\|\tilde{r}_0\| < \delta \times \varepsilon_2$ then stop, else set i = 0, $\omega = 0$, and let $\tilde{x}^p{}_0 = \tilde{x}^p$, go to step 3.

Therefore the overall parallel algorithm can be described as follows: at each Newton's iterative step we update the discretised *physical* state variables by solving the linear system in the first type subdomain in each processor, arrange the communication for each processor to have the discretised *physical* state variables in the third type subdomain, and then generate the components of the residual vector and the elements of the Jacobian matrix in all cells in the first type subdomain in each processor. Then we go to the next Newton's iterative step.

5.4 Numerical tests

The foregoing numerical tests have been carried out on the flow problem cases as in chapter 4.

First we test the linear solver. The Jacobian matrix is generated after 1000 steps of explicit iterations. The global grids in the flow cross section tested are 34×34, and 66×34 respectively, thus the unknown variables are in 32×32, and 64×32 respectively. The resulting large sparse non-symmetric linear systems to be solved are block 13-point structured matrices of order 32×32×5 and 64×32×5 corresponding to the different grids. Fig.5.4.1 shows the speedup achieved using from 1 to 8 processors for solving the linear

system using the α-GMRES algorithm. Fig.5.4.2 shows the convergence histories for different numbers of processors in the 34×34 grid case. The convergence criterion of the inner GMRES algorithm ε_1 is 10^{-1} and the convergence criterion of the outer loop of the α-GMRES algorithm ε_2 is 10^{-10}, the Krylov subspace k is 30, and the damping factor α is 0.1. When using 8 processors for the 34×34 grid the efficiency is equal to 81.5%, for the 66×34 grid the efficiency is equal to 92.2%. It is seen that better efficiencies are achieved with the large grid.

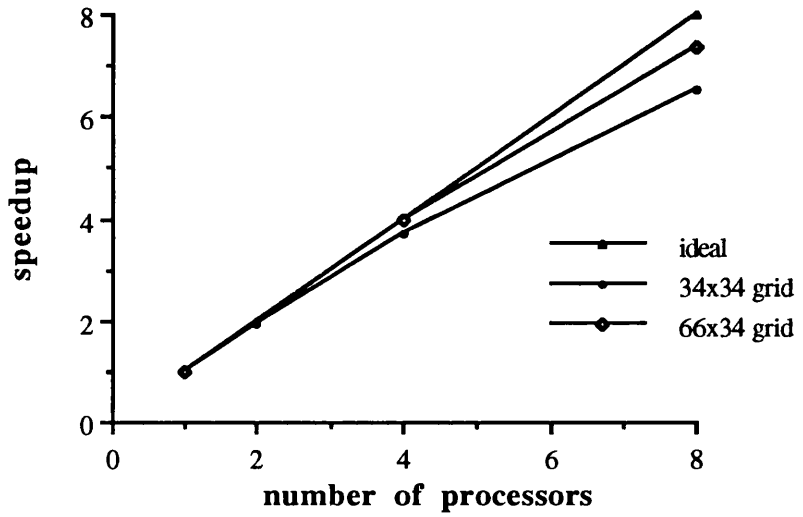


Fig.5.4.1 Speedup with different grids for α-GMRES algorithm

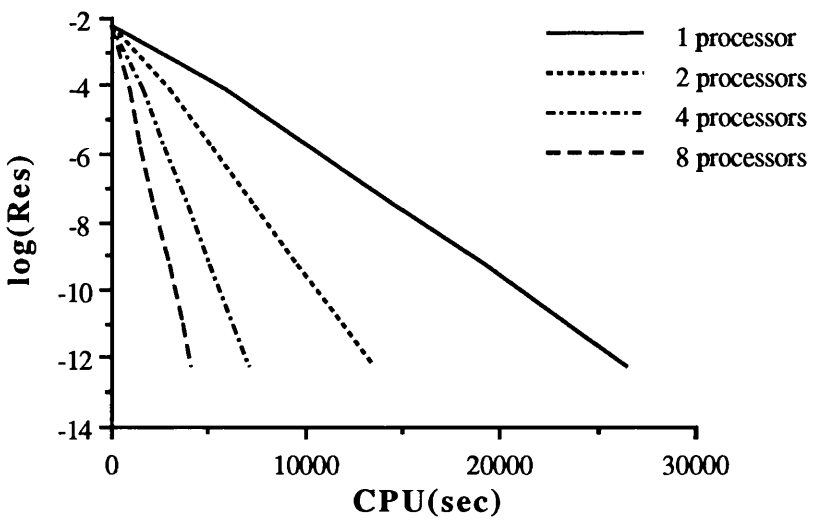


Fig.5.4.2 Convergence of α-GMRES algorithm with different number of processors

Fig.5.4.3 shows the speedup achieved using from 1 to 8 processors for solving the

complete locally conical Navier-Stokes equations. Fig.5.4.4 shows the convergence histories for different numbers of processors, the convergence criterion of the inner GMRES algorithm ε_1 is 10^{-1}, the convergence criterion of the outer loop of α-GMRES algorithm ε_2 is 10^{-2}, and the convergence criterion of the whole Navier-Stokes solution ε_3 is 10^{-10}, the Krylov subspace k is 30, and the damping factor α is 0.1. When using 8 processors for the 34×34 grid the efficiency is equal to 79.4%, for the 66×34 grid the efficiency is equal to 89.7%.

