ROLFES, HERMANN

# THE NUMERICAL SOLUTION OF TURBULENT FLOW IN THREE-DIMENSIONAL CURVILINEAR CO-ORDINATES

MEng UP 1992

# THE NUMERICAL SOLUTION OF TURBULENT FLOW IN THREE - DIMENSIONAL CURVILINEAR CO - ORDINATES

by

**Hermann Rolfes**

Dissertation presented in partial fulfilment
of the requirements for the degree of

**MASTER OF MECHANICAL ENGINEERING**

in the Department of Mechanical Engineering
University of Pretoria
Pretoria

August 1992

# THE NUMERICAL SOLUTION OF TURBULENT FLOW IN THREE – DIMENSIONAL CURVILINEAR CO – ORDINATES

by

**Hermann Rolfes**

Promoter: Prof J A Visser
Department: Mechanical Engineering
Degree: Master of Mechanical Engineering

## ABSTRACT

Many fluid flow applications that exist in engineering practice can be solved by means of numerical techniques. Most of these problems require complete three–dimensional modelling of flow in complex curvilinear geometries. This motivated the development of a numerical model for the solution of three–dimensional turbulent flow based on a general curvilinear co–ordinate system.

Three–dimensional turbulent flow is described by six highly non–linear partial differential equations. These include the three momentum equations, the continuity equation and the two equations of the $k - \epsilon$ turbulence model. In order to apply the conservation principles in the above equations to general curvilinear co–ordinates, transformation relations are used in formulating the equations in terms of general curvilinear form.

A finite volume numerical approach is used to discretize the relevant equations into a linear form. The equations are then solved simultaneously by an iterative process. A segregated approach based on the SIMPLE algorithm is used for this purpose whereby pressures and velocities are calculated separately. Due to the application of the segregated approach, decoupling between pressures and velocities occurs. A specific interpolation scheme is implemented whereby strong pressure–velocity coupling is ensured. Turbulence effects are included by calculating an additional turbulent viscosity, which has the effect of increasing the effective fluid viscosity. The computer program (3DFLO) is written in Fortran 77 and executed on an IBM f550 computer.

After each stage of the development process, various test cases were solved to verify the accuracy of the code. It is shown that the numerical results compare favourably to analytical, experimental and previous numerical results. The code was then applied to the modelling of three–dimensional atmospheric boundary layer flow over and around arbitrary shaped buildings. The use of non–orthogonal boundary fitted grids enabled the exact conformation of sharp ridge geometry and pitched roof inclines. The numerical predictions are in good agreement with full scale measurements and prove to be superior to previous numerical predictions. This can be mainly attributed to an improved representation of physical flow boundaries and to complete three–dimensional modelling.

# DIE NUMERIESE OPLOSSING VAN TURBULENTE VLOEI IN DRIE – DIMENSIONELE KROMLYNIGE KOÖRDINATE

deur

**Hermann Rolfes**

Promotor: Prof J A Visser
Departement: Meganiese Ingenieurswese
Graad: Magister in Meganiese Ingenieurswese

## UITTREKSEL

Vele vloeiprobleme kom in die ingenieurspraktyk voor, wat moeilik sonder die hulp van numeriese modelle opgelos kan word. Hierdie probleme gaan meestal gepaard met turbulensie en bestaan in die werklikheid gewoonlik in komplekse drie–dimensionele geometrieë. Dit het aanleiding gegee tot die ontwikkeling van 'n numeriese model vir die oplossing van drie–dimensionele turbulente vloei in nie–uniforme gebiede.

Drie–dimensionele laminêre vloei word beskryf deur vier nie–lineêre partiële differensiaalvergelykings naamlik die drie momentum vergelykings en die kontinuiteitsvergelyking. Wanneer vloei egter turbulent is, word verder ook van 'n turbulensie sluitingsmodel gebruik gemaak. In hierdie studie word die k–$\epsilon$ model gebruik wat uit twee verdere differentiaal–vergelykings bestaan. Om te verseker dat die behoudsbeginsels waarop die bogenoemde vergelykings gebasseer is steeds geld wanneer in nie–uniforme gebiede gewerk word, word die vergelykings getransformeer relatief tot 'n algemene kromlynige koördinaatstelsel.

Die numeriese tegniek wat vir die oplossing van die vergelykings gebruik word is 'n eindige volume metode. Hiervolgens word elkeen van die vergelykings gediskretiseer waarna dit op 'n iteratiewe basis opgelos word. Die oplosalgoritme, wat op die SIMPLE metode gebaseer is, behels die afsonderlike oplossing van snelhede en drukke wat lei tot swak koppeling tussen dié veranderlikes. Derhalwe word van 'n spesiale interpolasie metode gebruik gemaak ten einde voldoende koppeling te bewerkstellig. Die effek van turbulensie word in berekening gebring deur 'n addisionele turbulente viskositeit te bereken wat gevolglik die effektiewe vloeier viskositeit verhoog. Wanneer vloei in komplekse geometrieë opgelos word, word van grenspassende roosters gebruik gemaak sodat realistiese vloei-gebiede akkuraat nageboots kan word. Die rekenaar program (3DFLO) is ontwikkel in Fortran 77 en word op 'n IBM f550 rekenaar uitgevoer.

Gedurende die ontwikkeling van die model, is dit na elke ontwikkelingsfase geverifieer deur standaard toetsgevalle, waarvoor daar korrekte oplossings beskikbaar is, op te los. In alle gevalle toon die huidige resultate goeie ooreenkoms met analitiese, eksperimentele en aanvaarde numeriese resultate. Die model is voorts toegepas op die modellering van atmosferiese grenslaag vloei oor en rondom geboue van arbitrêre vorm. Volledige drie–dimensionele simulasie tesame met die akkurate modellering van dak geometrieë het tot gevolg dat die huidige resultate beter korrelleer met volskaal metings as enige van die vorige voorspellings.

# ACKNOWLEDGEMENTS

I am deeply endebted to the following people for their assistance during the course of this study:

Prof J A Visser for his guidance and continued assistance over the past two years. He not only contributed heavily on technical level but also strongly influenced my personal development.

Charles Crosby for his valued advice concerning numerical methods and his strong interest in my topic of research.

Louis Le Grange for getting me started on transformation to curvilinear co–ordinates and for the friendly manner in which he was always prepared to help me.

My parents for their continued support and motivation which were always sincerely appreciated,

and particularly to Liesel for always standing by me with love and encouragement.

Last of all, my Almighty Creator without Whom this would not have been possible.

# TABLE OF CONTENTS

## APPENDICES

# CHAPTER 1 INTRODUCTION

## 1.1 The problem considered

The rapid development of computer technology during the last decade, have resulted in growing emphasis being placed on the use of numerical methods for solving complex engineering problems. Such applications present a cost effective alternative to experimental testing by evaluating different design alternatives during the development stage. The numerical solution of fluid flow problems further contribute to improve the understanding of complicated flow phenomena.

In general, practical flows in engineering occur in non–uniformly shaped regions. The inability of classical co–ordinate systems to accurately model flow in such regions, motivated the search for methods to solve fluid flow by using a general curvilinear co–ordinate system. This enables the exact conformation of physical flow boundaries and removes inaccuracies due to boundary approximating assumptions.

Many applications exist where the ability of a computer code to model fluid motion in randomly shaped regions is especially useful. One of these is the simulation of wind flow over obscurely shaped buildings. With sufficient knowledge of the wind flow patterns in a certain area, such a computer code can provide detailed information regarding the velocity field and pressure distribution surrounding any proposed building at that location. This enables the study of wind loads on building structures and reflects the expected wind micro climate associated with those structures.

Like most practical flow situations, the flow around buildings is turbulent. The general fluid flow equations used to calculate laminar flow are therefore no longer valid and an additional closure model is required to include the effect of turbulence. The aim of the study is thus to develop a numerical model which is able to simulate turbulent flow over arbitrary geometries – more specific, the modelling of wind flow over obscurely shaped buildings.

## 1.2 Review of related literature

### 1.2.1 General overview

For many centuries, engineers have struggled with the solution of fluid flow problems. Until recently most problems were solved using either analytical or empirical methods based on experimental measurements. The limitations of these methods inspired the search for improved solution procedures. In the early 20th century, a few pioneers started using numerical methods to solve fluid flow problems [1]. In those days calculations had to be done by hand which required enormous amounts of time and effort. The advent of the digital computer made it possible to obtain numerical solutions with much greater ease. It resulted in increased interest in computational techniques applicable to fluid dynamic problems.

According to the literature, the actual beginning of computational fluid dynamics can be attributed to Richardson [2] in 1910, who presented a point iterative scheme for the solution of the Laplace equation. For the first time, problems requiring solution by relaxation were distinguished from those which required marching schemes. In 1940, Southwell [3] introduced a relaxation scheme which made use of point residuals in the calculation of the dependent flow variables. The method was applied to solving incompressible, viscous flow over a cylinder. During the same time, a great deal of research was done on the evaluation of the stability of numerical methods. This resulted in the Von Neumann method [4] which is still the most commonly used method for determining the stability of numerical solution procedures today [1].

In 1972, Patankar [5] developed the so–called SIMPLE algorithm for the calculation of pressure and velocity distributions describing an entire flow field. A number of similar methods were consequently developed and it will be shown later (Section 1.2.5), that this family of solution algorithms is still the most popular for solving elliptic fluid flow problems today.

The aspect of fluid flow turbulence also attracted the attention of a number of engineers and researchers [6]. In an attempt to describe the phenomenon of turbulence in fluid motion, various mathematical models have been developed. These closure models are included in a numerical model when turbulent flows are calculated. The most popular group of turbulence models to date, are the turbulent viscosity models which are based on the calculation of an effective fluid viscosity to include turbulent effects. A discussion of these and other turbulence models will be presented in Section 1.2.3

With the increasing complexity of modern engineering technology, it has become necessary to accurately model fluid flow in randomly shaped regions. In an attempt to address this issue, a so–called general curvilinear co–ordinate system is used to replace classical co–ordinate systems. Current research is aimed at employing general curvilinear boundary fitted grids to model almost any three–dimensional fluid flow problem.

### 1.2.2 General curvilinear co–ordinates

The development of new solution methodologies is one of the primary pacing items in computational fluid dynamics today. With the current rate of progress in this discipline, as well as with grid generation techniques and the enhancements in computing ability, it is now practical to simulate complicated fluid dynamic phenomena associated with realistic geometries [7]. Classical co–ordinate systems such as cartesian or cylindrical co–ordinates are limited in the extent to which randomly shaped spaces can be modelled. When co–ordinate lines do not conform the boundaries of the physical region, difficulties occur at these boundaries. It requires the tiresome interpolation of the values for the dependent variables at points closest to the boundaries. The process is not only time–consuming but decreases the accuracy of the solution.

The restrictions inherent to classical co–ordinate systems motivated the search for methods which enable the solution of fluid flow problems in realistic geometries. Wenquan et al. [8], developed a numerical model for the calculation of flow along arbitrarily twisted stream surfaces. The model makes use of a stream function–vorticity method in calculating two–dimensional velocity components. It includes a general curved surface fitting scheme to represent curved surfaces in turbo–machines. Cunsolo et al. [9] also used a stream function–vorticity method to model fluid flow in general non–orthogonal grid reference systems. The extension of these stream function–vorticity methods to three dimensions are, however, not straight forward, which leaves pressure–velocity methods to be considered.

Greyvenstein [10] applied a finite difference pressure–velocity method for the solution of elliptic flow by using orthogonal curvilinear co–ordinates. The use of such a system leads to the simplification of the equations of motion describing the problem and also avoids discretization errors coupled with non–orthogonal methods. According to Raithby et al. [11], boundary conditions can be applied more simply and with greater accuracy when the co–ordinate system is orthogonal. On the other hand, non–orthogonal grids can be generated with much greater ease, especially when randomly curved three–dimensional spaces are modelled. The aim is therefore to make use of non–orthogonal co–ordinates and to find ways and means of circumventing the above mentioned difficulties.

Ramachandra and Spalding [12] presented a non–orthogonal finite–difference formulation for three–dimensional duct flows with arbitrary cross sections. Swanson [13] noted, that while Ramachandra et al. [12] transformed cross–stream dimensions from polar co–ordinates, cross section planes remained orthogonal in the streamwise direction. This limited the model to duct flow applications where there is only one direction of main transverse flow. Swanson [13] extended the model to solve incompressible flow in pump impellers and diffusers and obtained good agreement between the elliptic– and an inviscid solution.

A number of other workers also identified the need for a numerical model that is generally applicable to most practical fluid flow problems [7,14,15]. Rhie and Chow [15] presented a finite volume numerical method for the solution of two–dimensional incompressible, steady Navier Stokes equations, in general curvilinear co–ordinates. The method is applied to two–dimensional turbulent flows over airfoils, with and without trailing edge separation. Numerical methods introduced by Thompson [16] were used in generating boundary fitted grids and the results obtained compared favourably with available experimental data. Thompson further used differential geometry to define the expressions required for transforming the partial differential equations into general co–ordinates.

Based on the same curvilinear principles, Le Grange [17] developed a computer code for the simulation of polymer melt flow in a mould during the injection moulding process. Although this model was extended to three dimensions, it was limited to the simulation of laminar flows. The model included the solution of the energy equation which presented the ability of predicting viscous heating effects. The results obtained were in good agreement with predictions by commercial codes and available measurements. According to Le Grange [17], the method is applicable to non–orthogonal grid reference systems but the accuracy decreases when grids become severely non–orthogonal. No quantitative limits for non–orthogonality were, however, specified.

### 1.2.3 Fluid flow turbulence

Very few practical flows can be classified as being laminar. Especially in fluids with low viscosities, such as atmospheric air motion over building structures, fluid flow turbulence is prevalent. A numerical model that is applied to simulate such flow situations should provide for the effect of turbulence. This can be done by the inclusion of a mathematical turbulence model next to the partial differential equations describing laminar fluid flow.

Turbulence can be defined as a three–dimensional time–dependent motion in which vortex stretching causes velocity fluctuations between a minimum determined by viscous forces, and a maximum determined by the flow boundary conditions [18]. It is the usual state of fluid motion except at low Reynolds numbers, and is associated with higher values of friction drag and steeper pressure drops than laminar flow. The diffusion rate of a scalar quantity is usually greater in turbulent flow which gives the impression of an increased fluid viscosity [19].

Despite this knowledge of turbulence, it has been recognized for more than 50 years that the understanding of turbulent flows is actually very incomplete. A quotation attributed to Sir Horace Lamb in 1932 [1] might be appropriate: "I am an old man now, and when I die and go to Heaven there are two matters on which I hope for enlightenment. One is quantum electrodynamics and the other is the turbulent motion in fluids. And about the former I am rather optimistic." Today, 60 years later, many scientists and researchers probably still feel the same way.

It can, however, be said that some progress has been made in trying to understand and predict turbulent fluid motion. It has been widely proposed that turbulent flow could be regarded as having an enhanced viscosity, a turbulent (or eddy) viscosity which presents similar effects as noted previously. Boussinesq [20], suggested that this value be constant and that the equations of mean motion become identical to those applicable for laminar flow. The value for this turbulent viscosity had to be determined from comparisons with experimental measurements. This approach did not prove adequate for wall bounded flows as the turbulent viscosity must vary with position, from a large value in the mainstream to be zero at the walls [19].

The simplest turbulence model to account for the variability of the turbulent mixing length with the use of only one empirical constant, is Prantl's mixing length model [1]. The basic idea in this model is that a fluid element, displaced perpendicularly to the main flow direction from its original position, would retain its original streamwise velocity. The major physical assumption underlying this hypothesis is that streamwise pressure forces and viscous stresses are unimportant, which may be shown justifiable for three–dimensional eddies that are flat in the sense that their streamwise dimensions are much greater than their cross–stream dimensions [19]. Prantl's algebraic formula treats the turbulent viscosity as a scalar and gives qualitatively correct trends, particularly near the wall. There is increasing experimental evidence, however, that in the outer layer, the turbulent viscosity should be treated as a tensor (ie. dependent on the direction of strain) in order to provide the best agreement with measurements. For flows in corners or in other geometries where a single "transverse" direction is not clearly defined, Prantl's formula must be further modified [18].

A large number of more complicated turbulence models have been proposed since Prantl's initial work. In 1972, Launder and Spalding [6] introduced the so–called k–$\epsilon$ turbulence model which applies to fully turbulent flows. The model entails the solution of two additional partial differential equations for the turbulence quantities k and $\epsilon$, which are used to calculate the turbulent viscosity. Several authors sought to devise turbulence model equations which are valid throughout the laminar, semi–laminar and fully turbulent regions. Launder and Spalding [21] recommend a separate pair of equations applicable to low Reynolds number flows. The model differs from the original method in that viscosity now influences levels of k and $\epsilon$ in two additional ways. Firstly, laminar diffusive transport becomes of increasing importance near the walls and secondly, extra destruction terms are included which become significant in transitional areas. For a complete discussion of this model, the reader is referred to reference [21].

Gosman and Ideriah [22], explains the handling of the k–$\epsilon$ model at various boundary conditions. In the near vicinity of fixed wall boundaries the use of wall functions to determine the value of $\epsilon$ is proposed. These values reach a maximum at the walls where the turbulence dissipation rate is very large. Concerning the kinetic energy near solid walls, the direct application of the effect of the wall shear force on the kinetic energy transport equation is suggested. At fixed walls, an additional shear force term is added to the streamwise momentum equation to represent the effect of flow becoming laminar at these locations. Despite the dependence of the method on empirical data, many other authors [7,10,14,15,18,23–28] successfully applied the techniques described by Launder et al. [21] and Gosman et al. [22] and presented results that are in good agreement with experimental measurements.

Many other turbulence models have been proposed such as algebraic stress models, Reynolds stress models, large eddy simulations and the full Navier Stokes equations [29]. Although the latter completely describes turbulent fluid motion, the small scale of turbulence could require approximately $10^5$ grid points for the simulation of 1 $cm^3$ of typical turbulent flow [1]. Of all the above mentioned methods only Reynolds stress models possibly offer a better approximation than the k–$\epsilon$ method.

Unlike the other methods mentioned, Reynolds stress models are not restricted by the Boussinesq approximation relating turbulent stresses to rates of mean strain (effective viscosity), but makes use of a great number of model partial differential equations to directly determine turbulent shear stresses independent of empirical constants. It would seem therefore that these Reynolds stress models ought to have the best chance of emerging as "ultimate" turbulence models if success is to be achieved at all through time–averaged Navier–Stokes equations. Nevertheless, these models must still utilize approximations and assumptions in modelling terms which presently cannot be measured. These models are perhaps still in their infancy and it may be some time yet before they have been refined and tested to the point that they become commonplace in engineering practice [1].

### 1.2.4 Flow over arbitrary shaped buildings

The satisfactory environmental performance of a building can depend critically upon the prediction and control of its wind microclimate. The design of an energy efficient building, for instance, requires a predictive understanding of its microclimate and particularly of the interaction between the wind, the building and its surroundings. Equally, comfort conditions in the spaces around and between buildings depend upon the wind speeds within them and their usefulness is determined in part by the adequate control of wind. In both cases there is a need to determine during design, the patterns of wind flow which will be generated by the proposed forms and layouts of buildings.

Wind flow around buildings has been investigated with scale models and wind tunnels for some time [30]. This process is expensive and time–consuming. Many workers have done full scale experimental measurements on existing buildings of various shapes in order to predict wind flows over similar buildings under design [31–34]. Instruments for such experiments are very sensitive and costly and the setup of these devices take many hours. It would be ideal to model the wind effects on buildings numerically rather than have to contend with the problems associated with most experiments.

The question naturally arises whether a computer simulation of wind flow around a building can accurately reproduce the information obtained from wind tunnel and full scale measurements. Acceptable agreement between numerical predicted and full scale measurements for wind flow over a film clad greenhouse was obtained by Meyer et al. [25], who made use of a two–dimensional finite difference computer model. Many other authors also presented numerical predictions of sufficient accuracy by using two–dimensional computer codes [26,35–38]. Crosby [18] applied a cartesian two–dimensional numerical model to simulate airflow through and around permeable windbreaks. Obstructions in the flow field were treated by prescribing very small porosities over the solid region and the resulting velocity and pressure distributions around house shaped buildings presented good agreement with experiments. These models are, however, limited in the extent to which three–dimensional effects can be included in the predictions.

The need to apply three–dimensional modelling to flow over buildings was identified by Hanson et al. [39] and by Paterson et al. [40]. Subsequently other workers also recognized the advantages of complete three–dimensional simulation [27,28,41,42]. In all the cases acceptable numerical results were presented. The numerical models used by these authors were, however, all limited to cartesian co–ordinates and square buildings. Similar to Häggkvist et al. [42], inclined roofs were represented by stepping the cartesian grid as shown below in Figure 1.1.

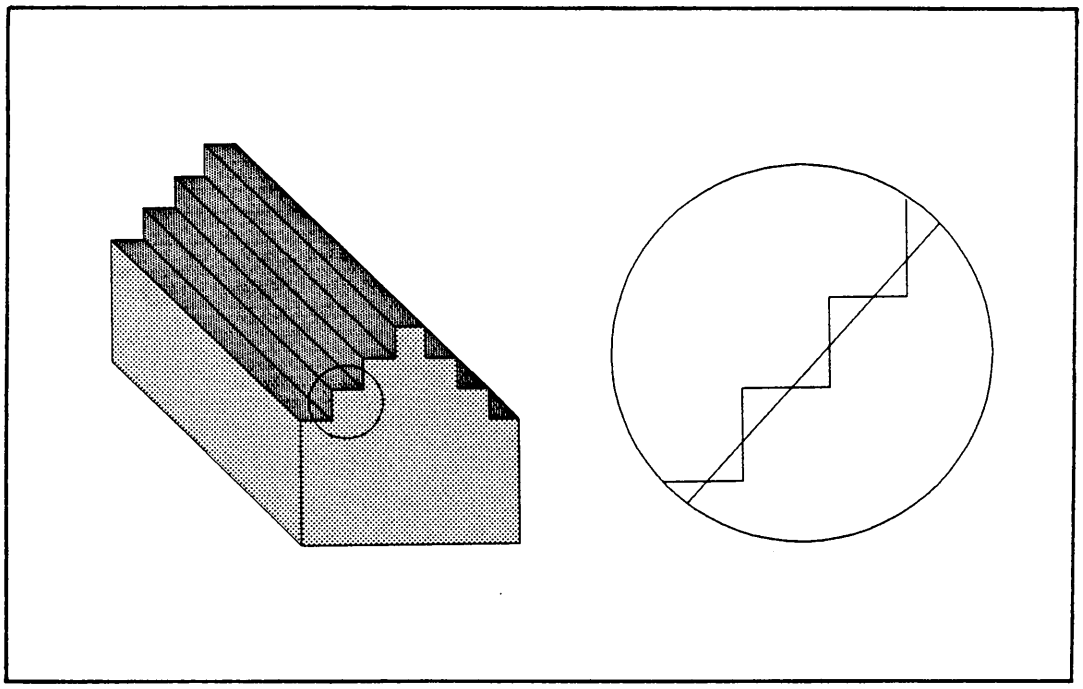

**Figure 1.1** Stepped representation of an inclined roof

Inaccuracies occur at these surfaces as a result of the uneven, stepped incline. Ironically this is the most sensitive and important part of the numerical solution. These difficulties can be removed by using boundary conforming grids to represent the exact regions.

Although acceptable correlation between numerical predictions and full scale measurements for single span pitched roof buildings were presented by a number of authors [18,42], no evidence could be found of numerical simulations on multi–span buildings. This could mean that such a problem requires either three–dimensional modelling or better representation of roof inclines, or both. In fact, no evidence could be found in the literature of three–dimensional numerical modelling, applied to body fitted grids in predicting atmospheric air motion over buildings.

### 1.2.5 Numerical solution algorithms

In the quest to solve the highly non–linear partial differential equations describing turbulent fluid flow, many solution procedures have been developed. Probably the most popular of them all is the SIMPLE algorithm (Semi–Implicit Method for Pressure Linked Equations) introduced by Patankar [5]. It entails the segregated solution of the pressures and velocities in an iterative manner. The values are then adjusted by calculating pressure and velocity corrections from the continuity equation. In order to improve the rate of convergence, the algorithm was modified and lead to the SIMPLER (Revised) procedure.

In 1984 Van Doornmaal and Raithby [43] introduced several modifications to the SIMPLE method that both simplified its implementation and reduced solution costs. The SIMPLEC algorithm removed the need for under–relaxation of the pressure corrections and arguably followed a more consistent approach in the manipulation of the pressure correction equation. In a comparison between the above mentioned three algorithms for two test cases, it was found that the new method reduced the computational effort of the original SIMPLE method by up to 50% using the same grid, differencing scheme and convergence parameters.

A new general segregated approach for solving a linear set of equations for pressure and velocity, which had better characteristics than previous segregated methods, were presented by Van Doornmaal et al. [44]. This method permits previous segregated methods to be derived from the same general equations and provides a means of qualitatively assessing the approximations introduced. The SIMPLEX method addressed the issue of degradation of segregated methods with grid refinement. It was concluded that the cost of the additional computational effort required by the SIMPLEX method, may be compensated for by its rate of convergence, that does not degrade with grid refinement.

In 1990, Thiart [45] presented a new difference method which removes the need for staggered grids in fluid dynamic computations. The method prevents pressure checkerboarding, as described by Patankar [5], through a differencing scheme that incorporates the influence of pressure on velocity gradients. The most important advantages of the SIMPLE based SIMPLEN method (for Non–staggered grids) are ease of programming and the relative ease with which boundary conditions can be applied. Following this development, Thiart published an improved differencing scheme for his original algorithm later the same year [46]. The method was extended by also upwinding cross–stream fluxes in addition to the upwinding of the source terms, and showed that the method is superior in accuracy compared to the previous method.

Other researchers also joined in the search for algorithms applicable to finite volume non–staggered grids. Rhie and Chow [15] presented a finite difference formulation, in which spurious pressure modes are detected and suppressed by a specific interpolation scheme. The scheme ensures strong pressure–velocity coupling by including pressure at consecutive points into the interpolation of cell wall velocities, which are needed to solve the pressure correction equation. Differencing of the local pressure gradient terms at these walls is then done over only one space interval instead of two. It was proven for some test cases that this procedure showed better convergence behaviour than the original SIMPLE method.

Other solution procedures such as the PISO (Pressure–Implicit Split–Operator) algorithm do not yet enjoy the same popularity as the SIMPLE family of methods. Benodekar et al. [23] used this algorithm in the prediction of turbulent flow over surface mounted ribs. The algorithm uses a two–stage predictor–corrector sequence which satisfies continuity and linearized momentum equations at each cycle more closely than hitherto. It resulted in an improved computational efficiency of three to four fold.

## 1.3 The need for this study

From the literature survey it is evident that computer models are fast becoming commonplace in engineering practice. Numerical models are becoming particularly popular for the solution of fluid flow problems that occur in nature and industry. These methods offer suitable alternatives to time–consuming and expensive experimental methods. Many of the flow problems encountered in practice, involve complex geometrical configurations and most of them require three–dimensional modelling to enable realistic simulation. A need is therefore identified to use three–dimensional boundary fitted grids and curvilinear co–ordinates when flow in arbitrary shaped regions is modelled.

Although many commercial codes exist that can simulate flow in randomly curved regions, they are not only expensive but often limited in applicability. Due to the complexity of the equations governing fluid motion, some simplifying assumptions are usually made during the development of these codes. Whether they be restricted to the solution of laminar flow, steady state or incompressible flow, the fact remains that they are seldom applicable to most practical engineering fluid flow problems. Without access to the source code, it is also not possible to modify the code for special applications. This provided the need to locally develop a general fluid flow code, that is applicable to laminar and turbulent fluid motion in realistic geometries.

One application for which numerical methods are becoming increasingly popular is the study of wind flow around buildings. Many workers used two–dimensional approximations, which were not always adequate, while other three–dimensional simulations were hampered by the geometrical restrictions inherent to fixed cartesian and orthogonal curvilinear co–ordinate systems. A general three–dimensional fluid flow code based on curvilinear co–ordinates is therefore required to include complete three–dimensional effects in the simulation and to enable exact representation of building structures thereby avoiding previous geometrical approximations of pitched roof ridges.

The code is further needed in order to simulate wind flow around multi–span pitched roof buildings – something which has evaded researchers in the past. This could lead to an interesting and unique contribution to the current knowledge of wind loads on and flow around arbitrary shaped buildings.

## 1.4 The outline of the study

The purpose of this study is to develop a three–dimensional computer code for the solution of laminar and turbulent flow in realistic geometries. The numerical model is based on a general three–dimensional curvilinear co–ordinate system and allows the use of non–orthogonal boundary conforming grids representing arbitrary curved regions.

In Chapter 2 a theoretical investigation into the mathematical models describing three–dimensional turbulent flow is provided. In Appendix A the derivation of a momentum equation from first principles is presented from which a General Transport Equation is formulated. The transformation relations utilized in transforming this equation relative to general co–ordinates is also discussed in this chapter, followed by an investigation of the k–$\epsilon$ turbulence model.

The above mentioned equations can, however, not be solved analytically. This presents the need to solve them numerically. The methods used and the assumptions made in linearizing these equations for numerical solution are discussed in Chapter 3. It also deals with the algorithms utilized in obtaining the numerical solution and attends specifically to the application of the different boundary conditions.

In Chapter 4 the results obtained from the computer simulations are compared to analytical and experimental data. The model is extensively verified for laminar as well as turbulent flows whereafter it is used to calculate flow over real buildings. The results obtained are compared with published full scale measurements of flow fields around buildings. The information is used to calculate dynamic pressure coefficients required to determine the effect of wind loads on buildings.

Chapter 5 concludes the report by presenting a summary of the thesis, a list of the main contributions of the study as well as the important conclusions drawn from the study. Finally a list is given of some related areas for further research which were identified during the course of this study.

# CHAPTER 2 THEORETICAL INVESTIGATION

## 2.1 Preamble

Laminar fluid motion can in general be described by a number of partial differential equations. These include a momentum equation for each of the velocity components and the continuity equation. By solving these partial differential equations, velocities and pressures can be calculated at any point within the flow region. For turbulent flow, a closure model is required to include the effect of turbulence in the fluid. In this study, a turbulent viscosity model is used which entails the solution of two additional partial differential equations.

The above mentioned equations derived for classical co–ordinate systems are, however, not applicable to the solution of flow in arbitrary curved regions. To solve flow in such curved regions, the governing flow equations should be cast into a general curvilinear form. This is done by applying transformation relations, developed from the principles of vector and tensor analysis, in the transformation from cartesian to general curvilinear co–ordinates. In this chapter, the relevant equations which include the momentum, continuity and turbulence equations, as well as the transformation relations used, are considered.

## 2.2 The Governing Equations

In an attempt to define fluid flow behaviour, several workers derived equations by which exact solutions for specific fluid flow problems could be obtained. This eventually lead to the formulation of the well–known Navier – Stokes equations describing complete elliptic fluid motion. The equations describe the law of conservation of momentum for a finite control volume as shown in Figure 2.1.

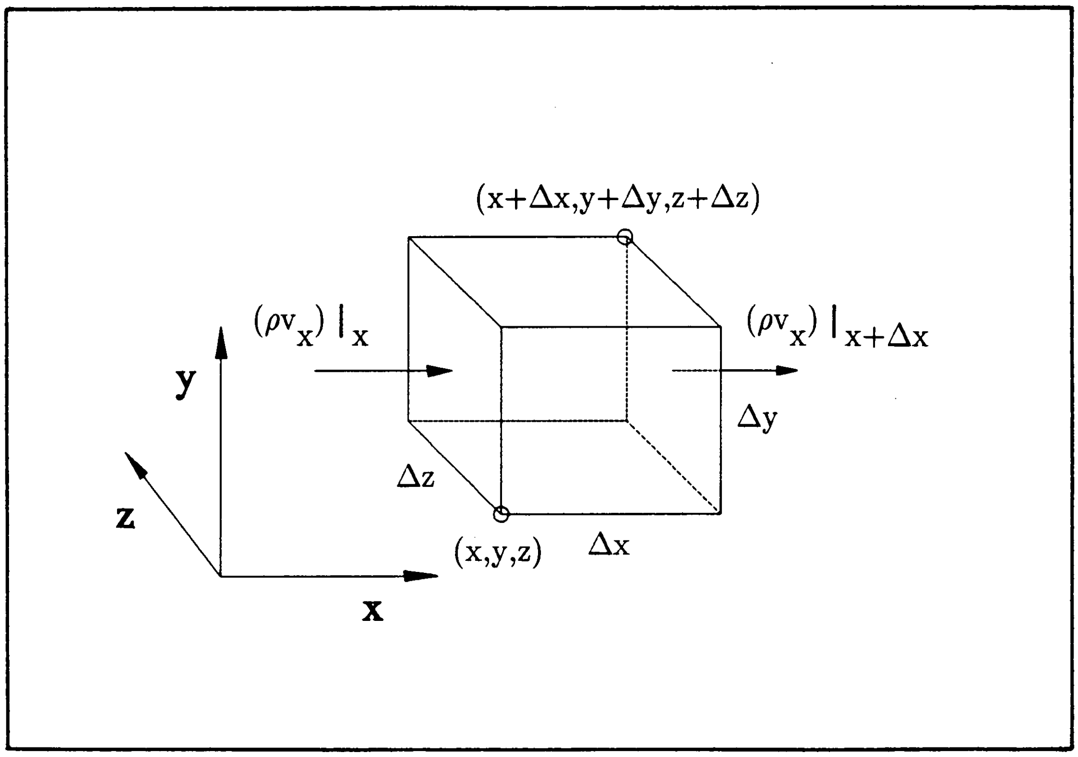

Figure 2.1 A general finite control volume element

The complete derivation of one of the momentum equations (for u–velocity) relative to cartesian co–ordinates is given in Appendix A. The same conservational principles also apply to the other two velocity components. These equations can be summarized by formulating the general transport equation, given below, which is independent of a co–ordinate system.

$$\frac{\partial}{\partial t}(\rho\phi) = -\nabla \bullet (\rho \boldsymbol{v} \phi) - \nabla \bullet (\Gamma \nabla \phi) + S^{\phi} \tag{2.1}$$

The equation is presented in vector form with $\phi$ representing any one of the three–dimensional velocity components, $\Gamma$ the diffusion coefficient and $S^{\phi}$ the source term which may contain pressure and gravity forces.

In addition to the momentum equations, fluid flow also obeys the law of conservation of mass. This implies that the total mass transferred into a control volume as shown in Figure 2.1, should equal the mass flux out of the volume plus the mass accumulated therein. Based on this principle, the continuity equation is derived. In vector notation the equation can be written as:

$$\frac{\partial}{\partial t}(\rho) = -\nabla \bullet (\rho \boldsymbol{v}) \tag{2.2}$$

where $\boldsymbol{v}$ represents the complete velocity vector consisting of the three velocity components (u,v,w).

The continuity equation can be combined with the momentum equations to present a mathematical model whereby three–dimensional laminar fluid flow can be calculated.

The equations presented above are general and independent of a co–ordinate system. Before the solution of these general equations can be addressed, the equations should be completely quantified relative to a reference co–ordinate system. For this purpose a general curvilinear co–ordinate system is used. In the next section, the process of formulating the equations in terms of general curvilinear co–ordinates is described.

## 2.3 Transformation to curvilinear co–ordinates

The choice of a co–ordinate system for the solution of three–dimensional fluid flow is very important because it affects the accuracy as well as the efficiency of the solution. Classical co–ordinate systems, such as cartesian and cylindrical co–ordinates, are limited in the extent to which obscurely shaped flow regions can be modelled. To overcome the restrictions inherent to such systems, a general curvilinear co–ordinate system is used.

For the solution of flow in general curvilinear co–ordinates, the governing equations are also required in the general form. The equations are transformed from classical to curvilinear co–ordinates. This is done using certain concepts from differential geometry and tensor analysis [16,48]. An arbitrarily shaped region is thereby transformed into a uniform calculation domain. (Figure 2.2)

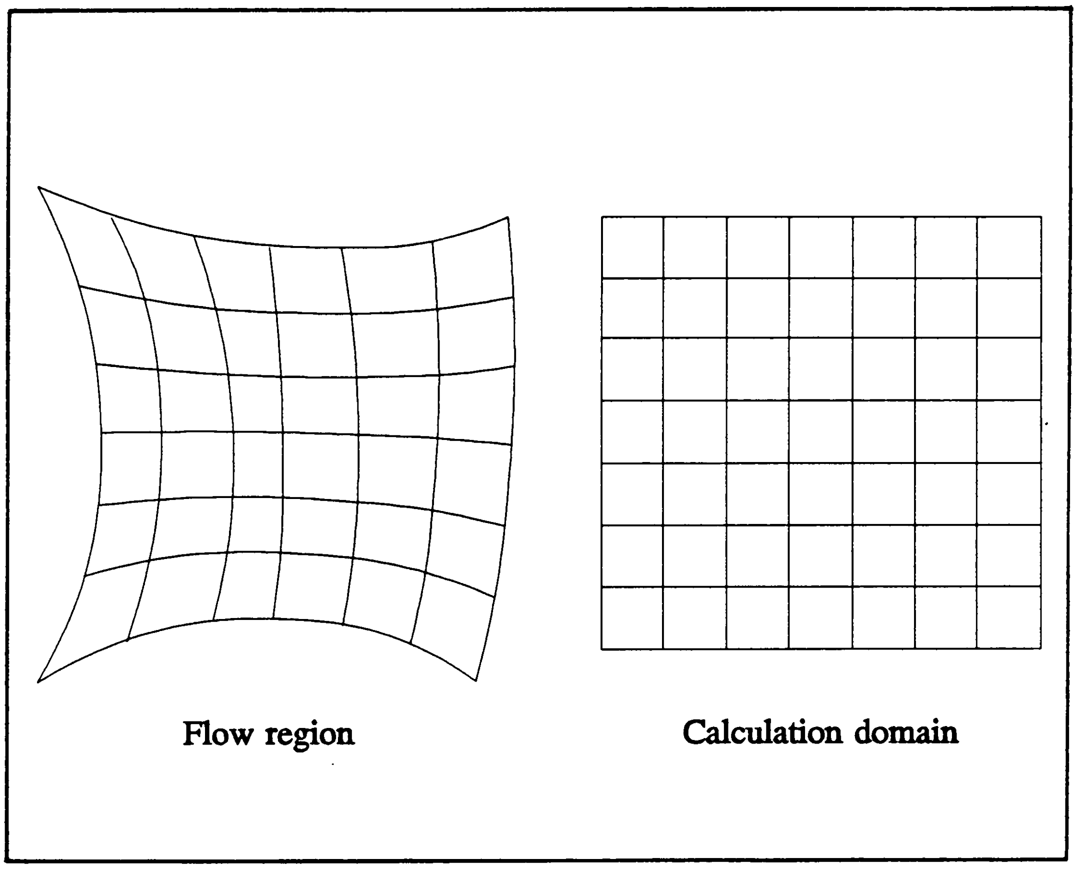

Figure 2.2 Effective transformation of the physical region by transforming the governing equations

Thompson [16] states the relationship between a cartesian co–ordinate system and a curvilinear co–ordinate system in the form of the relationship between their respective base vectors,

$$\mathbf{a}_i = \mathbf{r}_{\xi^i} \quad (i = 1,2,3) \tag{2.3}$$

where $\mathbf{a}_i$ represents the covariant base vectors relative to a curvilinear co–ordinate system while $\mathbf{r}$ consists of the cartesian unit vectors.

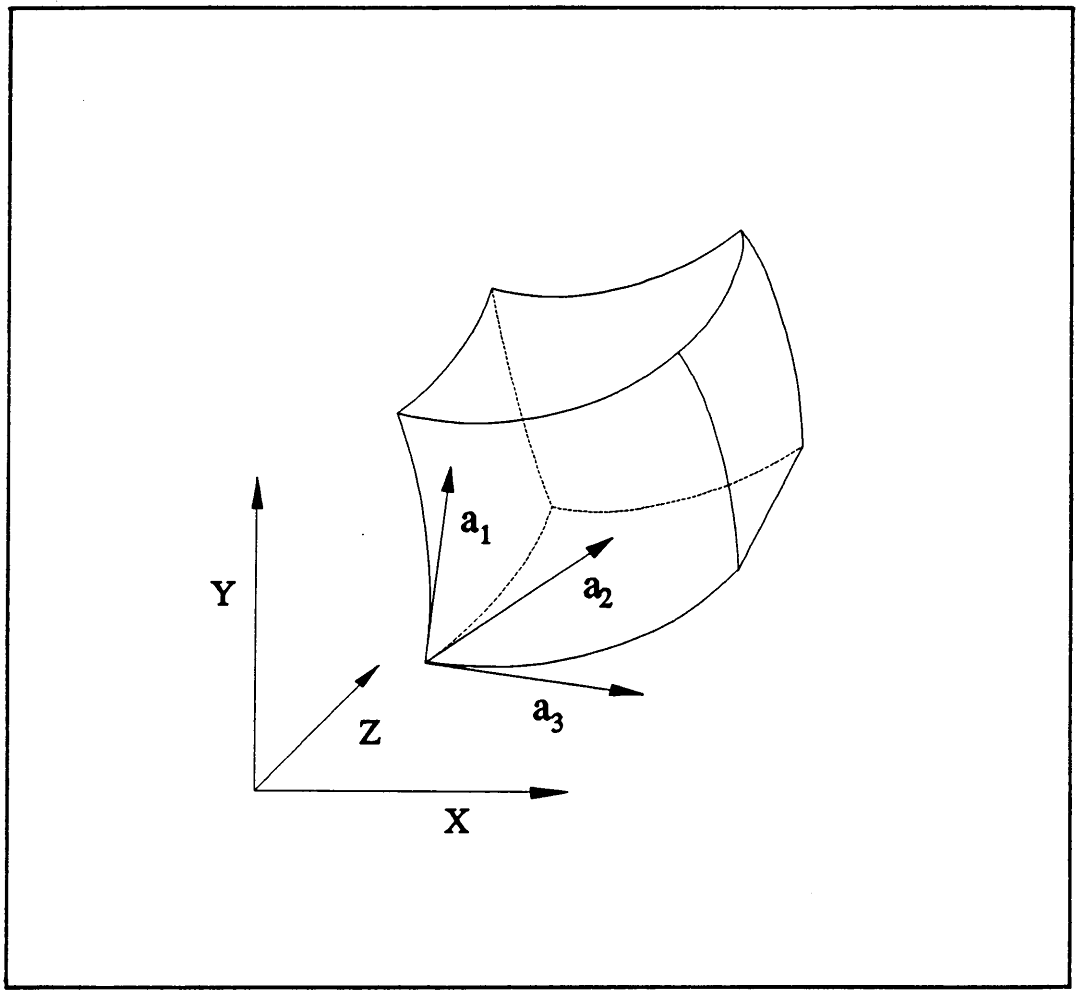

**Figure 2.3** Contravariant base vectors relating cartesian to curvilinear co—ordinates

The complete argument of transforming cartesian co–ordinates to curvilinear co–ordinates is explained in Appendix B where the covariant base vectors are utilized in defining expressions for arc length, surface area and volume element. These expressions are used in determining transformation relations for the differential operators occurring in the general transport equation.