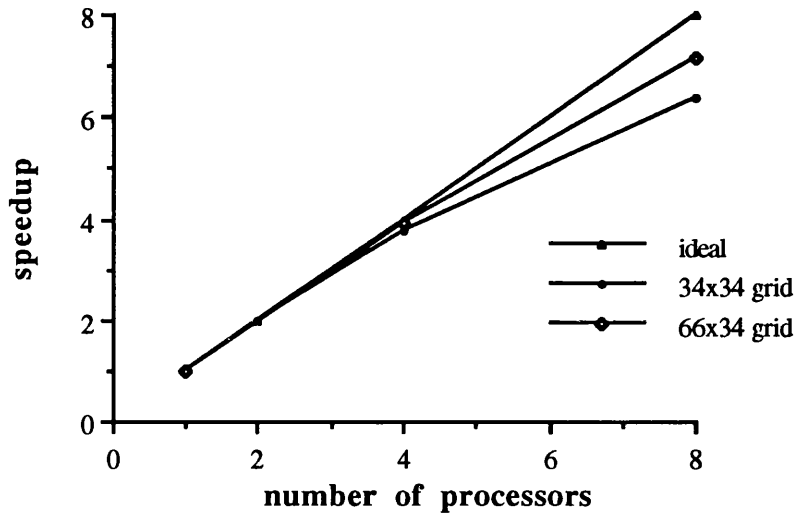


Fig.5.4.3 Speedup for the whole LCNS computation using different grids

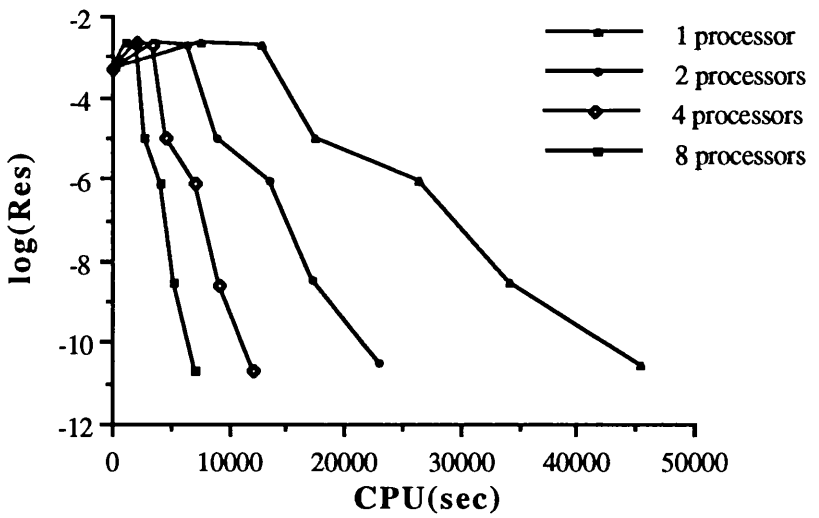


Fig.5.4.4 Convergence of the whole LCNS solution with different number of processors

Fig.5.4.5 shows the memory required on each processor for solving the Navier-

Stokes equation. As can be seen, the requirement on memory for each processor decreases as the number of the processors increases.

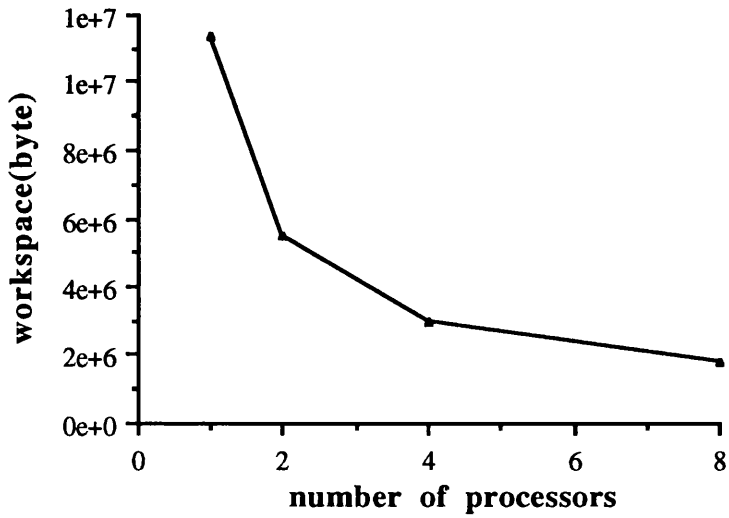


Fig.5.4.5 Memory requirement against processor number

5.5 Conclusions

The parallel implementation of the algorithm for solving the steady state Navier-Stokes equations has been developed. It includes the parallel implementation of Newton's method for solving the non-linear *algebraic* system. In this procedure an efficient data storage method is proposed for storing the Jacobian matrix of the non-linear systems, which has no data overlap in different processors and leads to a type of domain decomposition. This data storage method is not suitable for a non-sparse matrix since the number of communications will increase greatly in the matrix-vector multiplication. An alternative storage method can be used by storing the Jacobian matrix in rows if we could generate the non-zero elements of the Jacobian matrix in rows, and in this case the third type communication is not needed for doing the preconditioning. The linear solver, α-GMRES, is suitable for parallel computation without any sequential bottle-necks, which plays a key role in the parallelization of the overall algorithm. This parallel linear solver is also expected to be useful for solving general large sparse non-symmetric linear systems.

From the test results we can see that the larger the number of grid point (Int) the higher is the efficiency of the computation. Since Int = I / P, in which I is the number of control volumes in a particular direction and P is the number of processors used, and we cannot increase I arbitrary, increased Int means that we can only use limited processors. Therefore the algorithm developed in this work is more suitable for a parallel computing system with few powerful processors, rather than many small ones.

Chapter Six

General remarks

6.1 Concluding remarks

The Newton's method for solving the steady state locally conical Navier-Stokes equations for the hypersonic laminar flow has been presented. When a high order high resolution spatial discretisation scheme is used it is very difficult to solve the resulting large sparse non-symmetric linear system in the Newton's method. Although ILU factorization could provide an efficient preconditioner for robust and efficient CG type linear solvers and GMRES linear solver to tackle the linear system above, each overall linear solver is not suitable for parallel computation since it includes sequential bottle-necks in the preconditioning. The α-GMRES linear solver proposed in this work, which is robust and efficient, suitable for both sequential and parallel computations for the linear system above, is the emphasis of the thesis.

The α-GMRES linear solver was designed following a discovery that after a modification of the linear system by means of a simple block diagonal preconditioner and damping factor α, the new linear system could be solved very efficiently by the GMRES scheme. In this procedure the damping factor α plays a key role because when choosing α equal to zero the non-convergence phenomenon appears, however when choosing α equal to a small positive number a considerably fast convergence could be achieved. However, as observed from the distribution of eigenvalues of the new linear system, the difference between the α equal to zero, i.e., without damping factor, and α equal to a small positive number is that all eigenvalues are shifted by a positive value α. In this work the α-GMRES linear solver was constructed by a new iterative procedure, where at each iterative step the modified linear system is solved by the GMRES algorithm.