Mathematical expressions for gradient ($\nabla\phi$), divergence ($\nabla\bullet\phi$) and for the individual partial derivatives of velocity, are derived in Appendix B and given below.

Gradient: $$\nabla\phi = J\sum_{i=1}^{3}\left[(a_j \times a_k)\phi\right]_{\xi^i} \tag{2.4}$$

Divergence: $$\nabla \bullet \phi = J\sum_{i=1}^{3}\left[(a_j \times a_k) \bullet \phi\right]_{\xi^i} \tag{2.5}$$

Derivatives: $$\frac{\partial u}{\partial y} = J\left[\frac{\partial}{\partial \xi}\left[\frac{u\xi_y}{J}\right] + \frac{\partial}{\partial \eta}\left[\frac{u\eta_y}{J}\right] + \frac{\partial}{\partial \zeta}\left[\frac{u\zeta_y}{J}\right]\right] \tag{2.6}$$

In curvilinear co–ordinates, the general transport equation can be written as follows:

$$\begin{aligned}\frac{\partial}{\partial t}\left[\frac{U}{J}\right] &+ \frac{\partial}{\partial \xi}\left[\frac{1}{J}(\xi_x E + \xi_y F + \xi_z G)\right] \\ &+ \frac{\partial}{\partial \eta}\left[\frac{1}{J}(\eta_x E + \eta_y F + \eta_z G)\right] \\ &+ \frac{\partial}{\partial \zeta}\left[\frac{1}{J}(\zeta_x E + \zeta_y F + \zeta_z G)\right] = 0\end{aligned} \tag{2.7}$$

where $\xi$, $\eta$ and $\zeta$ represent the three–dimensional curvilinear space co–ordinates while the vectors **U**,**E**,**F** and **G** contain the convection, diffusion and pressure terms as defined in Appendix B.

## 2.4 <u>The Turbulence Equations</u>

Fluid flow turbulence is a phenomenon of great importance in many fields of engineering and science. As most flows encountered in nature are turbulent, a good understanding of its fundamental mechanisms is a necessity. It does, however, present some of the most difficult problems both in the fundamental understanding of its physics and in practical applications.

Turbulent fluid motion can be described as an irregular condition of flow in which the various quantities show a random variation in time and space co–ordinates, so that statistically distinct average values can be discerned. A better idea of what is meant can be gained from Figure 2.4, where the small scale fluctuations around a mean value ($\phi$) for any of the dependent variables is shown.

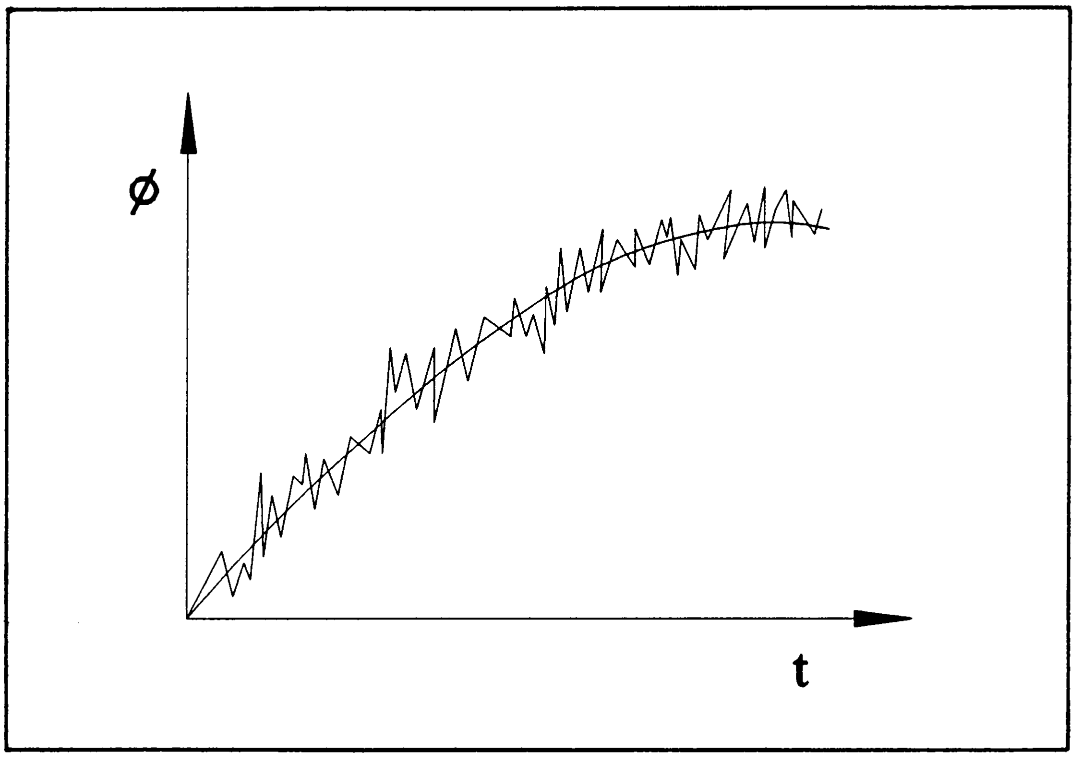

**Figure 2.4** Small scale fluctuations in turbulent flow

By now assuming that each of the dependent variables consist of an average value and a fluctuating value, as shown below, the Navier Stokes equations previously discussed, can be adapted to represent turbulent fluid motion.

$$u = \bar{u} + u' \qquad v = \bar{v} + v' \qquad w = \bar{w} + w'$$

$$\rho = \bar{\rho} + \rho' \qquad p = \bar{p} + p'$$

The complete argument is presented in Appendix C. It is shown that the momentum equations, which where derived from Newton's universal conservational principles, remain valid for turbulent flows with the mere substitution of the laminar viscosity by an effective viscosity. The effective viscosity is defined as the sum of the laminar and turbulent viscosities.

$$\mu_{\text{eff}} = \mu_{\text{lam}} + \mu_t \tag{2.8}$$

In this study the k–$\epsilon$ turbulence model is applied in the calculation of the turbulent viscosity value ($\mu_t$). The model entails the solution of two additional partial differential equations, one for the turbulence dissipation rate ($\epsilon$) and the other for the kinetic energy of turbulence (k). The turbulence equations which are also independent of a co–ordinate system are given below.

$$\underbrace{\frac{\partial\epsilon}{\partial t}}_{\text{time}} + \underbrace{(\mathbf{v}\bullet\nabla\epsilon)}_{\text{convection}} = \underbrace{\frac{1}{\rho}\left[\nabla\bullet\left(\frac{\mu_t}{\sigma_\epsilon}\nabla\epsilon\right)\right]}_{\text{diffusion}} + \underbrace{\frac{C_1\mu_t}{\rho}\frac{\epsilon}{k}\Gamma}_{\text{production}} - \underbrace{C_2\frac{\epsilon^2}{k}}_{\text{dissipation}} \tag{2.9}$$

$$\underbrace{\frac{\partial k}{\partial t}}_{\text{time}} + \underbrace{(\mathbf{v}\bullet\nabla k)}_{\text{convection}} = \underbrace{\frac{1}{\rho}\left[\nabla\bullet\left(\frac{\mu_t}{\sigma_k}\nabla k\right)\right]}_{\text{diffusion}} + \underbrace{\frac{\mu_t}{\rho}\Gamma}_{\text{production}} - \underbrace{\epsilon}_{\text{dissipation}} \tag{2.10}$$

By solving the above partial differential equations for the unknown variables, the value of the turbulent viscosity can be calculated by

$$\mu_t = C_\mu\,\rho\,k^2/\epsilon\,. \tag{2.11}$$

The constants $C_\mu$, $C_1$, $C_2$, $\sigma_\epsilon$ and $\sigma_k$ occurring in equations (2.9) and (2.10) are empirical and do not have fixed values. Their values differ slightly for various fluid flow applications and is published widely for these cases. For most general fluid flow problems the following numerical values are commonly assigned to each of the constants [18,24–28].

$$C_\mu = 0.09, \quad C_1 = 1.44, \quad C_2 = 1.92, \quad \sigma_\epsilon = 1.0 \quad \text{and} \quad \sigma_k = 1.33$$

When atmospheric boundary layer flow is considered, the following set of values are proposed [25].

$$C_\mu = 0.03, \quad C_1 = 1.54, \quad C_2 = 2.0, \quad \sigma_\epsilon = 1.0 \quad \text{and} \quad \sigma_k = 1.0$$

The k–$\epsilon$ turbulence model is, however, only valid for fully turbulent flows. Close to solid walls, there are inevitably regions where the local Reynolds number is so small that viscous effects predominate over turbulent ones [10]. Special attention is required at these regions. The greatest concern is not for the values on the walls but for the values right next to the wall which are in the flow field. Many authors successfully applied wall function methods developed by Spalding [21] and Gosman [22] by which wall shear stresses are calculated.

These viscous shear stresses are dominant in the sub–layer adjoining the wall where the flow is laminar and the local Reynolds number ($y^+$) is low. The local Reynolds number is defined by the following equation:

$$y^+ = y_p \, \rho \, C_\mu^{1/4} k^{1/2} / \mu \tag{2.12}$$

For a local Reynolds number of lower than 11,6 the shear stress can be calculated by

$$\tau_w = -\mu\, v_t / y_p \ , \tag{2.13}$$

where $y_p$ represents the distance from the solid wall and $v_t$ the tangential velocity at this distance from the wall. When the local Reynolds number is greater than 11,6 the shear stress is described by

$$\tau_w = (\rho C_\mu^{1/4} k^{1/2} \kappa\, v_t)/(\ln(\epsilon y^+) \tag{2.14}$$

Near solid walls, the partial differential equation describing k is modified somewhat by replacing the dissipation and production terms by the following expression, which depends on the wall shear stress.

$$-\Gamma + \rho\epsilon = -\tau_w \frac{\partial v_t}{\partial y_i} + \frac{C_\mu \rho^2 k^2}{\tau_w} \left[\frac{\partial v_t}{\partial y_i}\right] \tag{2.15}$$

In the above equation, $y_i$ represents the direction perpendicular to the wall so that $y_p$ is measured in this direction. The value of $\epsilon$ close to the wall can also be calculated as a function of the wall shear stress according to the equation below, where $\kappa$ represents the Von Karman constant.

$$\epsilon = \frac{C_\mu^{3/4} k^{1/2}}{\kappa\, y_p} \tag{2.16}$$

This outlines the basic mathematics behind the two equation k—$\epsilon$ turbulence model. The numerical application as well as the treatment of solid wall and other turbulence boundaries will be discussed in Chapter 3.

The k–$\epsilon$ turbulence model does not describe turbulent fluid motion from its basic principles since it depends on empirical values. It does, however, provide a useful compromise between accuracy and computability and is therefore the most widely used mathematical method for describing fluid flow turbulence in engineering practice today.

## 2.5 Summary

Turbulent fluid motion are described by six partial differential equations. They are the three momentum equations, the continuity equation and the two turbulence equations. Each of these equations is based on Newton's universal law of conservation for the transport of scalar variables and can be represented by a General Transport Equation. To solve turbulent fluid flow in curvilinear co–ordinates and in realistic geometries, transformation relations are used to formulate the governing equations in terms of general curvilinear co–ordinates.

Due to the non–linearity of the equations, they cannot be solved analytically for complex flow situations without making a large number of simplifying assumptions. The equations are, in fact, also interdependent and require simultaneous solution. In order to avoid such assumptions, the following chapter will deal with the aspects of developing a numerical solution method for solving the equations considered in this study.

# CHAPTER 3 NUMERICAL MODELLING

## 3.1 Preamble

In the previous chapter, the appropriate differential equations describing turbulent fluid flow in three–dimensional curvilinear co–ordinates were discussed. The next step is the development of a solution procedure which can be applied to obtain a numerical solution to the governing partial differential equations.

Two basic approaches can be followed to solve the partial differential equations numerically, namely finite element and finite difference methods. Although the ability of finite element methods for irregular grids have been recognized for some time, a number of difficulties have hampered progress in its application to fluid flow problems [5]. The main difficulty concerns the upwind nature of convection which is poorly addressed when straightforward finite element methods are used. These methods further often employ the direct simultaneous solution of all the equations, which is a expensive process. Alternatively, a finite difference method, which offers a better physical interpretation of fluid flow behaviour, can be used. Considerable progress has been made in applying finite difference methods to randomly shaped regions. By transforming the equations relative to general curvilinear co–ordinates (Section 2.3), flow in arbitrary curved regions can be adequately modelled. Due to the above considerations, the finite difference approach is used in this study to obtain a numerical solution to three–dimensional turbulent flow in curvilinear co–ordinates.

The first step in the development of a computational model, is the discretization of the differential equations relative to the problem. The process of obtaining linear discretization equations for the dependent variables will be discussed in Section 3.2. In Section 3.3 a pressure correction equation is derived from the equation of continuity, which is used to enforce the law of conservation of mass upon the flow field. As a result of the finite volume approach used during discretization and the separate solution of pressures and velocities, decoupling between the pressure and velocity fields occur. An interpolation scheme to remedy the situation is discussed in Section 3.4.

In order to ensure realistic simulation of any practical flow situation, the physical boundary conditions applicable to the problem should be correctly implemented in the numerical model. Section 3.5 deals with a number of general boundary conditions applicable to most flow situations, including flow over real buildings.

Once the above processes have been considered, a method is required whereby the values of the dependent variables can be obtained numerically at discrete points throughout the calculation domain. In Section 3.6, a method for separately solving each of the linearized equations is presented. Finally the entire solution algorithm for calculating turbulent fluid flow is given.

## 3.2 Discretization Equations

In order to calculate numerical values for the variables describing fluid flow, the governing equations should be discretized. This entails the use of finite differences to replace the partial derivatives in time and space occurring in the governing equations. The finite differences are obtained by evaluating the variables at discrete points throughout the field. In this way the highly non–linear partial differential equations are linearized.

In the discretization process, the control volume formulation is used. This is done to ensure that the conservation principle, which serves as the basis in the derivation of the General Transport Equation (2.1),is preserved [17]. A two–dimensional representation of the arrangement of these control volumes surrounding grid points is shown in Figure 3.1.

It is shown that the whole domain is covered with non–overlapping control volumes so that the boundaries of the region are exactly matched by a combination of control volume boundaries. This feature becomes useful when obstructions in the flow field have to be delt with, as such obstructions can be closely conformed by the cell boundaries. In setting up such an

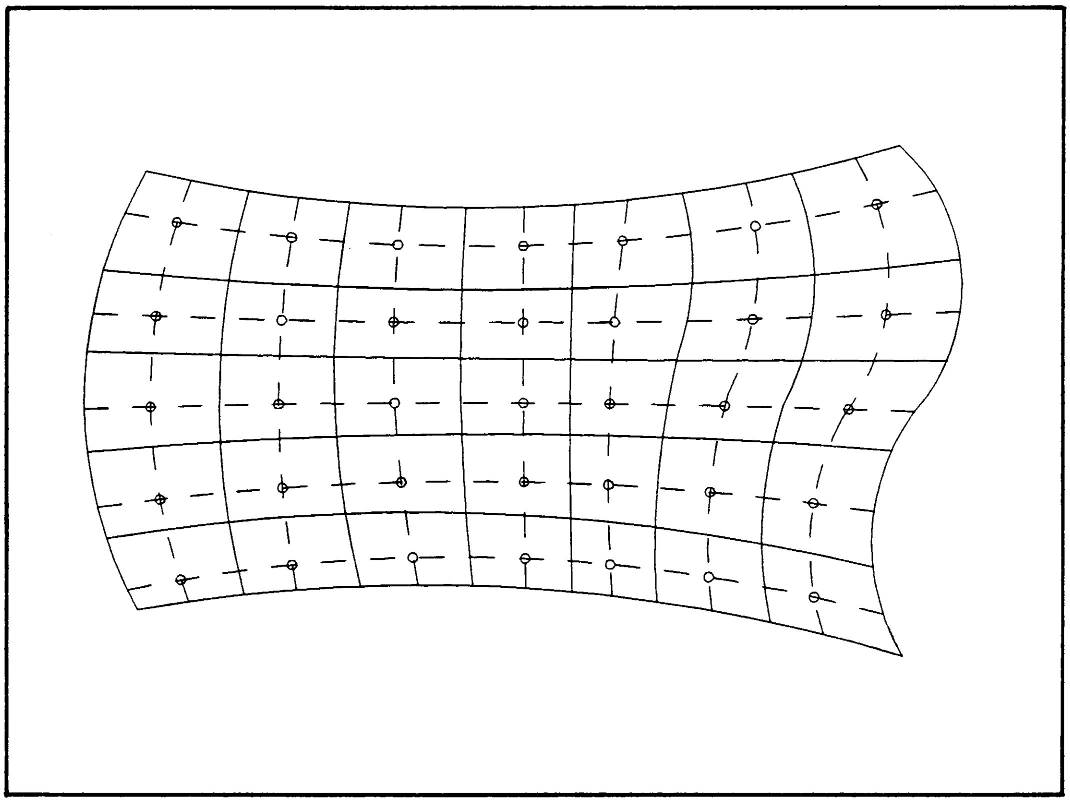

**Figure 3.1** Typical representation of a control volume arrangement associated with a finite volume methods

arrangement, a double refined grid is initially generated by positioning the grid points at the intersections of grid lines. Exact transformation values can then be calculated at cell wall boundaries instead of being determined by interpolation between values of neighbouring majour grid points. This improves the accuracy of the transformation values used in the differential equations to describe control volume geometries. In Figure 3.1 the control volume boundaries are therefore represented by the solid grid lines.

One such control volume is shown in Figure 3.2 where the position of the grid point under consideration is defined relative to its neighbours. It is assumed that the value at point P is only influenced by values at the points in its immediate surroundings [5].

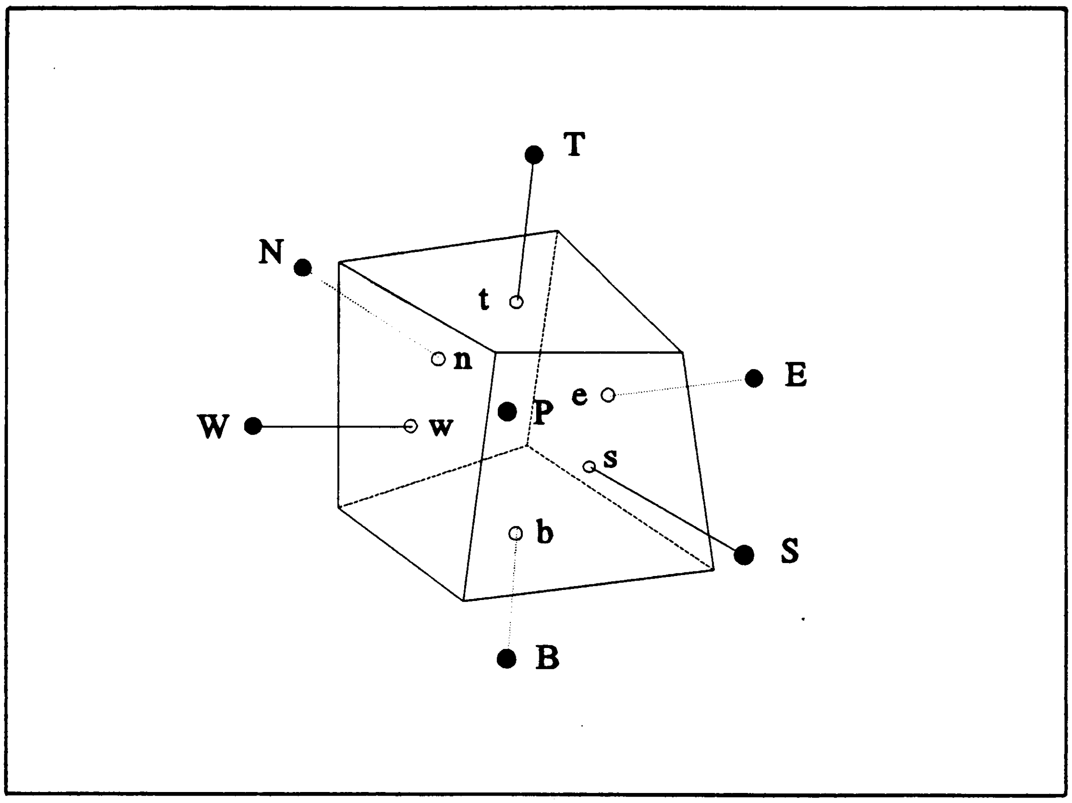

**Figure 3.2** A typical finite control volume element

The General Transport Equation is integrated over the control volume above. It is assumed that the density is constant over the whole volume and that cell wall velocities govern the mass flow over the whole face. These velocities are evaluated by applying linear interpolation between major points. A general form of the discretization equation is obtained for the calculation of the value at point P.

$$a_p\phi_P = a_e\phi_E + a_w\phi_W + a_n\phi_N + a_s\phi_S + a_b\phi_B + a_f\phi_F + a_p^o\phi_P^o + b \qquad (3.1)$$

The coefficients of the neighbouring points represent convection and diffusion flux while the source term b contains information concerning grid deformation and pressure effects. The coefficient $a_p^o$ represents the time dependence of the specific variable in cases where unsteady flow is modelled.

## 3.3 Pressure correction equation

A difficulty exists in calculating the velocity field using the expressions developed above. This is due to the fact that the pressure field is unknown. No explicit equation for obtaining the pressure exists, and therefore a special procedure for the calculation of the pressure field should be considered. A well documented segregated approach can be followed whereby pressure corrections are calculated with the use of the continuity equation [5].

If the pressure field is explicitly known, only the velocity field needs to be calculated. In the equation below, $\phi$ represents the values of any of the three–dimensional velocity components (u,v,w) to be calculated, and P the known correct pressures. The coefficients $B^{\phi}$, $C^{\phi}$ and $D^{\phi}$ involve density, area and metric coefficients and $S^{\phi}$ represents the source terms.

$$\phi_P = \Sigma\, a_{nb}\phi_{nb} + B^{\phi}P_{\xi} + C^{\phi}P_{\eta} + D^{\phi}P_{\zeta} + S^{\phi} \tag{3.2}$$

Usually the pressure field is not known explicitly, and approximate velocities ($\phi^*$) are calculated by guessing an initial pressure distribution ($P^*$).

$$\phi^*_P = \Sigma\, a_{nb}\phi^*_{nb} + B^{\phi}P^*_{\xi} + C^{\phi}P^*_{\eta} + D^{\phi}P^*_{\zeta} + S^{\phi} \tag{3.3}$$

However, in general these velocities do not satisfy the continuity equation and a net mass source is produced instead. In order to remove this mass source, the velocity values ($\phi^*$) can be corrected by a correction $\phi''$.

$$\phi'' = B^{\phi}P''_{\xi} + C^{\phi}P''_{\eta} + D^{\phi}P''_{\zeta} \tag{3.4}$$

By replacing velocity values (u,v,w) in the continuity equation by the sum of the approximate values and the corrections ($u^* + u''$,$v^* + v''$,$w^* + w''$), an equation is obtained whereby the pressure corrections can be calculated. According to Rhie and Chow [15] the cross derivatives of the pressure correction can be neglected if the grid is nearly orthogonal. For a detailed discussion the reader is referred to Appendix E.

Application of the discretization principles developed in Section 3.2 leads to the final equation for the calculation of the pressure correction at point P as a function of its neighbouring point values.

$$a_P P''_P = \Sigma\, a_{nb} P''_{nb} + S \tag{3.5}$$

where S represents the local imbalance of mass and gives an indication of how well the principle of conservation of mass is satisfied.

## 3.4 Pressure–velocity coupling

A finite volume approach to finite differences implies the calculation of all the dependent variables at the major grid points. This differs from the staggered grid approach which was used by Patankar [5] in the derivation of the solution algorithms of the next section. The use of the staggered grid arrangement involves the calculation of pressures and turbulence values at major grid points while velocity components are calculated at corresponding control volume surfaces [10,24]. This technique can, however, not be applied when considering the equations in general curvilinear co–ordinates, since the three–dimensional velocity components u, v and w are not related to control volume surface orientations.

The straightforward application of the mentioned algorithms to the finite volume method results in the occurrence of oscillatory pressure fields. The major source of this instability is the second–order centered difference approximation for the pressure gradient at point P.

$$\left.\frac{\partial P}{\partial \xi}\right|_P = \frac{P_W - P_E}{2\Delta\xi} \tag{3.6}$$

This means that the pressure at point P has no influence on the finite volume expression for the pressure derivative during the calculation of the velocity at point P. This causes the decoupling between the pressure and velocity fields.

As an alternative approach, a special method for the treatment of locally linearized convection terms at the control volume surfaces, was introduced by Rhie et al. [15]. These terms are calculated by linear interpolation between adjacent point values. This interpolation causes the decoupling of the pressures and velocities. Consider the equations used in calculating $u_P^*$ and $u_E^*$.

$$u_P^* = \Sigma\, a_{nb} u_{nb}^* + BB_P^u\left[\frac{P_W - P_E}{2\Delta\xi}\right] + b^u \tag{3.7a}$$

$$u_E^* = \Sigma\, a_{nb} u_{nb}^* + BB_E^u\left[\frac{P_P - P_{EE}}{2\Delta\xi}\right] + b^u \tag{3.7b}$$

By linear interpolation for the value of $\bar{u}_e^*$, $1\Delta\xi$–pressure variations cannot be detected. One may remedy the situation by correcting the pressure derivative through a $1\Delta\xi$–difference scheme on the cell boundary to yield

$$u_e^* = \bar{u}_e^* + \overline{BB}\left[\frac{P_E - P_P}{1\Delta\xi}\right] - \frac{1}{2}BB_P^u\left[\frac{P_W - P_E}{2\Delta\xi}\right] - \frac{1}{2}BB_E^u\left[\frac{P_P - P_{EE}}{2\Delta\xi}\right] \tag{3.8}$$

where the overbar denotes linearly interpolated values and EE indicates the value at the grid point situated east of point E (East).

By applying this procedure prior to the solution of the pressure correction equation strong pressure–velocity coupling is ensured.

## 3.5 Boundary Conditions

Once the mathematical equations describing any fluid flow problem have been correctly manipulated to enable numerical solution, only the boundary conditions affect the accuracy of the solution. The real effects influencing the physical flow situation must be modelled accurately to ensure that the numerical solution reflect the practical problem.

The different boundary conditions that are applicable to this study are discussed in two parts. First, the general boundary conditions applicable to most fluid flow problems are presented, followed by the boundary conditions applicable to flow over buildings. Special attention is given to the turbulence and velocity profiles in the atmospheric boundary layer.

### 3.5.1 General boundary conditions

#### a) Inflow boundaries

Values for dependent variables that are specified at inflow boundaries are usually determined experimentally by measurement. Fixed values include the three velocity components and the two turbulence properties. In modelling incompressible flow, pressure boundaries are usually not specified and are left to find their own values as determined by the flow field. Pressures on inflow boundaries are often set equal to or on a constant gradient with the numerically calculated pressures immediately downstream.

#### b) Solid walls

Most practical flow situations are bounded at some point by a fixed wall. At this wall, velocities tangential and perpendicular to the wall are zero. In Figure 3.3 an element next to a fixed wall is shown. The velocities at the western cell wall (w) are made zero when calculating the values at point P.

Since the pressure correction is not a measurable physical entity, the values are not known at the walls. When the pressure correction coefficient $a_w$ is set equal to zero, the value on the wall has no influence on the pressure corrections in the flow field. Within the flow region, the pressure corrections find their own values and eventually approach zero when the final solution is obtained.

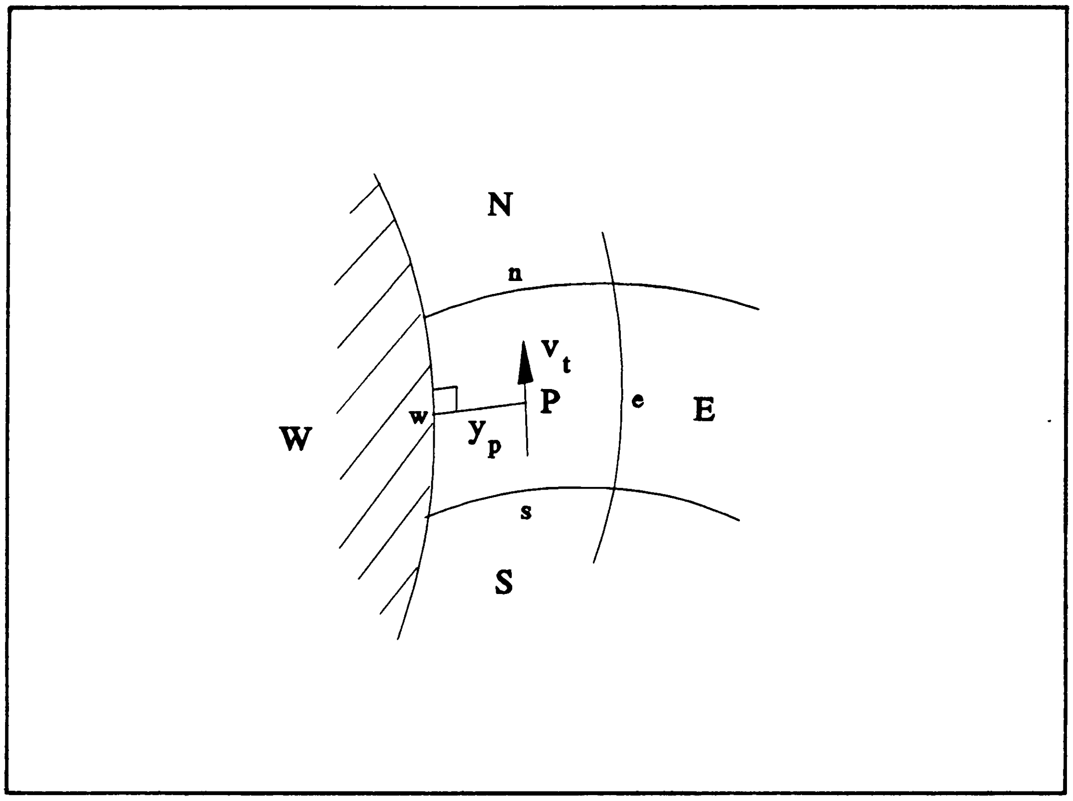

Figure 3.3 Control volume at fixed wall boundary

Due to the finite volume method used, the pressure on the wall is required in solving the velocities at point P. The assumption is made that the pressure on the wall equals the pressure at point P, hence a zero gradient.

The turbulence model used in this study, is valid only for fully turbulent flows. Close to solid walls, there are inevitably regions where the local Reynolds number is so small that viscous effects predominate over turbulent ones [21]. In these cases, the value of the kinetic energy of turbulence (k) on the wall is theoretically zero, as the fluid film adjoining the wall is stationary. Due to the extremely large gradients that occur in these areas, the use of a zero gradient for k gives a better approximation [10,24]:

$\frac{\partial k}{\partial \xi_i} = 0$ , $\xi_i$ perpendicular to the wall.

The value of $\epsilon$ at the wall is very large, as the dissipation rate of turbulent energy on the wall is virtually infinite. The wall function method is also used to calculate the value of $\epsilon_P$. Where necessary, the value on the wall is then set equal to the value of the immediate neighbouring point in the flow region. For both turbulence quantities, the wall coefficient ($a_w$) is set equal to zero so that it does not adversely affect the flow field.

**c) Symmetrical planes**

It is common practice to model only half of the problem if the flow region and the applicable boundary conditions are symmetrical. Values on opposite sides of the plane coincide and therefore zero gradients are used for all the flow variables.

$\frac{\partial \phi}{\partial \xi_i} = 0$ , $\xi_i$ perpendicular to the wall

Where $\phi$ represents any one of the dependent variables.

**d) Constant velocity boundaries**

These boundaries occur when the fluid is in contact with a moving wall, as in the case of the driven cavity (sliding lid) problem, or where the flow is bounded by neighbouring freestream velocity. For these cases, fixed velocities are specified and zero gradients are used for k, $\epsilon$ and the pressure corrections. For freestream boundaries, velocities are specified in such way that mass flow across these boundaries is prevented. Its application should be carefully investigated to ensure that the boundary condition is representative of the real boundary. In the case of modelling flow over buildings, the boundary should be placed far enough from the building to ensure that that the flow field around the building is not influenced.

**e) Outflow boundaries**

At a first glance one might underestimate the importance of outflow boundaries. Although they are usually situated downstream, an incorrect implementation could lead to inaccurate upstream numerical results and could even prevent convergence. The outflow boundary should ensure that global mass conservation is enforced. In most cases zero gradients are used for all the flow quantities and the velocities are adjusted to obey overall continuity.

### 3.5.2 Boundaries applicable to flow over buildings

Of great importance to the numerical simulation of wind motion over buildings is the correct specification of the boundary layer profiles. These profiles are usually dependent on the unique terrain associated with the problem. Large amounts of empirical data describing the nature of velocity and turbulence profiles in the atmospheric boundary layer exist. Two widely accepted empirical relations describing boundary layer flow are the logarithmic and the power law, the latter used during this study. The equation describing the boundary layer velocity profiles is:

$$\frac{V(y)}{V_{ref}} = \left[\frac{y}{h_{ref}}\right]^{\alpha} \tag{3.9}$$

where V(y) is the mean horizontal wind speed component at a height y and $V_{ref}$ is the mean horizontal wind speed at a reference height $h_{ref}$. The exponent $\alpha$ is the mean wind speed exponent which is dependent on the upstream terrain roughness. Values of $\alpha$ are widely published in the literature [53].

According to Scruton [53], the inflow length scale values for atmospheric longitudinal turbulence L are approximated by the following empirical relation:

$$L(y) = 151\ (y/10)^{\alpha} \tag{3.10}$$

where L(y) is the turbulence length scale in the flow direction at height y. This upstream length scale is included in the numerical model via inflow values for the turbulent dissipation rate $\epsilon(y)$. The relationship between $\epsilon(y)$ and L(y) is defined by [6]:

$$\epsilon(y) = [\ C_D\ \rho\ k(y)^{3/2}]\ /\ L(y) \tag{3.11}$$

where $C_D$ is a constant, with value 0,07 for full scale atmospheric turbulence [25].

An empirical relation can also be used to approximate the turbulence intensity I(y) of natural wind at the inflow boundary. Scruton [53] states the relation as:

$$I(y) = (6{,}7\, k_s)^{1/2} V_{ref}/V(y) \tag{3.12}$$

where $k_s$ is a surface roughness parameter which is a measure of the kinetic of the surface friction coefficient of the upstream terrain. The values for the kinetic energy of turbulence k(y) at the inflow boundary can be obtained from the following relation between k(y) and I(y) given by:

$$k(y) = 0{,}5\,[I(y)\, V(y)\,]^2 \tag{3.13}$$

Fixed values for V(y),$\epsilon$(y) and k(y) at the inflow boundary, are calculated using the equations presented above. Cross–stream velocities are set equal to zero while zero gradients are applied to pressures at these boundaries.

## 3.6 Solution algorithms

The general discretization equation derived in the previous section is not linked to any particular method of calculation. A suitable method for solving the linearized equations, for each of the dependent variables, at every point throughout the flow region is now required.

The simplest method is to make use of a Gauss–Seidel point–by–point iteration scheme. When large three–dimensional grids are used, this method becomes very time–consuming and inefficient in the spreading of the influence of boundary conditions. A better method is to use the TDMA–solver (Tri–Diagonal Matrix Algorithm) [5] whereby variables along a grid line are directly calculated. In two or more dimensions, the solver is applied in an iterative manner whereby variables are calculated line–by–line while sweeping across the calculation domain. The method is not perfectly suited for three–dimensional problems, but it can be improved by varying the sweeping direction for every solution of the field. This method speeds up the introduction of the boundary conditions and are used in this study. When unstructured grids are used, the method is not applicable and a direct solver becomes essential.

At this stage it is appropriate to consider the entire solution algorithm required to solve the complete set of discretization equations. Usually the pressures, velocities and turbulence values are unknown and a special procedure is necessary in order to obtain a solution. One such a solution procedure is the SIMPLE algorithm which was introduced by Patankar [5]. This method forms part of a group of segregated methods whereby the pressures and velocities are solved in an uncoupled manner. The process as it is applied in the current model is briefly outlined below.

- Guess initial pressure field $p^*$
- Solve momentum equations to obtain $u^*$, $v^*$ and $w^*$
- Interpolate velocities for pressure–velocity coupling
- Solve the pressure correction equation for $p''$

- Calculate pressure field ($p = p^* + p''$)
- Correct velocities ($u = u^* + u''$)
- Solve turbulence equations ($k$ and $\epsilon$)
- Calculate new viscosity
- Return to the first step using $p$ as $p^*$

For a detailed discussion of the pressure correction equation the reader is referred to Appendix E. The complete algorithm is also presented in [5].

Many alternative SIMPLE–based methods are also available, such as SIMPLER, SIMPLEC, SIMPLEX and SIMPLEN which basically apply the same process with minor modifications for specific applications [45]. According to Van Doornmaal et al. [44] the SIMPLE method is inconsistent in neglecting the underlined term in the equation below, during the derivation of the pressure correction equation.

$$a_p u''_p = \underline{\Sigma\, a_{nb} u''_{nb}} + \Delta p''$$

A SIMPLEC method is presented by introducing a consistent approximation by subtracting $\Sigma a_{nb} u''_e$ on both sides of the above equation and neglecting the term, underlined below, instead.

$$(a_p - \Sigma\, a_{nb})\, u''_e = \underline{\Sigma\, a_{nb}(u''_{nb} - u''_p)} + \Delta p''$$

The method further removes the need for pressure under–relaxation and proves to be more efficient than the SIMPLE method. Both of the methods were applied during the study and a comparison revealed an improvement of almost 30% in calculation time with the SIMPLEC method. A detailed description of this method is presented in [44].

## 3.7 Summary

In this chapter, the development of the numerical model for the simulation of turbulent fluid flow in three–dimensional curvilinear co–ordinates was described. Issues of discretization and linearization of the equations to enable their numerical solution were also discussed. A model was developed for the iterative solution of the governing equations of Chapter 2. A special interpolation scheme was included into the model to ensure strong pressure–velocity coupling. A segregated solution method based on the SIMPLE algorithm was suggested for the solution of the linearized equations while a TDMA–solver was proposed for the actual solution for each of the dependent variables at various nodes. Finally, the relevant boundary conditions and their implementation into the computer code were discussed.

# CHAPTER 4 PROGRAM APPLICATIONS

## 4.1 Preamble

The mathematical and numerical procedures developed in the previous chapters form the basis of the computer program 3DFLO. The development of the code can be divided into the following phases:

I. The development of a two–dimensional laminar fluid flow code in general curvilinear co–ordinates.

II. The extension of the model to simulate three–dimensional laminar fluid flow in curvilinear co–ordinates.

III. The incorporation of a turbulence model to account for turbulent effects in the fluid.

In order to establish confidence in the accuracy of the code it needs to be verified by solving various test cases. In this chapter, the test cases used to verify each new development phase, are discussed. Finally the code is applied in studying wind flow behaviour around arbitrary shaped buildings.

## 4.2 Two–dimensional laminar flow (Phase I)

On completion of Phase I, laminar flow between two parallel plates is modelled. The problem can be considered two–dimensionally if it is assumed that the plates are large compared to the distance between them. Near the middle, the flow is effectively two–dimensional as it is largely unaffected by the boundary conditions in the third dimension. As a result of viscous friction at the solid walls, velocity gradients exist in the boundary layer. The two boundary layers increase in height in the downstream direction and gradually merge to eventually result in a fully developed velocity profile.

Figure 4.1 compares the numerically predicted velocity distribution to the analytical solution for the same application given by Schlichting [47] and to a previous numerical result by Le Grange [17].

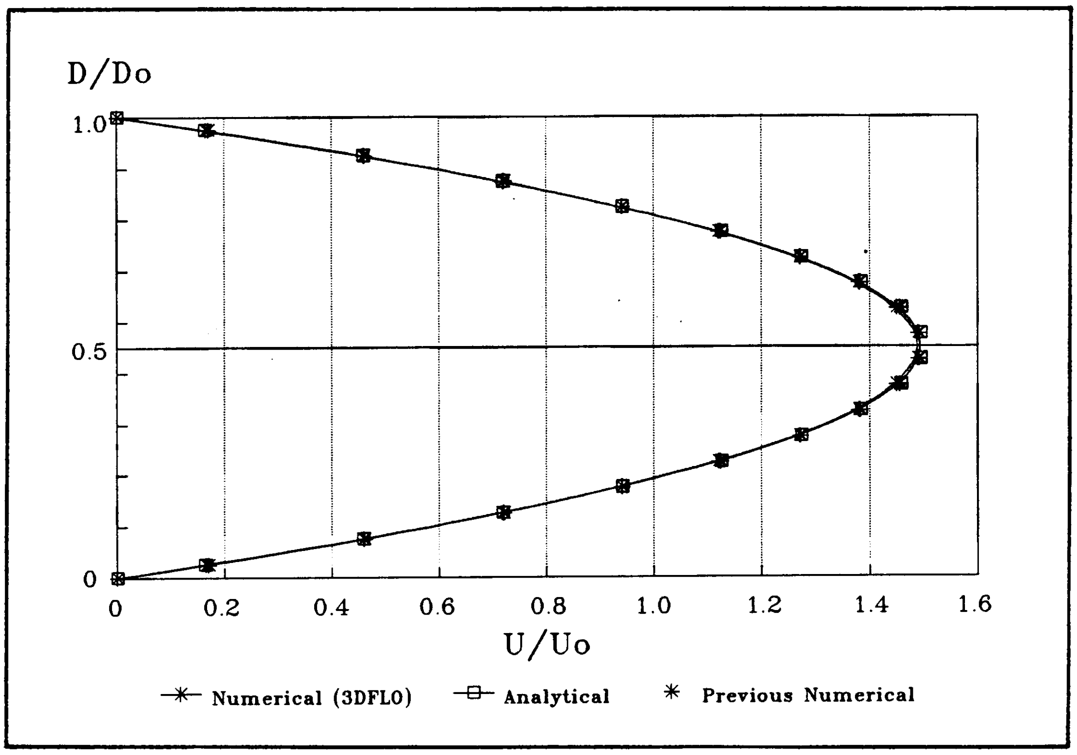

**Figure 4.1** Fully developed velocity profile of laminar channel flow

For ease of reference, the results are presented in dimensionless form. $U_0$ represents the constant stream–wise inlet velocity and $D_0$ the distance between the two plates. The fully developed velocity profile predicted by the numerical code compares favourably to the analytical solution and to a previous numerical result. This indicates the ability of the code to model two–dimensional laminar flow.

In testing the curvilinear abilities of the code, the problem of two–dimensional laminar channel flow, as discussed above, is solved by using two different grid reference systems. Figure 4.2 shows a classical cartesian as well as a curvilinear grid reference system representing the same flow region.

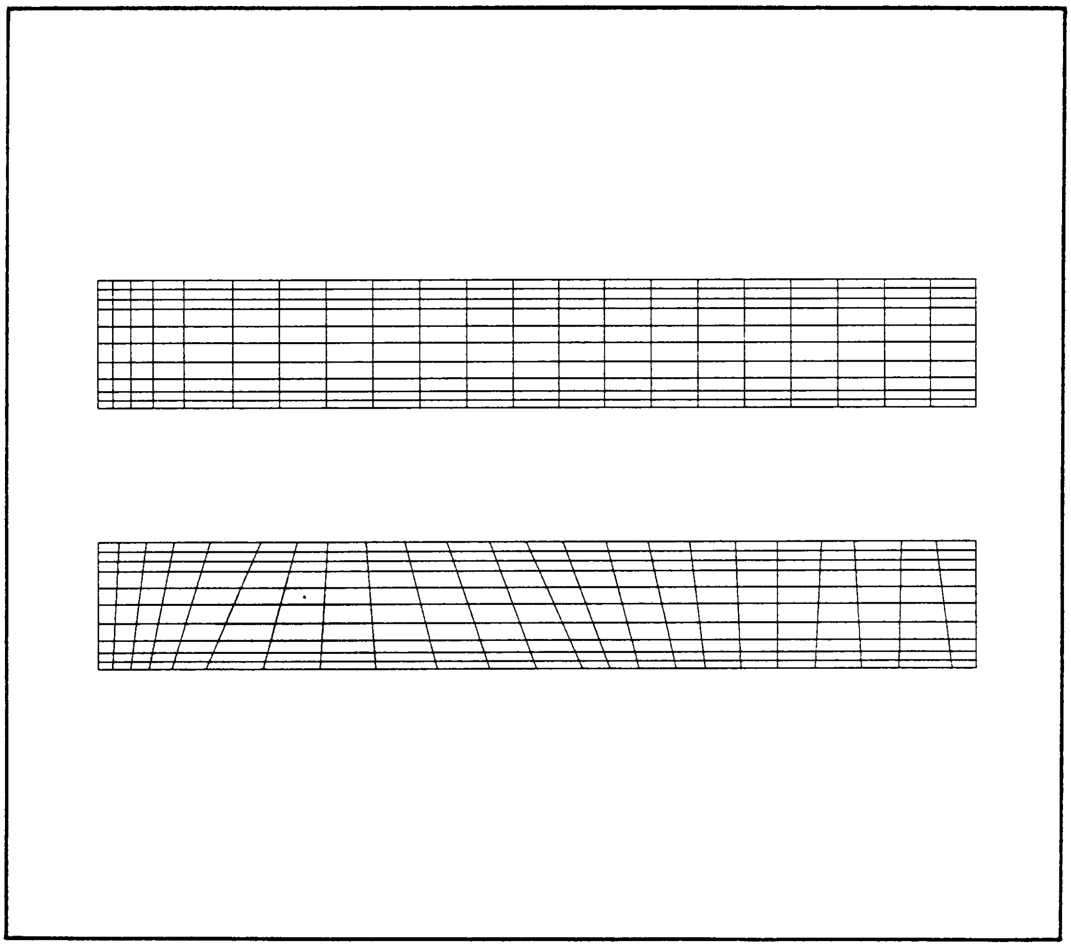

**Figure 4.2** Cartesian and Curvilinear grids for the same region

In the solution of the curvilinear grid, cross derivative terms are included to describe the curvilinear geometrical effects. For classical cartesian co–ordinates these terms are all equal to zero and therefore have no influence. In both cases the same velocity profile for a fully developed laminar channel flow is predicted which confirms the applicability of the code to general curvilinear co–ordinates.

In order to demonstrate the ability of the numerical model to predict recirculating laminar flow, the solution of flow in a two–dimensional driven cavity is considered. Such a cavity is shown below in Figure 4.3. It is indicated that the flow is bounded by three fixed walls and a sliding lid which drives recirculation of the fluid. The problem can be assumed to be two–dimensional when the depth dimension is sufficiently large compared to the other two dimensions of the cavity.

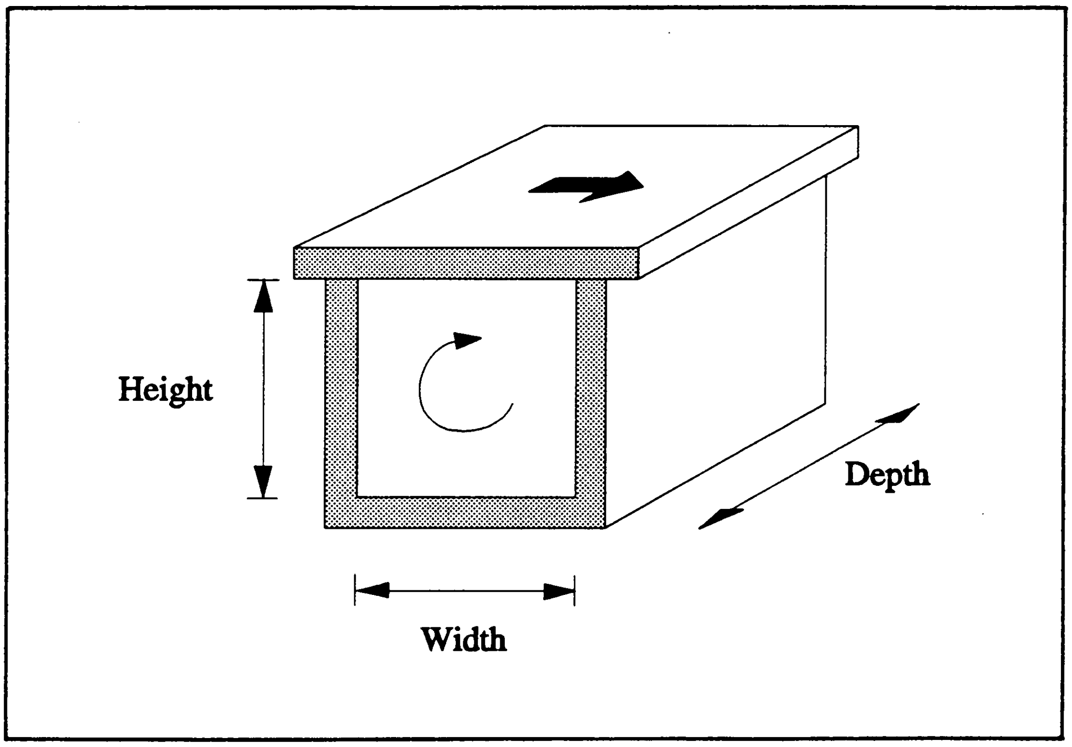

**Figure 4.3** Schematic representation of the driven cavity problem

For a low Reynolds number (Re = 1) the complete velocity distribution and the centreline velocities are given in Figures 4.4a and 4.4b. The results compare favourably to data published by Al–Sanea, Pun and Spalding [49].

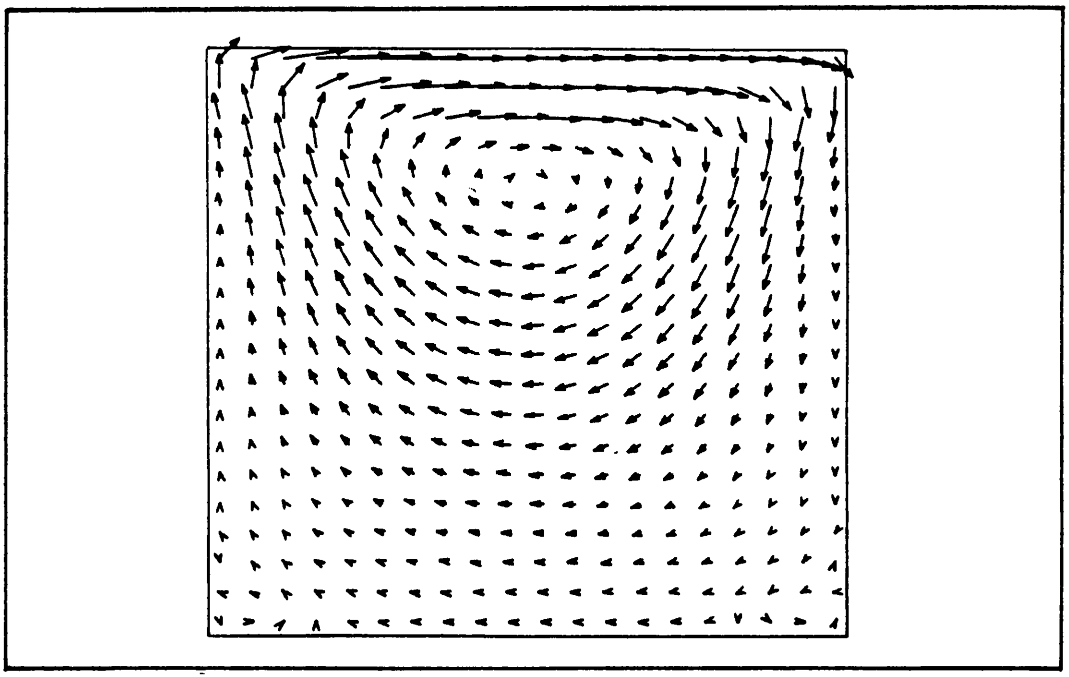

**Figure 4.4a** Driven cavity velocity distribution (Re = 1)

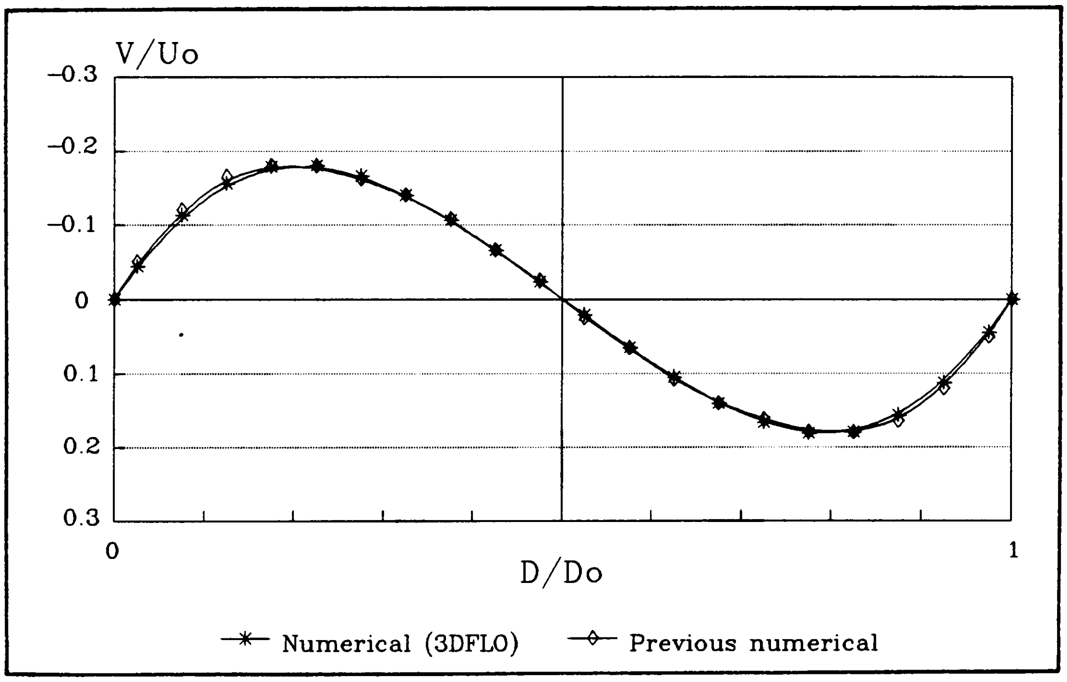

**Figure 4.4b** Driven cavity centreline velocity (Re = 1)

Good agreement is found by relating the position of the centre of rotation and the overall flow pattern. Each of the four different walls of the cavity are moved in turn to check the symmetry of the code. The flow fields generated in this way turn out to be identical.

The results are verified quantitatively by comparing the predicted velocity profile (vertical velocities) on the horizontal centreline to an accepted numerical solution [49] as shown in Figure 4.4b.

For an increased Reynolds number (Re = 400) the velocity distribution and the vertical velocities on the horizontal centreline are presented in Figures 4.5a and 4.5b.

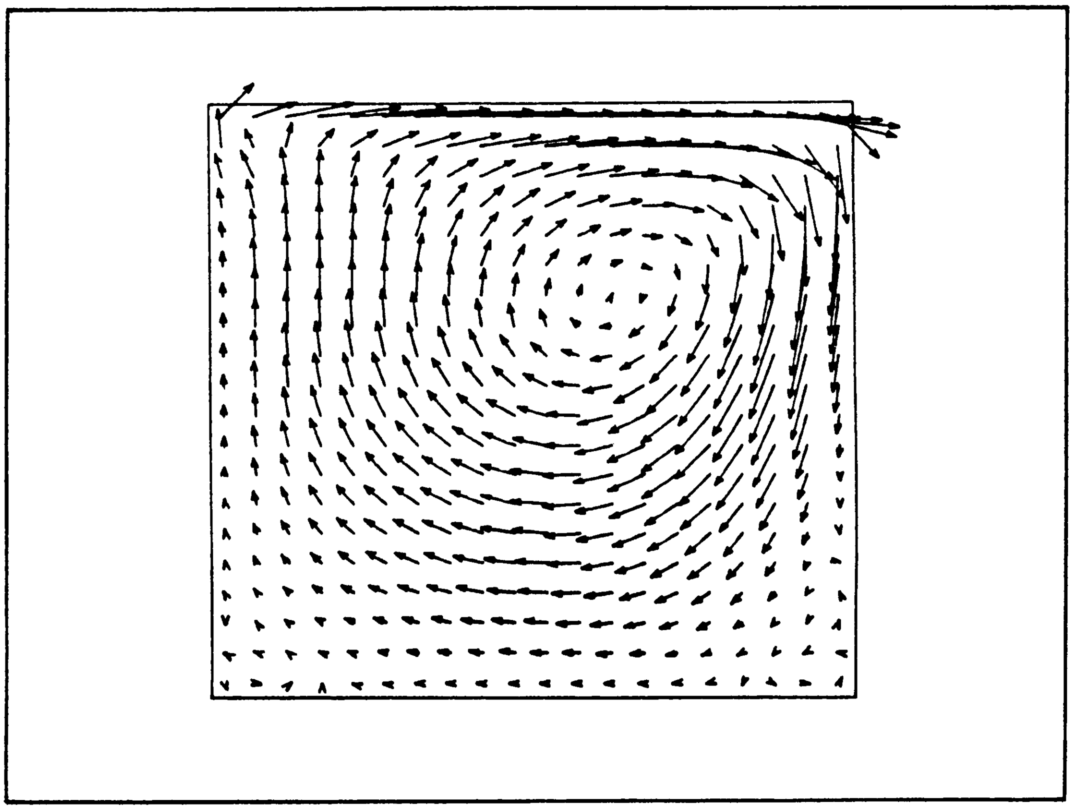

**Figure 4.5a** Driven cavity velocity distribution (Re = 400)

It can be seen that the centre of rotation moves downstream and somewhat away from the moving wall. This numerical solution is again found to be in good agreement with an accepted numerical solution presented by Al–Sanea, Pun and Spalding [49]. The good agreement between the results, suggests that the code is accurate in predicting two–dimensional laminar flow.

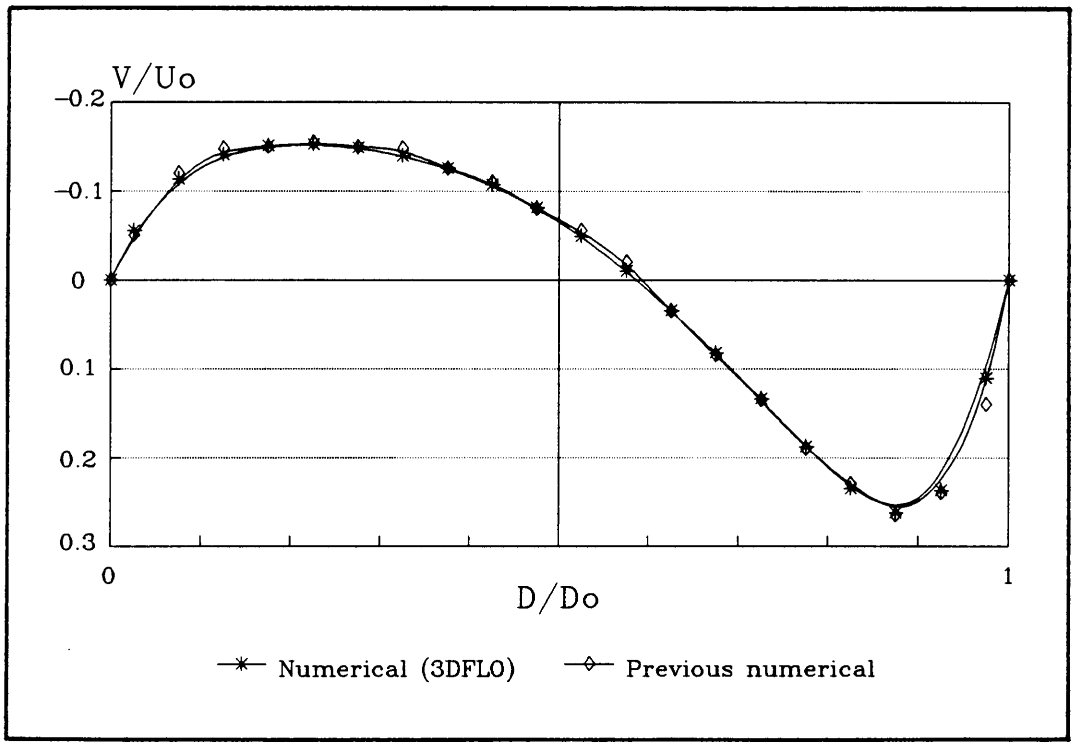

**Figure 4.5b** Driven cavity centreline velocity (Re = 400)

## 4.3 Three–dimensional laminar flow (Phase II)

As a next step in the development process, the code is the extended to model three–dimensional laminar flow. Verification of this phase is done by considering the problem of three–dimensional laminar recirculating flow over a backward facing step.

A schematic representation of the problem is given below in Figure 4.6. Flow enters the working section between two horizontal and two vertical plates and flows across a vertical step with a height of one third of the total distance between the top and bottom plates. The horizontal plates are 45 mm apart while the working section has a width of 305 mm between the vertical plates. At the step, the flow breaks away from the wall and forms a zone of recirculating flow behind the step. At some distance downstream of the step, the flow re–attaches to the bottom plate and eventually results in a fully developed velocity profile.

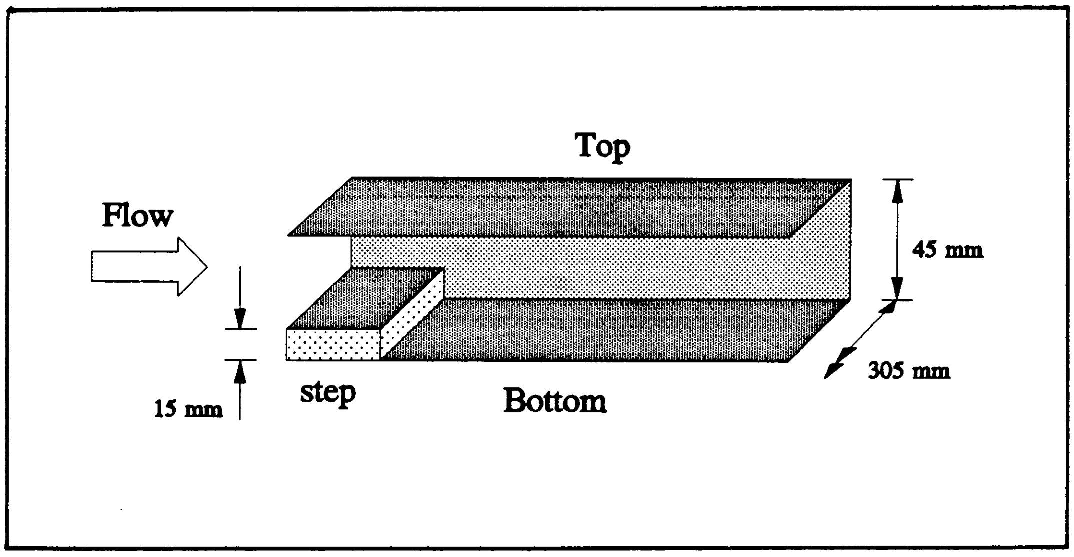

**Figure 4.6** Flow across a backward facing step

The modelled section shown above corresponds to the section which was used by Denham and Patrick [50] for experimental measurements. The comparison between the predicted and the measured velocities at different locations behind the backstep is shown in Figure 4.7. The numerical as well as the experimental results were obtained at a Reynolds number of 229, referenced to the height of the step.

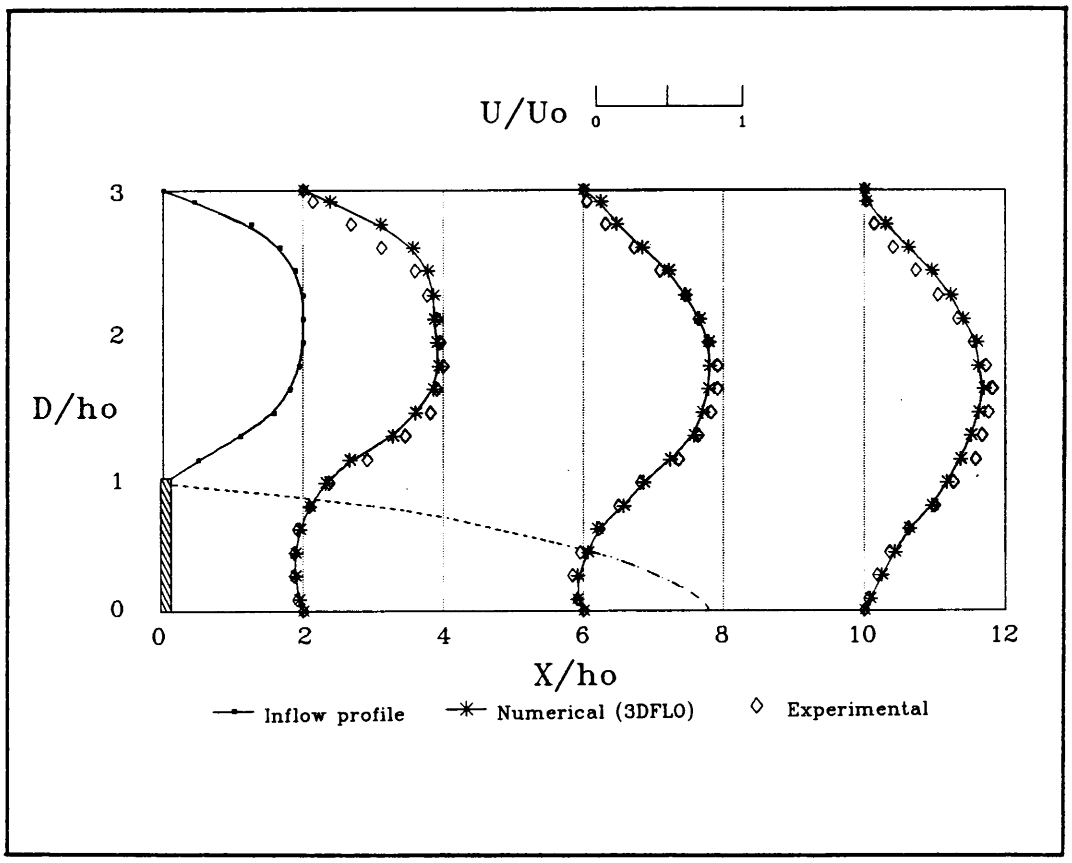

**Figure 4.7** Velocity profiles behind a backward facing step (Re = 229)

The predicted stream–wise velocity profiles at various distances behind the step closely match the measured values. The predicted re–attachment length of approximately eight times the step height ($8h_o$) does however differ from the measured re–attachment length ($9h_o$). This could be attributed to the discretization process, where partial derivatives are replaced by finite differences. The differencing scheme that is used in this process, affects the accuracy of the final solution, particularly in areas where recirculating flow occurs. The predicted re–attachment length does, however, compare favourably with a previous numerical prediction for this application by Visser [24].

Considering the above discrepancy between numerical and experimental results, it is interesting to evaluate the effect of various differencing schemes on the accuracy of the solution. Five different schemes (See Appendix D) as described by Patankar [5], are compared by evaluating the numerical results obtained in each case to analytical or experimental solutions. For two–dimensional flow between parallel plates, the resulting velocity profiles obtained by implementing each of the differencing techniques in turn, hardly varies at all. In the case of modelling two–dimensional laminar recirculating flow over a backward facing step, meaningful differences in accuracy is observed. This can be seen by comparing the flow re–attachment lengths behind the step.

| Differencing Scheme | Upwind | Central | Exponential | Hybrid | Power law | Experimental |
|---|---|---|---|---|---|---|
| Re-attachment length | $6{,}8h_o$ | $6{,}5h_o$ | $7h_o$ | $8h_o$ | $8h_o$ | $9h_o$ |

**Table 4.1** Influence of differencing scheme on re–attachment length

In Table 4.1 above, the predicted lengths as well as the experimentally measured distance at a Reynolds number of 229, are presented. It is shown that the Power law and the Hybrid differencing schemes provide superior accuracy compared to centered and upwind differencing schemes. This coincides with Patankar's theoretical comparison between these methods. Throughout the rest of this study, the Power law scheme is therefore employed.

## 4.4 Three–dimensional turbulent flow (Phase III)

Very few fluid flow applications in engineering practice can be adequately represented by laminar flow. In order to model turbulent fluid flow, a turbulence closure model is required. The k–$\epsilon$ turbulence model, as discussed in the previous chapters, is used for this purpose. Before applying the code to the simulation of full–scale atmospheric turbulence, it is necessary to verify the accuracy of the code in the prediction of turbulent flows. This is done by validating the numerical predictions against experimental results for cases of turbulent flow. The problem considered for this purpose, is the solution of turbulent flow over a backward facing step.

The current flow conditions differ from the laminar problem, addressed in Section 4.3, in that the Reynolds number (Re) is increased to 3025. This Reynolds number lies, however, close to the transition region from laminar to turbulent flow. At this point, the flow becomes extremely sensitive to small disturbances resulting from experimental measurements. For this reason Denham et al. [51] made use of laser anemometry to measure the velocity distribution in turbulent flow over a backstep.

Measured velocity profiles published by Denham et al. [51] were used to prescribe inflow velocities for the numerical simulation. Based upon this input, the complete velocity distribution behind the step is calculated. In Figure 4.8, the numerical solution is compared to the measured data at various distances behind the step.

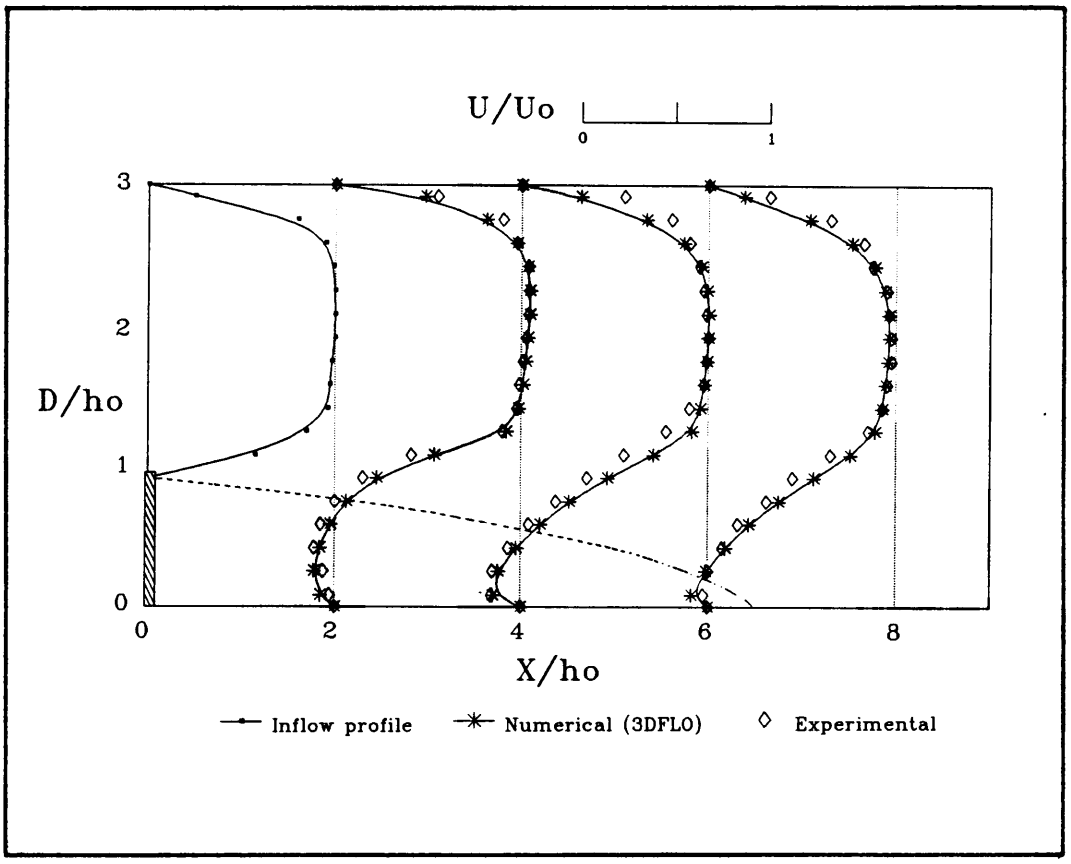

**Figure 4.8** Velocity profiles behind a backward facing step (Re = 3025)

It can be seen that the numerically predicted values compare favourably with the measured values at corresponding locations behind the step. The predicted reattachment length of 6,4$h_0$ is also in good agreement with the measured length of 7$h_0$. Two possible reasons for this difference can be noted. The first being the differencing scheme applied during discretization which affects the accuracy of the solution in recirculation zones. Secondly, the turbulence model may be inadequate for the Reynolds numbers which lie so close to the transitional region. The k–$\epsilon$ turbulence model applies specifically to fully turbulent flows and numerical predictions in the transitional region is known to be inaccurate. For engineering application, the results do, however, prove to be sufficiently accurate.

After this series of verifications, it can now be concluded that the numerical model is able to predict fully turbulent recirculating fluid flow with acceptable accuracy. The code will now be used to simulate atmospheric air motion around arbitrary shaped buildings.

## 4.5 Wind flow over buildings

Complete knowledge of the velocity fields and pressure distribution surrounding buildings are of great importance to the engineer and the architect. This information can be obtained by simulating the complete problem with the use of a computer code. Many workers contributed to improving the understanding of wind flow patterns around buildings by developing computer simulation codes. These methods were, however, all limited in the extent to which arbitrary geometries conformed to the orthogonal co–ordinate systems they employed. The exact geometry of an inclined house roof and a sharp ridge can only be modelled with the use of a non–orthogonal curvilinear grid reference system. In this section three–dimensional atmospheric air motion around arbitrary shaped buildings are simulated by applying appropriate curvilinear co–ordinates.

The first building to be considered is an ordinary pitched roof building with a roof set at an incline of $26^o$, as shown in Figure 4.9. The total span width of the house is 6,4 m and it has a length of 21,3 m. The ridge of the roof is at a height of 3,9 m. It should be noted that due to the symmetrical properties of the problem, only half of the entire flow field is modelled . This building is selected because full–scale pressure measurements are available in the literature [52], and previous two–dimensional numerical simulations had already been carried out [18]. In determining the inflow boundary layer profile a mean wind speed exponent of 0,15 is used to correspond with the measurements.

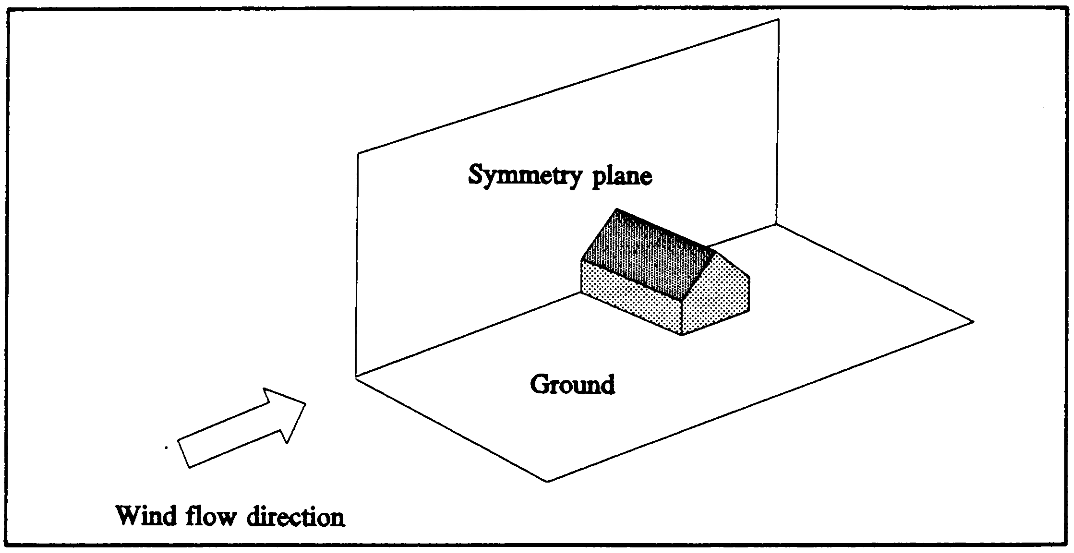

**Figure 4.9** Single span pitched roof building

A two–dimensional cross–section of the three–dimensional grid, used in modelling the problem, is shown in Figure 4.10. In order to maintain simplicity, the largest part of the flow region is covered with rectangular control volumes. Above the roof of the building, the elements are, however, non–orthogonal which makes it possible to represent the exact physical geometry of the inclined roof. In the close vicinity of the building the elements are smaller in order to improve the accuracy with which the complicated flow in these areas can be calculated. The elements gradually increase in size as the distance from the building increases. Behind the building, the diagonal grid lines are extended beyond the leeward wall which leads to better simulation of the flow behaviour on the leeward side of the ridge.

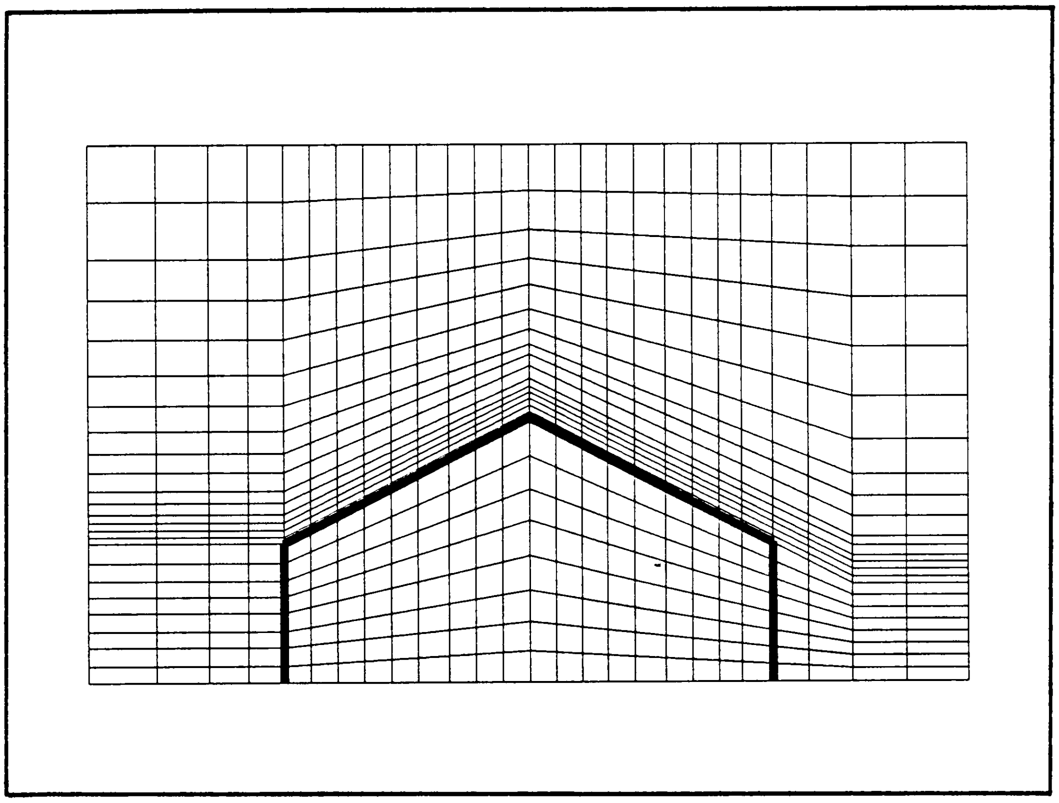

Figure 4.10 Cross—section of 3D grid around a single span building

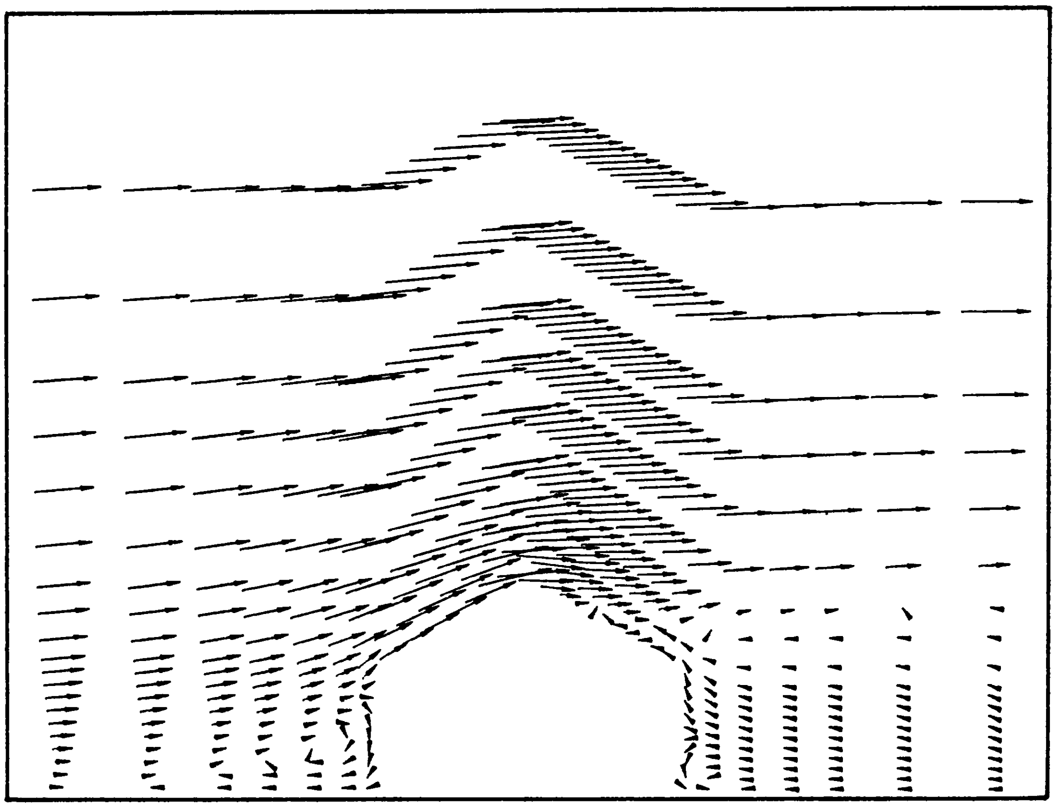

Figure 4.11 ZY—plane velocity distribution on symmetry plane

The complete velocity distribution on the symmetry plane surrounding the building is presented in Figure 4.11. Atmospheric air approaches the building according to prescribed boundary layer profiles. As the air hits the windward wall of the building a recirculation zone is formed. This forces the flow upward onto the roof of the building. Due to the control volume grid arrangement above the building, the figure might suggest that the flow goes up and then down again as it passes the ridge. A careful study of the velocity vectors does, however, show that this is not the case and that only an upward deflection is indicated. At the ridge of the roof, the flow breaks away from the surface and forms a large zone of primary recirculation behind the building. Smaller zones of secondary recirculation are also observed behind the ridge and the leeward wall. The effect of this flow pattern on the pressures around the building is reflected in the next paragraph.

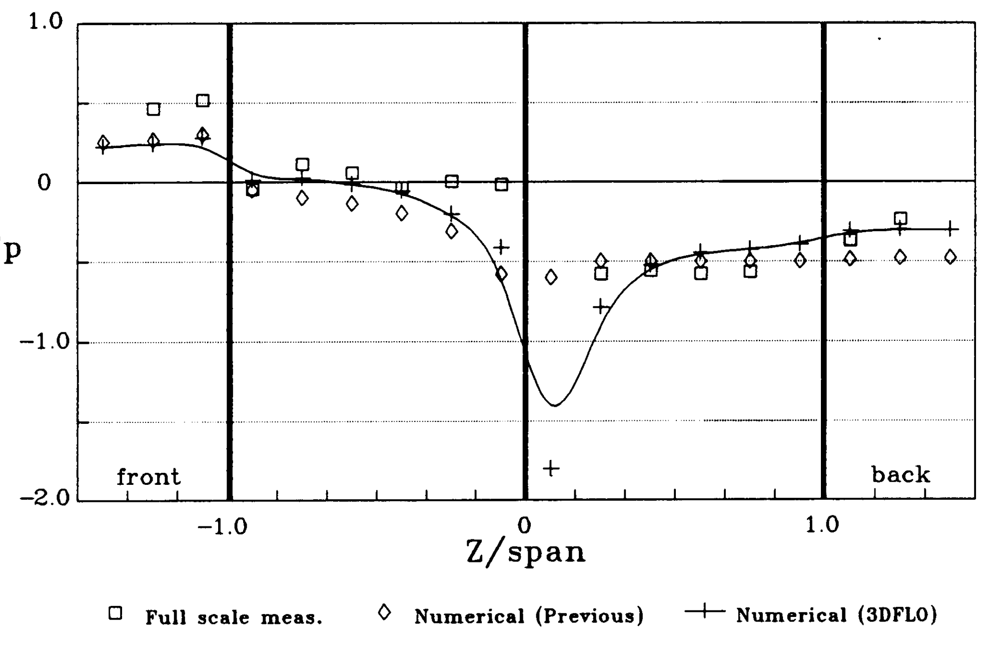

Figure 4.12 Single span pressure coefficients (Middle)

The numerical results obtained from the current model are compared with full–scale experimental measurements by Wells and Hoxey [52], and presented graphically in Figure 4.12. The figure shows pressure coefficients on the middle section (symmetry plane) of the building in Figure 4.9.