The storage of the non-zero elements of the matrix in the linear system constitutes the major overall storage of the Newton's method for solving the Navier-Stokes equations. A very efficient data storage method was proposed for storing the non-zero elements of matrix in the parallel computation, which has no data overlap in different processors. After the data decomposition the α-GMRES linear solver was implemented in a parallel manner without any change of the original procedure and without sequential bottle-necks.

A new simplified procedure was proposed for generating the numerically approximate Jacobian matrix, which speeds up the computation and minimises the cell extent in which the discretised *physical* state variables need to be used for generating a matrix element. It also contributes to decreasing the data re-storage in each processor.

The initial guess used in Newton's method was provided by an explicit time dependent approach using the Runge-Kutta method with local time stepping, which is robust when starting the solution from free stream conditions but slow in convergence. From the test flow problems we can see that when the Newton's method is used the quadratic convergence is nearly achieved. In this research the parallelization uses a basis of data decomposition following the data storage method for the Jacobian matrix. Since the parallel implementation does not change the original algorithm, every iterative step has its sequential counterparts on the global domain, and the convergence and the accuracy are maintained compared with the implementation on a single sequential computer.

6.2 Further research

The linear convergence of the outer iterative procedure in the α-GMRES linear solver could somewhat delay the convergence of the linear solver. Further sophistication of the technique is anticipated for an improved design of outer iterative procedure, so that convergence could be accelerated. In the α-GMRES linear solver there are many matrix-vector multiplications to be implemented, therefore any improvement in the calculation of matrix-vector multiplication will contribute to speed up of the computation.

6.3 Expanding the range of application of the scheme

Further expansion of research work using the Newton's method can be carried out in four main directions. (1) Using it to solve other pseudo-3-dimensional flow problems, e.g., axisymmetrical flow and the flow described by reduced Navier-Stokes equations, and 2-dimensional flow problems. Expansion to 3-dimensional flow problems will require a very large memory. The size of the Jacobian matrix increases and the block 13-point diagonal stencil for the 2-dimensional case changes then to a block 25-point diagonal stencil. (2) Using it solve turbulence flow problems. The Baldwin-Lomax model and the Johnson-King model are successful models enabling us to add the turbulence phenomenon in the computation, but they are unsuitable for use in the Newton's method due to their lack of differentiability and the large stencil that they involve. A more promising approach would be to use the κ-ε model. There are some simple strategies that could be used in the numerical calculation. (3) Using it to solve unsteady state flow problems. In each time step Newton's method can be used to solve a non-linear system. (4) Implement it on other parallel computers, such as, the INTEL iPSC/860 super-computer.

REFERENCE

1. Courant, R., Isaacson, E. and Reeves, M. "On the solution of nonlinear hyperbolic differential equations by finite differences", *Comm. Pure and Applied Mathematics*, Vol. 5, pp. 243-255, 1952

2. Steger, J.L. and Warming, R.F. "Flux vector splitting of the inviscid gas-dynamic equations with applications to finite difference methods", *J. Comp. Phys.* Vol. 40, pp. 263-293, 1981

3. Van Leer, B. "Flux-Vector Splitting for the Euler Equations", *Lect. Notes in Phys.* Vol. 170, pp. 507-512, 1982.

4. Godunov, S.K. "A Difference Scheme for Numerical Computation of Discontinuous Solution of Hydrodynamic Equations", Math. Sbornik, 47, pp. 271-306, 1959 (in Russian). Translated US Joint Publ. Res. Service, JPRS 7226, 1969.

5. Osher, S. and Solomon, F. "Upwind Difference Schemes for Hyperbolic Systems of Conservation Laws", *Math. Comp.* Vol. 38, pp. 339-374, 1982.

6. Roe, P.L. " Approximate Riemann Solvers, Parameters Vectors and Difference Schemes", *J. Comp. Phys.* Vol. 43, pp. 357-372, 1981.

7. Qin, N., Scriba, K.W., and Richards, B.E. "Shock-shock, Shock-vortex Interaction and Aerodynamic Heating in Hypersonic Corner Flow", Aeronautical J. Vol. 95, No. 945, pp. 152-160, 1991.

8. Spekreijse, S.P. "Multigrid Solution of the Steady Euler Equations", Ph. D.Thesis, CWI, Amsterdam, 1987.

9. Engquist, B. and Osher, S. "One-sided Difference Approximations for Nonlinear Conservation Laws", *Math. Comp.* Vol. 36, pp. 321-353, 1981.

10. Harten, A. "High Resolution Schemes for hyperbolic Conservation Laws", *J. Comp. Phys.* Vol 49, pp. 357-393, 1983.

11. Harten, A. "On a Class of High Resolution Total-Variation-Stable Finite-Difference Schemes", *SIAM J. Numerical Analysis*, Vol. 21, pp. 1-23, 1984.

12. Osher, S. "Riemann Solvers, the Entropy Condition and Difference Approximations", *SIAM J. Numerical Analysis*, Vol. 21, pp. 217-235, 1984.

13. Van Leer, B. "Towards the Ultimate Conservative Difference Scheme. I. The Quest of Monotonicity", *Lect. Notes in Phys.* Vol. 18, pp. 163-168, 1973. Springer-Verlag, Berlin.

14. Van Leer, B. "Towards the Ultimate Conservative Difference Scheme. II. Monotonicity and Conservation Combined in a Second Order Scheme", *J. Comp. Phys.* Vol. 14, pp. 361-370, 1974.

15. Boris, J.P. and Book, D.L. "Flux Corrected Transport: I. SHASTA, a Fluid Transport Algorithm that Works", *J. Comp. Phys.* Vol. 11, pp. 38-69, 1973.

16. Boris, J.P. and Book, D.L. "Solution of the Continuity Equation by the Method of Flux Corrected Transport", *J. Comp. Phys.* Vol. 16, pp. 85-129, 1976.

17. Venkatakrishnan, V., "Newton Solution of Inviscid and Viscous Problems", *AIAA J.* Vol. 27, (7), pp. 885-891, 1989.

18. Venkatakrishnan, V., "Preconditioned Conjugate Gradient Methods for the Compressible Navier-Stokes Equations", *AIAA J.* Vol. 29, (7), pp. 1092-1100, 1991.

19. Qin, N. and Richards, B.E. "Sparse Quasi-Newton Method for High Resolution Schemes", *Notes in Numerical Fluid Mechanics* Vol. 20, pp. 310-317, 1988.

20. Qin, N. and Richards B. E. "Sparse Quasi-Newton Method for Navier-Stokes Solutions", *Notes in Numerical Fluid Mechanics*, Vol. 29, pp. 474-483, 1990.

21. Schubert, L.K. "Modification of a Quasi-Newton Method for Nonlinear Equations with a Sparse Jacobian", *Math. Comp.* Vol. 24, pp. 27-30, 1970

22. Curtis, A., Powell, M.J.D and Reid, J.K. "On the Estimation of Sparse Jacobian Matrices", *J. Inst. Maths. Applics.* Vol. 13, pp. 117-119, 1974.

23. Hemker, P.W. "Defect Correction and High Order Schemes for Multigrid Solution of the Steady Euler Equations", *Lect. Notes in Math.*, Vol. 1228, pp. 150-165, 1985.

24. Koren, B. "Defect Correction and Multigrid for an Efficient and Accurate Computation of Airfoil Flows", *J. Comp. Phys.* Vol. 77, pp. 183-206, 1988.

25. Whitfield, D.L. and Taylor, L.K. "Discretized Newton-Relaxation Solution of High Resolution Flux--Difference Split Schemes", AIAA Paper 91-1539, 1991.

26. Orkwis, P.D. and McRae, D.S. "A Newton's Method Solver for the Navier-Stokes Equations", AIAA Paper 90-1524, 1990.

27. Orkwis, P.D. and McRae, D.S. "A Newton's Method Solver for the Axisymmetric Navier-Stokes Equations", AIAA Paper 91-1554, 1991.

28. Orkwis, P.D. and McRae, D.S. "Newton's Method Solver for High-Speed Viscous Separated Flowfields", *AIAA J.* Vol. 30, (1), pp. 78-85, 1992.

29. Mallet, M. Periaux J. and Stoufflet, B. "Convergence Acceleration of Finite Element Methods for the Solution of the Euler and Navier-Stokes Equations of Compressible Flows", *Notes in Numerical Fluid Mechanics*, Vol. 20, pp. 199-210, 1988.

30. Wigton, L.B., Yu, N.J., and Young, D.P. "GMRES Acceleration of Compressible Fluid Dynamics Codes", AIAA Paper 85-1494, 1985.

31. Radicati di Brozolo, G. and Robert, Y. "Parallel Conjugate Gradient-like Algorithms for Solving Sparse Nonsymmetric Linear Systems on a Vector Multiprocessor", *Parallel Computing*, Vol. 11, pp. 223-239, 1989.

32. Venkatakrishnan, V., Saltz, J.H., and Mavriplis, D.J. "Parallel Preconditioned Iterative Methods for the Compressible Navier-Stokes Equations", *Lect. Notes in Phys.* Vol. 371, pp. 233-237, 1990.

33. Hwang, K. and Briggs, F.A. "Computer Architecture and Parallel Processing", McGraw-Hill, NY, 1984.

34. Flynn, M.J. "Very High-speed Computing Systems", *Proc. IEEE*, Vol. 54, (12), pp. 1901-1909, 1966.

35. Flynn, M.J. "Some Computer Organizations and Their Effectiveness", *IEEE Trans. on Computers*, Vol. C-21 (9), pp. 948-960, 1972.

36. Xu, X., Qin, N., and Richards, B.E. "α-GMRES: A New Parallelizable Iterative Solver for Large Sparse Non-symmetric Linear System Arising from CFD", *Int. J. Numer. Methods Fluids.* Vol. 15, pp. 615-625, 1992.

37. Xu, X., Qin, N., and Richards, B.E. "Parallelization of Discretized Newton Method for Navier-Stokes Equations", *Proc. on Parallel CFD '92 Conf.*, Rutgers University, New Jersey, May 1992. Elsevier, Amsterdam, 1993.

38. Osher, S. and Chakravarthy, S.R. "Upwind Schemes and Boundary Conditions with Applications to Euler Equations in General Coordinates", *J. Comp. Phys.* Vol. 50, pp. 447-481, 1983.

39. Anderson, W.K., Thomas, J.L. and Van Leer, B. "A Comparison of Finite Volume Flux Vector Splittings for the Euler Equations", *AIAA J.* Vol. 24, (9), pp. 1453-1460, 1986.

40. Mollenstadt, W. "Experimentelle Untersuchngen an Langsangestromten, Gepfeilten Echenkonfigurationen im Hyperschallbereich", Dissertation TU Braunschweig, 1984.

41. Tracy, R.R. "Hypersonic Flow over a Yawed Circular Cone", California Institute of Technology Aeronautical Laboratory Memorandum, No. 69, 1962.

42. Cross, E.J., Jr. and Hankey, W.L. "Investigation of the Leeward Side of a Delta Wing at Hypersonic Speeds", *AIAA J.* Vol. 6, (2), pp. 185-190, 1968.

43. Feldhun, R.F., Winkelman, A.E. and Pasiuk, L. "An Experimental Investigation of the Flowfield Around a Yawed Cone", *AIAA J.* Vol. 9, (6), pp. 1074-1081, 1971.

44. Stetson, K.F. Experimental Results of a Laminar Boundary Layer Separation on a Slender Cone at Angle of Attack at M=14.2", ARL 71-0127, Aerospace Research Laboratories, Wright-Patterson AFB, Ohio, 1971.

45. MacRae, D.S. "A Numerical Study of Supersonic Viscous Cone Flow at High Angle of Attack", AIAA Paper 76-97, 1976.

46. Bluford, G.S., Jr. "Numerical Simulation of the Supersonic and Hypersonic Viscous Flow around Thin Delta Wings", *AIAA J.* Vol. 17, (9), pp. 942-949, 1979.