The graph is presented in dimensionless form of pressure coefficient versus span–wise position. The pressure coefficient $C_p$ is calculated from Equation 4.1, where P is the pressure on the surface of the building , $P_{ref}$ is the upstream pressure at ridge height and $V_{ref}$ is the upstream velocity also at ridge height.

$$C_p = \frac{P - P_{ref}}{\rho V^2_{ref}/2} \tag{4.1}$$

In order to present the pressures on the walls of the buildings on the same figures, they are included as values less than –1 (windward wall) and values greater than 1 (leeward wall) of the dimensionless span–wise position Z/Span. Z is the distance measured in either direction from the ridge while the Span represents the total distance between the building edge and the ridge in the middle of the section.

It can be seen from the figure above, that the pressure distributions on the windward section of the roof were predicted more accurately than those from previous work. It is, however, noted that towards the ridge, the current predicted values drift slightly away from the full–scale measurements. This can be explained by considering the interdependence between pressures and velocities in the numerical model. Point velocities are calculated by including the effect of pressure differences between neighbouring points while pressure corrections are, in turn, calculated using interpolated velocity values on the cell walls. These velocities are interpolated by applying pressure differences between consecutive grid points. The low pressure spike, occurring behind the ridge, therefore, has a lowering effect on the upstream pressures. Theoretically this is not incorrect, and the problem can only be overcome by excessive grid refinement. This would limit the effect to the close vicinity of the ridge, to correspond with expected practical effects.

On the leeward side, the previous prediction, the measured values and the current prediction of the pressure coefficients differ only slightly. The current model indicates the occurrence of a low pressure spike immediately behind the ridge of the roof. This is due to the break–away of flow at the ridge. The resulting suction leads to flow recirculation, and hence a low pressure spike. Visser et al.[38] also indicated the occurrence of such a spike while numerical predictions by Crosby [18] did not. In neither of these two cases, where cartesian co–ordinates were used, could the exact geometry of the roof ridge be modelled as accurately as in the current model and the results, therefore, depend largely on the staggering of grid points to approximate the ridge geometry. Although a full–scale measurement at this point is unfortunately not available, it is the opinion of the author that the current numerical result can be accepted with confidence.

The advantage of three–dimensional modelling is that it enables the study of atmospheric air flow behaviour at the building edge where flow is strongly influenced by three–dimensional effects. The complete velocity distribution on a ZY–plane at the building edge is shown in Figure 4.13.

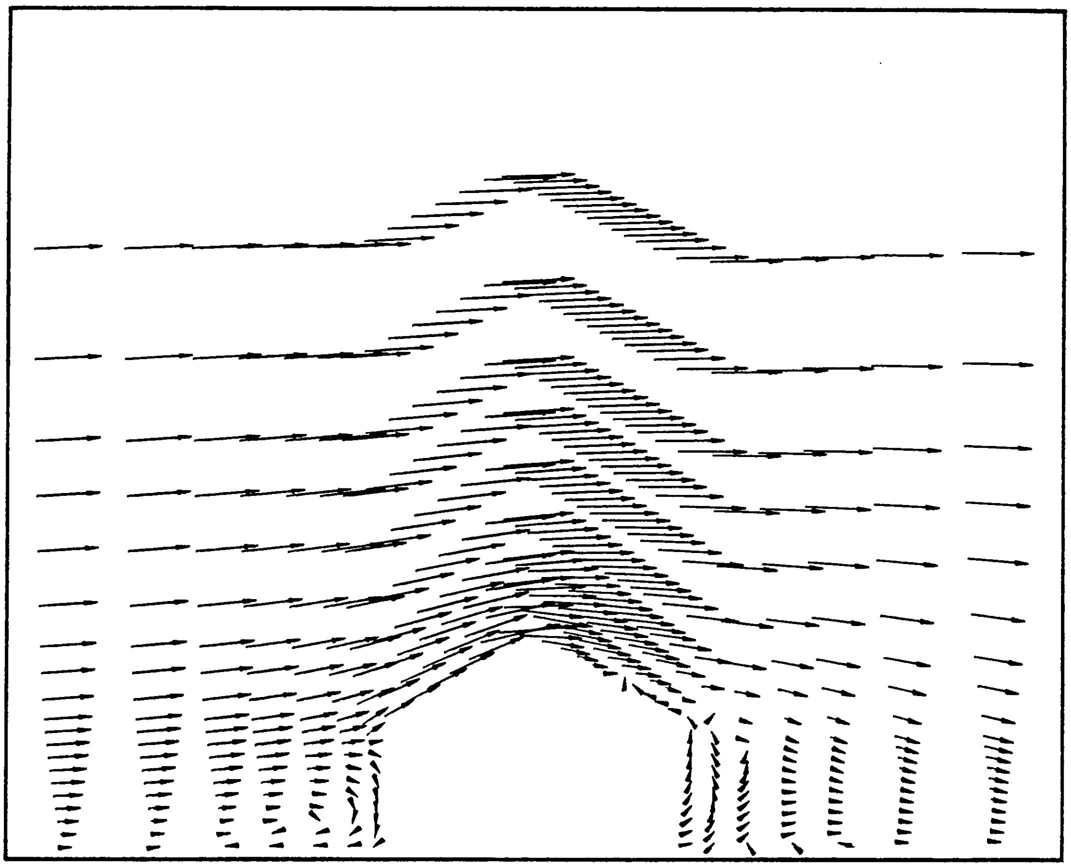

**Figure 4.13** ZY–plane velocity distribution at building edge

The velocity field differs from the field in Figure 4.11 in that a recirculation zone behind the building is not present. At this point, the flow is dominated by flow break–away and recirculation in the third dimension.

The numerical results at the building edge can be evaluated quantitatively by comparing them with full–scale measurements. The Figure 4.14 shows pressure coefficient versus span–wise position (Z/Span) on a ZY–plane at the edge of the building.

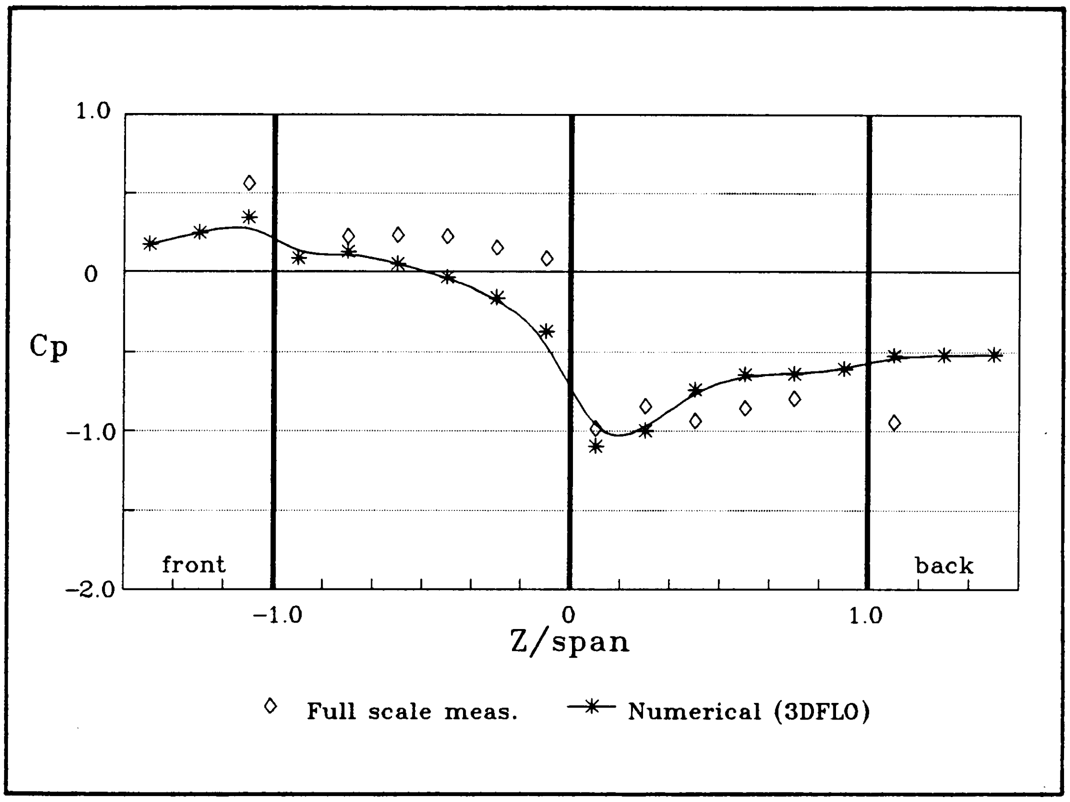

**Figure 4.14** Single span pressure coefficients (Building edge)

It can be seen that the current numerical prediction correlates favourably with the full–scale measurements in absolute value as well as in trend. The slight variation may be ascribed to the insufficient modelling of turbulence at the edge where the flow is highly sensitive and unsteady. One limitation affecting the three–dimensional modelling, is that due to limited computer memory, the grid size cannot be refined indefinitely. This factor may contribute to the slight discrepancies observed. On the whole, the results still remain acceptable for most engineering applications.

The second building to be considered, is shown below in Figure 4.15. The building has a double span pitched roof with the all inclines set at angles of 26$^o$. The span width is 12,8 m and the building has a length of 39,6 m. The ridge is 7,1 m high, the valley 4,25 m and the eave stands at height of 3,4 m.

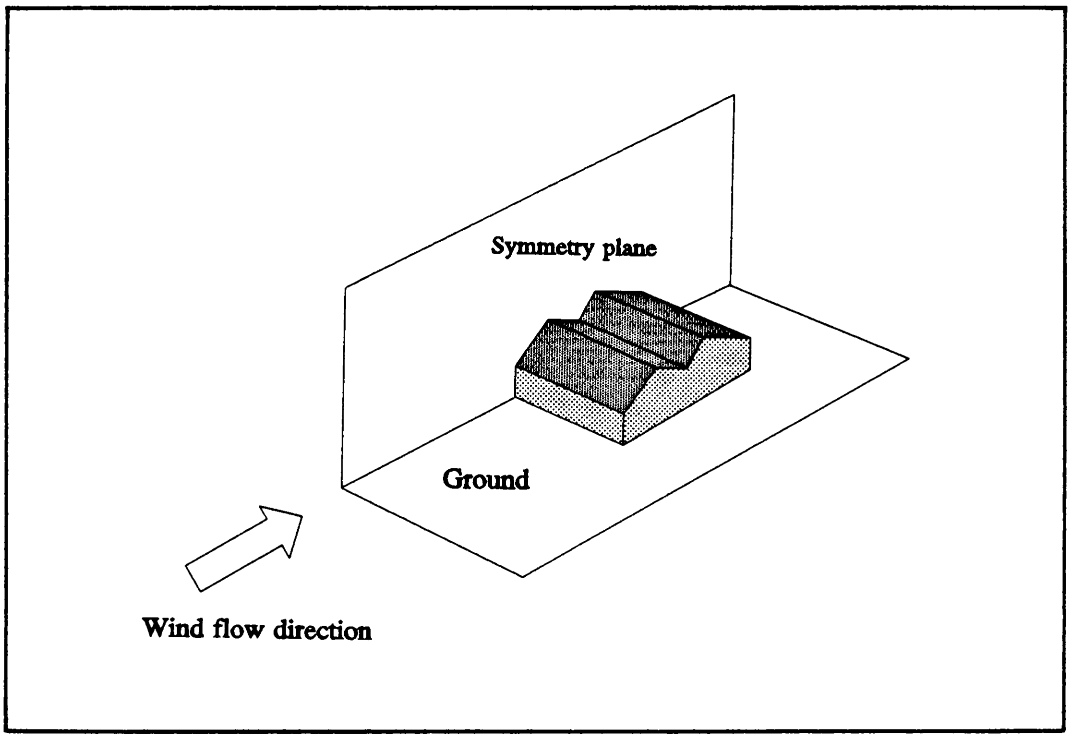

**Figure 4.15** Double span pitched roof building

A literature survey provided no evidence of previous numerical simulations on a building of this shape. This motivated the use of the current code to model such a complex building in an attempt to establish the accuracy and applicability of the current model to practical engineering fluid flow problems. Due to symmetry only half of the problem is modelled, as is shown in Figure 4.15 above. A zero gradient symmetry boundary condition, as discussed in Chapter 4, is imposed on the symmetry plane. The inflow boundary layer profiles where determined, as explained in Section 3.5.2, by using a value of 0,21 for the mean wind speed exponent ($\alpha$). This corresponds to conditions under which Wells et al. [52] did full–scale experimental measurements on such a building.

A cross–section of the three–dimensional grid generated for this problem, is shown below in Figure 4.16. Once again the grid is cartesian, except above the building, where boundary fitting non–orthogonal control volume elements are used. The finite volume cells are small near the building to enable accurate modelling of boundary layer flow on the building walls. They increase in size at larger distances from the building where the flow is less complex.

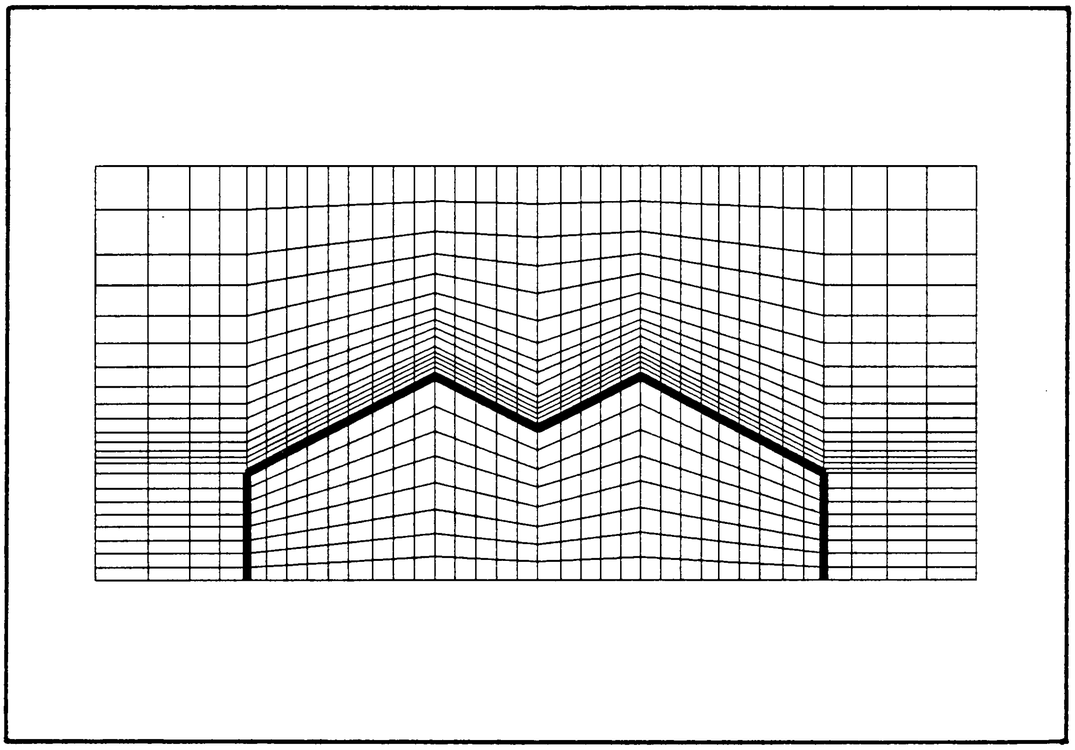

**Figure 4.16** Cross–section of 3D grid around double span building

In Figure 4.17, the predicted symmetry plane velocity distribution surrounding the building is presented. Similar to the flow field around the previous single span building, zones of recirculating flow are observed in front of and behind the building, with a small zone of secondary recirculation right behind the leeward wall. An additional zone of recirculation is identified in the valley between the ridges of the roof. This is a result of flow break–away at the first ridge. The break–away of flow at the second ridge leads to the formation of the large zone of primary recirculation.

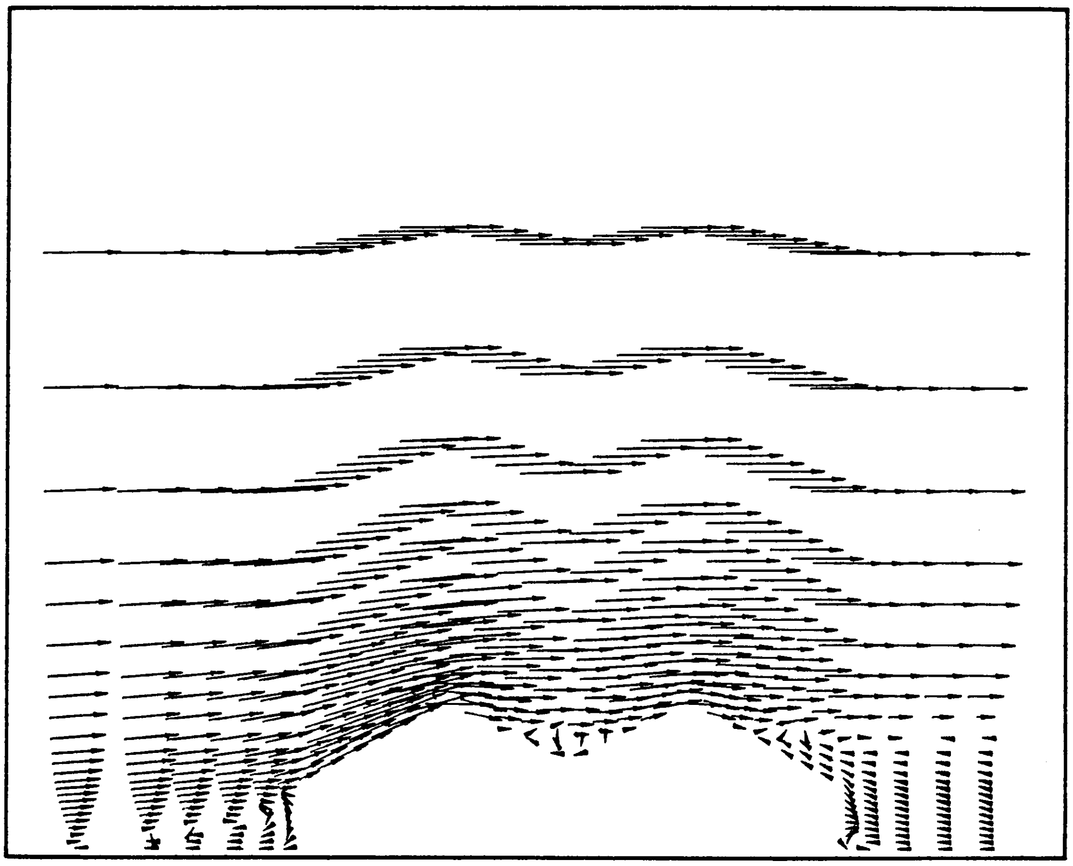

**Figure 4.17** Symmetry plane velocity distribution around double span building

The numerical prediction is quantitatively verified by comparing the pressures around the building to the full–scale experimental measurements of Wells and Hoxey [52]. Figure 4.18 presents a graph of pressure coefficient ($C_p$) versus span–wise position (Z/Span). The pressure coefficients are calculated by using Equation 4.1 with $V_{ref}$ and $P_{ref}$ obtained at a ridge height of 7,1 m. Pressures on windward and leeward walls are included in the graph at span–wise values of less that –1 and greater than 1 respectively.

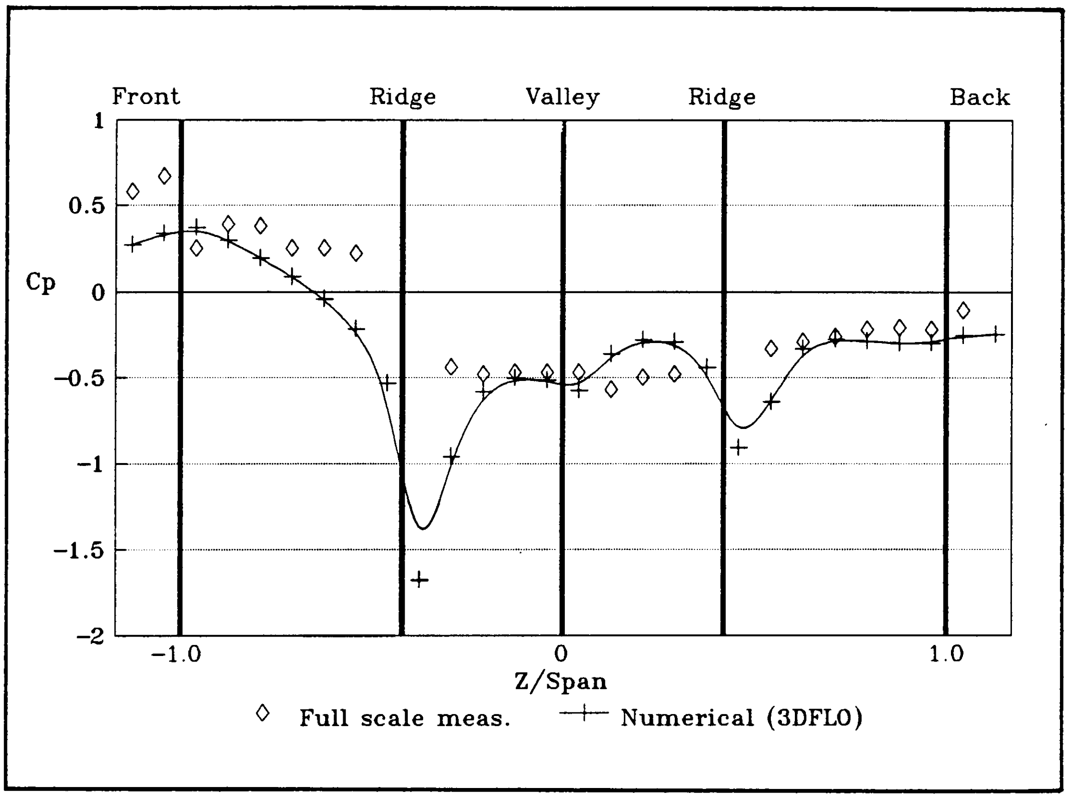

**Figure 4.18** Double span pressure coefficients (symmetry plane)

The figure shows that the current numerical predictions are in good agreement with available full–scale measurements. Two outstanding aspects noted from the figure, are the occurrences of low pressure spikes behind the ridges. As explained earlier, this is a result of flow break–away at the ridge, which has the effect of suction on the roof. Unfortunately experimental measurements at these points are not available. Wells and Hoxey [52] do not give a reason for the absence of a measurement, but it is believed that it might be a result of highly unsteady experimental readings due to repeated flow de–attachment and re–attachment at these points. As explained previously, the low pressure following the first ridge is, once again, responsible for the discrepancy between the measured and predicted pressures on the windward positive incline towards the ridge.

Considering the valley section of the roof (between ridges), it can be seen that the pressures remain relatively low and steady due to the existence of a recirculation zone. From the low pressure coefficients on the second incline towards the second ridge, it is evident that the flow is not allowed to re–attach to the roof. Therefore the break–away at the second ridge is not so abrupt and clearly defined, and, therefore, the low pressure spike behind the second ridge is not as intense as before. On the leeward decline towards the end of the building, the numerical prediction in the recirculation zone closely corresponds to the full–scale measurements for that section.

By comparing the absolute values of the pressure coefficients as well as the trends, overall good agreement between the numerical prediction and the full–scale measurements is obtained.

## 4.6 Summary

Before any computer code can be applied with confidence to practical problems, it has to be verified by comparing the predicted results with analytical solutions, experimental measurements or accepted numerical results. In this chapter, various fluid flow test cases were modelled to test the accuracy of the current numerical model at different phases throughout the development process. The results provided in this chapter presented good agreement in all of the case studies considered.

The model was then applied to the simulation of wind flow over different buildings. Complete turbulent atmospheric boundary layer flow was simulated in three dimensions with the use of boundary–fitted grids. It included the simulation of complete three–dimensional turbulent recirculation, which made it possible to determine the influence of end effects on the wind loads. The predictions were compared to full–scale measurements and the model was shown to be superior to previous two–dimensional cartesian approximations for this purpose.

# CHAPTER 5 CONCLUSIONS

## 5.1 Synopsis

The steady increase in computer capabilities, combined with the growing need for highly specialized engineering technology, present many opportunities for the use of numerical methods today. Particularly in the field of Fluid Dynamics, numerical methods are becoming exceedingly popular for the solution of practical fluid flow problems. Many researchers developed theoretical models describing different aspects of fluid flow behaviour. The need was recognized to inclüde these models in the development of a computer code that is generally applicable to most fluid flow problems in nature and in industry.

The current model is based on a finite volume numerical method for the solution of the partial differential equations describing three–dimensional turbulent flow. These equations include the full Navier–Stokes equations, the continuity equation and two turbulence equations of the $k$–$\epsilon$ turbulence model. In order to enable the solution of flow in regions of arbitrary geometrical shape, the equations are transformed to general curvilinear co–ordinates. The equations are discretized and linearized and are solved simultaneously by an iterative process. The solution algorithm based on the SIMPLE method and employs a specific interpolation scheme to ensure strong pressure–velocity coupling. The computer code 3DFLO provides complete information on the velocities, pressures and turbulence quantities as final results.

The model has been validated by extensive verification of the predicted results against analytical, experimental and numerical data at each stage of the development process. The method was then applied to the modelling of atmospheric air motion over and around angularly shaped buildings. The numerical predictions compared favourably with available full–scale measurements of pressure distributions on pitched roof buildings. The main conclusions drawn from the study, as well as the main contributions of the study, is presented in the following section.

## 5.2 <u>Conclusions and Contributions</u>

This study leads to a number of conclusions regarding the development of the numerical model. Each of these will be noted and discussed in this section.

* A comparison of the numerical results, obtained by applying a number of different finite differencing schemes, shows that the Power Law scheme as proposed by Patankar [5], provides numerical results of superior accuracy. Particularly where recirculating flow is

considered, a significant improvement is observed when using this scheme. The difference in accuracy is less severe when a simple channel flow is considered. This numerical finding agrees with Patankar's analytical comparison between the schemes.

* During the development of the model, two similar though different solution algorithms were applied for the segregated solution of pressures and velocities. The difference between the two methods (SIMPLE and SIMPLEC) are discussed in Section 4.5. A comparison using identical differencing schemes and convergence parameters, indicates an acceleration in convergence using the SIMPLEC algorithm. An increase in solution efficiency of around 30% is obtained for all fluid flow problems considered.

* The good agreement between the experimental results and the numerical prediction for various bench–mark test cases, indicate the accuracy of the code 3DFLO in predicting laminar and turbulent fluid flow in three dimensions. In all cases considered, the accuracy of the numerical results coincides with the accuracies of previous numerical models.

* Close correlation of the numerical predictions with full–scale measurements of pressure coefficients surrounding different buildings, further confirms the ability of the code to model flow in and around arbitrary curvilinear geometries.

*  Numerical results of pressure fields surrounding the buildings show that complete three–dimensional modelling provide better results than previous two–dimensional approximations. This is due to a more complete simulation of the real physical problem, including three–dimensional effects.

The application of the computer code 3DFLO to the simulation of wind flow around arbitrary shaped buildings leads to a number of contributions which are outlined below.

*  For the first time, atmospheric boundary layer flow over buildings was modelled in three dimensions using curvilinear boundary conforming grids.

*  The use of boundary fitted grids in precisely representing inclines and declines on building roofs leads to a more accurate prediction of pressure coefficients and resulting wind loads on pitched roof buildings.

*  The ability to exactly model the sharp ridge geometry on building roofs, leads to an improved understanding of flow phenomena in the near vicinity of the ridge. The existence of a low pressure spike immediately behind the ridge is confirmed.

*  Three–dimensional modelling made it possible to study wind loading on pitched roof building edges. For the first time pressure

distributions at building edges (in the third dimension) were numerically predicted with accepted accuracy.

* The effect of wind flow around multispan pitched roof buildings were modelled successfully for the first time. The results indicate that three–dimensional effects strongly influence flow patterns in the downstream direction behind the first ridge.

* Throughout the course of this study, the wind approach angle was constant (right form the front of the building). With the current model it is now possible to vary the direction of wind flow across the buildings in the computer simulations to correspond with reality.

## 5.3 Recommendations for further research

During the course of this study several areas justifying further research were identified. These are briefly outlined below.

* A great need exists for the combination of a numerical grid generation scheme with the current flow model in order to improve the userfriendliness of the code and avoid tiresome file transferring.

* An even greater need exists for the development and implementation of a more effective numerical solver for three–dimensional purposes to replace the current TDMA–solver. This would improve the efficiency of the code by decreasing and economizing on computer running time.

* The energy equation can be incorporated into the current model to enable the solution of fully compressible flows. The model would then be able to solve supersonic flows that is characteristic of aeronautical applications.

* Special attention should be given to the prescription of outflow boundary conditions where the outflow surface is not perpendicular to one of the cartesian velocity components. A problem arises with the enforcement of continuity across angular control volumes at outflow boundaries. This prevents complete convergence of the numerical solution.

* It will be profitable to extend the model to make use of unstructured grids especially where flow around buildings are considered. This would enable the grid to be dense in areas around the building but course at large distances form the buildings where flow is less complex. In such a way the number of grid points can be greatly decreased, resulting in more efficient computer modelling.

* The use of adaptive grids should be investigated for extending the model to the solution of free surface flows. It would also increase the accuracy of the solution in high gradient areas.

* The modelling of flow over arbitrary shaped buildings with variable wind approach angle should be investigated.

# NOMENCLATURE

**Roman Alphabet**

| | |
|---|---|
| $\mathbf{a}_i$ | Covariant base vector |
| a | Discretization equation coefficients |
| b | Discretization equation source term |
| $B^\phi, C^\phi, D^\phi$ | Pressure correction coefficients for variable $\phi$ |
| $C_1, C_2, C_\mu$ | Turbulence constants |
| $C_D$ | Constant for atmospheric boundary layer profile |
| **E,F,G,U** | General vectors defined in Appendix B |
| I | Turbulence intensity |
| J | Jacobian of the transformation |
| k | Kinetic energy of turbulence |
| $k_s$ | Surface roughness parameter |
| **r** | 3D space vector made up by cartesian unit vectors |
| P | Point pressures |
| $h_{ref}$ | Reference height for boundary layer profiles |
| L | Length scale values for atmospheric turbulence |
| u | X–direction velocity component |
| v | Y–direction velocity component |
| w | Z–direction velocity component |
| $V_{ref}$ | Reference velocity |
| $v_t$ | Velocity tangential to solid wall |
| $S^\phi$ | Source term in $\phi$–equation |
| t | Time |
| $y^+$ | Local Reynolds number |

| | |
|---|---|
| $y_p$ | Perpendicular distance from the wall to the node point |
| x,y,z | Cartesian three–dimensional space co–ordinates |

**Greek Alphabet**

| | |
|---|---|
| $\alpha$ | Mean wind speed exponent |
| $\epsilon$ | Turbulence dissipation rate |
| $\rho$ | Density |
| $\mu$ | Fluid viscosity |
| $\tau_w$ | Wall shear stress |
| $\kappa$ | Von Karman constant |
| $\boldsymbol{v}$ | Three–dimensional velocity vector |
| $\phi$ | Representing any of the dependant scalar variables |
| $\boldsymbol{\phi}$ | Representing any 3D vector |
| $\Delta$ | Difference between consecutive grid nodes |
| $\nabla$ | Del operator |
| $\Gamma$ | Diffusion coefficient |
| $\xi,\eta,\zeta$ | General curvilinear three–dimensional space co–ordinates |
| $\sigma_k,\sigma_\epsilon$ | Turbulence constants |
| $\frac{\partial}{\partial\theta}$ | Partial derivatives to any variable $\theta$ |

**Superscripts**

| | |
|---|---|
| i | Equals 1,2 or 3 for each of the general co–ordinate axes |
| ' | Indicates fluctuating value due to small scale turbulence |
| — | Indicates average value |
| o | Value at previous time step |

| | |
|---|---|
| * | Approximate values (guessed) |
| '' | Corrections |

## Subscripts

| | |
|---|---|
| i | Equals 1,2 or 3 for each of the general co–ordinate axes |
| $\xi^i$ | Partial derivative to $\xi^i$ |
| eff | Indicates effective value |
| lam | Indicates laminar flow |
| t | Indicates turbulent flow |
| P,p | Grid point under consideration |
| e,w,n,s,t,b | Indicates cell wall positions |
| E,W,N,S,T,B | Indicates neighbouring grid points |

# REFERENCES

1. **ANDERSON, D.A., TANNEHILL, J.C. and PLETCHER, R.H.;** *Computational Fluid Mechanics and Heat Transfer,* Hemisphere Publishing Corporation, USA, 1984.

2. **RICHARDSON, L.F.;** *The Approximate Arithmetical Solution by Finite Difference of Physical Problems Involving Differential Equations, with an Application to the Stresses in a Masonry Dam,* Philos. Trans., Royal Society, London, Ser. A, Vol. 210, pp 307–357.

3 **SOUTHWELL, R.V.;** *Relaxation Methods in Engineering Science,* Oxford University Press, London, 1940.

4. **O'BRIEN, G.G., HYMAN, M.A. and KAPLAN, S.;** *A Study of the Numerical Solution of Partial Differential Equations,* Journal of Math. Phys., Vol. 29, pp 223–251, 1950.

5. **PATANKAR, S.V.;** *Numerical Heat Transfer and Fluid Flow,* Hemisphere Publishing CorpOration, New York, 1980.

6. **LAUNDER, B.E. and SPALDING, D.B.;** *Lectures in Mathematical Models of Turbulence, Academic Press,* London, 1972.

7. **KWAK, D. and CHAKRAVARTHY, S.R.;** *A Three–Dimensional Incompressible Navier–Stokes Flow Solver Using Primitive Variables,* AIAA Journal, Vol. 24, No. 3, pp 390–396, 1985.

8. **WENQUAN, W.U. and HAOYO, Y.U.**; *General Curved Surface Fitting and Calculation of Flow Along Arbitrarily Twisted Stream Surface,* ASME Journal, 85–GT–97, 1985.

9. **CUNSOLO, D. and ORLANDI, P.**; *Accuracy in Non Orthogonal Grid Reference Systems,* Aerodynamics Institute of Rome, Italy, 1978.

10. **GREYVENSTEIN, G.P.**; *Snelheidsdruk Numeriese Metode vir die Berekening van Tweedimensionele Elliptiese Vloei,* Ph.D. Dissertation, University of Pretoria, 1981.

11. **RAITHBY, G.D., GALPIN, P.F. and VAN DOORNMAAL, J.P.**; *Prediction of Heat and Fluid Flow in Complex Geometries using General Orthogonal Coordinates,* Numerical Heat Transfer, Vol. 9, pp 125–142, 1986.

12. **RAMACHANDRA, V. and SPALDING, D.B.**; *A Non–orthogonal Finite Difference Formulation for Three–dimensional Duct Flows,* HTS/78/1, Imperial College, Feb.1978.

13 **SWANSON, B.W.**; *Solutions of the Three–Dimensional Equations in Non–Orthogonal Coordinates to Calculate the Flow in a Log Spiral Impeller,* ASME Journal, 82–GT–268, 1982.

14. **ANDO, Y., FUJIMORI, T., TOH, H., KAWAI, M., MASUKO, A. and MIYAMAE, S.**; *Development of Three–Dimensional Numerical Analysis Code (VEGA–3) for Turbulent Flow Field with Heat/Mass Transfer Phenomena,* IHI Engineering Review, Vol.22, No. 3, 1989

15. **RHIE, C.M. and CHOW, W.L.**; *Numerical Study of the Turbulent Flow Past an Airfoil with Trailing Edge Separation,* AIAA Journal, 22 Nov., 1983.

16. **THOMPSON, J.F., WARSI, Z.U.A. and MASTIN, C.W.**; *Numerical Grid Generation Foundations and Applications,* Elsevier Science Publishing Co. Inc.,New York, 1985.

17. **LE GRANGE, L.A.**; *Numerical Simulation of Polymer Melt Flow in a Mould using a Boundary–Fitted Coordinate System,* Thesis for Master in Engineering, PU for CHE, Vanderbijlpark, 1990.

18. **CROSBY, C.P.**; *The Numerical Prediction of Airflow through and around Permeable Windbreaks and Buildings,* Dissertation presented for Master's degree, University of Pretoria, 1989.

19. **LANDAHL, M.T. and MOLLO–CHRISTENSEN, E.**; *Turbulence and Random Processes in Fluid Mechanics,* Cambridge University Press, Cambridge, 1986.

20. **BOUSSINESQ, J.**; *Essai Sur La Theorie Des Eaux Courantes,* Mem. Presentes Acad. Sie., Paris, Vol. 23, pp46, 1877

21. **LAUNDER, B.E. and SPALDING,D.B.**; *The Numerical Computation of Turbulent Flows, Computer Methods in Applied Mechanics and Engineering,* pp. 269–289, North–Holland Publishing Company, 1974.

22. **GOSMAN, A.D. and IDERIAH, F.J.K.**; *A General Computer Program for Two–Dimensional, Turbulent,* Recirculating Flows, Imperial College, London, 1976.

23. **BENODEKAR, R.W., GODDARD, A.J.H., GOSMAN, A.D. and ISSA, R.I.**; *Numerical Prediction of Turbulent Flow over Surface Mounted Ribs,* ASME Journal, 83–FE–13, 1983.

24. **VISSER, J.A.**; *Numerical Modelling of Combustion,* Ph.D. Dissertation, University of Pretoria, 1989.

25. **MATHEWS, E.H. and MEYER, J.P.**; *Numerical Modelling of Wind Loading on Film Clad Greenhouse,* Building and Environment, Pergamon Journals Ltd., Great Britain, 1987.

26. **MATHEWS, E.H.**; *Prediction of the Wind Generated Pressure Distribution around Buildings,* Journal of Wind Engineering and Industrial Aerodynamics, 29, Silsoe, UK., 225–233, 1988.

27. **MURAKAMI, S. and MOCHIDA, A.**; *Three–Dimensional Numerical Simulation of Turbulent Flow Around Buildings using the $k-\epsilon$ Turbulence Model,* Building and Environment, Vol. 24, No. 1, pp 51–64, 1989.

28. **PATERSON, D.A. and APELT, C.J.**; *Simulation of Wind Flow Around Three–Dimensional Buildings,* Building and Environment, Vol.24, No. 1, pp 39–50, 1989.

29. **HAINES, A.B.**; *Turbulence Modelling,* Working Party Report, Aeronautical Journal, pp 269–277, Aug./Sept. 1982.

30. **SAVORY, E. and TOY, N.**; *Hemispheres and Hemisphere–Cylinders in Turbulent Boundary Layers,* Journal of Wind Engineering and Industrial Aerodynamics, vol.23, pp 345–364, 1986.

31. **HOXEY, R.P. and RICHARDSON, G.M.**; *Measurements of Wind Loads on Full–Scale Film Plastic Clad Greenhouses,* J.of Wind Eng. and Ind. Aerod., no. 16, pp57–83, 1984.

32. **HOXEY, R.P. and RICHARDSON, G.M.**; *Wind Loads on Film Plastic Greenhouses,* J.of Wind Eng. and Ind. Aerod., no. 11, pp225–237, 1983.

33. **HOXEY, R.P. and POLLARD, P.R.**; *Full–Scale Measurements of Wind Loads on Full–Scale Film Plastic Greenhouses,* NIAE, Departmental Note, G/1003/04025.

34. **ROBERTSON, A.P., HOXEY, R.P. and MORAN, P.**; *A Full–Scale Study of Wind Loads on Agricultural Ridged Conapy Roof Structures and Proposals for Design,* J.of Wind Eng. and Ind. Aerod., no. 21, pp167–205, 1985.

35. **HANSON, T., SMITH, F., SUMMERS, D.M. and WILSON, C.B.;** *Computer Simulation of Wind Flow around Buildings,* Computer Aided Design, Vol. 14, 1982.

36. **HANSON, T., SUMMERS, D.M. and WILSON, C.B.;** Numerical Modelling of Wind Flow over Buildings in Two Dimensions, International Journal for Numerical Methods in Fluids, Vol. 4, pp25–41, 1984.

37. **SUMMERS, D.M., HANSON, T. and WILSON, C.B.;** A Random Vortex Simulation of Wind Flow over a Building, International Journal for Numerical Methods in Fluids, Vol. 5, pp849–871, 1985.

38. **MATHEWS, E.H., CROSBY, C.P., VISSER, J.A. and MEYER, J.P.;** Numerical Prediction of Wind Loads on Buildings, J.of Wind Eng. and Ind. Aerod., no. 31, pp241–250, 1988.

39. **HANSON, T., SUMMERS, D.M. and WILSON, C.B.;** A Three–Dimensional Simulation of Wind Flow around Buildings, International Journal for Numerical Methods in Fluids, Vol. 6, pp113–127, 1986.

40. **PATERSON, D.A. and APELT, C.J.;** Computation of Wind Flows over Three–Dimensional Buildings, J.of Wind Eng. and Ind. Aerod., no. 24, pp193–213, 1986.

41. **SUMMERS, D.M., HANSON, T. and WILSON, C.B.**; Validation of a Computer Simulation of Wind Flow over a Building Model, Building and Environment, Vol. 21, pp97–111, 1986.

42. **HÄGGKVIST, K., SVENSSON, U. and TAESLER, R.**; Numerical Simulations of Pressure Fields Around Buildings, Building and Environment, Vol. 24, pp65–72, 1989.

43. **VAN DOORNMAAL, J.P. and RAITHBY, G.D.**; *Enhancements of the Simple Method for Predicting Incompressible Fluid Flows,* Numerical Heat Transfer, Vol. 7, pp. 147–163, 1984.

44. **VAN DOORNMAAL, J.P. and RAITHBY, G.D.**; *An Evaluation of the Segregated Approach for Predicting Incompressible Fluid Flows,* Presented at the National Heat Transfer Conference, Colorado, August 4–7, 1985.

45. **THIART, G.D.**; *Finite Difference Scheme for the Numerical Solution of Fluid Flow and Heat Transfer Problems on Non–staggered Grids,* Numerical Heat Transfer, Part B, Vol. 17, pp. 43–62, 1990.

46. **THIART, G.D.**; *Improved Finite Difference Scheme for the Solution of Convection–Diffusion Problems with the Simplen Algorithm,* Numerical Heat Transfer, Part B, Vol. 18, pp. 81–95,1990.

47. **SCHLICHTING, H**; Boundary–Layer Theory, Seventh Edition, McGraw–Hill, 1979.

48. **MINKOWYCZ, W.J., SPARROW, E.M., SCHNEIDER, G.E. and PLETCHER, R.H.**; *Handbook of Numerical Heat Transfer,* A Wiley–Interscience Publication, New York, 1988.

49. **AL–SANEA, S.A., PUN, W.M. and SPALDING, D.B.**; Computation of Two–Dimensional Elliptic Flows, Including Heat Transfer, Imperial College of Science and Technology, HTS/78/5, London, 1978.