47. Qin, N. and Richards, B.E. "Numerical Experiments with Hypersonic Flows beneath a Cone-Delta-Wing Combination", AGARD-CP-428, Paper 20, 1987.

48. Newsome, R.W. "A Comparison of Euler and Navier-Stokes Solutions for Supersonic Flow Over a Conical Delta Wing", AIAA Paper 85-0111, 1985.

49. Cebeci, T. and Bradshaw, P. "Physical and Computational Aspects of Convective Heat Transfer", New York: Springer-Verlag, 1984

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

51. Dennis, J.E., Jr. and Schnabel, R.B. "Numerical Methods for Unconstrained Optimization and Nonlinear Equations", Prentice Hall, Englewood Cliffs, N.Y. 1983.

52. Saad, Y and Schultz, M.H. "GMRES: A General Minimal Residual Algorithm for Solving Nonsymmetric Linear System", *SIAM J. Stat. Comp.*, Vol. 7, No. 3, pp. 856-869, 1986.

53. Sonneweld, P., Wesseling, P. and de Zeeuw, P.M. "Multigrid and Conjugate Gradient methods as Convergence Acceleration Techniques", in: D.J. Paddon and M. Holstein, eds., *Multigrid Methods for Integral and Differential Equations* (Claredon Press, Oxford), pp. 117-167, 1985.

54. Fletcher, R. "Conjugate Gradient Methods for Indefinite Systems", *Lect. Notes in Math.*, Vol. 506, pp. 73-89, 1975.

55. Qin, N., Xu, X., and Richards, B.E. "SFDN–α-GMRES and SQN–α-GMRES Methods for Fast High Resolution NS Simulations", *Proc. Conference on Numerical Methods for Fluid Dynamics.*, Reading, April 1992. Oxford University Press 1992.

56. Fletcher, C.A.J. "Computational Techniques for Fluid Dynamics", Vol. 1&2, Springer Series in Computational Techniques, Springer-Verlag.

BIBLIOGRAPHY

Hirsch, C.

"Numerical Computation of Internal and External Flows", Vol. 1: Fundamentals of Numerical Discretization, and Vol. 2: Computational Methods for inviscid and Viscous Flows, John Wiley & sons, 1988.

Appendix 1: Non-dimensionalization

The 3-D N-S equations can be written as following conservative form:

$$\frac{\partial \tilde{\mathbf{Q}}}{\partial \tilde{t}} + \frac{\partial(\tilde{\mathbf{E}}_i - \tilde{\mathbf{E}}_v)}{\partial \tilde{x}_1} + \frac{\partial(\tilde{\mathbf{F}}_i - \tilde{\mathbf{F}}_v)}{\partial \tilde{x}_2} + \frac{\partial(\tilde{\mathbf{G}}_i - \tilde{\mathbf{G}}_v)}{\partial \tilde{x}_3} = \tilde{\mathbf{S}} \tag{A1.1}$$

where

$$\tilde{\mathbf{Q}} = \begin{bmatrix} \tilde{\rho} \\ \tilde{\rho}\tilde{v}_1 \\ \tilde{\rho}\tilde{v}_2 \\ \tilde{\rho}\tilde{v}_3 \\ \tilde{\rho}\tilde{E} \end{bmatrix} \tag{A1.2}$$

$$\tilde{\mathbf{E}}_i = \begin{bmatrix} \tilde{\rho}\tilde{v}_1 \\ \tilde{\rho}\tilde{v}_1^2 + \tilde{p} \\ \tilde{\rho}\tilde{v}_1\tilde{v}_2 \\ \tilde{\rho}\tilde{v}_1\tilde{v}_3 \\ \tilde{\rho}\tilde{v}_1\tilde{H} \end{bmatrix} \qquad \tilde{\mathbf{E}}_v = \begin{bmatrix} 0 \\ \tilde{\tau}_{11} \\ \tilde{\tau}_{12} \\ \tilde{\tau}_{13} \\ \tilde{\tau}_{11}\tilde{v}_1 + \tilde{\tau}_{12}\tilde{v}_2 + \tilde{\tau}_{13}\tilde{v}_3 + \tilde{q}_1 \end{bmatrix} \tag{A1.3a}$$

$$\tilde{\mathbf{F}}_i = \begin{bmatrix} \tilde{\rho}\tilde{v}_2 \\ \tilde{\rho}\tilde{v}_1\tilde{v}_2 \\ \tilde{\rho}\tilde{v}_2^2 + \tilde{p} \\ \tilde{\rho}\tilde{v}_2\tilde{v}_3 \\ \tilde{\rho}\tilde{v}_2\tilde{H} \end{bmatrix} \qquad \tilde{\mathbf{F}}_v = \begin{bmatrix} 0 \\ \tilde{\tau}_{21} \\ \tilde{\tau}_{22} \\ \tilde{\tau}_{23} \\ \tilde{\tau}_{21}\tilde{v}_1 + \tilde{\tau}_{22}\tilde{v}_2 + \tilde{\tau}_{23}\tilde{v}_3 + \tilde{q}_2 \end{bmatrix} \tag{A1.3b}$$

$$\tilde{\mathbf{G}}_i = \begin{bmatrix} \tilde{\rho}\tilde{v}_3 \\ \tilde{\rho}\tilde{v}_1\tilde{v}_3 \\ \tilde{\rho}\tilde{v}_2\tilde{v}_3 \\ \tilde{\rho}\tilde{v}_3^2 + \tilde{p} \\ \tilde{\rho}\tilde{v}_3\tilde{H} \end{bmatrix} \qquad \tilde{\mathbf{G}}_v = \begin{bmatrix} 0 \\ \tilde{\tau}_{31} \\ \tilde{\tau}_{32} \\ \tilde{\tau}_{33} \\ \tilde{\tau}_{31}\tilde{v}_1 + \tilde{\tau}_{32}\tilde{v}_2 + \tilde{\tau}_{33}\tilde{v}_3 + \tilde{q}_3 \end{bmatrix} \tag{A1.3c}$$

and $\tilde{\mathbf{S}}$ is source terms.