50. **DENHAM, M.K.and PATRICK, M.A.**; *Laminar Flow over a Downstream–Facing Step in a Two–Dimensional Flow Channel,* Transactions of the Institution of Chemical Engineers, Vol. 52, 1974

51. **DENHAM, M.K., BRIARD, P. AND PATRICK, M.A.**; *A Directionally–sensitive Laser Anemometer for Velocity Measurements in Highly Turbulent Flows,* Journal of Physics E, Scientific Instruments, Vol. 8, 1975.

52. **WELLS, D.A. and HOXEY, R.P.**; *Measurements of Wind Loads on Full–scale Glasshouses,Journal of Wind Engineering and Industrial Aerodynamics 16,* Silsoe, UK., 139–167, 1980.

53. **SCRUTON, C.**; *An introduction to wind effects on structures,* Oxford University Press, 1981.

# APPENDIX A

# DERIVATION OF GENERAL TRANSPORT EQUATION

In this appendix the x–momentum equation is derived from basic principles in cartesian co–ordinates. The equation is then written in the general form of the transport equation which is valid for the conservation of mass, energy as well as momentum. These equations describe the velocity profiles and pressure distribution in any laminar flow situation. It will be shown in Appendix D that this equation is also applicablein the case of the turbulence equations of the k–$\epsilon$ turbulence model, used to describe turbulent flows.

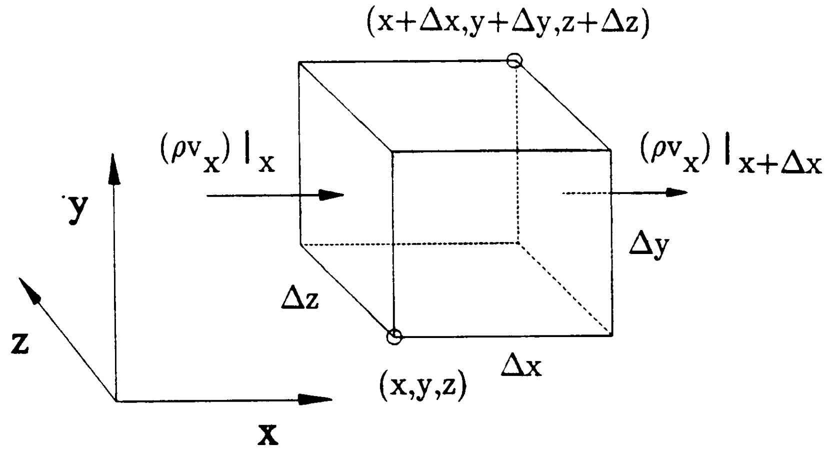

**Figure A.1** A cartesian finite control volume element

For a volume element $\Delta x \Delta y \Delta z$ as shown in Figure A.1 the momentum balance can be written as follows.

Rate of momentum accumulation = rate of momentum in
+ rate of momentum out
+ sum of forces acting on system (A.1)

The complete three–dimensional unsteady behaviour of momentum into and out of the control volume in Figure A.1 will be considered.

The first mechanism whereby momentum enters or leaves the control volume is by means of bulk fluid flow or rather ***convection***. The rate at which the x–component of momentum enters the face at x is given by

$$\rho v_x v_x |_x \Delta y \Delta z \ ,$$

and the rate at which it leaves the control volume at $x+\Delta x$ is

$$\rho v_x v_x |_{x+\Delta x} \Delta y \Delta z \ .$$

Similar to this, the rate of x–momentum in at y and at z are

$$\rho v_y v_x |_y \Delta x \Delta z \quad \text{and}$$

$$\rho v_z v_x |_z \Delta y \Delta x$$

respectively, and the rate at which x–momentum leaves at faces y+Δy and z+Δz are given by the following two expressions.

$$\rho v_y v_x|_{y+\Delta y}\Delta x\Delta z$$

$$\rho v_z v_x|_{z+\Delta z}\Delta y\Delta x$$

The nett convective x–momentum flow into the control volume element can now be determined by subtracting the outflow x–momentum from the total inflow x–momentum. The total convective x–momentum contribution to the element can be written as follows.

$$\begin{aligned}&\left\{\rho v_x v_x|_x + \rho v_x v_x|_{x+\Delta x}\right\}\Delta y\Delta z +\\&\left\{\rho v_y v_x|_y + \rho v_y v_x|_{y+\Delta y}\right\}\Delta x\Delta z +\\&\left\{\rho v_z v_x|_z + \rho v_z v_x|_{z+\Delta z}\right\}\Delta y\Delta x\end{aligned} \qquad \text{(A.2)}$$

The second mechanism by which x–momentum can enter the control volume is by means of ***diffusion*** (molecular transport), or better explained as a result of shear forces acting on the element. Similar to the convective x–momentum inflow, the gain of x–momentum as a result of the shear forces can be expressed by

$$\tau_{xx}|_x\Delta y\Delta z$$

and the loss of x–momentum through molecular transport can be written as

$$\tau_{xx}|_{x+\Delta x}\Delta y\Delta z \ .$$

The total contribution to the x–momentum of the element as a result of shear forces, caused by viscous action in the fluid, can now be written as follows:

$$\begin{aligned}&\left\{\tau_{xx}\big|_x + \tau_{xx}\big|_{x+\Delta x}\right\}\Delta y\Delta z + \\ &\left\{\tau_{yx}\big|_y + \tau_{yx}\big|_{y+\Delta y}\right\}\Delta x\Delta z + \\ &\left\{\tau_{zx}\big|_z + \tau_{zx}\big|_{z+\Delta z}\right\}\Delta y\Delta x\end{aligned} \qquad (A.3)$$

Momentum is also transported to the control volume element by the forces acting on the element. In most cases the only important forces are those arising from the fluid ***pressure*** and the effect of ***gravity.*** The influence of these forces in the x–direction is:

$$\left\{ p\big|_x - p\big|_{x+\Delta x} \right\}\Delta y\Delta z + \rho g_x\Delta x\Delta y\Delta z \qquad (A.4)$$

Finally, the rate of accumulation of x–momentum in the control volume element can be expressed as

$$\left\{ \frac{\partial(\rho v)_z}{\partial t} \right\}\Delta x\Delta y\Delta z \ . \qquad (A.5)$$

By now combining the contribution of each of the four mechanisms of momentum transfer, the x–component of the equation of motion in cartesian co–ordinates can be obtained.

$$\frac{\partial}{\partial t}\left[\rho v_x\right] = -\left[\frac{\partial}{\partial t}\left[\rho v_x v_x\right] + \frac{\partial}{\partial t}\left[\rho v_y v_x\right] + \frac{\partial}{\partial t}\left[\rho v_z v_x\right]\right]$$

$$-\left[\frac{\partial}{\partial t}\left[\tau_{xx}\right] + \frac{\partial}{\partial t}\left[\tau_{yx}\right] + \frac{\partial}{\partial t}\left[\tau_{zx}\right]\right] - \frac{\partial p}{\partial x} + \rho g_x \tag{A.6}$$

By making use of vector and tensor calculus the equation can be generalized to give

$$\frac{\partial}{\partial t}(\rho \overline{v}) = -(\nabla \bullet \rho\overline{vv}) - \nabla p - (\nabla \bullet \overline{\tau}) + \rho g \quad . \tag{A.7}$$

As previously mentioned, similar equations for the y– and z–momentum components and also for the continuity equation can be derived by applying the law of mass conservation. The same can be done for the energy equation but does not form part of this study. The equations can be summarised by a General Transport Equation given below,

$$\frac{\partial}{\partial t}(\rho\phi) = -\nabla \bullet (\rho v \phi) - \nabla p - (\Gamma \nabla \phi) + S^{\phi} \tag{A.8}$$

where $\phi$ represents any one of the dependent variables u,v,or w with $\Gamma$ the diffusion coefficient and $S^{\phi}$ the source term.

# APPENDIX B

# TRANSFORMATION TO CURVILINEAR CO-ORDINATES

In order to solve three–dimensional flow in curvilinear co–ordinates the governing equations are transformed into at set of equations that is generally applicable to a non–orthogonal control volume element, as shown in Figure B.1. In this section the transformation relations from cartesian co–ordinates to a general curvilinear co–ordinate system is developed by using certain concepts from differential geometry and vector and tensor analysis. The equations are transformed in such a way that the cartesian velocity components (u,v,w) are maintained as the dependent variables.

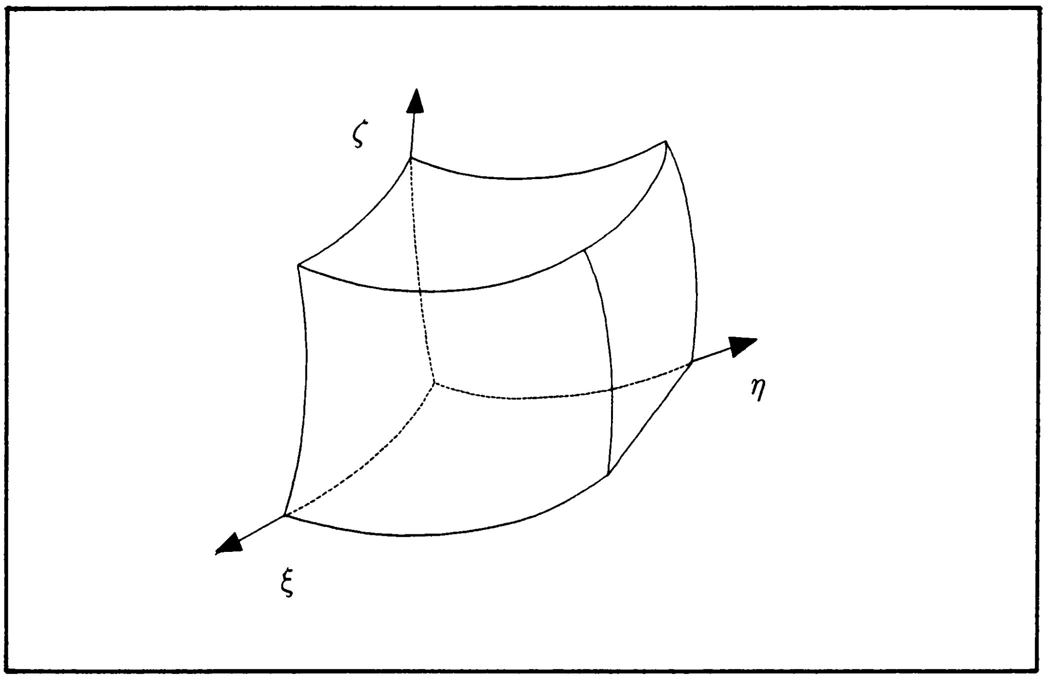

**Figure B.1** General curvilinear finite control volume

Partial derivatives with respect to cartesian co–ordinates are related to partial derivatives with respect to curvilinear co–ordinates by the chain rule which can be written as

$$A_{x_i} = \sum_{j=1}^{3} A_{\xi^j} (\xi^j)_{x_i} \qquad (i = 1,2,3) , \qquad \text{(B.1)}$$

where A is a scalar–valued function. In order to relate the two co–ordinate systems, the covariant base vectors are evaluated as

$$\mathbf{a}_i = \mathbf{r}_{\xi^i} \quad (i = 1,2,3) \quad , \qquad \text{(B.2)}$$

where the curvilinear co–ordinates are represented by $\xi^i$ (i = 1,2,3), and the superscript i indicates the base vector corresponding to the $\xi^i$ co–ordinate. In partial derivative form, the expression for the base vectors can be written as follows.

$$\mathbf{a}_i = \left( \hat{\mathbf{i}}\frac{\partial x_i}{\partial \xi^i} + \hat{\mathbf{j}}\frac{\partial x_j}{\partial \xi^j} + \hat{\mathbf{k}}\frac{\partial x_k}{\partial \xi^k} \right) \qquad (i = 1,2,3;\ i,j,k \text{ cyclic}) \qquad \text{(B.3)}$$

The differential increments of arc length, surface and volume, which are needed in the formulation of the derivative operators can now be developed. An **increment of arc length** on a co–ordinate line along which $\xi^i$ varies, is given by

$$ds^i = |\mathbf{a}_i| d\xi^i \ . \qquad \text{(B.4)}$$

An **increment of area** on a co–ordinate surface of constant $\xi^i$ is given by

$$dS^i = (\mathbf{a}_j \times \mathbf{a}_k)d\xi^j d\xi^k \tag{B.5}$$

and a **volume increment** is given by

$$dV = \mathbf{a}_i \bullet (\mathbf{a}_j \times \mathbf{a}_k)d\xi^i d\xi^j d\xi^k \ . \tag{B.6}$$

The **Jacobian** of the transformation is evaluated as follows:

$$J = \frac{1}{\mathbf{a}_1 \bullet (\mathbf{a}_2 \times \mathbf{a}_3)} \tag{B.7}$$

Expressions for the derivative operators, such as gradient, divergence, curl and Laplacian are obtained by applying the Divergence Theorem to a differential volume increment bounded by co–ordinate surfaces. For any tensor **A** the Theorem gives

$$\iiint_V \nabla \bullet \tilde{A}\, dV = \iint_S \tilde{A} \bullet \bar{n}\, dS \quad , \tag{B.8}$$

where $\bar{n}$ is the outward directed unit normal to the closed surface S enclosing the volume V. For a differential surface element lying on a co–ordinate surface, equation (B.5) gives

$$\bar{n}\, dS^i = \pm (\mathbf{a}_j \times \mathbf{a}_k)d\xi^j d\xi^k \quad , \tag{B.9}$$

with the choice of sign being dependent on the location of the volume relative to the surface. Considering a differential element of volume, $\delta V$, bounded by six faces lying on co–ordinate surfaces, equation (B.6),(B.7) and (B.9) lead to

$$\iiint_{\delta V} (\nabla \bullet \mathrm{A}) \frac{1}{J} d\xi^1 d\xi^2 d\xi^3 = \sum_{i=1}^{3} \left\{ \iint_{\delta S^i_+} \mathrm{A} \bullet (a_j \times a_k) d\xi^j d\xi^k - \iint_{\delta S^i_-} \mathrm{A} \bullet (a_j \times a_k) d\xi^j d\xi^k \right\} \qquad \text{(B.10)}$$

where $\delta S^i_+$ and $\delta S^i_-$ indicate the elements on two sides of which $\xi^i$ is constant and which are located at larger and smaller values, respectively, of $\xi^i$.

Proceeding to the limit as the element of volume shrinks to zero, an expression is obtained for the **divergence** in the conservative form,

$$\nabla \bullet \mathrm{A} = J \sum_{i=1}^{3} \left[ (\mathbf{a}_j \times \mathbf{a}_k) \bullet \mathrm{A} \right]_{\xi^i} \; . \qquad \text{(B.11)}$$

It is important to note that since the conservative form of the divergence and of the gradient and Laplacian to follow, is obtained directly from the closed surface integral in the Divergence Theorem, the use of conservative difference forms for these derivative operators is equivalent to using difference forms for that closed surface integral. Therefore, the finite volume difference formulation can be implemented by using these conservative forms directly in the differential equations of motion without the necessity of returning to the integral form of the equation of motion.[10]

Equation (B.8) is also valid with **A** replaced by a scalar A and the dot product by simple multiplication. The conservative form of the **gradient** follows directly from equation (B.11) as

$$\nabla A = J \sum_{i=1}^{3} \left[ (\mathbf{a}_j \times \mathbf{a}_k) A \right]_{\xi^i} \ . \qquad \text{(B.12)}$$

By replacing A by ∇A in equation (B.12) the expression for the **Laplacian** is as follows:

$$\begin{aligned} \nabla^2 A &= \nabla \bullet (\nabla A) \\ &= J \sum_{i=1}^{3} \sum_{l=1}^{3} \left\{ J(\mathbf{a}_j \times \mathbf{a}_k) \bullet \left[ (\mathbf{a}_m \times \mathbf{a}_n) A \right]_{\xi^l} \right\}_{\xi^i} \end{aligned} \qquad \text{(B.13)}$$

(i,j,k) cyclic & $(l,m,n)$ cyclic

By making use of these transformation relations, the general transport equation derived in appendix A,

$$\frac{\partial}{\partial t}(\rho\phi) = -\nabla \bullet (\rho \mathrm{v} \phi) - \nabla p - \nabla \bullet (\Gamma \nabla \phi) + S^{\phi} \ , \qquad \text{(B.14)}$$

can be transformed to curvilinear co–ordinates. Complete transformation and expansion lead to the following set of equations that are used to describe three–dimensional flow in curvilinear co–ordinates.

Let ($\xi^i$, i = 1,2,3) be substituted by ($\xi,\eta,\zeta$) and ($x^i$, i = 1,2,3) by (x,y,z).

$$\begin{aligned}\frac{\partial}{\partial t}\left[\frac{U}{J}\right] + \frac{\partial}{\partial \xi}\left[\frac{1}{J}(\xi_x E + \xi_y F + \xi_z G)\right] \\ + \frac{\partial}{\partial \eta}\left[\frac{1}{J}(\eta_x E + \eta_y F + \eta_z G)\right] \\ + \frac{\partial}{\partial \zeta}\left[\frac{1}{J}(\zeta_x E + \zeta_y F + \zeta_z G)\right] = 0 \qquad \text{(B.15)}\end{aligned}$$

where the **U**,**E**,**F** and **G** vectors are given by

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix}$$

$$E = \begin{bmatrix} \rho u \\ \rho u^2 + p - \tau_{xx} \\ \rho uv - \tau_{xy} \\ \rho uw - \tau_{xz} \end{bmatrix}$$

$$F = \begin{bmatrix} \rho v \\ \rho uv - \tau_{xy} \\ \rho v^2 + p - \tau_{yy} \\ \rho uw - \tau_{yz} \end{bmatrix}$$

$$G = \begin{bmatrix} \rho w \\ \rho uw - \tau_{xz} \\ \rho uv - \tau_{yz} \\ \rho w^2 + p - \tau_{zz} \end{bmatrix} .$$

The transformation values and the shear stresses are defined as follows:

$$\begin{array}{lll} \xi_x = J(y_\eta z_\zeta - y_\zeta z_\eta) & \xi_y = -J(x_\eta z_\zeta - x_\zeta z_\eta) & \xi_z = J(x_\eta z_\zeta - x_\zeta z_\eta) \\ \eta_x = -J(y_\xi z_\zeta - y_\zeta z_\xi) & \eta_y = J(x_\xi z_\zeta - x_\zeta z_\xi) & \eta_z = -J(x_\xi z_\zeta - x_\zeta z_\xi) \\ \zeta_x = J(y_\xi z_\eta - y_\eta z_\xi) & \zeta_y = -J(x_\xi z_\eta - x_\eta z_\xi) & \zeta_z = J(x_\xi z_\eta - x_\eta z_\xi) \end{array}$$

$$\begin{aligned}
\tau_{xx} &= \tfrac{2}{3}\mu\Big[\,2\big[\xi_x u_\xi + \eta_x u_\eta + \zeta_x u_\zeta\big] - \big[\xi_y v_\xi + \eta_y v_\zeta + \zeta_y v_\zeta\big] \\
&\quad - \big[\xi_z w_\xi + \eta_z w_\eta + \zeta_z w_\zeta\big]\Big] \\
\tau_{yy} &= \tfrac{2}{3}\mu\Big[\,2\big[\xi_y v_\xi + \eta_y v_\eta + \zeta_y v_\zeta\big] - \big[\xi_x u_\xi + \eta_x u_\zeta + \zeta_x u_\zeta\big] \\
&\quad - \big[\xi_z w_\xi + \eta_z w_\eta + \zeta_z w_\zeta\big]\Big] \\
\tau_{zz} &= \tfrac{2}{3}\mu\Big[\,2\big[\xi_z w_\xi + \eta_z w_\eta + \zeta_z w_\zeta\big] - \big[\xi_x u_\xi + \eta_x u_\zeta + \zeta_x u_\zeta\big] \\
&\quad - \big[\xi_y v_\xi + \eta_y v_\eta + \zeta_y v_\zeta\big]\Big] \\
\tau_{xy} &= \mu\big[\xi_y u_\xi + \eta_y u_\eta + \zeta_y u_\zeta + \xi_x v_\xi + \eta_x v_\eta + \zeta_x v_\zeta\big] \\
\tau_{yz} &= \mu\big[\xi_z v_\xi + \eta_z v_\eta + \zeta_z v_\zeta + \xi_y w_\xi + \eta_y w_\eta + \zeta_y w_\zeta\big] \\
\tau_{xz} &= \mu\big[\xi_z u_\xi + \eta_z u_\eta + \zeta_z u_\zeta + \xi_x w_\xi + \eta_x w_\eta + \zeta_x w_\zeta\big] \ .
\end{aligned}$$

# APPENDIX C

# SMALL SCALE TURBULENCE FLUCTUATIONS

A very similar equation as the one obtained in Appendix B also applies to turbulent flow. The only difference is the occurrence of small scale fluctuations in the flow field. In this section the laminar x–momentum equation is modified to provide for these small scale fluctuations due to turbulence.

Let each dependent variable consist of an average value as well as a fluctuating term. The variables considered can therefore be written as follows:

$$u = \bar{u} + u'$$
$$v = \bar{v} + v'$$
$$w = \bar{w} + w'$$
$$p = \bar{p} + p'$$
$$\rho = \bar{\rho} + \rho'$$

By replacing the variables in the laminar equation (C.1) with the above expressions,

$$\frac{\partial}{\partial t}\left[\frac{\rho u}{J}\right] + \frac{\partial}{\partial \xi}\left[\frac{1}{J}\left[\xi_x \rho u^2\right] + \frac{1}{J}\left[\xi_y \rho uv\right] + \frac{1}{J}\left[\xi_z \rho uw\right]\right]$$
$$+ \frac{\partial}{\partial \eta}\left[\frac{1}{J}\left[\eta_x \rho u^2\right] + \frac{1}{J}\left[\eta_y \rho uv\right] + \frac{1}{J}\left[\eta_z \rho uw\right]\right]$$
$$+ \frac{\partial}{\partial \zeta}\left[\frac{1}{J}\left[\zeta_x \rho u^2\right] + \frac{1}{J}\left[\zeta_y \rho uv\right] + \frac{1}{J}\left[\zeta_z \rho uw\right]\right] = S^u \qquad (C.1)$$

the following equation for compressible turbulent flows can be derived:

$$\frac{\partial}{\partial t}\left[(\bar\rho + \rho')(\bar u + u')\right]$$
$$\frac{\partial}{\partial \xi}\frac{1}{J}\left[\xi_x(\bar\rho + \rho')(\bar u + u')(\bar u + u') + \xi_y(\bar\rho + \rho')(\bar u + u')(\bar v + v')\right.$$
$$\left. + \xi_z(\bar\rho + \rho')(\bar u + u')(\bar w + w')\right]$$
$$\frac{\partial}{\partial \xi}\frac{1}{J}\left[\eta_x(\bar\rho + \rho')(\bar u + u')(\bar u + u') + \eta_y(\bar\rho + \rho')(\bar u + u')(\bar v + v')\right.$$
$$\left. + \eta_z(\bar\rho + \rho')(\bar u + u')(\bar w + w')\right]$$
$$\frac{\partial}{\partial \xi}\frac{1}{J}\left[\zeta_x(\bar\rho + \rho')(\bar u + u')(\bar u + u') + \zeta_y(\bar\rho + \rho')(\bar u + u')(\bar v + v')\right.$$
$$\left. + \zeta_z(\bar\rho + \rho')(\bar u + u')(\bar w + w')\right] = S^u \qquad , (C.2)$$

where $S^u$ (the right hand side of the equation) consists of the pressure and cross derivative terms. These terms are not influenced by the small scale fluctuations since the dependent variables are merely replaced by their time averages. Therefore, only the right hand side of the equation will be considered in detail.

By applying the time averages for each variable, the time averaged form of the right hand side of the momentum equation is obtained.

$$\begin{aligned}
\text{RH:}\quad \frac{\partial}{\partial t}\left[\overline{\rho u} + \rho' u'\right] &+ \frac{\partial}{\partial \xi}\frac{1}{J}\Big[\xi_x(\overline{\rho uu} + 2\rho' u'\bar{u} + \bar{\rho} u' u') \\
&\qquad + \xi_y(\overline{\rho uv} + \rho' u'\bar{v} + \rho'\bar{u}v' + \bar{\rho} u' v') \\
&\qquad + \xi_z(\overline{\rho uw} + \rho' u'\bar{w} + \rho'\bar{u}w' + \bar{\rho} u' w')\Big] \\
&+ \frac{\partial}{\partial \eta}\frac{1}{J}\Big[\eta_x(\overline{\rho uu} + 2\rho' u'\bar{u} + \bar{\rho} u' u') \\
&\qquad + \eta_y(\overline{\rho uv} + \rho' u'\bar{v} + \rho'\bar{u}v' + \bar{\rho} u' v') \\
&\qquad + \eta_z(\overline{\rho uw} + \rho' u'\bar{w} + \rho'\bar{u}w' + \bar{\rho} u' w')\Big] \\
&+ \frac{\partial}{\partial \zeta}\frac{1}{J}\Big[\xi_x(\overline{\rho uu} + 2\rho' u'\bar{u} + \bar{\rho} u' u') \\
&\qquad + \zeta_y(\overline{\rho uv} + \rho' u'\bar{v} + \rho'\bar{u}v' + \bar{\rho} u' v') \\
&\qquad + \zeta_z(\overline{\rho uw} + \rho' u'\bar{w} + \rho'\bar{u}w' + \bar{\rho} u' w')\Big] \quad . \qquad \text{(C.3)}
\end{aligned}$$

By now assuming that $\rho'$ is equal to zero, the above expression is reduced to

$$\begin{aligned}
\text{RH:}\quad \frac{\partial}{\partial t}\left[\overline{\rho u}\right] &+ \frac{\partial}{\partial \xi}\frac{1}{J}\Big[\xi_x(\overline{\rho uu} + \underline{\bar{\rho} u' u'}) + \xi_y(\overline{\rho uv} + \underline{\bar{\rho} u' v'}) + \xi_z(\overline{\rho uw} + \underline{\bar{\rho} u' w'})\Big] \\
&+ \frac{\partial}{\partial \eta}\frac{1}{J}\Big[\eta_x(\overline{\rho uu} + \underline{\bar{\rho} u' u'}) + \eta_y(\overline{\rho uv} + \underline{\bar{\rho} u' v'}) + \eta_z(\overline{\rho uw} + \underline{\bar{\rho} u' w'})\Big] \\
&+ \frac{\partial}{\partial \zeta}\frac{1}{J}\Big[\xi_x(\overline{\rho uu} + \underline{\bar{\rho} u' u'}) + \zeta_y(\overline{\rho uv} + \underline{\bar{\rho} u' v'}) + \zeta_z(\overline{\rho uw} + \underline{\bar{\rho} u' w'})\Big]
\end{aligned}$$

(C.4)

The underlined terms in equation (C.4) are the new terms that result from the provision for small scale fluctuations and is the only addition to the laminar equation that was derived in Appendix B, necessary for solving turbulent flow. Each of these terms can, in turn, be represented by a shear force (or diffusion) term by making use of the turbulent viscosity $\mu_t$.

$$\bar{\rho} u' u' = -\mu_t \frac{\partial u}{\partial \xi} \qquad \text{(C.5)}$$

By combining the above with the other diffusion terms, the final equation that describes the conservation of x–momentum is

$$\begin{aligned}\frac{\partial}{\partial t}\left[\frac{U}{J}\right] + \frac{\partial}{\partial \xi}\left[\frac{1}{J}(\xi_x E + \xi_y F + \xi_z G)\right] \\ + \frac{\partial}{\partial \eta}\left[\frac{1}{J}(\eta_x E + \eta_y F + \eta_z G)\right] \\ + \frac{\partial}{\partial \zeta}\left[\frac{1}{J}(\zeta_x E + \zeta_y F + \zeta_z G)\right] = 0 \,, \end{aligned} \tag{C.6}$$

where the **U**,**E**,**F** and **G** vectors are as in Appendix B and new expressions for the shear stresses are defined as follows:

$$\begin{aligned}\tau_{xx} &= \frac{2}{3}\mu_{eff}\left[2\left[\xi_x u_\xi + \eta_x u_\eta + \zeta_x u_\zeta\right] - \left[\xi_y v_\xi + \eta_y v_\zeta + \zeta_y v_\zeta\right] \right. \\ &\qquad \left. - \left[\xi_z w_\xi + \eta_z w_\eta + \zeta_z w_\zeta\right]\right] \\ \tau_{yy} &= \frac{2}{3}\mu_{eff}\left[2\left[\xi_y v_\xi + \eta_y v_\eta + \zeta_y v_\zeta\right] - \left[\xi_x u_\xi + \eta_x u_\zeta + \zeta_x u_\zeta\right] \right. \\ &\qquad \left. - \left[\xi_z w_\xi + \eta_z w_\eta + \zeta_z w_\zeta\right]\right] \\ \tau_{zz} &= \frac{2}{3}\mu_{eff}\left[2\left[\xi_z w_\xi + \eta_z w_\eta + \zeta_z w_\zeta\right] - \left[\xi_x u_\xi + \eta_x u_\zeta + \zeta_x u_\zeta\right] \right. \\ &\qquad \left. - \left[\xi_y v_\xi + \eta_y v_\eta + \zeta_y v_\zeta\right]\right]\end{aligned}$$

$$\begin{aligned}\tau_{xy} &= \mu_{eff}\left[\xi\, u_\xi + \eta_y u_\eta + \zeta_y u_\zeta + \xi_x v_\xi + \eta_x v_\eta + \zeta_x v_\zeta\right] \\ \tau_{yz} &= \mu_{eff}\left[\xi_z v_\xi + \eta_z v_\eta + \zeta_z v_\zeta + \xi_y w_\xi + \eta_y w_\eta + \zeta_y w_\zeta\right] \\ \tau_{xz} &= \mu_{eff}\left[\xi_z u_\xi + \eta_z u_\eta + \zeta_z u_\zeta + \xi_x w_\xi + \eta_x w_\eta + \zeta_x w_\zeta\right]\end{aligned}$$

The effective viscosity in the above expressions is now defined as the sum of the laminar and the turbulent viscosities:

$$\mu_{\mathrm{eff}} = \mu + \mu_{\mathrm{t}} \tag{C.7}$$

The same procedure can be followed for the y– and z–momentum equations which finally result in the replacement of the laminar viscosity $\mu$ by the the effective viscosity $\mu_{\mathrm{eff}}$.

# APPENDIX D

# DIFFERENCING AND DISCRETIZATION

In order to solve the partial differential equations that describe three–dimensional flow in curvilinear co–ordinates numerically, the equations have to be discretized. In this section a finite volume approach is used to obtain the linear finite difference expressions. This method entails the integration of the partial differential equations over a finite control volume. The general x–momentum equation describing turbulent flow, with respect to curvilinear co–ordinates, is discretized and the y– and z–momentum equations are treated similarly. The principles described in this section apply to the discretization of the General Transport Equation which includes the k and $\epsilon$ equations describing turbulence properties as well as the continuity equation.

From Appendix A the complete x–momentum equation relative to a curvilinear co–ordinate system can be written as

$$\frac{\partial}{\partial t}\left[\frac{\rho u}{J}\right] + \frac{\partial}{\partial \xi}\left[\frac{1}{J}\left[\xi_x\left[\rho u^2+p-\tau_{xx}\right] + \xi_y\left[\rho uv-\tau_{xy}\right] + \xi_z\left[\rho uw-\tau_{xz}\right]\right]\right]$$
$$+ \frac{\partial}{\partial \eta}\left[\frac{1}{J}\left[\eta_x\left[\rho u^2+p-\tau_{xx}\right] + \eta_y\left[\rho uv-\tau_{xy}\right] + \eta_z\left[\rho uw-\tau_{xz}\right]\right]\right]$$
$$+ \frac{\partial}{\partial \zeta}\left[\frac{1}{J}\left[\zeta_x\left[\rho u^2+p-\tau_{xx}\right] + \zeta_y\left[\rho uv-\tau_{xy}\right] + \zeta_z\left[\rho uw-\tau_{xz}\right]\right]\right] = 0 \quad \text{(D.1)}$$

with the shear forces being

$$
\begin{aligned}
\tau_{xx} &= \frac{2}{3}\mu_{eff}\left[ 2\left[\xi_x u_\xi + \eta_x u_\eta + \zeta_x u_\zeta\right] - \left[\xi_y v_\xi + \eta_y v_\zeta + \zeta_y v_\zeta\right] \right. \\
&\qquad \left. - \left[\xi_z w_\xi + \eta_z w_\eta + \zeta_z w_\zeta\right]\right] \\
\tau_{xy} &= \mu_{eff}\left[\xi_y u_\xi + \eta_y u_\eta + \zeta_y u_\zeta + \xi_x v_\xi + \eta_x v_\eta + \zeta_x v_\zeta\right] \\
\tau_{xz} &= \mu_{eff}\left[\xi_z u_\xi + \eta_z u_\eta + \zeta_z u_\zeta + \xi_x w_\xi + \eta_x w_\eta + \zeta_x w_\zeta\right]
\end{aligned}
$$

and the transformation values $\xi_x$, $\xi_y$, $\xi_z$ etc. evaluated as described in Appendix B. By substituting the expressions as above, and by ordering the convection, diffusion, pressure and cross derivative terms, the left hand side of the equation becomes

$$
\begin{aligned}
\text{LH:}\quad & \frac{\partial}{\partial t}\left[\frac{\rho u}{J}\right] + \frac{\partial}{\partial \xi}\left[\frac{1}{J}\left[\xi_x \rho u^2 + \xi_y \rho uv + \xi_z \rho uw - \mu\left[\frac{4}{3}\xi_x^2 + \xi_y^2 + \xi_z^2\right]\frac{\partial u}{\partial \xi}\right]\right] \\
& + \frac{\partial}{\partial \eta}\left[\frac{1}{J}\left[\eta_x \rho u^2 + \eta_y \rho uv + \eta_z \rho uw - \mu\left[\frac{4}{3}\eta_x^2 + \eta_y^2 + \eta_z^2\right]\frac{\partial u}{\partial \eta}\right]\right] \\
& + \frac{\partial}{\partial \zeta}\left[\frac{1}{J}\left[\zeta_x \rho u^2 + \zeta_y \rho uv + \zeta_z \rho uw - \mu\left[\frac{4}{3}\zeta_x^2 + \zeta_y^2 + \zeta_z^2\right]\frac{\partial u}{\partial \zeta}\right]\right] \quad , \qquad \text{(D.2)}
\end{aligned}
$$

while the rest of the equation goes into the source term on the right hand side:

$$
\begin{aligned}
\text{RH:}\quad & -\frac{\partial}{\partial \xi}\left[\frac{\xi_x P}{J}\right] - \frac{\partial}{\partial \eta}\left[\frac{\eta_x P}{J}\right] - \frac{\partial}{\partial \zeta}\left[\frac{\zeta_x P}{J}\right] \\
& -\frac{\partial}{\partial \xi}\left[\frac{\mu}{J}\left[BB1\frac{\partial u}{\partial \eta} + BB2\frac{\partial u}{\partial \zeta} + BB3\frac{\partial v}{\partial \xi} + BB4\frac{\partial v}{\partial \eta}\right.\right.
\end{aligned}
$$

$$
\begin{aligned}
&\quad + \mathrm{BB5}\frac{\partial v}{\partial \zeta} + \mathrm{BB6}\frac{\partial w}{\partial \xi} + \mathrm{BB7}\frac{\partial w}{\partial \eta} + \mathrm{BB8}\frac{\partial w}{\partial \zeta} \Bigg]\Bigg] \\
&- \frac{\partial}{\partial \eta}\Bigg[ \frac{\mu}{J}\Bigg[ \mathrm{BB9}\frac{\partial u}{\partial \xi} + \mathrm{BB10}\frac{\partial u}{\partial \zeta} + \mathrm{BB11}\frac{\partial v}{\partial \xi} + \mathrm{BB12}\frac{\partial v}{\partial \eta} \\
&\quad + \mathrm{BB13}\frac{\partial v}{\partial \zeta} + \mathrm{BB14}\frac{\partial w}{\partial \xi} + \mathrm{BB15}\frac{\partial w}{\partial \eta} + \mathrm{BB16}\frac{\partial w}{\partial \zeta} \Bigg]\Bigg] \\
&- \frac{\partial}{\partial \xi}\Bigg[ \frac{\mu}{J}\Bigg[ \mathrm{BB17}\frac{\partial u}{\partial \eta} + \mathrm{BB18}\frac{\partial u}{\partial \zeta} + \mathrm{BB19}\frac{\partial v}{\partial \xi} + \mathrm{BB20}\frac{\partial v}{\partial \eta} \\
&\quad + \mathrm{BB21}\frac{\partial v}{\partial \zeta} + \mathrm{BB22}\frac{\partial w}{\partial \xi} + \mathrm{BB23}\frac{\partial w}{\partial \eta} + \mathrm{BB24}\frac{\partial w}{\partial \zeta} \Bigg]\Bigg]
\end{aligned}
\tag{D.3}
$$

The BB–values are evaluated as follows:

$$
\begin{array}{ll}
\mathrm{BB1} = -\frac{4}{3}\xi_x\eta_x - \xi_y\eta_y - \xi_z\eta_z & \mathrm{BB2} = -\frac{4}{3}\xi_x\zeta_x - \xi_y\zeta_y - \xi_z\zeta_z \\
\mathrm{BB3} = -\frac{1}{3}\xi_x\xi_y & \mathrm{BB4} = \frac{2}{3}\xi_x\eta_y - \xi_y\eta_x \\
\mathrm{BB5} = \frac{2}{3}\xi_x\zeta_y - \xi_y\zeta_x & \mathrm{BB6} = -\frac{1}{3}\xi_x\xi_z \\
\mathrm{BB7} = \frac{2}{3}\xi_x\eta_z - \xi_z\eta_x & \mathrm{BB8} = \frac{2}{3}\xi_x\zeta_z - \xi_z\zeta_x
\end{array}
$$

$$
\begin{array}{ll}
\mathrm{BB9} = -\frac{4}{3}\eta_x\xi_x - \eta_y\xi_y - \eta_z\xi_z & \mathrm{BB10} = -\frac{4}{3}\eta_x\zeta_x - \eta_y\zeta_y - \eta_z\zeta_z \\
\mathrm{BB11} = \frac{2}{3}\eta_x\xi_y - \eta_y\xi_x & \mathrm{BB12} = -\frac{1}{3}\eta_x\eta_y \\
\mathrm{BB13} = \frac{2}{3}\eta_x\zeta_y - \eta_y\zeta_x & \mathrm{BB14} = \frac{2}{3}\eta_x\xi_z - \eta_z\xi_x \\
\mathrm{BB15} = -\frac{1}{3}\eta_x\eta_z & \mathrm{BB16} = \frac{2}{3}\eta_x\zeta_z - \eta_z\zeta_x
\end{array}
$$

$$
\begin{array}{ll}
\mathrm{BB17} = -\frac{4}{3}\zeta_x\xi_x - \zeta_y\xi_y - \zeta_z\xi_z & \mathrm{BB18} = -\frac{4}{3}\eta_x\xi_x - \eta_y\xi_y - \eta_z\xi_z \\
\mathrm{BB19} = \frac{2}{3}\zeta_x\xi_y - \zeta_y\xi_x & \mathrm{BB20} = \frac{2}{3}\zeta_x\eta_y - \zeta_y\eta_x \\
\mathrm{BB21} = -\frac{1}{3}\zeta_x\zeta_y & \mathrm{BB22} = \frac{2}{3}\zeta_x\xi_z - \zeta_z\xi_x \\
\mathrm{BB23} = \frac{2}{3}\zeta_x\eta_z - \zeta_z\eta_x & \mathrm{BB24} = -\frac{1}{3}\zeta_x\zeta_z
\end{array}
$$

Referring to Eq. (D.2) the following concepts can now be defined:

Convection flux:

$$F_\xi = \xi_x \rho u + \xi_y \rho v + \xi_z \rho w$$
$$F_\eta = \eta_x \rho u + \eta_y \rho v + \eta_z \rho w$$
$$F_\zeta = \zeta_x \rho u + \zeta_y \rho v + \zeta_z \rho w$$

Diffusion flux:

$$D_\xi = \mu_{eff}\left[\frac{4}{3}\xi_x^2 + \xi_y^2 + \xi_z^2\right]/\Delta\xi$$
$$D_\eta = \mu_{eff}\left[\frac{4}{3}\eta_x^2 + \eta_y^2 + \eta_z^2\right]/\Delta\eta$$
$$D_\zeta = \mu_{eff}\left[\frac{4}{3}\zeta_x^2 + \zeta_y^2 + \zeta_z^2\right]/\Delta\zeta$$

Peclet Number:

$$P_\xi = \frac{F_\xi}{D_\xi}$$
$$P_\eta = \frac{F_\eta}{D_\eta}$$
$$P_\zeta = \frac{F_\zeta}{D_\zeta} \ .$$

The Peclet Number is the ratio of the strength of convection to the strength of diffusion. The magnitude of the Peclet Number influences the differencing.

| <u>Differencing scheme</u> | <u>Formula for A(\|P\|)</u> |
|---|---|
| Central Difference | $1 - 0.5\|P\|$ |
| Upwind Difference | $1$ |
| Hybrid Difference | $\ll 0, 1 - 0.5\|P\| \gg$ |
| Power law Difference | $\ll 0, (1 - 0.1\|P\|)^5$ |
| Exponential Difference | $\|P\|/[\exp(\|P\|) - 1]$ |

Following Patankar [15], the use of this number is implemented by calculating the function value A as indicated in the table above.

The operator $\langle\!\langle x, y \rangle\!\rangle$ means "select the greater of x and y". Programming the differencing schemes in such a way, enables the comparison and the optimal use of the various differencing schemes. It should be noted that this type of differencing is applied only to the main convection and diffusion parts of the equation, and the pressure and cross derivative terms are subsequently discussed. Patankar [5] recommends the power law scheme as a good compromise between accuracy and economy of calculation.

By making use of the expressions developed so far, the discretized equation for a general curvilinear control volume can be written as follows:

$$a_p u_P = a_e u_E + a_w u_W + a_n u_N + a_s U_S + a_b u_B + a_f u_F + a_p^o u_P^o + b \qquad \text{(D.4)}$$

where

$$\begin{aligned}
a_e &= D_e A(|P_e|) + \langle\!\langle -F_e, 0 \rangle\!\rangle \\
a_w &= D_w A(|P_w|) + \langle\!\langle F_w, 0 \rangle\!\rangle \\
a_n &= D_n A(|P_n|) + \langle\!\langle -F_n, 0 \rangle\!\rangle \\
a_s &= D_s A(|P_s|) + \langle\!\langle F_s, 0 \rangle\!\rangle \\
a_b &= D_b A(|P_b|) + \langle\!\langle -F_b, 0 \rangle\!\rangle \\
a_f &= D_f A(|P_f|) + \langle\!\langle F_f, 0 \rangle\!\rangle \\
a_p^o &= \rho^o / \Delta t \quad , \quad \text{and}
\end{aligned}$$

$$a_p = a_e + a_w + a_n + a_s + a_b + a_f + a_p^o \ .$$

Using the expressions for convection and diffusion flux developed earlier, the values at the control volume walls are determined in the following way:

$$F_e = \xi_{x_e}\rho_e u_e + \xi_{y_e}\rho_e v_e + \xi_{z_e}\rho_e w_e \ ,$$
$$D_e = \mu_{\mathrm{eff}_e}\left[\tfrac{4}{3}\xi^2_{x_e} + \xi^2_{y_e} + \xi^2_{z_e}\right] \ , \text{ and}$$
$$P_e = \frac{F_e}{D_e} \ ,$$

since $\Delta\xi$, $\Delta\eta$ and $\Delta\zeta$ are equal to unity.

The equations, relevant to this study, contain several first derivatives, for example, the **unsteady term** in the x–momentum equation. At grid point P, this term can be discretized as:

$$\frac{\partial}{\partial t}\left(\frac{\rho\phi}{J}\right) \simeq \frac{(\rho\phi - \rho^o\phi^o)_P}{J_P\Delta t}$$

where $\Delta t$ represents the time interval in the solution of unsteady flow problems, and J is the Jacobian of the transformation, as calculated in Appendix B. Another first order derivative is the **pressure term** that form part of the source term in Eq. (D.3) The three pressure terms are treated by applying the central difference scheme between opposite cell walls. Therefore, the pressures used, are linearly interpolated at the control volume walls.

The terms $\quad -\frac{\partial}{\partial\xi}\left[\frac{\xi_x P}{J}\right] - \frac{\partial}{\partial\eta}\left[\frac{\eta_x P}{J}\right] - \frac{\partial}{\partial\xi}\left[\frac{\zeta_x P}{J}\right]$

are discretized as

$$\frac{\left[\frac{1}{J_w}\xi_{x_w}P_w - \frac{1}{J_e}\xi_{x_e}P_e\right]}{\Delta\xi} + \frac{\left[\frac{1}{J_s}\eta_{x_s}P_s - \frac{1}{J_n}\eta_{x_n}P_n\right]}{\Delta\eta} + \frac{\left[\frac{1}{J_f}\zeta_{x_f}P_f - \frac{1}{J_b}\zeta_{x_b}P_b\right]}{\Delta\zeta} \quad ,$$

but with the dimensions in the calculation domain,$\Delta\xi$, $\Delta\eta$ and $\Delta\zeta$, taking the value of unity due the method of transformation used, it can be further reduced to

$$\left[\ \xi_{x_w}P_w - \xi_{x_e}P_e\ \right] + \left[\ \eta_{x_s}P_s - \eta_{x_n}P_n\ \right] + \left[\ \zeta_{x_f}P_f - \zeta_{x_b}P_b\ \right] \quad .$$

Referring to Equation (D.3), the cross derivative terms are all treated similarly and therefore only one of the discretizations will be provided. Consider the **diffusion term** of coefficient BB1:

$$-\frac{\partial}{\partial\xi}\left[\frac{\mu}{J}\left[\ \mathrm{BB1}\frac{\partial u}{\partial\eta}\right]\right] \quad .$$

The expression BB1 is evaluated at opposing cell walls as follows:

$$\mathrm{BB1}_e = -\frac{4}{3}\xi_{x_e}\eta_{x_e} - \xi_{y_e}\eta_{y_e} - \xi_{z_e}\eta_{z_e}$$
$$\mathrm{BB1}_w = -\frac{4}{3}\xi_{x_w}\eta_{x_w} - \xi_{y_w}\eta_{y_w} - \xi_{z_w}\eta_{z_w}$$

Discretizing the first derivative leads to

$$\mu_{\mathrm{eff}_w}\mathrm{BB1}_w\left[\frac{\partial u}{\partial\eta}\right]_w - \mu_{\mathrm{eff}_e}\mathrm{BB1}_e\left[\frac{\partial u}{\partial\eta}\right]_e$$

and discretization of the second derivative, completely expanded, provides the final difference expression as given below:

$$\mu_{\mathrm{eff}_w} \mathrm{BB1}_w \left[ u_{nw} - u_{sw} \right] - \mu_{\mathrm{eff}_e} \mathrm{BB1}_e \left[ u_{ne} - u_{se} \right] \quad .$$

The velocity values at the control volume corners $u_{ne}$ etc. are determined by linear interpolation between the eight surrounding grid points. Each of the terms in the difference equation has now been evaluated, and the discretization equations can, therefore, be solved to determine the point values of the dependent variables.

# APPENDIX E

## DISCRETIZATION OF TURBULENCE EQUATIONS

The partial differential equations included in the k$-\epsilon$ turbulence model are given below. The equations are presented in the generalized vector form independent of a co–ordinate system. They describe the turbulent kinetic energy (k) and the turbulence dissipation rate ($\epsilon$) from which an additional turbulent viscosity can be calculated.

$$\frac{\partial \epsilon}{\partial t} + (\mathbf{v} \bullet \nabla \epsilon) = \frac{1}{\rho}\left[\nabla \bullet \left(\frac{\mu_t}{\sigma_\epsilon} \nabla \epsilon\right)\right] + \frac{C_1 \mu_t}{\rho} \frac{\epsilon}{k} \Gamma - C_2 \frac{\epsilon^2}{k} \tag{E.1}$$

$$\frac{\partial k}{\partial t} + (\mathbf{v} \bullet \nabla k) = \frac{1}{\rho}\left[\nabla \bullet \left(\frac{\mu_t}{\sigma_k} \nabla k\right)\right] + \frac{\mu_t}{\rho} \Gamma - \epsilon \tag{E.2}$$

The above equations, which are also based on the general conservation principle, are discretized using the same procedures as described in Appendix D for the discretization of the General Transport Equation. This applies for the unsteady–, the convection– and the diffusion terms (the first three terms) in the above equation. Further attention is given to the discretization of the source term which is made up by the production– and the dissipation terms (last two terms) in each of the two equations.

Spalding [2] and Gosman [4] describe a specific linearization process of the source terms which provides stability to the solution and also yields accurate approximations. This process is briefly considered for each of the turbulence quantities.

For the k–equation the source term is:

$$S_k = \mu_t \Gamma - \rho\epsilon \tag{E.3}$$

The linearization is as follows:

$$S_k = - C_p^{'} k + B \tag{E.4}$$

where

$$C_p^{'} = \frac{\left[ \; C_2\rho\epsilon \; + \; \frac{1}{2}\mu_t\Gamma \; \right]}{k_p} \quad \text{, and} \tag{E.5}$$

$$B = \left[ \frac{3}{2}\mu_t\Gamma + (C_2 - 1)\rho\epsilon \right] \; . \tag{E.6}$$

For the $\epsilon$–equation the source term

$$S_\epsilon = C_1\mu_t\Gamma\frac{\epsilon}{k} - C_2\rho\frac{\epsilon^2}{k} \tag{E.7}$$

is linearized as follows:

$$S_\epsilon = -C_p^{'}\epsilon + B \tag{E.8}$$

where

$$C_p^{'} = \rho(2C_2 - 1)\frac{\epsilon}{k} \quad , \tag{E.9}$$

$$B = \left[ C_1 \mu_t \Gamma + (C_2 - 1) \rho \frac{\epsilon^2}{k} \right] \quad . \tag{E.10}$$

This leads to the final discretization equation

$$a_p \phi_p = \Sigma a_{nb} \phi_{nb} + B \tag{E.11}$$

where $a_p = \Sigma a_{nb} + C_p' \quad .$ (E.12)

On fixed wall boundaries the source term in the k–equation is replaced by the term below (as discussed in Section 2.3), in order to obtain a better description of the effect of wall shear stresses.

$$- \mu_t \Gamma + \rho \epsilon = - \tau_w \frac{\partial v_t}{\partial s} + \frac{\rho^2 k^2 C_\mu}{\tau_w} \mu \left[ \frac{\partial v_t}{\partial s} \right] \tag{E.13}$$

where $v_t$ is the velocity tangential to the wall and the partial derivative with respect to $\partial s$ represents the distance and direction perpendicular to the wall. The same form of linearization of the source term is applied to the relation above, which yields

$$C_p' = \frac{C_\mu v_t \rho^2 k w \delta s}{J \tau_w} \tag{E.14}$$

and $B = \dfrac{\tau_w v_t \delta s}{J}$ (E.15)

where $\delta s$ represents the distance, perpendicular from the point of the adjoining control volume to the wall.

## APPENDIX F

# THE PRESSURE CORRECTION EQUATION

Following Patankar's SIMPLE method, the continuity equation is used to derive a pressure correction equation. The pressure corrections are used to adjust the pressure field and to calculate velocity corrections. Its main function is, therefore, to enforce the law of conservation of mass by adjusting the velocity distribution to comply with the existing pressure field.

In curvilinear co–ordinates the continuity equation is:

$$\begin{aligned}\frac{\partial}{\partial t}\left[\frac{\rho}{J}\right] = \frac{\partial}{\partial \xi}\left[\frac{1}{J}\left[\xi_x\rho + \xi_y\rho + \xi_z\rho\right]\right] \\ + \frac{\partial}{\partial \eta}\left[\frac{1}{J}\left[\eta_x\rho + \eta_y\rho + \eta_z\rho\right]\right] \\ + \frac{\partial}{\partial \zeta}\left[\frac{1}{J}\left[\zeta_x\rho + \zeta_y\rho + \zeta_z\rho\right]\right]\end{aligned} \qquad \text{(F.1)}$$

and can be discretized using the same principles as for the General Transport Equation in Appendix D. Applying the finite volume approach in discretizing each of the terms as indicated below

$$\frac{\partial}{\partial \xi}\left[\frac{\xi}{J}{}_x\rho u\right] = \frac{\xi_{x_e}\rho_e u_e}{J_e} - \frac{\xi_{x_w}\rho_w u_w}{J_w} \quad ,$$

and by letting

$$B^u_e = \frac{\xi_{x_e}\rho_e}{J_e} \qquad \text{and} \qquad B^u_w = \frac{\xi_{x_w}\rho_w}{J_w} \ ,$$

the continuity equation can be written in the following differenced form:

$$\begin{aligned}\frac{1}{J}\left[\frac{\rho - \rho o}{\Delta t}\right] &= B^u_e u_e - B^u_w u_w + B^v_e v_e - B^v_w v_w + B^w_e w_e - B^w_w w_w \\ &+ B^u_n u_n - B^u_s u_s + B^v_n v_n - B^v_s v_s + B^w_n w_n - B^w_s w_s \\ &+ B^u_b u_b - B^u_f u_f + B^v_b v_b - B^v_f v_f + B^w_b w_b - B^w_f w_f \ .\end{aligned} \tag{F.2}$$

The reader is referred to Appendix D for expressions for the cell wall velocities (eg. $u_P$). Expanding the pressure terms included in those expressions leads to

$$\begin{aligned}a_p u_p = \Sigma\, a_{nb} u_{nb} &+ \left[\xi_{x_w} P_w - \xi_{x_e} P_e\right] \\ &+ \left[\eta_{x_s} P_s - \eta_{x_n} P_n\right] \\ &+ \left[\zeta_{x_f} P_f - \zeta_{x_b} P_b\right] + S^u \ .\end{aligned} \tag{F.3}$$

A similar expression is also valid for the newly calculated velocity values ($u^*_P$) that are obtained after the solution of the momentum equations

$$\begin{aligned}a_p u^*_p = \Sigma\, a_{nb} u^*_{nb} &+ \left[\xi_{x_w} P^*_w - \xi_{x_e} P^*_e\right] \\ &+ \left[\eta_{x_s} P^*_s - \eta_{x_n} P^*_n\right] \\ &+ \left[\zeta_{x_f} P^*_f - \zeta_{x_b} P^*_b\right] + S^u \ .\end{aligned} \tag{F.4}$$

These velocities are, however, not the correct velocities that satisfy the law of conservation of mass, and can ultimately be corrected by assuming that the correct velocities consist of a calculated value plus a correction (u'),

$$u = u^{*} + u' \quad . \tag{F.5}$$

Substituting equations (F.3) and (F.4) into (F.5), an expression for the value of the velocity correction at point p is obtained as a function of the pressure correction derivatives (P'):

$$u'_p = \frac{\left[ -\frac{\partial}{\partial \xi}\left[\frac{\xi_x P'}{J}\right] - \frac{\partial}{\partial \eta}\left[\frac{\eta_x P'}{J}\right] - \frac{\partial}{\partial \zeta}\left[\frac{\zeta_x P'}{J}\right] \right]}{a_p} \quad . \tag{F.6}$$

Similar expressions can be obtained for each of the cell wall velocities in equation (F.2). Substitution and differencing of the pressure correction derivatives at neighbouring grid points yield a discretization equation from which the pressure corrections throughout the flow field can be calculated:

$$a_p P'_p = \Sigma a_{nb} P'_{nb} + a^o_p P'^o_p + S^p \quad . \tag{F.7}$$

The source term $S^p$ gives a indication of the extent to which continuity is satisfied, as it presents the local mass inbalance due to the newly calculated velocity field, and is commonly used in evaluating convergence parameters. Its theoretical value is zero when complete convergence is achieved.