In the above formulations the $\tilde{\rho}$, $\tilde{p}$, $\tilde{v}_i$, $\tilde{E}$, $\tilde{H}$ are the density, pressure, velocity, total energy, and total enthalpy respectively, and we have

$$\tilde{q}_i = k\frac{\partial \tilde{T}}{\partial \tilde{x}_i} \tag{A1.4}$$

where k is the thermal conductivity, and the shear stress tensor is

$$\tilde{\tau}_{ij} = 2\tilde{\mu}\left(\frac{1}{2}\left(\frac{\partial \tilde{v}_i}{\partial \tilde{x}_j} + \frac{\partial \tilde{v}_j}{\partial \tilde{x}_i}\right) - \frac{1}{3}\,\mathrm{div}\,(\tilde{v}_k)\,\delta_{ij}\right) \tag{A1.5}$$

where $\tilde{\mu}$ is the viscosity coefficient and $\delta_{ij} = 1$ if i=j or $\delta_{ij} = 0$ if i≠j.

The non-dimensional variables are $\tilde{L}$, $\tilde{\rho}_\infty$, $\tilde{v}_\infty$, $\tilde{\mu}_\infty$, $\tilde{T}_\infty$.

We have

$$x_i = \frac{\tilde{x}_i}{\tilde{L}} \qquad \rho = \frac{\tilde{\rho}}{\tilde{\rho}_\infty} \qquad v_i = \frac{\tilde{v}_i}{\tilde{v}_\infty} \qquad T = \frac{\tilde{T}}{\tilde{T}_\infty} \qquad \mu = \frac{\tilde{\mu}}{\tilde{\mu}_\infty} \tag{A1.6}$$

and

$$t = \frac{\tilde{t}}{\tilde{L}/\tilde{v}_\infty} \qquad p = \frac{\tilde{p}}{\tilde{\rho}_\infty \tilde{v}_\infty^2} \qquad E = \frac{\tilde{E}}{\tilde{v}_\infty^2} \qquad H = \frac{\tilde{H}}{\tilde{v}_\infty^2} \tag{A1.7}$$

By doing the following calculations, i.e., the mass equation multiplied by $\tilde{L}/(\tilde{\rho}_\infty \tilde{v}_\infty)$, the momentum equations multiplied by $\tilde{L}/(\tilde{\rho}_\infty \tilde{v}_\infty^2)$, and the energy equation multiplied by $\tilde{L}/(\tilde{\rho}_\infty \tilde{v}_\infty^3)$, we have

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial (\mathbf{E}_i - \mathbf{E}_v)}{\partial x_1} + \frac{\partial (\mathbf{F}_i - \mathbf{F}_v)}{\partial x_2} + \frac{\partial (\mathbf{G}_i - \mathbf{G}_v)}{\partial x_3} = \mathbf{S} \tag{A1.8}$$

where

$$\mathbf{Q} = \begin{bmatrix} \rho \\ \rho v_1 \\ \rho v_2 \\ \rho v_3 \\ \rho E \end{bmatrix} \tag{A1.9}$$

$$\mathbf{E}_i = \begin{bmatrix} \rho v_1 \\ \rho v_1^2 + p \\ \rho v_1 v_2 \\ \rho v_1 v_3 \\ \rho v_1 H \end{bmatrix} \qquad \mathbf{E}_v = \begin{bmatrix} 0 \\ \tau_{11} \\ \tau_{12} \\ \tau_{13} \\ \tau_{11} v_1 + \tau_{12} v_2 + \tau_{13} v_3 + q_1 \end{bmatrix} \tag{A1.10a}$$

$$\mathbf{F_i} = \begin{bmatrix} \rho v_2 \\ \rho v_1 v_2 \\ \rho v_2^2 + p \\ \rho v_2 v_3 \\ \rho v_2 H \end{bmatrix} \qquad \mathbf{F_v} = \begin{bmatrix} 0 \\ \tau_{21} \\ \tau_{22} \\ \tau_{23} \\ \tau_{12} v_1 + \tau_{22} v_2 + \tau_{23} v_3 + q_2 \end{bmatrix} \tag{A1.10b}$$

$$\mathbf{G_i} = \begin{bmatrix} \rho v_3 \\ \rho v_1 v_3 \\ \rho v_2 v_3 \\ \rho v_3^2 + p \\ \rho v_3 H \end{bmatrix} \qquad \mathbf{G_v} = \begin{bmatrix} 0 \\ \tau_{31} \\ \tau_{32} \\ \tau_{33} \\ \tau_{13} v_1 + \tau_{23} v_2 + \tau_{33} v_3 + q_3 \end{bmatrix} \tag{A1.10c}$$

and S represents the source terms, where

$$\tau_{ij} = \frac{2\mu}{Re_L}\left(\frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) - \frac{1}{3}\,\mathrm{div}\,(v_k)\,\delta_{ij}\right) \tag{A1.11}$$

and

$$Re_L = \frac{\tilde{\rho}_\infty \tilde{v}_\infty \tilde{L}}{\tilde{\mu}_\infty} \tag{A1.12}$$

By using the formulas $p = \rho RT$, $R = C_p - C_v$, $\gamma = \frac{C_p}{C_v}$, $a = \frac{v}{M}$, $a^2 = \gamma\frac{p}{\rho}$, and $Pr = \frac{\mu C_p}{k}$

we have

$$q_i = \frac{\mu}{(\gamma-1)M_\infty^2 Re_L Pr}\frac{\partial T}{\partial x_i} \tag{A1.13}$$

Appendix 2: 2-dimensional grid generation

We will describe the algebraic grid generation method used. In this procedure we require that each line, which is drawn from the solid wall, is represented by a quadratic with orthogonality to the solid wall.

We first define a 1-dimensional stretching function [56],

$$s = p\,\eta + (1-p)\,(1-\tanh^{-1}(q\,(1-\eta))\,/\,\tanh(q)) \qquad \text{(A2.1)}$$

where p and q are the control parameters. This formulation shows that as η proceeds from 0 to 1 with consecutive increments proportional to 1/(N-1) in N steps, the function s will give N values from 0 to 1 with a stretch determined by p and q. Fig.A2.1 illustrates the stretching function s for the N = 11, p = 0.2, and q = 2.0 result.

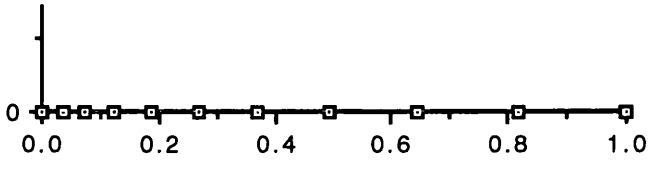


Fig.A2.1 Stretched points distribution

Assume that a 2-dimensional space point can be described as z = (x,y), we will describe a method to construct the curve represented by a quadratic equation. The quadratic is generated from three points z^{i-1}, z_0, and z^{i+1} on a solid wall and a point z_1 outside it, and is drawn from point z_0 to z_1 with the orthogonality at point z_0 with the curve z^{i-1}, z_0, and z^{i+1}, Fig.A2.2. When a stretch function is given we can define all the nodes in the curve.

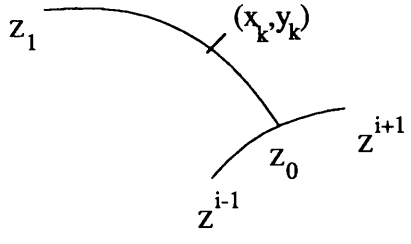


Fig.A2.2 A quadratic in (x,y) plane

A coordinate translation from (x,y) to (ξ,η) is developed as follows:

$$\begin{aligned} \xi &= (x-x_0)\cos\theta + (y-y_0)\sin\theta \\ \eta &= -(x-x_0)\sin\theta + (y-y_0)\cos\theta \end{aligned} \qquad \text{(A2.2)}$$

where θ is the rotational angle. The inverse coordinate translation is then

$$\begin{aligned} x &= x_0 + \xi \cos\theta - \eta \sin\theta \\ y &= y_0 + \xi \sin\theta + \eta \cos\theta \end{aligned} \tag{A2.3}$$

It is obvious that when $x = x_0$, $y = y_0$ we have $\xi = \xi_0 = 0$ and $\eta = \eta_0 = 0$. Now we can define the coordinate translation by defining θ. Without losing generality we can impose that $\eta_1 = 0$. The curves in the ξ,η plane are then illustrated in Fig.A2.3. Thus we obtain

$$\operatorname{tg}\theta = (y_1 - y_0) / (x_1 - x_0) \tag{A2.4}$$

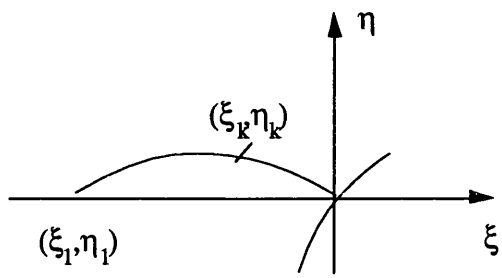


Fig.A2.3 A quadratic in (ξ,η) plane

If $x_0 = x_1$, when $y_1 > y_0$ we obtain $\theta = \pi / 2$, and $\xi_1 = y_1 - y_0$, when $y_1 < y_0$ we obtain $\theta = -\pi / 2$, and $\xi_1 = y_0 - y_1$.

If $x_0 \neq x_1$, we obtain $\theta = \operatorname{tg}^{-1}((y_1-y_0) / (x_1-x_0))$, and

$$\begin{aligned} \xi_1 &= (x_1-x_0)\cos\left(\tan^{-1}\frac{y_1-y_0}{x_1-x_0}\right) + (y_1-y_0)\sin\left(\tan^{-1}\frac{y_1-y_0}{x_1-x_0}\right) \\ &= (x_1-x_0) / \sqrt{1+\left(\frac{y_1-y_0}{x_1-x_0}\right)^2} + \frac{(y_1-y_0)^2}{x_1-x_0} / \sqrt{1+\left(\frac{y_1-y_0}{x_1-x_0}\right)^2} \end{aligned} \tag{A2.5}$$

Therefore the points z^{i-1}, z^{i+1} in (ξ,η) coordinates are

$$\begin{cases} \xi^{i-1} = (x^{i-1}-x_0)\cos\theta + (y^{i-1}-y_0)\sin\theta \\ \eta^{i-1} = -(x^{i-1}-x_0)\sin\theta + (y^{i-1}-y_0)\cos\theta \end{cases} \tag{A2.6}$$

$$\begin{cases} \xi^{i+1} = (x^{i+1}-x_0)\cos\theta + (y^{i+1}-y_0)\sin\theta \\ \eta^{i+1} = -(x^{i+1}-x_0)\sin\theta + (y^{i+1}-y_0)\cos\theta \end{cases} \tag{A2.7}$$

Assuming that η_{ξ_0} is the orthonormal direction of curve z^{i-1}, z_0, and z^{i+1} at z_0, we obtain

$$\eta_{\xi_0} = \frac{1}{2}\left(-\frac{\xi_0 - \xi^{i-1}}{\eta_0 - \eta^{i-1}} - \frac{\xi^{i+1} - \xi_0}{\eta^{i+1} - \eta_0}\right) = -\frac{1}{2}\left(\frac{\xi^{i-1}}{\eta^{i-1}} + \frac{\xi^{i+1}}{\eta^{i+1}}\right) \quad \text{(A2.8)}$$

Because we require the curve to be represented by a quadratic, then

$$\eta = A\,\xi^2 + B\,\xi + C\,, \quad \text{(A2.9)}$$

and $\eta_\xi = 2A\,\xi + B$.