# APPENDIX G

## THE COMPUTER PROGRAM 3DFLO

```
**********************************************
*                                            *
*      3333  DDDD   FFFFF L       000        *
*     3    3 D   D  F     L      0   0       *
*        33  D    D FFF   L     0     0      *
*     3    3 D   D  F     L      0   0       *
*      3333  DDDD   F     LLLLL   000        *
*                                            *
*     THIS PROGRAM SOLVES 3D COMPRESSIBLE    *
*       LAMINAR OR TURBULENT FLOWS IN        *
*         CURVILINEAR CO-ORDINATES           *
*                                            *
*               DEVELOPED BY                 *
*                                            *
*              HERMANN ROLFES                *
*                                            *
**********************************************
```

```
************************************************************************
      Program MAIN
************************************************************************

      INCLUDE 'COMM'

*=======================================================================
*************   CHAPTER 1 : INPUT AND INITIALIZATION   ***************
*=======================================================================

***   Input of fluid and geometrical data

      Call INPUT
      Call GEOM
      Call PROP
      Call OBSTR

***   Choose restart option

      if(irstart.eq.1)Call RESTART
      write(*,*)'Read complete!'

***   Award initial boundary conditions

      Call SETBND
      RESTKEN = 0.1
      RESEPS = 0.1
      if(iturb.eq.1)Call EFFVISC
c     if(irstart.eq.1)Call PROP

***   Interpolate the velocities for the first iteration

      Call INTPOL
      Call STEP
      Call BOUND
      Call INTPOL

***   Calculate mass flow rate at the inflow boundary

      Call MASSIN
      write(*,4)Re
    4 FORMAT(/,1X,'REYNOLDS NUMBER (RE) = ',F9.1)
      if(iglobal.eq.1)write(*,2)fmin
    2 FORMAT(1X,'INLET MASSFLOW = ',E12.4,' (KG/S)')
      if(isetup.eq.1)then
      Call OUTPUT
      Call PLOTOUT
      write(*,*)'**********  Check initial setup  **********'
      stop
      endif

*=======================================================================
*************   CHAPTER 2 : SOLUTION OF THE FLOW FIELD   ***************
*=======================================================================
***   A new time step starts here

      ntime = 0.0
 1000 ntime = ntime + 1
      time = time + delt

      do 20 i = 1,in
       do 20 j = 1,jn
        do 20 k = 1,kn
         vxold(i,j,k) = vx(i,j,k)
         vyold(i,j,k) = vy(i,j,k)
         vzold(i,j,k) = vz(i,j,k)
         ppold(i,j,k) = pp(i,j,k)
         epsold(i,j,k) = eps(i,j,k)
         tkenold(i,j,k) = tken(i,j,k)

   20 continue

***   A new iteration of the total flow field starts here

      if(irstart.ne.1)then
       isweep = 0.0
       iwrite = 0.0
      endif
      write(*,1)isweep,imax,ccheck
    1 format(/,1x,'ISWEEP  =  ',I4,2X,'IMAX  =  ',I4,2X,
     *            'CCHECK  =  ',E10.4,/)
  200 isweep = isweep + 1
      iwrite = iwrite + 1

***   Update current corrected velocity values to star values
***   before solving for new star values

      do 40 i = 1,in
       do 40 j = 1,jn
        do 40 k = 1,kn
         vxstar(i,j,k) = vx(i,j,k)
         vystar(i,j,k) = vy(i,j,k)
         vzstar(i,j,k) = vz(i,j,k)
   40 continue

*---------------------------------------------------------------------
***   STEP 1 : Solve the momentum equation for star w-velocities
*---------------------------------------------------------------------

      Call WCOEFF

***   Sweeping in I - direction

      if(kisweep.eq.0) goto 55
      do 50 k = 2,knm1
       do 50 i = 2,inm1
        do 50 mm = 1,mobs(i,k)
         njend = jjend(mm,i,k)
         njbeg = jjbeg(mm,i,k)
```

```
            Call WPREPI (i,k,njend,njbeg)
            Call TDMAI (i,k,vzstar,njend,njbeg)
   50 continue

***   Sweeping in J - direction

   55 if(kjsweep.eq.0) goto 65
      do 60 k = 2,knm1
       do 60 j = 2,jnm1
        do 60 ll = 1,lobs(j,k)
         niend = iiend(ll,j,k)
         nibeg = iibeg(ll,j,k)
         Call WPREPJ (j,k,niend,nibeg)
         Call TDMAJ (j,k,vzstar,niend,nibeg)
   60 continue

*-----------------------------------------------------------------------
***   STEP 2 : Solve the momentum equation for star v-velocities
*-----------------------------------------------------------------------

   65 Call VCOEFF

***   Sweeping in I - direction

      if(kisweep.eq.0) goto 75
      do 70 k = 2,knm1
       do 70 i = 2,inm1
        do 70 mm = 1,mobs(i,k)
         njend = jjend(mm,i,k)
         njbeg = jjbeg(mm,i,k)
         Call VPREPI (i,k,njend,njbeg)
         Call TDMAI (i,k,vystar,njend,njbeg)
   70 continue

**    Sweeping in J -direction

   75 if(kjsweep.eq.0) goto 85
      do 80 k = 2,knm1
       do 80 j = 2,jnm1
        do 80 ll = 1,lobs(j,k)
         niend = iiend(ll,j,k)
         nibeg = iibeg(ll,j,k)
         Call VPREPJ (j,k,niend,nibeg)
         Call TDMAJ (j,k,vystar,niend,nibeg)
   80 continue

*-----------------------------------------------------------------------
***   STEP 3 : Solve the momentum equation for star u-velocities
*-----------------------------------------------------------------------

   85 Call UCOEFF

***   Sweeping in I - direction

      if(kisweep.eq.0) goto 95
      do 90 k = 2,knm1
       do 90 i = 2,inm1
        do 90 mm = 1,mobs(i,k)
         njend = jjend(mm,i,k)
         njbeg = jjbeg(mm,i,k)
         Call UPREPI (i,k,njend,njbeg)
         Call TDMAI (i,k,vxstar,njend,njbeg)
   90 continue

**    Sweeping in J -direction

   95 if(kjsweep.eq.0) goto 105
      do 100 k = 2,knm1
       do 100 j = 2,jnm1
        do 100 ll = 1,lobs(j,k)
         niend = iiend(ll,j,k)
         nibeg = iibeg(ll,j,k)
         Call UPREPJ (j,k,niend,nibeg)
         Call TDMAJ (j,k,vxstar,niend,nibeg)
  100 continue

*-----------------------------------------------------------------------
***   STEP 4 : Interpolate the velocities at the cell walls with
***            strong pressure - velocity coupling
*-----------------------------------------------------------------------

  105 Call INTPOLV

*-----------------------------------------------------------------------
***   STEP 5 : Solve the pressure correction equation
*-----------------------------------------------------------------------

      Call PCOEFF

      do 106 ip = 1,npress

***   Sweeping in I - direction

      do 120 k = 2,knm1
       do 120 i = 2,inm1
        do 120 mm =1,mobs(i,k)
         njend = jjend(mm,i,k)
         njbeg = jjbeg(mm,i,k)
         Call PPREPI (i,k,njend,njbeg)
         Call TDMAI (i,k,pcor,njend,njbeg)
  120 continue

***   Sweeping in J - direction

      do 130 k = 2,knm1
       do 130 j = 2,jnm1
        do 130 ll = 1,lobs(j,k)
         niend = iiend(ll,j,k)
         nibeg = iibeg(ll,j,k)
         Call PPREPJ (j,k,niend,nibeg)
        Call TDMAJ (j,k,pcor,niend,nibeg)
  130 continue

***   Apply zero gradient boundaries for pressure corrections
***   also at boundaries than are not implicitly solved

  131 do 110 k = 1,kn
       do 110 j = 1,jn
        do 110 i = 1,in

***     Back and front ends

         pcor(i,j,1) = pcor(i,j,2)
         pcor(i,j,kn) = pcor(i,j,knm1)
         if(jbound(i,j,k).eq.0.and.jbound(i,j,k-1).eq.nbnd)
     *    pcor(i,j,k-1) = pcor(i,j,k)
         if(jbound(i,j,k).eq.0.and.jbound(i,j,k+1).eq.nbnd)
     *    pcor(i,j,k+1) = pcor(i,j,k)

  110 continue

  106 continue

*-----------------------------------------------------------------------
***   STEP 6 : Correct the pressures and velocities
*-----------------------------------------------------------------------

      Call ADJUST

*-----------------------------------------------------------------------
***   STEP 7 : Adjust velocities on outflow boundaries
*-----------------------------------------------------------------------

      Call BOUND

*-----------------------------------------------------------------------
***   STEP 8 : Interpolate pressures and velocities
*-----------------------------------------------------------------------

      Call INTPOL
      if(iturb.eq.0)goto 190

*-----------------------------------------------------------------------
***   STEP 9 : Solve the kinetic energy turbulence equation - k
*-----------------------------------------------------------------------

      Call GAMM

      do 135 it = 1,nturb

      Call KCOEFF

***   Sweeping in I - direction

      do 140 k = 2,knm1
      do 140 i = 2,inml
       do 140 mm =1,mobs(i,k)
        njend = jjend(mm,i,k)
        njbeg = jjbeg(mm,i,k)
        Call KPREPI (i,k,njend,njbeg)
        Call TDMAI (i,k,tken,njend,njbeg)
  140 continue

***   Sweeping in J - direction

      do 150 k = 2,knml
       do 150 j = 2,jnml
        do 150 ll = 1,lobs(j,k)
         niend = iiend(ll,j,k)
         nibeg = iibeg(ll,j,k)
         Call KPREPJ (j,k,niend,nibeg)
         Call TDMAJ (j,k,tken,niend,nibeg)
  150 continue

*------------------------------------------------------------------------
***   STEP 10 : Solve the dissiption turbulence equation - epsilon
*------------------------------------------------------------------------

      Call ECOEFF

***   Sweeping in I - direction

      do 160 k = 2,knml
       do 160 i = 2,inml
        do 160 mm =1,mobs(i,k)
         njend = jjend(mm,i,k)
         njbeg = jjbeg(mm,i,k)
         Call EPREPI (i,k,njend,njbeg)
         Call TDMAI (i,k,eps,njend,njbeg)
  160 continue

***   Sweeping in J - direction

      do 170 k = 2,knml
       do 170 j = 2,jnml
        do 170 ll = 1,lobs(j,k)
         niend = iiend(ll,j,k)
         nibeg = iibeg(ll,j,k)
         Call EPREPJ (j,k,niend,nibeg)
         Call TDMAJ (j,k,eps,niend,nibeg)
  170 continue

***   Apply turbulence back and front zero gradients

      do 180 k = 1,kn
       do 180 j = 1,jn
        do 180 i = 1,in

         eps(i,j,kn) = eps(i,j,knml)
         if(jbound(i,j,k).eq.0.and.jbound(i,j,k-1).eq.nbnd)
```

```
     *    eps(i,j,k-1) = eps(i,j,k)
         if(jbound(i,j,k).eq.0.and.jbound(i,j,k+1).eq.nbnd)
     *    eps(i,j,k+1) = eps(i,j,k)
         tken(i,j,kn) = tken(i,j,knm1)
         if(jbound(i,j,k).eq.0.and.jbound(i,j,k-1).eq.nbnd)
     *    tken(i,j,k-1) = tken(i,j,k)
         if(jbound(i,j,k).eq.0.and.jbound(i,j,k+1).eq.nbnd)
     *    tken(i,j,k+1) = tken(i,j,k)
         if(jbound(i,j,k).ne.0)goto 180
         vmuturb(i,j,k) = (cmu*rhop(i,j,k)*tken(i,j,k)**2)/eps(i,j,k)
  180 continue

*-----------------------------------------------------------------------
***   STEP 12 : Interpolate for k and epsilon
*-----------------------------------------------------------------------

      if(igrid.eq.0)Call INTPOLKE

  135 continue

*-----------------------------------------------------------------------
***   STEP 11 : Determine the new effective viscosity
*-----------------------------------------------------------------------

      Call EFFVISC

*-----------------------------------------------------------------------
***   STEP 12 : Test for convergence
*-----------------------------------------------------------------------

      if(iwrite.eq.nwrite)then
       IWRITE = 0.0
  190  WRITE(*,3)ISWEEP,DIFFX,DIFFY,DIFFZ,DIFFP,DIFFTKEN,DIFFEPS
       WRITE(9,3)ISWEEP,DIFFX,DIFFY,DIFFZ,DIFFP,DIFFTKEN,DIFFEPS
    3  FORMAT(1X,I5,1X,'X=',E9.3,1X,'Y=',E9.3,1X,
     * 'Z=',E9.3,1X,'P=',E9.3,1X,'K=',E9.3,1X,'E=',E9.3)
      ENDIF

      rmax = dmax1(diffx,diffy,diffz,diffp)
      if(isweep.lt.imax.and.rmax.gt.ccheck)goto 200
      if(rmax.gt.ccheck)then
      write(*,*)'!!!!!!!  NO CONVERGENCE  !!!!!!!'
      else
      write(*,*)'!!!!!!!  CONVERGENCE ACHIEVED  !!!!!!!'
      endif

***   Output required data

      Call OUTPUT
      Call PLOTOUT

      goto 10000
 9000 write(*,*)'!!! CANNOT OPEN OUTPUT FILE IN MAIN !!!'
      stop
```

```
10000 Close(9)
      Close(19)
      Close(8)

      stop
      end

**************************************************************************
      Subroutine INPUT
**************************************************************************

***   In this subroutine reads the input data

      INCLUDE 'COMM'

      write(*,*)'**********  READING INPUT DATA  **********'

      read(4,'(a)')dummy
      read(4,'(a)')dummy
      read(4,*)in,inm1,jn,jnm1,kn,knm1
      inm2 = in-2
      jnm2 = jn-2
      knm2 = kn-2

      read(4,'(a)')dummy
      read(4,*)kapecl,iglobal,isetup,irstart,iturb,igrid

      read(4,'(a)')dummy
      read(4,*)c1,c2,sigmk,sigme,cmu,cappa,epsinit,tkeninit

      read(4,'(a)')dummy
      read(4,*)kisweep,kjsweep,nwrite,relax,relaxt,ccheck,imax

      read(4,'(a)')dummy
      read(4,*)delt,npress,nturb,kfst,kbst,istep,jstep

      read(4,'(a)')dummy
      read(4,*)rho,vmulam

      read(4,'(a)')dummy
      read(4,*)velmean,chardim

      read(4,'(a)')dummy
      read(4,*)vxinit,vyinit,vzinit

      Re = rho*velmean*chardim/vmulam
      great = 1.0d40
      tiny = 1.0d-30

      goto 10000
 9000 write(*,*)'!!! CANNOT OPEN INPUT DATA FILE IN INPUT !!!'
      stop

10000 return
      end
```

```
************************************************************************
      Subroutine RESTART
************************************************************************

***   In this subroutine reads the restart input data

      INCLUDE 'COMM'

      write(*,*)'**********  READING RESTART DATA   **********'

      read(13,*)ninml,njnml,nknml,isweep
      if(ninml.ne.inml.or.njnml.ne.jnml.or.nknml.ne.knml)then
       write(*,*)'!!!  We have a major f___-up in RESTART  !!!'
       stop
      endif

      if(iturb.eq.1)then
      do 20 k = 2,knml
       do 20 j = 2,jnml
        do 20 i = 2,inml
        read(13,*)x(i,j,k),y(i,j,k),z(i,j,k),tken(i,j,k),eps(i,j,k)
        read(13,*)vx(i,j,k),vy(i,j,k),vz(i,j,k),pp(i,j,k),viscp(i,j,k)
   20 continue
      endif

      if(iturb.eq.0)then
      do 30 k = 2,knml
       do 30 j = 2,jnml
        do 30 i = 2,inml
        read(13,*)x(i,j,k),y(i,j,k),z(i,j,k)
        read(13,*)vx(i,j,k),vy(i,j,k),vz(i,j,k),pp(i,j,k)
   30 continue
      endif

      goto 10000
 9000 write(*,*)'!!! UNABLE TO OPEN RESTART FILE !!!'
      stop

10000 return
      end

************************************************************************
      Subroutine PROP
************************************************************************

***   This subroutine awards fluid properties

      INCLUDE 'COMM'

      do 10 k = 1,kn
       do 10 j = 1,jn
        do 10 i = 1,in

        viscp(i,j,k) = vmulam
        visce(i,j,k) = vmulam
        viscn(i,j,k) = vmulam
        viscb(i,j,k) = vmulam

        rhop(i,j,k) = rho
        rhoo(i,j,k) = rho
        rhoe(i,j,k) = rho
        rhon(i,j,k) = rho
        rhob(i,j,k) = rho

   10 continue

***   Effective viscosity on inflow boundary

      return
      end

************************************************************************
      Subroutine OBSTR
************************************************************************

      INCLUDE 'COMM'

***   This subroutine controls solving around obstructions

      Dimension istart(20),istop(20),jstart(20),jstop(20),
     *          kstart(20),kstop(20)

***---------------------------------------------------------------------
***   Part 1 identifies internal and wall obstructions
***---------------------------------------------------------------------

***   Read obstruction input data

      read(4,'(a)')dummy
      read(4,*)nmatls,nomts,nrfunk

      do 10 k = 1,kn
       do 10 j = 1,jn
        do 10 i = 1,in
        np(i,j,k) = 1
   10 continue

      do 20 nr = 1,nrfunk
       read(4,'(a)')dummy
       read(4,*)istart(nr),istop(nr),jstart(nr),jstop(nr),
     *          kstart(nr),kstop(nr)
   20 continue

***   Find the nodes that fall within this physical obstruction

      do 30 k = 1,kn
       do 30 j = 1,jn
        do 30 i = 1,in
         do 30 nr = 1,nrfunk
```

```
          if(i.ge.istart(nr).and.i.le.istop(nr).and.
     *         j.ge.jstart(nr).and.j.le.jstop(nr).and.
     *         k.ge.kstart(nr).and.k.le.kstop(nr))then
           np(i,j,k) = nmatls
          endif
   30 continue

***-----------------------------------------------------------------------
***   Part 2 determines TDMA limits
***-----------------------------------------------------------------------

***   FOR SWEEPING IN I - DIRECTION

      do 60 k = 2,knml
       do 60 i = 2,inml
        m = 0.0
        itel = 0.0
        do 70 j = 1,jn

***   In case of inflow and outflow boundaries

         if(j.eq.1.and.np(i,j,k).le.nomts)then
          itel = itel + 1
          l = l + 1
          jjbeg(m,i,k) = j
         endif

         if(j.eq.jn.and.np(i,j,k).le.nomts)then
          itel = 0.0
          jjend(m,i,k) = j
         endif
***
         if(np(i,j,k).le.nomts)goto 70

         if(np(i,j,k).gt.nomts.and.np(i,j+1,k).le.nomts.
     *      and.itel.eq.0)then
          itel = itel + 1
          m = m + 1
          jjbeg(m,i,k) = j
          goto 70
         endif

         if(np(i,j,k).gt.nomts.and.np(i,j-1,k).le.nomts.
     *      and.itel.eq.1)then
          itel = 0.0
          jjend(m,i,k) = j
         endif

   70  continue
      mobs(i,k) = m
   60 continue

***   FOR SWEEPING IN J - DIRECTION

      do 80 k = 2,knml
```

```
        do 80 j = 2,jnm1
         l = 0.0
         itel = 0.0
         do 90 i = 1,in

***    In case of and outflow boundaries

          if(i.eq.1.and.np(i,j,k).le.nomts)then
           itel = itel + 1
           l = l + 1
           iibeg(l,j,k) = i
          endif

          if(i.eq.in.and.np(i,j,k).le.nomts)then
           itel = 0.0
           iiend(l,j,k) = i
          endif
***
          if(np(i,j,k).le.nomts)goto 90

          if(np(i,j,k).gt.nomts.and.np(i+1,j,k).le.nomts.
     *       and.itel.eq.0)then
           itel = itel + 1
           l = l + 1
           iibeg(l,j,k) = i
           goto 90
          endif

          if(np(i,j,k).gt.nomts.and.np(i-1,j,k).le.nomts.
     *       and.itel.eq.1)then
           itel = 0.0
           iiend(l,j,k) = i
          endif

   90   continue
        lobs(j,k) = l
   80 continue

***   FOR SWEEPING IN K - DIRECTION

      do 100 j = 2,jnm1
       do 100 i = 2,inm1
        n = 0.0
        itel = 0.0
        do 110 k = 1,kn

***   In case of and outflow boundaries

          if(k.eq.1.and.np(i,j,k).le.nomts)then
           itel = itel + 1
           n = n + 1
           kkbeg(n,i,j) = k
          endif

          if(k.eq.kn.and.np(i,j,k).le.nomts)then
```

```
           itel = 0.0
           kkend(n,i,j) = k
          endif
***
          if(np(i,j,k).le.nomts)goto 110

          if(np(i,j,k).gt.nomts.and.np(i,j,k+1).le.nomts.
     *       and.itel.eq.0)then
           itel = itel + 1
           n = n + 1
           kkbeg(n,i,j) = k
           goto 110
          endif

          if(np(i,j,k).gt.nomts.and.np(i,j,k-1).le.nomts.
     *       and.itel.eq.1)then
           itel = 0.0
           kkend(n,i,j) = k
          endif

  110   continue
        nobs(i,j) = n
  100 continue

***-----------------------------------------------------------------
***   Part 3 produces an output of the obstruction data
***-----------------------------------------------------------------

      kx = 1
      jx = -1
      if(kn.gt.18) kx = 2
      if(kn.gt.36) kx = 3
      if(kn.gt.54) kx = 4
      if(kn.gt.72) kx = 5

      write(9,22)

      do 120 i = 1,in
       write(9,25)i
       do 120 kkk = 1,kx
        write(9,*)' '
        kend = 18*kkk
        kbeg = kend-17
        if(kend.gt.kn) kend = kn
        write(9,24) (kk,kk = kbeg,kend)
        write(9,21)
        do 130 j = jn,1,jx
         write(9,23)j,(np(i,j,kk),kk = kbeg,kend)
  130   continue
  120 continue

   21 format(6x,74('-'))
   22 format(//1x,'FLOW FIELD (NM)'/)
   23 format(1x,I2,'  |',23(1x,I3))
   24 format(1x,' J/K = ',23(I2,2x))
   25 format(/,1x,'I-SURFACE NO. ',I3)

      return
      end

*********************************************************************
      Subroutine UCOEFF
*********************************************************************

***   This subroutine calculates the coefficients to solve the
***   u-velocity components

      INCLUDE 'COMM'

***   Set all the constants equal to zero

      do 10 k = 2,knm1
       do 10 j = 2,jnm1
        do 10 i = 2,inm1
         an(i,j,k) = 0.0
         as(i,j,k) = 0.0
         ae(i,j,k) = 0.0
         aw(i,j,k) = 0.0
         ab(i,j,k) = 0.0
         af(i,j,k) = 0.0
         apu(i,j,k) = 0.0
         apo(i,j,k) = 0.0
         source(i,j,k) = 0.0
         resux(i) = 0.0
         resuy(j) = 0.0
         resuz(k) = 0.0
   10 continue

***   Calculate the coefficients

      diffx = 0.0
      do 20 k = 2,knm1
       do 20 j = 2,jnm1
        do 20 i = 2,inm1

        if(jbound(i,j,k).gt.0)goto 20
        ii = i-1
        jj = j-1
        kk = k-1

        conve = rhoe(i,j,k)*(exixe(i,j,k)*ue(i,j,k)
     *         +             exiye(i,j,k)*ve(i,j,k)
     *         +             exize(i,j,k)*we(i,j,k))/tjace(i,j,k)
        convw = rhoe(ii,j,k)*(exixe(ii,j,k)*ue(ii,j,k)
     *         +              exiye(ii,j,k)*ve(ii,j,k)
     *         +              exize(ii,j,k)*we(ii,j,k))/tjace(ii,j,k)
        convn = rhon(i,j,k)*(etaxn(i,j,k)*un(i,j,k)
     *         +             etayn(i,j,k)*vn(i,j,k)
     *         +             etazn(i,j,k)*wn(i,j,k))/tjacn(i,j,k)
        convs = rhon(i,jj,k)*(etaxn(i,jj,k)*un(i,jj,k)
     *        +               etayn(i,jj,k)*vn(i,jj,k)
     *        +               etazn(i,jj,k)*wn(i,jj,k))/tjacn(i,jj,k)
        convb = rhob(i,j,k)*(zetxb(i,j,k)*ub(i,j,k)
     *        +              zetyb(i,j,k)*vb(i,j,k)
     *        +              zetzb(i,j,k)*wb(i,j,k))/tjacb(i,j,k)
        convf = rhob(i,j,kk)*(zetxb(i,j,kk)*ub(i,j,kk)
     *        +               zetyb(i,j,kk)*vb(i,j,kk)
     *        +               zetzb(i,j,kk)*wb(i,j,kk))/tjacb(i,j,kk)

        diffe = visce(i,j,k)*(exiye(i,j,k)**2 + exize(i,j,k)**2
     *  + 4*(exixe(i,j,k)**2)/3)/(tjace(i,j,k)*delexie(i,j,k))
        diffw = visce(ii,j,k)*(exiye(ii,j,k)**2 + exize(ii,j,k)**2
     *  + 4*(exixe(ii,j,k)**2)/3)/(tjace(ii,j,k)*delexie(ii,j,k))
        diffn = viscn(i,j,k)*(etayn(i,j,k)**2 + etazn(i,j,k)**2
     *  + 4*(etaxn(i,j,k)**2)/3)/(tjacn(i,j,k)*deletan(i,j,k))
        diffs = viscn(i,jj,k)*(etayn(i,jj,k)**2 + etazn(i,jj,k)**2
     *  + 4*(etaxn(i,jj,k)**2)/3)/(tjacn(i,jj,k)*deletan(i,jj,k))
        diffb = viscb(i,j,k)*(zetyb(i,j,k)**2 + zetzb(i,j,k)**2
     *  + 4*(zetxb(i,j,k)**2)/3)/(tjacb(i,j,k)*delzetb(i,j,k))
        difff = viscb(i,j,kk)*(zetyb(i,j,kk)**2 + zetzb(i,j,kk)**2
     *  + 4*(zetxb(i,j,kk)**2)/3)/(tjacb(i,j,kk)*delzetb(i,j,kk))

        pecle = dabs(conve/diffe)
        peclw = dabs(convw/diffw)
        pecln = dabs(convn/diffn)
        pecls = dabs(convs/diffs)
        peclb = dabs(convb/diffb)
        peclf = dabs(convf/difff)

***   Determine the finte difference coefficients

        ae(i,j,k) = diffe*apecl(pecle) + dmax1(-conve,zero)
        aw(i,j,k) = diffw*apecl(peclw) + dmax1( convw,zero)
        an(i,j,k) = diffn*apecl(pecln) + dmax1(-convn,zero)
        as(i,j,k) = diffs*apecl(pecls) + dmax1( convs,zero)
        ab(i,j,k) = diffb*apecl(peclb) + dmax1(-convb,zero)
        af(i,j,k) = difff*apecl(peclf) + dmax1( convf,zero)
        apo(i,j,k) = rhoo(i,j,k)/(delt*tjac(i,j,k))
        apu(i,j,k) = ae(i,j,k) + aw(i,j,k) + an(i,j,k) + as(i,j,k)
     *            + ab(i,j,k) + af(i,j,k) + apo(i,j,k)

***   Calculate the source term coefficients

      bb1e=visce(i,j,k)*(4*exixe(i,j,k)*etaxe(i,j,k)/3
     *    +               exiye(i,j,k)*etaye(i,j,k)
     *    +               exize(i,j,k)*etaze(i,j,k))/tjace(i,j,k)
      bb1w=visce(ii,j,k)*(4*exixe(ii,j,k)*etaxe(ii,j,k)/3
     *    +                exiye(ii,j,k)*etaye(ii,j,k)
     *    +                exize(ii,j,k)*etaze(ii,j,k))/tjace(ii,j,k)
      bb2e=visce(i,j,k)*(4*exixe(i,j,k)*zetxe(i,j,k)/3
     *    +               exiye(i,j,k)*zetye(i,j,k)
     *    +               exize(i,j,k)*zetze(i,j,k))/tjace(i,j,k)
      bb2w=visce(ii,j,k)*(4*exixe(ii,j,k)*zetxe(ii,j,k)/3
     *    +                exiye(ii,j,k)*zetye(ii,j,k)
     *    +                exize(ii,j,k)*zetze(ii,j,k))/tjace(ii,j,k)
```

```
      bb3e=visce(i,j,k)*exiye(i,j,k)*exixe(i,j,k)/(3*tjace(i,j,k))
      bb3w=visce(ii,j,k)*exiye(ii,j,k)*exixe(ii,j,k)/(3*tjace(ii,j,k))
      bb4e=visce(i,j,k)*(exiye(i,j,k)*etaxe(i,j,k)
     *    -           2*exixe(i,j,k)*etaye(i,j,k)/3)/tjace(i,j,k)
      bb4w=visce(ii,j,k)*(exiye(ii,j,k)*etaxe(ii,j,k)
     *    -            2*exixe(ii,j,k)*etaye(ii,j,k)/3)/tjace(ii,j,k)
      bb5e=visce(i,j,k)*(exiye(i,j,k)*zetxe(i,j,k)
     *    -           2*exixe(i,j,k)*zetye(i,j,k)/3)/tjace(i,j,k)
      bb5w=visce(ii,j,k)*(exiye(ii,j,k)*zetxe(ii,j,k)
     *    -            2*exixe(ii,j,k)*zetye(ii,j,k)/3)/tjace(ii,j,k)
      bb6e=visce(i,j,k)*exize(i,j,k)*exixe(i,j,k)/(3*tjace(i,j,k))
      bb6w=visce(ii,j,k)*exize(ii,j,k)*exixe(ii,j,k)/(3*tjace(ii,j,k))
      bb7e=visce(i,j,k)*(exize(i,j,k)*etaxe(i,j,k)
     *    -           2*exixe(i,j,k)*etaze(i,j,k)/3)/tjace(i,j,k)
      bb7w=visce(ii,j,k)*(exize(ii,j,k)*etaxe(ii,j,k)
     *    -            2*exixe(ii,j,k)*etaze(ii,j,k)/3)/tjace(ii,j,k)
      bb8e=visce(i,j,k)*(exize(i,j,k)*zetxe(i,j,k)
     *    -           2*exixe(i,j,k)*zetze(i,j,k)/3)/tjace(i,j,k)
      bb8w=visce(ii,j,k)*(exize(ii,j,k)*zetxe(ii,j,k)
     *    -            2*exixe(ii,j,k)*zetze(ii,j,k)/3)/tjace(ii,j,k)

      bb9n=viscn(i,j,k)*(4*etaxn(i,j,k)*exixn(i,j,k)/3
     *    +                etayn(i,j,k)*exiyn(i,j,k)
     *    +                etazn(i,j,k)*exizn(i,j,k))/tjacn(i,j,k)
      bb9s=viscn(i,jj,k)*(4*etaxn(i,jj,k)*exixn(i,jj,k)/3
     *    +                 etayn(i,jj,k)*exiyn(i,jj,k)
     *    +                 etazn(i,jj,k)*exizn(i,jj,k))/tjacn(i,jj,k)
      bb10n=viscn(i,j,k)*(4*etaxn(i,j,k)*zetxn(i,j,k)/3
     *     +                etayn(i,j,k)*zetyn(i,j,k)
     *     +                etazn(i,j,k)*zetzn(i,j,k))/tjacn(i,j,k)
      bb10s=viscn(i,jj,k)*(4*etaxn(i,jj,k)*zetxn(i,jj,k)/3
     *     +                 etayn(i,jj,k)*zetyn(i,jj,k)
     *     +                 etazn(i,jj,k)*zetzn(i,jj,k))/tjacn(i,jj,k)
      bb11n=viscn(i,j,k)*(etayn(i,j,k)*exixn(i,j,k)
     *     -           2*etaxn(i,j,k)*exiyn(i,j,k)/3)/tjacn(i,j,k)
      bb11s=viscn(i,jj,k)*(etayn(i,jj,k)*exixn(i,jj,k)
     *     -            2*etaxn(i,jj,k)*exiyn(i,jj,k)/3)/tjacn(i,jj,k)
      bb12n=viscn(i,j,k)*etayn(i,j,k)*etaxn(i,j,k)/(3*tjacn(i,j,k))
      bb12s=viscn(i,jj,k)*etayn(i,jj,k)*etaxn(i,jj,k)/(3*tjacn(i,jj,k))
      bb13n=viscn(i,j,k)*(etayn(i,j,k)*zetxn(i,j,k)
     *     -           2*etaxn(i,j,k)*zetyn(i,j,k)/3)/tjacn(i,j,k)
      bb13s=viscn(i,jj,k)*(etayn(i,jj,k)*zetxn(i,jj,k)
     *     -            2*etaxn(i,jj,k)*zetyn(i,jj,k)/3)/tjacn(i,jj,k)
      bb14n=viscn(i,j,k)*(etazn(i,j,k)*exixn(i,j,k)
     *     -           2*etaxn(i,j,k)*exizn(i,j,k)/3)/tjacn(i,j,k)
      bb14s=viscn(i,jj,k)*(etazn(i,jj,k)*exixn(i,jj,k)
     *     -            2*etaxn(i,jj,k)*exizn(i,jj,k)/3)/tjacn(i,jj,k)
      bb15n=viscn(i,j,k)*etazn(i,j,k)*etaxn(i,j,k)/(3*tjacn(i,j,k))
      bb15s=viscn(i,jj,k)*etazn(i,jj,k)*etaxn(i,jj,k)/(3*tjacn(i,jj,k))
      bb16n=viscn(i,j,k)*(etazn(i,j,k)*zetxn(i,j,k)
     *     -           2*etaxn(i,j,k)*zetzn(i,j,k)/3)/tjacn(i,j,k)
      bb16s=viscn(i,jj,k)*(etazn(i,jj,k)*zetxn(i,jj,k)
     *     -            2*etaxn(i,jj,k)*zetzn(i,jj,k)/3)/tjacn(i,jj,k)

      bb17b=viscb(i,j,k)*(4*zetxb(i,j,k)*exixb(i,j,k)/3
     *     +                 zetyb(i,j,k)*exiyb(i,j,k)
     *     +                 zetzb(i,j,k)*exizb(i,j,k))/tjacb(i,j,k)
      bb17f=viscb(i,j,kk)*(4*zetxb(i,j,kk)*exixb(i,j,kk)/3
     *     +                 zetyb(i,j,kk)*exiyb(i,j,kk)
     *     +                 zetzb(i,j,kk)*exizb(i,j,kk))/tjacb(i,j,kk)
      bb18b=viscb(i,j,k)*(4*zetxb(i,j,k)*etaxb(i,j,k)/3
     *     +                 zetyb(i,j,k)*etayb(i,j,k)
     *     +                 zetzb(i,j,k)*etazb(i,j,k))/tjacb(i,j,k)
      bb18f=viscb(i,j,kk)*(4*zetxb(i,j,kk)*etaxb(i,j,kk)/3
     *     +                  zetyb(i,j,kk)*etayb(i,j,kk)
     *     +                  zetzb(i,j,kk)*etazb(i,j,kk))/tjacb(i,j,kk)
      bb19b=viscb(i,j,k)*(zetyb(i,j,k)*exixb(i,j,k)
     *     -             2*zetxb(i,j,k)*exiyb(i,j,k)/3)/tjacb(i,j,k)
      bb19f=viscb(i,j,kk)*(zetyb(i,j,kk)*exixb(i,j,kk)
     *     -              2*zetxb(i,j,kk)*exiyb(i,j,kk)/3)/tjacb(i,j,kk)
      bb20b=viscb(i,j,k)*(zetyb(i,j,k)*etaxb(i,j,k)
     *     -             2*zetxb(i,j,k)*etayb(i,j,k)/3)/tjacb(i,j,k)
      bb20f=viscb(i,j,kk)*(zetyb(i,j,kk)*etaxb(i,j,kk)
     *     -              2*zetxb(i,j,kk)*etayb(i,j,kk)/3)/tjacb(i,j,kk)
      bb21b=viscb(i,j,k)*zetyb(i,j,k)*zetxb(i,j,k)/(3*tjacb(i,j,k))
      bb21f=viscb(i,j,kk)*zetyb(i,j,kk)*zetxb(i,j,kk)/(3*tjacb(i,j,kk))
      bb22b=viscb(i,j,k)*(zetzb(i,j,k)*exixb(i,j,k)
     *     -             2*zetxb(i,j,k)*exizb(i,j,k)/3)/tjacb(i,j,k)
      bb22f=viscb(i,j,kk)*(zetzb(i,j,kk)*exixb(i,j,kk)
     *     -              2*zetxb(i,j,kk)*exizb(i,j,kk)/3)/tjacb(i,j,kk)
      bb23b=viscb(i,j,k)*(zetzb(i,j,k)*etaxb(i,j,k)
     *     -             2*zetxb(i,j,k)*etazb(i,j,k)/3)/tjacb(i,j,k)
      bb23f=viscb(i,j,kk)*(zetzb(i,j,kk)*etaxb(i,j,kk)
     *     -              2*zetxb(i,j,kk)*etazb(i,j,kk)/3)/tjacb(i,j,kk)
      bb24b=viscb(i,j,k)*zetzb(i,j,k)*zetxb(i,j,k)/(3*tjacb(i,j,k))
      bb24f=viscb(i,j,kk)*zetzb(i,j,kk)*zetxb(i,j,kk)/(3*tjacb(i,j,kk))

      ss1 = exixe(ii,j,k)*pie(ii,j,k)/tjace(ii,j,k)
     *    - exixe(i,j,k)*pie(i,j,k)/tjace(i,j,k)
     *    + etaxn(i,jj,k)*pin(i,jj,k)/tjacn(i,jj,k)
     *    - etaxn(i,j,k)*pin(i,j,k)/tjacn(i,j,k)
     *    + zetxb(i,j,kk)*pib(i,j,kk)/tjacb(i,j,kk)
     *    - zetxb(i,j,k)*pib(i,j,k)/tjacb(i,j,k)
      ss2 = bb1e*(une(i,j,k)  - une(i,jj,k))
     *    - bb1w*(une(ii,j,k) - une(ii,jj,k))
     *    + bb2e*(ube(i,j,k)  - ube(i,j,kk))
     *    - bb2w*(ube(ii,j,k) - ube(ii,j,kk))
     *    + bb3e*(vy(i+1,j,k) - vy(i,j,k))/delexie(i,j,k)
     *    - bb3w*(vy(i,j,k)   - vy(ii,j,k))/delexie(ii,j,k)
     *    + bb4e*(vne(i,j,k)  - vne(i,jj,k))
     *    - bb4w*(vne(ii,j,k) - vne(ii,jj,k))
     *    + bb5e*(vbe(i,j,k)  - vbe(i,j,kk))
     *    - bb5w*(vbe(ii,j,k) - vbe(ii,j,kk))
     *    + bb6e*(vz(i+1,j,k) - vz(i,j,k))/delexie(i,j,k)
     *    - bb6w*(vz(i,j,k)   - vz(ii,j,k))/delexie(ii,j,k)
     *    + bb7e*(wne(i,j,k)  - wne(i,jj,k))
     *    - bb7w*(wne(ii,j,k) - wne(ii,jj,k))
     *    + bb8e*(wbe(i,j,k)  - wbe(i,j,kk))
     *    - bb8w*(wbe(ii,j,k) - wbe(ii,j,kk))
```

```
       ss3 = bb9n*(une(i,j,k)   - une(ii,j,k))
     *     - bb9s*(une(i,jj,k)  - une(ii,jj,k))
     *     + bb10n*(unb(i,j,k)  - unb(i,j,kk))
     *     - bb10s*(unb(i,jj,k) - unb(i,jj,kk))
     *     + bb11n*(vne(i,j,k)  - vne(ii,j,k))
     *     - bb11s*(vne(i,jj,k) - vne(ii,jj,k))
     *     + bb12n*(vy(i,j+1,k) - vy(i,j,k))/deletan(i,j,k)
     *     - bb12s*(vy(i,j,k)   - vy(i,jj,k))/deletan(i,jj,k)
     *     + bb13n*(vnb(i,j,k)  - vnb(i,j,kk))
     *     - bb13s*(vnb(i,jj,k) - vnb(i,jj,kk))
     *     + bb14n*(wne(i,j,k)  - wne(ii,j,k))
     *     - bb14s*(wne(i,jj,k) - wne(ii,jj,k))
     *     + bb15n*(vz(i,j+1,k) - vz(i,j,k))/deletan(i,j,k)
     *     - bb15s*(vz(i,j,k)   - vz(i,jj,k))/deletan(i,jj,k)
     *     + bb16n*(wnb(i,j,k)  - wnb(i,j,kk))
     *     - bb16s*(wnb(i,jj,k) - wnb(i,jj,kk))

       ss4 = bb17b*(ube(i,j,k)  - ube(ii,j,k))
     *     - bb17f*(ube(i,j,kk) - ube(ii,j,kk))
     *     + bb18b*(unb(i,j,k)  - unb(i,jj,k))
     *     - bb18f*(unb(i,j,kk) - unb(i,jj,kk))
     *     + bb19b*(vbe(i,j,k)  - vbe(ii,j,k))
     *     - bb19f*(vbe(i,j,kk) - vbe(ii,j,kk))
     *     + bb20b*(vnb(i,j,k)  - vnb(i,jj,k))
     *     - bb20f*(vnb(i,j,kk) - vnb(i,jj,kk))
     *     + bb21b*(vy(i,j,k+1) - vy(i,j,k))/delzetb(i,j,k)
     *     - bb21f*(vy(i,j,k)   - vy(i,j,kk))/delzetb(i,j,kk)
     *     + bb22b*(wbe(i,j,k)  - wbe(ii,j,k))
     *     - bb22f*(wbe(i,j,kk) - wbe(ii,j,kk))
     *     + bb23b*(wnb(i,j,k)  - wnb(i,jj,k))
     *     - bb23f*(wnb(i,j,kk) - wnb(i,jj,kk))
     *     + bb24b*(vz(i,j,k+1) - vz(i,j,k))/delzetb(i,j,k)
     *     - bb24f*(vz(i,j,k)   - vz(i,j,kk))/delzetb(i,j,kk)

       source(i,j,k) = ss1 + ss2 + ss3 + ss4

       res = apu(i,j,k)*vx(i,j,k) - apo(i,j,k)*vx(i,j,k)
     *     - ae(i,j,k)*vx(i+1,j,k) - aw(i,j,k)*vx(ii,j,k)
     *     - an(i,j,k)*vx(i,j+1,k) - as(i,j,k)*vx(i,jj,k)
     *     - ab(i,j,k)*vx(i,j,k+1) - af(i,j,k)*vx(i,j,kk)
     *     - source(i,j,k)
       diffx = diffx + dabs(res)
       resux(i) = resux(i) + dabs(res)
       resuy(j) = resuy(j) + dabs(res)
       resuz(k) = resuz(k) + dabs(res)

    20 continue

       return

c-----------------------------------------------------------------------
       Entry UPREPI (iii,kkk,jne,jnb)
c-----------------------------------------------------------------------

***    This section prepares the TDMA coefficients
***   sweeping in I - direction

      jnb1 = jnb + 1
      jne1 = jne - 1

      jb = jbound(iii,jnb,kkk)
      aaj(jnb) = 1.0
      bbj(jnb) = bu(jb)
      ccj(jnb) = 0.0
      ddj(jnb) = cu(jb)

      do 30 j = jnb1,jne1
        aaj(j) = apu(iii,j,kkk)/relax
        bbj(j) = an(iii,j,kkk)
        ccj(j) = as(iii,j,kkk)
        ddj(j) = ae(iii,j,kkk)*vxstar(iii+1,j,kkk)
     *         + aw(iii,j,kkk)*vxstar(iii-1,j,kkk)
     *         + ab(iii,j,kkk)*vxstar(iii,j,kkk+1)
     *         + af(iii,j,kkk)*vxstar(iii,j,kkk-1)
     *         + apo(iii,j,kkk)*vxold(iii,j,kkk) + source(iii,j,kkk)
     *         + apu(iii,j,kkk)*vxstar(iii,j,kkk)*(1-relax)/relax
   30 continue

      jb = jbound(iii,jne,kkk)
      aaj(jne) = 1.0
      bbj(jne) = 0.0
      ccj(jne) = bu(jb)
      ddj(jne) = cu(jb)

      return

c-----------------------------------------------------------------------
      Entry UPREPJ (jjj,kkk,ine,inb)
c-----------------------------------------------------------------------

***   This section prepares the TDMA coefficients
***   sweeping in J - direction

      inb1 = inb + 1
      ine1 = ine - 1

      jb = jbound(inb,jjj,kkk)
      aai(inb) = 1.0
      bbi(inb) = bu(jb)
      cci(inb) = 0.0
      ddi(inb) = cu(jb)

      do 40 i = inb1,ine1
        aai(i) = apu(i,jjj,kkk)/relax
        bbi(i) = ae(i,jjj,kkk)
        cci(i) = aw(i,jjj,kkk)
        ddi(i) = an(i,jjj,kkk)*vxstar(i,jjj+1,kkk)
     *         + as(i,jjj,kkk)*vxstar(i,jjj-1,kkk)
     *         + ab(i,jjj,kkk)*vxstar(i,jjj,kkk+1)
     *         + af(i,jjj,kkk)*vxstar(i,jjj,kkk-1)
     *         + apo(i,jjj,kkk)*vxold(i,jjj,kkk) + source(i,jjj,kkk)
     *         + apu(i,jjj,kkk)*vxstar(i,jjj,kkk)*(1-relax)/relax
   40 continue

      jb = jbound(ine,jjj,kkk)
      aai(ine) = 1.0
      bbi(ine) = 0.0
      cci(ine) = bu(jb)
      ddi(ine) = cu(jb)

      return
      end

*************************************************************************
      Subroutine VCOEFF
*************************************************************************

***   This subroutine calculates the coefficients to solve the
***   v-velocity components

      INCLUDE 'COMM'

***   Set all the constants equal to zero

      do 10 k = 2,knml
       do 10 j = 2,jnml
        do 10 i = 2,inml
         an(i,j,k) = 0.0
         as(i,j,k) = 0.0
         ae(i,j,k) = 0.0
         aw(i,j,k) = 0.0
         ab(i,j,k) = 0.0
         af(i,j,k) = 0.0
         apv(i,j,k) = 0.0
         apo(i,j,k) = 0.0
         source(i,j,k) =0.0
         resvx(i) = 0.0
         resvy(j) = 0.0
         resvz(k) = 0.0
   10 continue

***   Calculate the coefficients

      diffy = 0.0
      do 20 k = 2,knml
       do 20 j = 2,jnml
        do 20 i = 2,inml

        if(jbound(i,j,k).gt.0)goto 20
        ii = i-1
        jj = j-1
        kk = k-1

        conve = rhoe(i,j,k)*(exixe(i,j,k)*ue(i,j,k)
     *       +             exiye(i,j,k)*ve(i,j,k)
     *        +              exize(i,j,k)*we(i,j,k))/tjace(i,j,k)
        convw = rhoe(ii,j,k)*(exixe(ii,j,k)*ue(ii,j,k)
     *        +               exiye(ii,j,k)*ve(ii,j,k)
     *        +               exize(ii,j,k)*we(ii,j,k))/tjace(ii,j,k)
        convn = rhon(i,j,k)*(etaxn(i,j,k)*un(i,j,k)
     *        +              etayn(i,j,k)*vn(i,j,k)
     *        +              etazn(i,j,k)*wn(i,j,k))/tjacn(i,j,k)
        convs = rhon(i,jj,k)*(etaxn(i,jj,k)*un(i,jj,k)
     *        +               etayn(i,jj,k)*vn(i,jj,k)
     *        +               etazn(i,jj,k)*wn(i,jj,k))/tjacn(i,jj,k)
        convb = rhob(i,j,k)*(zetxb(i,j,k)*ub(i,j,k)
     *        +              zetyb(i,j,k)*vb(i,j,k)
     *        +              zetzb(i,j,k)*wb(i,j,k))/tjacb(i,j,k)
        convf = rhob(i,j,kk)*(zetxb(i,j,kk)*ub(i,j,kk)
     *        +               zetyb(i,j,kk)*vb(i,j,kk)
     *        +               zetzb(i,j,kk)*wb(i,j,kk))/tjacb(i,j,kk)

        diffe = visce(i,j,k)*(exixe(i,j,k)**2 + exize(i,j,k)**2
     *  + 4*(exiye(i,j,k)**2)/3)/(tjace(i,j,k)*delexie(i,j,k))
        diffw = visce(ii,j,k)*(exixe(ii,j,k)**2 + exize(ii,j,k)**2
     *  + 4*(exiye(ii,j,k)**2)/3)/(tjace(ii,j,k)*delexie(ii,j,k))
        diffn = viscn(i,j,k)*(etaxn(i,j,k)**2 + etazn(i,j,k)**2
     *  + 4*(etayn(i,j,k)**2)/3)/(tjacn(i,j,k)*deletan(i,j,k))
        diffs = viscn(i,jj,k)*(etaxn(i,jj,k)**2 + etazn(i,jj,k)**2
     *  + 4*(etayn(i,jj,k)**2)/3)/(tjacn(i,jj,k)*deletan(i,jj,k))
        diffb = viscb(i,j,k)*(zetxb(i,j,k)**2 + zetzb(i,j,k)**2
     *  + 4*(zetyb(i,j,k)**2)/3)/(tjacb(i,j,k)*delzetb(i,j,k))
        difff = viscb(i,j,kk)*(zetxb(i,j,kk)**2 + zetzb(i,j,kk)**2
     *  + 4*(zetyb(i,j,kk)**2)/3)/(tjacb(i,j,kk)*delzetb(i,j,kk))

        pecle = dabs(conve/diffe)
        peclw = dabs(convw/diffw)
        pecln = dabs(convn/diffn)
        pecls = dabs(convs/diffs)
        peclb = dabs(convb/diffb)
        peclf = dabs(convf/difff)

***   Determine the finte difference coefficients

        ae(i,j,k) = diffe*apecl(pecle) + dmax1(-conve,zero)
        aw(i,j,k) = diffw*apecl(peclw) + dmax1( convw,zero)
        an(i,j,k) = diffn*apecl(pecln) + dmax1(-convn,zero)
        as(i,j,k) = diffs*apecl(pecls) + dmax1( convs,zero)
        ab(i,j,k) = diffb*apecl(peclb) + dmax1(-convb,zero)
        af(i,j,k) = difff*apecl(peclf) + dmax1( convf,zero)
        apo(i,j,k) = rhoo(i,j,k)/(delt*tjac(i,j,k))
        apv(i,j,k) = ae(i,j,k) + aw(i,j,k) + an(i,j,k) + as(i,j,k)
     *             + ab(i,j,k) + af(i,j,k) + apo(i,j,k)

***   Calculate the source term coefficients

      bble=visce(i,j,k)*(4*exiye(i,j,k)*etaye(i,j,k)/3
     *    +               exixe(i,j,k)*etaxe(i,j,k)
     *    +               exize(i,j,k)*etaze(i,j,k))/tjace(i,j,k)
      bblw=visce(ii,j,k)*(4*exiye(ii,j,k)*etaye(ii,j,k)/3
     *    +                 exixe(ii,j,k)*etaxe(ii,j,k)
     *    +                 exize(ii,j,k)*etaze(ii,j,k))/tjace(ii,j,k)
      bb2e=visce(i,j,k)*(4*exiye(i,j,k)*zetye(i,j,k)/3
     *    +                exixe(i,j,k)*zetxe(i,j,k)
     *    +                exize(i,j,k)*zetze(i,j,k))/tjace(i,j,k)
      bb2w=visce(ii,j,k)*(4*exiye(ii,j,k)*zetye(ii,j,k)/3
     *    +                 exixe(ii,j,k)*zetxe(ii,j,k)
     *    +                 exize(ii,j,k)*zetze(ii,j,k))/tjace(ii,j,k)
      bb3e=visce(i,j,k)*exiye(i,j,k)*exixe(i,j,k)/(3*tjace(i,j,k))
      bb3w=visce(ii,j,k)*exiye(ii,j,k)*exixe(ii,j,k)/(3*tjace(ii,j,k))
      bb4e=visce(i,j,k)*(exixe(i,j,k)*etaye(i,j,k)
     *    -            2*exiye(i,j,k)*etaxe(i,j,k)/3)/tjace(i,j,k)
      bb4w=visce(ii,j,k)*(exixe(ii,j,k)*etaye(ii,j,k)
     *    -             2*exiye(ii,j,k)*etaxe(ii,j,k)/3)/tjace(ii,j,k)
      bb5e=visce(i,j,k)*(exixe(i,j,k)*zetye(i,j,k)
     *    -            2*exiye(i,j,k)*zetxe(i,j,k)/3)/tjace(i,j,k)
      bb5w=visce(ii,j,k)*(exixe(ii,j,k)*zetye(ii,j,k)
     *    -             2*exiye(ii,j,k)*zetxe(ii,j,k)/3)/tjace(ii,j,k)
      bb6e=visce(i,j,k)*exize(i,j,k)*exiye(i,j,k)/(3*tjace(i,j,k))
      bb6w=visce(ii,j,k)*exize(ii,j,k)*exiye(ii,j,k)/(3*tjace(ii,j,k))
      bb7e=visce(i,j,k)*(exize(i,j,k)*etaye(i,j,k)
     *    -            2*exiye(i,j,k)*etaze(i,j,k)/3)/tjace(i,j,k)
      bb7w=visce(ii,j,k)*(exize(ii,j,k)*etaye(ii,j,k)
     *    -             2*exiye(ii,j,k)*etaze(ii,j,k)/3)/tjace(ii,j,k)
      bb8e=visce(i,j,k)*(exize(i,j,k)*zetye(i,j,k)
     *    -            2*exiye(i,j,k)*zetze(i,j,k)/3)/tjace(i,j,k)
      bb8w=visce(ii,j,k)*(exize(ii,j,k)*zetye(ii,j,k)