Since $\eta_0 = 0$, we obtain $C = 0$; since $\eta_1 = 0$, we obtain $A\,\xi_1 + B = 0$; and since $\xi_0 = 0$, we obtain $\eta_{\xi_0} = 2A\,\xi_0 + B = B$. Thus we get the equation of the curve as follows:

$$\eta = \xi\,\eta_{\xi_0}\left(1 - \frac{\xi}{\xi_1}\right) \quad \text{(A2.10)}$$

Assuming that the stretching function has been given, and $0<s_n<1$, $n = 2, \ldots, N-1$, we can use a circle to cut the above curve, in which the semidiameter equals $\xi_1 \times s_n$ and the equation is

$$\xi_n^2 + \eta_n^2 = \xi_1^2\, s_n^2\,. \quad \text{(A2.11)}$$

Substituting the (A2.11) into (A2.10) we get

$$\xi_n^2 + \xi_n^2\left[\eta_{\xi_0}\left(1 - \xi_n/\xi_1\right)\right]^2 = \xi_1^2\, s_n^2\,, \quad \text{(A2.12)}$$

resulting in

$$\xi_n = \frac{\xi_1\, s_n}{\sqrt{1 + \left[\eta_{\xi_0}\left(1 - \xi_n/\xi_1\right)\right]^2}} \approx \frac{\xi_1\, s_n}{\sqrt{1 + \left[\eta_{\xi_0}\left(1 - s_n\right)\right]^2}}\,. \quad \text{(A2.13)}$$

Using (A2.10) we also obtain η_n, $n = 2, \ldots, N-1$.

Using (A2.3) we obtain x_n, y_n for $n = 2, \ldots, N-1$. Together with the $x_1 = x_0$, $y_1 = y_0$ and $x_N = x^1$, $y_N = y^1$ we get all N coordinate values of the point.

Generally speaking for generating a 2-dimensional structured grid we first should define all the nodes on the solid wall then construct the other pertinent boundaries and finally connect each pair of nodes following the above constraints. For a simple case of four boundaries described by equations of curves such that the arc lengths of the curves can be stretched to give all the nodes on boundaries according to particular requirements, then we

can use the above method to generate the grid. The stretch parameters for the interior curves can be given according to the particular proportions of the stretch parameters on the two sides of the boundaries. The grid generation method can be also used for a block domain, i.e., different blocks can use different stretched grids according to requirements.

Fig.A2.4-6 show three 2-dimensional grid generation cases.

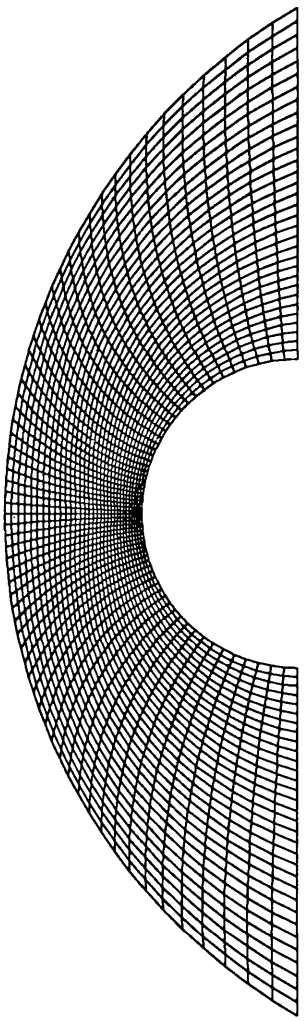


Fig.A2.4 2 block grid

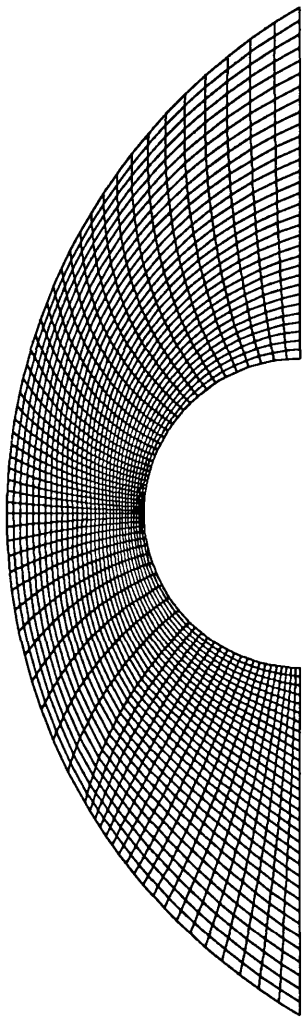


Fig.A2.5 3 block grid

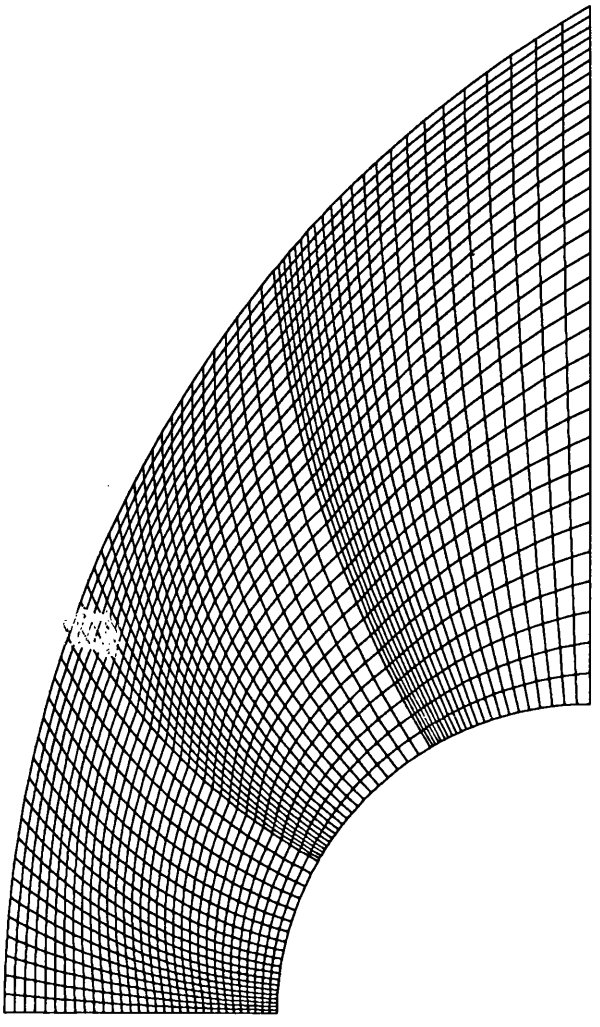


Fig.A2.6 3 block grid for a quarter circlar body

GLASGOW
UNIVERSITY
LIBRARY