     *    -             2*exiye(ii,j,k)*zetze(ii,j,k)/3)/tjace(ii,j,k)

      bb9n=viscn(i,j,k)*(4*etayn(i,j,k)*exiyn(i,j,k)/3
     *    +               etaxn(i,j,k)*exixn(i,j,k)
     *    +               etazn(i,j,k)*exizn(i,j,k))/tjacn(i,j,k)
      bb9s=viscn(i,jj,k)*(4*etayn(i,jj,k)*exiyn(i,jj,k)/3
     *    +                etaxn(i,jj,k)*exixn(i,jj,k)
     *    +                etazn(i,jj,k)*exizn(i,jj,k))/tjacn(i,jj,k)
      bb10n=viscn(i,j,k)*(4*etayn(i,j,k)*zetyn(i,j,k)/3
     *     +               etaxn(i,j,k)*zetxn(i,j,k)
     *     +               etazn(i,j,k)*zetzn(i,j,k))/tjacn(i,j,k)
      bb10s=viscn(i,jj,k)*(4*etayn(i,jj,k)*zetyn(i,jj,k)/3
     *     +                etaxn(i,jj,k)*zetxn(i,jj,k)
     *     +                etazn(i,jj,k)*zetzn(i,jj,k))/tjacn(i,jj,k)
      bb11n=viscn(i,j,k)*(etaxn(i,j,k)*exiyn(i,j,k)
     *     -            2*etayn(i,j,k)*exixn(i,j,k)/3)/tjacn(i,j,k)
      bb11s=viscn(i,jj,k)*(etaxn(i,jj,k)*exiyn(i,jj,k)
     *     -             2*etayn(i,jj,k)*exixn(i,jj,k)/3)/tjacn(i,jj,k)
      bb12n=viscn(i,j,k)*etayn(i,j,k)*etaxn(i,j,k)/(3*tjacn(i,j,k))
      bb12s=viscn(i,jj,k)*etayn(i,jj,k)*etaxn(i,jj,k)/(3*tjacn(i,jj,k))
      bb13n=viscn(i,j,k)*(etaxn(i,j,k)*zetyn(i,j,k)
     *     -            2*etayn(i,j,k)*zetxn(i,j,k)/3)/tjacn(i,j,k)
      bb13s=viscn(i,jj,k)*(etaxn(i,jj,k)*zetyn(i,jj,k)
     *     -             2*etayn(i,jj,k)*zetxn(i,jj,k)/3)/tjacn(i,jj,k)
      bb14n=viscn(i,j,k)*(etazn(i,j,k)*exiyn(i,j,k)
     *     -            2*etayn(i,j,k)*exizn(i,j,k)/3)/tjacn(i,j,k)
      bb14s=viscn(i,jj,k)*(etazn(i,jj,k)*exiyn(i,jj,k)
     *     -             2*etayn(i,jj,k)*exizn(i,jj,k)/3)/tjacn(i,jj,k)
```

```
      bb15n=viscn(i,j,k)*etazn(i,j,k)*etayn(i,j,k)/(3*tjacn(i,j,k))
      bb15s=viscn(i,jj,k)*etazn(i,jj,k)*etayn(i,jj,k)/(3*tjacn(i,jj,k))
      bb16n=viscn(i,j,k)*(etazn(i,j,k)*zetyn(i,j,k)
     *     -              2*etayn(i,j,k)*zetzn(i,j,k)/3)/tjacn(i,j,k)
      bb16s=viscn(i,jj,k)*(etazn(i,jj,k)*zetyn(i,jj,k)
     *     -               2*etayn(i,jj,k)*zetzn(i,jj,k)/3)/tjacn(i,jj,k)

      bb17b=viscb(i,j,k)*(4*zetyb(i,j,k)*exiyb(i,j,k)/3
     *     +                zetxb(i,j,k)*exixb(i,j,k)
     *     +                zetzb(i,j,k)*exizb(i,j,k))/tjacb(i,j,k)
      bb17f=viscb(i,j,kk)*(4*zetyb(i,j,kk)*exiyb(i,j,kk)/3
     *     +                 zetxb(i,j,kk)*exixb(i,j,kk)
     *     +                 zetzb(i,j,kk)*exizb(i,j,kk))/tjacb(i,j,kk)
      bb18b=viscb(i,j,k)*(4*zetyb(i,j,k)*etayb(i,j,k)/3
     *     +                zetxb(i,j,k)*etaxb(i,j,k)
     *     +                zetzb(i,j,k)*etazb(i,j,k))/tjacb(i,j,k)
      bb18f=viscb(i,j,kk)*(4*zetyb(i,j,kk)*etayb(i,j,kk)/3
     *     +                 zetxb(i,j,kk)*etaxb(i,j,kk)
     *     +                 zetzb(i,j,kk)*etazb(i,j,kk))/tjacb(i,j,kk)
      bb19b=viscb(i,j,k)*(zetxb(i,j,k)*exiyb(i,j,k)
     *     -              2*zetyb(i,j,k)*exixb(i,j,k)/3)/tjacb(i,j,k)
      bb19f=viscb(i,j,kk)*(zetxb(i,j,kk)*exiyb(i,j,kk)
     *     -               2*zetyb(i,j,kk)*exixb(i,j,kk)/3)/tjacb(i,j,kk)
      bb20b=viscb(i,j,k)*(zetxb(i,j,k)*etayb(i,j,k)
     *     -              2*zetyb(i,j,k)*etaxb(i,j,k)/3)/tjacb(i,j,k)
      bb20f=viscb(i,j,kk)*(zetxb(i,j,kk)*etayb(i,j,kk)
     *     -               2*zetyb(i,j,kk)*etaxb(i,j,kk)/3)/tjacb(i,j,kk)
      bb21b=viscb(i,j,k)*zetyb(i,j,k)*zetxb(i,j,k)/(3*tjacb(i,j,k))
      bb21f=viscb(i,j,kk)*zetyb(i,j,kk)*zetxb(i,j,kk)/(3*tjacb(i,j,kk))
      bb22b=viscb(i,j,k)*(zetzb(i,j,k)*exiyb(i,j,k)
     *     -              2*zetyb(i,j,k)*exizb(i,j,k)/3)/tjacb(i,j,k)
      bb22f=viscb(i,j,kk)*(zetzb(i,j,kk)*exiyb(i,j,kk)
     *     -               2*zetyb(i,j,kk)*exizb(i,j,kk)/3)/tjacb(i,j,kk)
      bb23b=viscb(i,j,k)*(zetzb(i,j,k)*etayb(i,j,k)
     *     -              2*zetyb(i,j,k)*etazb(i,j,k)/3)/tjacb(i,j,k)
      bb23f=viscb(i,j,kk)*(zetzb(i,j,kk)*etayb(i,j,kk)
     *     -               2*zetyb(i,j,kk)*etazb(i,j,kk)/3)/tjacb(i,j,kk)
      bb24b=viscb(i,j,k)*zetzb(i,j,k)*zetyb(i,j,k)/(3*tjacb(i,j,k))
      bb24f=viscb(i,j,kk)*zetzb(i,j,kk)*zetyb(i,j,kk)/(3*tjacb(i,j,kk))

      ss1 = exiye(ii,j,k)*pie(ii,j,k)/tjace(ii,j,k)
     *    - exiye(i,j,k)*pie(i,j,k)/tjace(i,j,k)
     *    + etayn(i,jj,k)*pin(i,jj,k)/tjacn(i,jj,k)
     *    - etayn(i,j,k)*pin(i,j,k)/tjacn(i,j,k)
     *    + zetyb(i,j,kk)*pib(i,j,kk)/tjacb(i,j,kk)
     *    - zetyb(i,j,k)*pib(i,j,k)/tjacb(i,j,k)
      ss2 = bb1e*(vne(i,j,k)   - vne(i,jj,k))
     *    - bb1w*(vne(ii,j,k)  - vne(ii,jj,k))
     *    + bb2e*(vbe(i,j,k)   - vbe(i,j,kk))
     *    - bb2w*(vbe(ii,j,k)  - vbe(ii,j,kk))
     *    + bb3e*(vx(i+1,j,k)  - vx(i,j,k))/delexie(i,j,k)
     *    - bb3w*(vx(i,j,k)    - vx(ii,j,k))/delexie(ii,j,k)
     *    + bb4e*(une(i,j,k)   - une(i,jj,k))
     *    - bb4w*(une(ii,j,k)  - une(ii,jj,k))
     *    + bb5e*(ube(i,j,k)   - ube(i,j,kk))
     *    - bb5w*(ube(ii,j,k) - ube(ii,j,kk))
     *    + bb6e*(vz(i+1,j,k) - vz(i,j,k))/delexie(i,j,k)
     *    - bb6w*(vz(i,j,k)   - vz(ii,j,k))/delexie(ii,j,k)
     *    + bb7e*(wne(i,j,k)  - wne(i,jj,k))
     *    - bb7w*(wne(ii,j,k) - wne(ii,jj,k))
     *    + bb8e*(wbe(i,j,k)  - wbe(i,j,kk))
     *    - bb8w*(wbe(ii,j,k) - wbe(ii,j,kk))

      ss3 = bb9n*(vne(i,j,k)   - vne(ii,j,k))
     *    - bb9s*(vne(i,jj,k)  - vne(ii,jj,k))
     *    + bb10n*(vnb(i,j,k)  - vnb(i,j,kk))
     *    - bb10s*(vnb(i,jj,k) - vnb(i,jj,kk))
     *    + bb11n*(une(i,j,k)  - une(ii,j,k))
     *    - bb11s*(une(i,jj,k) - une(ii,jj,k))
     *    + bb12n*(vx(i,j+1,k) - vx(i,j,k))/deletan(i,j,k)
     *    - bb12s*(vx(i,j,k)   - vx(i,jj,k))/deletan(i,jj,k)
     *    + bb13n*(unb(i,j,k)  - unb(i,j,kk))
     *    - bb13s*(unb(i,jj,k) - unb(i,jj,kk))
     *    + bb14n*(wne(i,j,k)  - wne(ii,j,k))
     *    - bb14s*(wne(i,jj,k) - wne(ii,jj,k))
     *    + bb15n*(vz(i,j+1,k) - vz(i,j,k))/deletan(i,j,k)
     *    - bb15s*(vz(i,j,k)   - vz(i,jj,k))/deletan(i,jj,k)
     *    + bb16n*(wnb(i,j,k)  - wnb(i,j,kk))
     *    - bb16s*(wnb(i,jj,k) - wnb(i,jj,kk))

      ss4 = bb17b*(vbe(i,j,k)  - vbe(ii,j,k))
     *    - bb17f*(vbe(i,j,kk) - vbe(ii,j,kk))
     *    + bb18b*(vnb(i,j,k)  - vnb(i,jj,k))
     *    - bb18f*(vnb(i,j,kk) - vnb(i,jj,kk))
     *    + bb19b*(ube(i,j,k)  - ube(ii,j,k))
     *    - bb19f*(ube(i,j,kk) - ube(ii,j,kk))
     *    + bb20b*(unb(i,j,k)  - unb(i,jj,k))
     *    - bb20f*(unb(i,j,kk) - unb(i,jj,kk))
     *    + bb21b*(vx(i,j,k+1) - vx(i,j,k))/delzetb(i,j,k)
     *    - bb21f*(vx(i,j,k)   - vx(i,j,kk))/delzetb(i,j,kk)
     *    + bb22b*(wbe(i,j,k)  - wbe(ii,j,k))
     *    - bb22f*(wbe(i,j,kk) - wbe(ii,j,kk))
     *    + bb23b*(wnb(i,j,k)  - wnb(i,jj,k))
     *    - bb23f*(wnb(i,j,kk) - wnb(i,jj,kk))
     *    + bb24b*(vz(i,j,k+1) - vz(i,j,k))/delzetb(i,j,k)
     *    - bb24f*(vz(i,j,k)   - vz(i,j,kk))/delzetb(i,j,kk)

      source(i,j,k) = ss1 + ss2 + ss3 + ss4

      res = apv(i,j,k)*vy(i,j,k) - apo(i,j,k)*vy(i,j,k)
     *    - ae(i,j,k)*vy(i+1,j,k) - aw(i,j,k)*vy(ii,j,k)
     *    - an(i,j,k)*vy(i,j+1,k) - as(i,j,k)*vy(i,jj,k)
     *    - ab(i,j,k)*vy(i,j,k+1) - af(i,j,k)*vy(i,j,kk)
     *    - source(i,j,k)
      diffy = diffy + dabs(res)
      resvx(i) = resvx(i) + dabs(res)
      resvy(j) = resvy(j) + dabs(res)
      resvz(k) = resvz(k) + dabs(res)

   20 continue
```

```
      return

c------------------------------------------------------------------
      Entry VPREPI (iii,kkk,jne,jnb)
c------------------------------------------------------------------

***   This section prepares the TDMA coefficients
***   sweeping in I - direction

      jnb1 = jnb + 1
      jne1 = jne - 1

      jb = jbound(iii,jnb,kkk)
      aaj(jnb) = 1.0
      bbj(jnb) = bv(jb)
      ccj(jnb) = 0.0
      ddj(jnb) = cv(jb)

      do 30 j = jnb1,jne1
        aaj(j) = apv(iii,j,kkk)/relax
        bbj(j) = an(iii,j,kkk)
        ccj(j) = as(iii,j,kkk)
        ddj(j) = ae(iii,j,kkk)*vystar(iii+1,j,kkk)
     *          + aw(iii,j,kkk)*vystar(iii-1,j,kkk)
     *          + ab(iii,j,kkk)*vystar(iii,j,kkk+1)
     *          + af(iii,j,kkk)*vystar(iii,j,kkk-1)
     *          + apo(iii,j,kkk)*vyold(iii,j,kkk) + source(iii,j,kkk)
     *          + apv(iii,j,kkk)*vystar(iii,j,kkk)*(1-relax)/relax
   30 continue

      jb = jbound(iii,jne,kkk)
      aaj(jne) = 1.0
      bbj(jne) = 0.0
      ccj(jne) = bv(jb)
      ddj(jne) = cv(jb)

      return

c------------------------------------------------------------------
      Entry VPREPJ (jjj,kkk,ine,inb)
c------------------------------------------------------------------

***   This section prepares the TDMA coefficients
***   sweeping in J - direction

      inb1 = inb + 1
      ine1 = ine - 1

      jb = jbound(inb,jjj,kkk)
      aai(inb) = 1.0
      bbi(inb) = bv(jb)
      cci(inb) = 0.0
      ddi(inb) = cv(jb)
      do 40 i = inbl,inel
        aai(i) = apv(i,jjj,kkk)/relax
        bbi(i) = ae(i,jjj,kkk)
        cci(i) = aw(i,jjj,kkk)
        ddi(i) = an(i,jjj,kkk)*vystar(i,jjj+1,kkk)
     *         + as(i,jjj,kkk)*vystar(i,jjj-1,kkk)
     *         + ab(i,jjj,kkk)*vystar(i,jjj,kkk+1)
     *         + af(i,jjj,kkk)*vystar(i,jjj,kkk-1)
     *         + apo(i,jjj,kkk)*vyold(i,jjj,kkk) + source(i,jjj,kkk)
     *         + apv(i,jjj,kkk)*vystar(i,jjj,kkk)*(1-relax)/relax
   40 continue

      jb = jbound(ine,jjj,kkk)
      aai(ine) = 1.0
      bbi(ine) = 0.0
      cci(ine) = bv(jb)
      ddi(ine) = cv(jb)

      return
      end

************************************************************************
      Subroutine WCOEFF
************************************************************************

***   This subroutine calculates the coefficients to solve the
***   w-velocity components

      INCLUDE 'COMM'

***   Set all the constants equal to zero

      do 10 k = 2,knm1
       do 10 j = 2,jnm1
        do 10 i = 2,inm1
         an(i,j,k) = 0.0
         as(i,j,k) = 0.0
         ae(i,j,k) = 0.0
         aw(i,j,k) = 0.0
         ab(i,j,k) = 0.0
         af(i,j,k) = 0.0
         apw(i,j,k) = 0.0
         apo(i,j,k) = 0.0
         source(i,j,k) =0.0
         reswx(i) = 0.0
         reswy(j) = 0.0
         reswz(k) = 0.0
   10 continue

***   Calculate the coefficients

      diffz = 0.0
      do 20 k = 2,knm1
       do 20 j = 2,jnm1
        do 20 i = 2,inm1
      if(jbound(i,j,k).gt.0)goto 20
      ii = i-1
      jj = j-1
      kk = k-1

      conve = rhoe(i,j,k)*(exixe(i,j,k)*ue(i,j,k)
     *      +             exiye(i,j,k)*ve(i,j,k)
     *      +             exize(i,j,k)*we(i,j,k))/tjace(i,j,k)
      convw = rhoe(ii,j,k)*(exixe(ii,j,k)*ue(ii,j,k)
     *      +              exiye(ii,j,k)*ve(ii,j,k)
     *      +              exize(ii,j,k)*we(ii,j,k))/tjace(ii,j,k)
      convn = rhon(i,j,k)*(etaxn(i,j,k)*un(i,j,k)
     *      +             etayn(i,j,k)*vn(i,j,k)
     *      +             etazn(i,j,k)*wn(i,j,k))/tjacn(i,j,k)
      convs = rhon(i,jj,k)*(etaxn(i,jj,k)*un(i,jj,k)
     *      +              etayn(i,jj,k)*vn(i,jj,k)
     *      +              etazn(i,jj,k)*wn(i,jj,k))/tjacn(i,jj,k)
      convb = rhob(i,j,k)*(zetxb(i,j,k)*ub(i,j,k)
     *      +             zetyb(i,j,k)*vb(i,j,k)
     *      +             zetzb(i,j,k)*wb(i,j,k))/tjacb(i,j,k)
      convf = rhob(i,j,kk)*(zetxb(i,j,kk)*ub(i,j,kk)
     *      +              zetyb(i,j,kk)*vb(i,j,kk)
     *      +              zetzb(i,j,kk)*wb(i,j,kk))/tjacb(i,j,kk)

      diffe = visce(i,j,k)*(exixe(i,j,k)**2 + exiye(i,j,k)**2
     * + 4*(exize(i,j,k)**2)/3)/(tjace(i,j,k)*delexie(i,j,k))
      diffw = visce(ii,j,k)*(exixe(ii,j,k)**2 + exiye(ii,j,k)**2
     * + 4*(exize(ii,j,k)**2)/3)/(tjace(ii,j,k)*delexie(ii,j,k))
      diffn = viscn(i,j,k)*(etaxn(i,j,k)**2 + etayn(i,j,k)**2
     * + 4*(etazn(i,j,k)**2)/3)/(tjacn(i,j,k)*deletan(i,j,k))
      diffs = viscn(i,jj,k)*(etaxn(i,jj,k)**2 + etayn(i,jj,k)**2
     * + 4*(etazn(i,jj,k)**2)/3)/(tjacn(i,jj,k)*deletan(i,jj,k))
      diffb = viscb(i,j,k)*(zetxb(i,j,k)**2 + zetyb(i,j,k)**2
     * + 4*(zetzb(i,j,k)**2)/3)/(tjacb(i,j,k)*delzetb(i,j,k))
      difff = viscb(i,j,kk)*(zetxb(i,j,kk)**2 + zetyb(i,j,kk)**2
     * + 4*(zetzb(i,j,kk)**2)/3)/(tjacb(i,j,kk)*delzetb(i,j,kk))

      pecle = dabs(conve/diffe)
      peclw = dabs(convw/diffw)
      pecln = dabs(convn/diffn)
      pecls = dabs(convs/diffs)
      peclb = dabs(convb/diffb)
      peclf = dabs(convf/difff)

***   Determine the finte difference coefficients

      ae(i,j,k) = diffe*apecl(pecle) + dmax1(-conve,zero)
      aw(i,j,k) = diffw*apecl(peclw) + dmax1( convw,zero)
      an(i,j,k) = diffn*apecl(pecln) + dmax1(-convn,zero)
      as(i,j,k) = diffs*apecl(pecls) + dmax1( convs,zero)
      ab(i,j,k) = diffb*apecl(peclb) + dmax1(-convb,zero)
      af(i,j,k) = difff*apecl(peclf) + dmax1( convf,zero)
      apo(i,j,k) = rhoo(i,j,k)/(delt*tjac(i,j,k))
```

```
***   Adjust main flow direction for turbulent flows near the wall
***   by making use of a additional shear force term

        apx = 0.0
        if(iturb.ne.1) goto 15
        if(vz(i,j,k).lt.0.0001)goto 15

        if(jbound(i,j,k).eq.0.and.jbound(i+1,j,k).eq.nbnd)then
         exiss = (exix(i,j,k)+exiy(i,j,k)+exiz(i,j,k))/tjac(i,j,k)
         apx = dabs(tauw(i,j,k)*exiss/vz(i,j,k))
        endif
        if(jbound(i,j,k).eq.0.and.jbound(i-1,j,k).eq.nbnd)then
         exiss = (exix(i,j,k)+exiy(i,j,k)+exiz(i,j,k))/tjac(i,j,k)
         apx = dabs(tauw(i,j,k)*exiss/vz(i,j,k))
        endif
        if(jbound(i,j,k).eq.0.and.jbound(i,j+1,k).eq.nbnd)then
         etass = (etax(i,j,k)+etay(i,j,k)+etaz(i,j,k))/tjac(i,j,k)
         apx = dabs(tauw(i,j,k)*etass/vz(i,j,k))
        endif
        if(jbound(i,j,k).eq.0.and.jbound(i,j-1,k).eq.nbnd)then
         etass = (etax(i,j,k)+etay(i,j,k)+etaz(i,j,k))/tjac(i,j,k)
         apx = dabs(tauw(i,j,k)*etass/vz(i,j,k))
        endif

   15   apw(i,j,k) = ae(i,j,k) + aw(i,j,k) + an(i,j,k) + as(i,j,k)
     *             + ab(i,j,k) + af(i,j,k) + apo(i,j,k) + apx

***   Calculate the source term coefficients

       bb1e=visce(i,j,k)*(4*exize(i,j,k)*etaze(i,j,k)/3
     *    +                 exixe(i,j,k)*etaxe(i,j,k)
     *    +                 exiye(i,j,k)*etaye(i,j,k))/tjace(i,j,k)
       bb1w=visce(ii,j,k)*(4*exize(ii,j,k)*etaze(ii,j,k)/3
     *    +                  exixe(ii,j,k)*etaxe(ii,j,k)
     *    +                  exiye(ii,j,k)*etaye(ii,j,k))/tjace(ii,j,k)
       bb2e=visce(i,j,k)*(4*exize(i,j,k)*zetze(i,j,k)/3
     *    +                 exixe(i,j,k)*zetxe(i,j,k)
     *    +                 exiye(i,j,k)*zetye(i,j,k))/tjace(i,j,k)
       bb2w=visce(ii,j,k)*(4*exize(ii,j,k)*zetze(ii,j,k)/3
     *    +                  exixe(ii,j,k)*zetxe(ii,j,k)
     *    +                  exiye(ii,j,k)*zetye(ii,j,k))/tjace(ii,j,k)
       bb3e=visce(i,j,k)*exize(i,j,k)*exixe(i,j,k)/(3*tjace(i,j,k))
       bb3w=visce(ii,j,k)*exize(ii,j,k)*exixe(ii,j,k)/(3*tjace(ii,j,k))
       bb4e=visce(i,j,k)*(exixe(i,j,k)*etaze(i,j,k)
     *    -             2*exize(i,j,k)*etaxe(i,j,k)/3)/tjace(i,j,k)
       bb4w=visce(ii,j,k)*(exixe(ii,j,k)*etaze(ii,j,k)
     *    -              2*exize(ii,j,k)*etaxe(ii,j,k)/3)/tjace(ii,j,k)
       bb5e=visce(i,j,k)*(exixe(i,j,k)*zetze(i,j,k)
     *    -             2*exize(i,j,k)*zetxe(i,j,k)/3)/tjace(i,j,k)
       bb5w=visce(ii,j,k)*(exixe(ii,j,k)*zetze(ii,j,k)
     *    -              2*exize(ii,j,k)*zetxe(ii,j,k)/3)/tjace(ii,j,k)
       bb6e=visce(i,j,k)*exize(i,j,k)*exiye(i,j,k)/(3*tjace(i,j,k))
       bb6w=visce(ii,j,k)*exize(ii,j,k)*exiye(ii,j,k)/(3*tjace(ii,j,k))
       bb7e=visce(i,j,k)*(exiye(i,j,k)*etaze(i,j,k)
     *    -             2*exize(i,j,k)*etaye(i,j,k)/3)/tjace(i,j,k)
```

```
      bb7w=visce(ii,j,k)*(exiye(ii,j,k)*etaze(ii,j,k)
     *    -              2*exize(ii,j,k)*etaye(ii,j,k)/3)/tjace(ii,j,k)
      bb8e=visce(i,j,k)*(exiye(i,j,k)*zetze(i,j,k)
     *    -              2*exize(i,j,k)*zetye(i,j,k)/3)/tjace(i,j,k)
      bb8w=visce(ii,j,k)*(exiye(ii,j,k)*zetze(ii,j,k)
     *    -              2*exize(ii,j,k)*zetye(ii,j,k)/3)/tjace(ii,j,k)

      bb9n=viscn(i,j,k)*(4*etazn(i,j,k)*exizn(i,j,k)/3
     *    +                 etaxn(i,j,k)*exixn(i,j,k)
     *    +                 etayn(i,j,k)*exiyn(i,j,k))/tjacn(i,j,k)
      bb9s=viscn(i,jj,k)*(4*etazn(i,jj,k)*exizn(i,jj,k)/3
     *    +                 etaxn(i,jj,k)*exixn(i,jj,k)
     *    +                 etayn(i,jj,k)*exiyn(i,jj,k))/tjacn(i,jj,k)
      bb10n=viscn(i,j,k)*(4*etazn(i,j,k)*zetzn(i,j,k)/3
     *    +                 etaxn(i,j,k)*zetxn(i,j,k)
     *    +                 etayn(i,j,k)*zetyn(i,j,k))/tjacn(i,j,k)
      bb10s=viscn(i,jj,k)*(4*etazn(i,jj,k)*zetzn(i,jj,k)/3
     *    +                  etaxn(i,jj,k)*zetxn(i,jj,k)
     *    +                  etayn(i,jj,k)*zetyn(i,jj,k))/tjacn(i,jj,k)
      bb11n=viscn(i,j,k)*(etaxn(i,j,k)*exizn(i,j,k)
     *    -              2*etazn(i,j,k)*exixn(i,j,k)/3)/tjacn(i,j,k)
      bb11s=viscn(i,jj,k)*(etaxn(i,jj,k)*exizn(i,jj,k)
     *    -               2*etazn(i,jj,k)*exixn(i,jj,k)/3)/tjacn(i,jj,k)
      bb12n=viscn(i,j,k)*etazn(i,j,k)*etaxn(i,j,k)/(3*tjacn(i,j,k))
      bb12s=viscn(i,jj,k)*etazn(i,jj,k)*etaxn(i,jj,k)/(3*tjacn(i,jj,k))
      bb13n=viscn(i,j,k)*(etaxn(i,j,k)*zetzn(i,j,k)
     *    -              2*etazn(i,j,k)*zetxn(i,j,k)/3)/tjacn(i,j,k)
      bb13s=viscn(i,jj,k)*(etaxn(i,jj,k)*zetzn(i,jj,k)
     *    -               2*etazn(i,jj,k)*zetxn(i,jj,k)/3)/tjacn(i,jj,k)
      bb14n=viscn(i,j,k)*(etayn(i,j,k)*exizn(i,j,k)
     *    -              2*etazn(i,j,k)*exiyn(i,j,k)/3)/tjacn(i,j,k)
      bb14s=viscn(i,jj,k)*(etayn(i,jj,k)*exizn(i,jj,k)
     *    -               2*etazn(i,jj,k)*exiyn(i,jj,k)/3)/tjacn(i,jj,k)
      bb15n=viscn(i,j,k)*etazn(i,j,k)*etayn(i,j,k)/(3*tjacn(i,j,k))
      bb15s=viscn(i,jj,k)*etazn(i,jj,k)*etayn(i,jj,k)/(3*tjacn(i,jj,k))
      bb16n=viscn(i,j,k)*(etayn(i,j,k)*zetzn(i,j,k)
     *    -             2*etazn(i,j,k)*zetyn(i,j,k)/3)/tjacn(i,j,k)
      bb16s=viscn(i,jj,k)*(etayn(i,jj,k)*zetzn(i,jj,k)
     *    -               2*etazn(i,jj,k)*zetyn(i,jj,k)/3)/tjacn(i,jj,k)

      bb17b=viscb(i,j,k)*(4*zetzb(i,j,k)*exizb(i,j,k)/3
     *    +                 zetxb(i,j,k)*exixb(i,j,k)
     *    +                 zetyb(i,j,k)*exiyb(i,j,k))/tjacb(i,j,k)
      bb17f=viscb(i,j,kk)*(4*zetzb(i,j,kk)*exizb(i,j,kk)/3
     *    +                  zetxb(i,j,kk)*exixb(i,j,kk)
     *    +                  zetyb(i,j,kk)*exiyb(i,j,kk))/tjacb(i,j,kk)
      bb18b=viscb(i,j,k)*(4*zetzb(i,j,k)*etazb(i,j,k)/3
     *    +                 zetxb(i,j,k)*etaxb(i,j,k)
     *    +                 zetyb(i,j,k)*etayb(i,j,k))/tjacb(i,j,k)
      bb18f=viscb(i,j,kk)*(4*zetzb(i,j,kk)*etazb(i,j,kk)/3
     *    +                  zetxb(i,j,kk)*etaxb(i,j,kk)
     *    +                  zetyb(i,j,kk)*etayb(i,j,kk))/tjacb(i,j,kk)
      bb19b=viscb(i,j,k)*(zetxb(i,j,k)*exizb(i,j,k)
     *    -             2*zetzb(i,j,k)*exixb(i,j,k)/3)/tjacb(i,j,k)
      bb19f=viscb(i,j,kk)*(zetxb(i,j,kk)*exizb(i,j,kk)
*     -              2*zetzb(i,j,kk)*exixb(i,j,kk)/3)/tjacb(i,j,kk)
 bb20b=viscb(i,j,k)*(zetxb(i,j,k)*etazb(i,j,k)
*     -            2*zetzb(i,j,k)*etaxb(i,j,k)/3)/tjacb(i,j,k)
 bb20f=viscb(i,j,kk)*(zetxb(i,j,kk)*etazb(i,j,kk)
*     -              2*zetzb(i,j,kk)*etaxb(i,j,kk)/3)/tjacb(i,j,kk)
 bb21b=viscb(i,j,k)*zetzb(i,j,k)*zetxb(i,j,k)/(3*tjacb(i,j,k))
 bb21f=viscb(i,j,kk)*zetzb(i,j,kk)*zetxb(i,j,kk)/(3*tjacb(i,j,kk))
 bb22b=viscb(i,j,k)*(zetyb(i,j,k)*exizb(i,j,k)
*     -             2*zetzb(i,j,k)*exiyb(i,j,k)/3)/tjacb(i,j,k)
 bb22f=viscb(i,j,kk)*(zetyb(i,j,kk)*exizb(i,j,kk)
*     -              2*zetzb(i,j,kk)*exiyb(i,j,kk)/3)/tjacb(i,j,kk)
 bb23b=viscb(i,j,k)*(zetyb(i,j,k)*etazb(i,j,k)
*     -             2*zetzb(i,j,k)*etayb(i,j,k)/3)/tjacb(i,j,k)
 bb23f=viscb(i,j,kk)*(zetyb(i,j,kk)*etazb(i,j,kk)
*     -              2*zetzb(i,j,kk)*etayb(i,j,kk)/3)/tjacb(i,j,kk)
 bb24b=viscb(i,j,k)*zetzb(i,j,k)*zetyb(i,j,k)/(3*tjacb(i,j,k))
 bb24f=viscb(i,j,kk)*zetzb(i,j,kk)*zetyb(i,j,kk)/(3*tjacb(i,j,kk))

 ss1 = exize(ii,j,k)*pie(ii,j,k)/tjace(ii,j,k)
*    - exize(i,j,k)*pie(i,j,k)/tjace(i,j,k)
*    + etazn(i,jj,k)*pin(i,jj,k)/tjacn(i,jj,k)
*    - etazn(i,j,k)*pin(i,j,k)/tjacn(i,j,k)
*    + zetzb(i,j,kk)*pib(i,j,kk)/tjacb(i,j,kk)
*    - zetzb(i,j,k)*pib(i,j,k)/tjacb(i,j,k)
 ss2 = bb1e*(wne(i,j,k)  - wne(i,jj,k))
*    - bb1w*(wne(ii,j,k) - wne(ii,jj,k))
*    + bb2e*(wbe(i,j,k)  - wbe(i,j,kk))
*    - bb2w*(wbe(ii,j,k) - wbe(ii,j,kk))
*    + bb3e*(vx(i+1,j,k) - vx(i,j,k))/delexie(i,j,k)
*    - bb3w*(vx(i,j,k)   - vx(ii,j,k))/delexie(ii,j,k)
*    + bb4e*(une(i,j,k)  - une(i,jj,k))
*    - bb4w*(une(ii,j,k) - une(ii,jj,k))
*    + bb5e*(ube(i,j,k)  - ube(i,j,kk))
*    - bb5w*(ube(ii,j,k) - ube(ii,j,kk))
*    + bb6e*(vy(i+1,j,k) - vy(i,j,k))/delexie(i,j,k)
*    - bb6w*(vy(i,j,k)   - vy(ii,j,k))/delexie(ii,j,k)
*    + bb7e*(vne(i,j,k)  - vne(i,jj,k))
*    - bb7w*(vne(ii,j,k) - vne(ii,jj,k))
*    + bb8e*(vbe(i,j,k)  - vbe(i,j,kk))
*    - bb8w*(vbe(ii,j,k) - vbe(ii,j,kk))

 ss3 = bb9n*(wne(i,j,k)    - wne(ii,j,k))
*    - bb9s*(wne(i,jj,k)   - wne(ii,jj,k))
*    + bb10n*(wnb(i,j,k)   - wnb(i,j,kk))
*    - bb10s*(wnb(i,jj,k)  - wnb(i,jj,kk))
*    + bb11n*(une(i,j,k)   - une(ii,j,k))
*    - bb11s*(une(i,jj,k)  - une(ii,jj,k))
*    + bb12n*(vx(i,j+1,k)  - vx(i,j,k))/deletan(i,j,k)
*    - bb12s*(vx(i,j,k)    - vx(i,jj,k))/deletan(i,jj,k)
*    + bb13n*(unb(i,j,k)   - unb(i,j,kk))
*    - bb13s*(unb(i,jj,k)  - unb(i,jj,kk))
*    + bb14n*(vne(i,j,k)   - vne(ii,j,k))
*    - bb14s*(vne(i,jj,k)  - vne(ii,jj,k))
*    + bb15n*(vy(i,j+1,k)  - vy(i,j,k))/deletan(i,j,k)
*    - bb15s*(vy(i,j,k)    - vy(i,jj,k))/deletan(i,jj,k)
```

```
     *    + bb16n*(vnb(i,j,k)   - vnb(i,j,kk))
     *    - bb16s*(vnb(i,jj,k)  - vnb(i,jj,kk))

      ss4 = bb17b*(wbe(i,j,k)   - wbe(ii,j,k))
     *    - bb17f*(wbe(i,j,kk)  - wbe(ii,j,kk))
     *    + bb18b*(wnb(i,j,k)   - wnb(i,jj,k))
     *    - bb18f*(wnb(i,j,kk)  - wnb(i,jj,kk))
     *    + bb19b*(ube(i,j,k)   - ube(ii,j,k))
     *    - bb19f*(ube(i,j,kk)  - ube(ii,j,kk))
     *    + bb20b*(unb(i,j,k)   - unb(i,jj,k))
     *    - bb20f*(unb(i,j,kk)  - unb(i,jj,kk))
     *    + bb21b*(vx(i,j,k+1)  - vx(i,j,k))/delzetb(i,j,k)
     *    - bb21f*(vx(i,j,k)    - vx(i,j,kk))/delzetb(i,j,kk)
     *    + bb22b*(vbe(i,j,k)   - vbe(ii,j,k))
     *    - bb22f*(vbe(i,j,kk)  - vbe(ii,j,kk))
     *    + bb23b*(vnb(i,j,k)   - vnb(i,jj,k))
     *    - bb23f*(vnb(i,j,kk)  - vnb(i,jj,kk))
     *    + bb24b*(vy(i,j,k+1)  - vy(i,j,k))/delzetb(i,j,k)
     *    - bb24f*(vy(i,j,k)    - vy(i,j,kk))/delzetb(i,j,kk)

      source(i,j,k) = ss1 + ss2 + ss3 + ss4

      res = apw(i,j,k)*vz(i,j,k) - apo(i,j,k)*vz(i,j,k)
     *    - ae(i,j,k)*vz(i+1,j,k) - aw(i,j,k)*vz(ii,j,k)
     *    - an(i,j,k)*vz(i,j+1,k) - as(i,j,k)*vz(i,jj,k)
     *    - ab(i,j,k)*vz(i,j,k+1) - af(i,j,k)*vz(i,j,kk)
     *    - source(i,j,k)
      diffz = diffz + dabs(res)
      reswx(i) = reswx(i) + dabs(res)
      reswy(j) = reswy(j) + dabs(res)
      reswz(k) = reswz(k) + dabs(res)

   20 continue

      return

c-----------------------------------------------------------------
      Entry WPREPI (iii,kkk,jne,jnb)
c-----------------------------------------------------------------

***   This section prepares the TDMA coefficients
***   sweeping in I - direction

      jnb1 = jnb + 1
      jne1 = jne - 1

      jb = jbound(iii,jnb,kkk)
      aaj(jnb) = 1.0
      bbj(jnb) = bw(jb)
      ccj(jnb) = 0.0
      ddj(jnb) = cw(jb)

      do 30 j = jnb1,jne1
        aaj(j) = apw(iii,j,kkk)/relax
        bbj(j) = an(iii,j,kkk)
```

```
         ccj(j) = as(iii,j,kkk)
         ddj(j) = ae(iii,j,kkk)*vzstar(iii+1,j,kkk)
     *          + aw(iii,j,kkk)*vzstar(iii-1,j,kkk)
     *          + ab(iii,j,kkk)*vzstar(iii,j,kkk+1)
     *          + af(iii,j,kkk)*vzstar(iii,j,kkk-1)
     *          + apo(iii,j,kkk)*vzold(iii,j,kkk) + source(iii,j,kkk)
     *          + apw(iii,j,kkk)*vzstar(iii,j,kkk)*(1-relax)/relax
   30 continue

       jb = jbound(iii,jne,kkk)
       aaj(jne) = 1.0
       bbj(jne) = 0.0
       ccj(jne) = bw(jb)
       ddj(jne) = cw(jb)

       return

c-----------------------------------------------------------------------
       Entry WPREPJ (jjj,kkk,ine,inb)
c-----------------------------------------------------------------------

***    This section prepares the TDMA coefficients
***    sweeping in J - direction

       inb1 = inb + 1
       ine1 = ine - 1

       jb = jbound(inb,jjj,kkk)
       aai(inb) = 1.0
       bbi(inb) = bw(jb)
       cci(inb) = 0.0
       ddi(inb) = cw(jb)

       do 40 i = inb1,ine1
         aai(i) = apw(i,jjj,kkk)/relax
         bbi(i) = ae(i,jjj,kkk)
         cci(i) = aw(i,jjj,kkk)
         ddi(i) = an(i,jjj,kkk)*vzstar(i,jjj+1,kkk)
     *          + as(i,jjj,kkk)*vzstar(i,jjj-1,kkk)
     *          + ab(i,jjj,kkk)*vzstar(i,jjj,kkk+1)
     *          + af(i,jjj,kkk)*vzstar(i,jjj,kkk-1)
     *          + apo(i,jjj,kkk)*vzold(i,jjj,kkk) + source(i,jjj,kkk)
     *          + apw(i,jjj,kkk)*vzstar(i,jjj,kkk)*(1-relax)/relax
   40 continue

       jb = jbound(ine,jjj,kkk)
       aai(ine) = 1.0
       bbi(ine) = 0.0
       cci(ine) = bw(jb)
       ddi(ine) = cw(jb)

       return
       end

***********************************************************************
```

```
      Subroutine PCOEFF
************************************************************************

***   This subroutine calculates the coefficients to solve the pressure
***   correction equation

      INCLUDE 'COMM'

***   At each new time step the pressure  corrections are zero
***   bacause the correct pressure field is known

      do 10 k = 1,kn
       do 10 j = 1,jn
        do 10 i = 1,in
         pcor(i,j,k) = 0.0
   10 continue

***   Set all the constants equal to zero

      diffp = 0.0
      do 20 k = 2,knml
       do 20 j = 2,jnml
        do 20 i = 2,inml
         ae(i,j,k) = 0.0
         aw(i,j,k) = 0.0
         an(i,j,k) = 0.0
         as(i,j,k) = 0.0
         ab(i,j,k) = 0.0
         af(i,j,k) = 0.0
         app(i,j,k) = 0.0
         bb(i,j,k) = 0.0
   20 continue

***   Calculate the finite difference coefficients

      do 30 k = 2,knml
       resp(k) = 0.0
       do 30 j = 2,jnml
        do 40 i = 2,inml
        if(jbound(i,j,k).gt.0)goto 40
        ii = i-1
        jj = j-1
        kk = k-1

***
        if(jbound(i,j,k).eq.0.and.jbound(i+1,j,k).ne.0)then
         aae = 0.0
         ae(i,j,k) = 0.0
        else
         apue = (apu(i,j,k)+apu(i+1,j,k))/2
         apve = (apv(i,j,k)+apv(i+1,j,k))/2
         apwe = (apw(i,j,k)+apw(i+1,j,k))/2
         aae = rhoe(i,j,k)*(exixe(i,j,k)*exix(i,j,k)/apue
     *                      +exiye(i,j,k)*exiy(i,j,k)/apve
     *                      +exize(i,j,k)*exiz(i,j,k)/apwe)
     *                   /(tjace(i,j,k)*tjac(i,j,k))
          ae(i,j,k)=rhoe(i,j,k)*(exixe(i,j,k)*exix(i+1,j,k)/apue
     *                          +exiye(i,j,k)*exiy(i+1,j,k)/apve
     *                          +exize(i,j,k)*exiz(i+1,j,k)/apwe)
     *                         /(tjace(i,j,k)*tjac(i+1,j,k))
         endif
***
         if(jbound(i,j,k).eq.0.and.jbound(i-1,j,k).ne.0)then
          aaw = 0.0
          aw(i,j,k) = 0.0
         else
          apuw = (apu(i,j,k)+apu(ii,j,k))/2
          apvw = (apv(i,j,k)+apv(ii,j,k))/2
          apww = (apw(i,j,k)+apw(ii,j,k))/2
          aaw = rhoe(ii,j,k)*(exixe(ii,j,k)*exix(i,j,k)/apuw
     *                       +exiye(ii,j,k)*exiy(i,j,k)/apvw
     *                       +exize(ii,j,k)*exiz(i,j,k)/apww)
     *                      /(tjace(ii,j,k)*tjac(i,j,k))
          aw(i,j,k)=rhoe(ii,j,k)*(exixe(ii,j,k)*exix(ii,j,k)/apuw
     *                           +exiye(ii,j,k)*exiy(ii,j,k)/apvw
     *                           +exize(ii,j,k)*exiz(ii,j,k)/apww)
     *                          /(tjace(ii,j,k)*tjac(ii,j,k))
         endif
***
         if(jbound(i,j,k).eq.0.and.jbound(i,j+1,k).ne.0)then
          aan = 0.0
          an(i,j,k) = 0.0
         else
          apun = (apu(i,j,k)+apu(i,j+1,k))/2
          apvn = (apv(i,j,k)+apv(i,j+1,k))/2
          apwn = (apw(i,j,k)+apw(i,j+1,k))/2
          aan = rhon(i,j,k)*(etaxn(i,j,k)*etax(i,j,k)/apun
     *                      +etayn(i,j,k)*etay(i,j,k)/apvn
     *                      +etazn(i,j,k)*etaz(i,j,k)/apwn)
     *                     /(tjacn(i,j,k)*tjac(i,j,k))
          an(i,j,k)=rhon(i,j,k)*(etaxn(i,j,k)*etax(i,j+1,k)/apun
     *                          +etayn(i,j,k)*etay(i,j+1,k)/apvn
     *                          +etazn(i,j,k)*etaz(i,j+1,k)/apwn)
     *                         /(tjacn(i,j,k)*tjac(i,j+1,k))
         endif
***
         if(jbound(i,j,k).eq.0.and.jbound(i,j-1,k).ne.0)then
          aas = 0.0
          as(i,j,k) = 0.0
         else
          apus = (apu(i,j,k)+apu(i,jj,k))/2
          apvs = (apv(i,j,k)+apv(i,jj,k))/2
          apws = (apw(i,j,k)+apw(i,jj,k))/2
          aas = rhon(i,jj,k)*(etaxn(i,jj,k)*etax(i,j,k)/apus
     *                       +etayn(i,jj,k)*etay(i,j,k)/apvs
     *                       +etazn(i,jj,k)*etaz(i,j,k)/apws)
     *                      /(tjacn(i,jj,k)*tjac(i,j,k))
          as(i,j,k)=rhon(i,jj,k)*(etaxn(i,jj,k)*etax(i,jj,k)/apus
     *                           +etayn(i,jj,k)*etay(i,jj,k)/apvs
     *                           +etazn(i,jj,k)*etaz(i,jj,k)/apws)
     *                          /(tjacn(i,jj,k)*tjac(i,jj,k))
        endif
***
        if(jbound(i,j,k).eq.0.and.jbound(i,j,k+1).ne.0)then
         aab = 0.0
         ab(i,j,k) = 0.0
        else
         apub = (apu(i,j,k)+apu(i,j,k+1))/2
         apvb = (apv(i,j,k)+apv(i,j,k+1))/2
         apwb = (apw(i,j,k)+apw(i,j,k+1))/2
         aab = rhob(i,j,k)*(zetxb(i,j,k)*zetx(i,j,k)/apub
     *                      +zetyb(i,j,k)*zety(i,j,k)/apvb
     *                      +zetzb(i,j,k)*zetz(i,j,k)/apwb)
     *                     /(tjacb(i,j,k)*tjac(i,j,k))
         ab(i,j,k)=rhob(i,j,k)*(zetxb(i,j,k)*zetx(i,j,k+1)/apub
     *                          +zetyb(i,j,k)*zety(i,j,k+1)/apvb
     *                          +zetzb(i,j,k)*zetz(i,j,k+1)/apwb)
     *                         /(tjacb(i,j,k)*tjac(i,j,k+1))
        endif
***
        if(jbound(i,j,k).eq.0.and.jbound(i,j,k-1).ne.0)then
         aaf = 0.0
         af(i,j,k) = 0.0
        else
         apuf = (apu(i,j,k)+apu(i,j,kk))/2
         apvf = (apv(i,j,k)+apv(i,j,kk))/2
         apwf = (apw(i,j,k)+apw(i,j,kk))/2
         aaf = rhob(i,j,kk)*(zetxb(i,j,kk)*zetx(i,j,k)/apuf
     *                       +zetyb(i,j,kk)*zety(i,j,k)/apvf
     *                       +zetzb(i,j,kk)*zetz(i,j,k)/apwf)
     *                      /(tjacb(i,j,kk)*tjac(i,j,k))
         af(i,j,k)=rhob(i,j,kk)*(zetxb(i,j,kk)*zetx(i,j,kk)/apuf
     *                           +zetyb(i,j,kk)*zety(i,j,kk)/apvf
     *                           +zetzb(i,j,kk)*zetz(i,j,kk)/apwf)
     *                          /(tjacb(i,j,kk)*tjac(i,j,kk))
        endif
***
        app(i,j,k) = aae + aaw + aan + aas + aab + aaf

        dens = (rhoo(i,j,k)-rhop(i,j,k))/(delt*tjac(i,j,k))
        beast =
     *             - (exixe(i,j,k)*ue(i,j,k)
     *               +exiye(i,j,k)*ve(i,j,k)
     *               +exize(i,j,k)*we(i,j,k))*rhoe(i,j,k)/tjace(i,j,k)
        bwest =
     *          + (exixe(ii,j,k)*ue(ii,j,k)
     *            +exiye(ii,j,k)*ve(ii,j,k)
     *            +exize(ii,j,k)*we(ii,j,k))*rhoe(ii,j,k)/tjace(ii,j,k)
        bnorth =
     *             - (etaxn(i,j,k)*un(i,j,k)
     *               +etayn(i,j,k)*vn(i,j,k)
     *               +etazn(i,j,k)*wn(i,j,k))*rhon(i,j,k)/tjacn(i,j,k)
        bsouth =
     *          + (etaxn(i,jj,k)*un(i,jj,k)
     *            +etayn(i,jj,k)*vn(i,jj,k)
     *           +etazn(i,jj,k)*wn(i,jj,k))*rhon(i,jj,k)/tjacn(i,jj,k)
         bback =
     *            - (zetxb(i,j,k)*ub(i,j,k)
     *              +zetyb(i,j,k)*vb(i,j,k)
     *              +zetzb(i,j,k)*wb(i,j,k))*rhob(i,j,k)/tjacb(i,j,k)
         bfront =
     *          + (zetxb(i,j,kk)*ub(i,j,kk)
     *            +zetyb(i,j,kk)*vb(i,j,kk)
     *            +zetzb(i,j,kk)*wb(i,j,kk))*rhob(i,j,kk)/tjacb(i,j,kk)

         bb(i,j,k) = beast+bwest+bnorth+bsouth+bback+bfront+dens

         resp(k) = resp(k) + dabs(bb(i,j,k))
         diffp = diffp + dabs(bb(i,j,k))

   40  continue
   30 continue

        diffp = diffp/fmin

        return

c---------------------------------------------------------------------
        Entry PPREPI (iii,kkk,jne,jnb)
c---------------------------------------------------------------------

***     This section prepares the TDMA coefficients
***     sweeping in I - direction

        jnb1 = jnb + 1
        jne1 = jne - 1

        jb = jbound(iii,jnb,kkk)
        aaj(jnb) = 1.0
        bbj(jnb) = bp(jb)
        ccj(jnb) = 0.0
        ddj(jnb) = cp(jb)

        do 60 j = jnb1,jne1
          aaj(j) = app(iii,j,kkk)
          bbj(j) = an(iii,j,kkk)
          ccj(j) = as(iii,j,kkk)
          ddj(j) = ae(iii,j,kkk)*pcor(iii+1,j,kkk)
     *          + aw(iii,j,kkk)*pcor(iii-1,j,kkk)
     *          + ab(iii,j,kkk)*pcor(iii,j,kkk+1)
     *          + af(iii,j,kkk)*pcor(iii,j,kkk-1)+bb(iii,j,kkk)
   60 continue

        jb = jbound(iii,jne,kkk)
        aaj(jne) = 1.0
        bbj(jne) = 0.0
        ccj(jne) = bp(jb)
        ddj(jne) = cp(jb)

        return
```

```
c-----------------------------------------------------------------------
      Entry PPREPJ (jjj,kkk,ine,inb)
c-----------------------------------------------------------------------

***   This section prepares the TDMA coefficients
***   sweeping in J - direction

      inbl = inb + 1
      inel = ine - 1

      jb = jbound(inb,jjj,kkk)
      aai(inb) = 1.0
      bbi(inb) = bp(jb)
      cci(inb) = 0.0
      ddi(inb) = cp(jb)

      do 70 i = inbl,inel
        aai(i) = app(i,jjj,kkk)
        bbi(i) = ae(i,jjj,kkk)
        cci(i) = aw(i,jjj,kkk)
        ddi(i) = an(i,jjj,kkk)*pcor(i,jjj+1,kkk)
     *         + as(i,jjj,kkk)*pcor(i,jjj-1,kkk)
     *         + ab(iii,j,kkk)*pcor(i,jjj,kkk+1)
     *         + af(iii,j,kkk)*pcor(i,jjj,kkk-1)+bb(i,jjj,kkk)
   70 continue

      jb = jbound(ine,jjj,kkk)
      aai(ine) = 1.0
      bbi(ine) = 0.0
      cci(ine) = bp(jb)
      ddi(ine) = cp(jb)

      return
      end

**********************************************************************
      Subroutine GAMM
**********************************************************************

      INCLUDE 'COMM'

      restken = 0.0
      reseps = 0.0

      do 10 k = 2,knml
       do 10 j = 2,jnml
        do 10 i = 2,inml
         if(jbound(i,j,k).ne.0)goto 10

         ii = i-1
         jj = j-1
         kk = k-1

      dudx = tjac(i,j,k)*(exixe(i,j,k)*ue(i,j,k)/tjace(i,j,k)
     *                    - exixe(ii,j,k)*ue(ii,j,k)/tjace(ii,j,k)
     *                    + etaxn(i,j,k)*un(i,j,k)/tjacn(i,j,k)
     *                    - etaxn(i,jj,k)*un(i,jj,k)/tjacn(i,jj,k)
     *                    + zetxb(i,j,k)*ub(i,j,k)/tjacb(i,j,k)
     *                    - zetxb(i,j,kk)*ub(i,j,kk)/tjacb(i,j,kk))
      dudy = tjac(i,j,k)*(exiye(i,j,k)*ue(i,j,k)/tjace(i,j,k)
     *                    - exiye(ii,j,k)*ue(ii,j,k)/tjace(ii,j,k)
     *                    + etayn(i,j,k)*un(i,j,k)/tjacn(i,j,k)
     *                    - etayn(i,jj,k)*un(i,jj,k)/tjacn(i,jj,k)
     *                    + zetyb(i,j,k)*ub(i,j,k)/tjacb(i,j,k)
     *                    - zetyb(i,j,kk)*ub(i,j,kk)/tjacb(i,j,kk))
      dudz = tjac(i,j,k)*(exize(i,j,k)*ue(i,j,k)/tjace(i,j,k)
     *                    - exize(ii,j,k)*ue(ii,j,k)/tjace(ii,j,k)
     *                    + etazn(i,j,k)*un(i,j,k)/tjacn(i,j,k)
     *                    - etazn(i,jj,k)*un(i,jj,k)/tjacn(i,jj,k)
     *                    + zetzb(i,j,k)*ub(i,j,k)/tjacb(i,j,k)
     *                    - zetzb(i,j,kk)*ub(i,j,kk)/tjacb(i,j,kk))
      dvdx = tjac(i,j,k)*(exixe(i,j,k)*ve(i,j,k)/tjace(i,j,k)
     *                    - exixe(ii,j,k)*ve(ii,j,k)/tjace(ii,j,k)
     *                    + etaxn(i,j,k)*vn(i,j,k)/tjacn(i,j,k)
     *                    - etaxn(i,jj,k)*vn(i,jj,k)/tjacn(i,jj,k)
     *                    + zetxb(i,j,k)*vb(i,j,k)/tjacb(i,j,k)
     *                    - zetxb(i,j,kk)*vb(i,j,kk)/tjacb(i,j,kk))
      dvdy = tjac(i,j,k)*(exiye(i,j,k)*ve(i,j,k)/tjace(i,j,k)
     *                    - exiye(ii,j,k)*ve(ii,j,k)/tjace(ii,j,k)
     *                    + etayn(i,j,k)*vn(i,j,k)/tjacn(i,j,k)
     *                    - etayn(i,jj,k)*vn(i,jj,k)/tjacn(i,jj,k)
     *                    + zetyb(i,j,k)*vb(i,j,k)/tjacb(i,j,k)
     *                    - zetyb(i,j,kk)*vb(i,j,kk)/tjacb(i,j,kk))
      dvdz = tjac(i,j,k)*(exize(i,j,k)*ve(i,j,k)/tjace(i,j,k)
     *                    - exize(ii,j,k)*ve(ii,j,k)/tjace(ii,j,k)
     *                    + etazn(i,j,k)*vn(i,j,k)/tjacn(i,j,k)
     *                    - etazn(i,jj,k)*vn(i,jj,k)/tjacn(i,jj,k)
     *                    + zetzb(i,j,k)*vb(i,j,k)/tjacb(i,j,k)
     *                    - zetzb(i,j,kk)*vb(i,j,kk)/tjacb(i,j,kk))
      dwdx = tjac(i,j,k)*(exixe(i,j,k)*we(i,j,k)/tjace(i,j,k)
     *                    - exixe(ii,j,k)*we(ii,j,k)/tjace(ii,j,k)
     *                    + etaxn(i,j,k)*wn(i,j,k)/tjacn(i,j,k)
     *                    - etaxn(i,jj,k)*wn(i,jj,k)/tjacn(i,jj,k)
     *                    + zetxb(i,j,k)*wb(i,j,k)/tjacb(i,j,k)
     *                    - zetxb(i,j,kk)*wb(i,j,kk)/tjacb(i,j,kk))
      dwdy = tjac(i,j,k)*(exiye(i,j,k)*we(i,j,k)/tjace(i,j,k)
     *                    - exiye(ii,j,k)*we(ii,j,k)/tjace(ii,j,k)
     *                    + etayn(i,j,k)*wn(i,j,k)/tjacn(i,j,k)
     *                    - etayn(i,jj,k)*wn(i,jj,k)/tjacn(i,jj,k)
     *                    + zetyb(i,j,k)*wb(i,j,k)/tjacb(i,j,k)
     *                    - zetyb(i,j,kk)*wb(i,j,kk)/tjacb(i,j,kk))
      dwdz = tjac(i,j,k)*(exize(i,j,k)*we(i,j,k)/tjace(i,j,k)
     *                    - exize(ii,j,k)*we(ii,j,k)/tjace(ii,j,k)
     *                    + etazn(i,j,k)*wn(i,j,k)/tjacn(i,j,k)
     *                    - etazn(i,jj,k)*wn(i,jj,k)/tjacn(i,jj,k)
     *                    + zetzb(i,j,k)*wb(i,j,k)/tjacb(i,j,k)
     *                    - zetzb(i,j,kk)*wb(i,j,kk)/tjacb(i,j,kk))

      Gamma(i,j,k) = 2*(dudx**2) + 2*(dvdy**2) + 2*(dwdz**2) + dudy**2
     *      + dudz**2 + dvdx**2 + dvdz**2 + dwdx**2 + dwdy**2
     *      + 2*dudy*dvdx + 2*dvdz*dwdy + 2*dwdx*dudz

      restken = restken + tken(i,j,k)
      reseps = reseps + eps(i,j,k)

   10 continue

      return
      end

************************************************************************
      Subroutine KCOEFF
************************************************************************

***   This subroutine calculates the coefficients to solve the
***   kinetic energy turbulence equation

      INCLUDE 'COMM'

***   Set all the constants equal to zero

      do 10 k = 2,knml
       do 10 j = 2,jnml
        do 10 i = 2,inml
         an(i,j,k) = 0.0
         as(i,j,k) = 0.0
         ae(i,j,k) = 0.0
         aw(i,j,k) = 0.0
         ab(i,j,k) = 0.0
         af(i,j,k) = 0.0
         apk(i,j,k) = 0.0
         apo(i,j,k) = 0.0
         source(i,j,k) = 0.0
   10 continue

***   Calculate the coefficients

      do 20 k = 2,knml
       do 20 i = 2,inml
        do 20 j = 2,jnml

        if(jbound(i,j,k).gt.0)goto 20
        ii = i-1
        jj = j-1
        kk = k-1

        conve = (exixe(i,j,k)*ue(i,j,k)
     *         +  exiye(i,j,k)*ve(i,j,k)
     *         +  exize(i,j,k)*we(i,j,k))*rhoe(i,j,k)/tjace(i,j,k)
        convw = (exixe(ii,j,k)*ue(ii,j,k)
     *         +  exiye(ii,j,k)*ve(ii,j,k)
     *         +  exize(ii,j,k)*we(ii,j,k))*rhoe(ii,j,k)/tjace(ii,j,k)
        convn = (etaxn(i,j,k)*un(i,j,k)
     *         +  etayn(i,j,k)*vn(i,j,k)
     *         +  etazn(i,j,k)*wn(i,j,k))*rhon(i,j,k)/tjacn(i,j,k)
        convs = (etaxn(i,jj,k)*un(i,jj,k)
     *         +  etayn(i,jj,k)*vn(i,jj,k)
     *         +  etazn(i,jj,k)*wn(i,jj,k))*rhon(i,jj,k)/tjacn(i,jj,k)
        convb = (zetxb(i,j,k)*ub(i,j,k)
     *         +  zetyb(i,j,k)*vb(i,j,k)
     *         +  zetzb(i,j,k)*wb(i,j,k))*rhob(i,j,k)/tjacb(i,j,k)
        convf = (zetxb(i,j,kk)*ub(i,j,kk)
     *         +  zetyb(i,j,kk)*vb(i,j,kk)
     *         +  zetzb(i,j,kk)*wb(i,j,kk))*rhob(i,j,kk)/tjacb(i,j,kk)

        diffe = (exixe(i,j,k)**2+exiye(i,j,k)**2+exize(i,j,k)**2)
     *         *vmuturb(i,j,k)/(tjace(i,j,k)*delexie(i,j,k)*sigmk)
        diffw = (exixe(ii,j,k)**2+exiye(ii,j,k)**2+exize(ii,j,k)**2)
     *         *vmuturb(i,j,k)/(tjace(ii,j,k)*delexie(ii,j,k)*sigmk)
        diffn = (etaxn(i,j,k)**2+etayn(i,j,k)**2+etazn(i,j,k)**2)
     *         *vmuturb(i,j,k)/(tjacn(i,j,k)*deletan(i,j,k)*sigmk)
        diffs = (etaxn(i,jj,k)**2+etayn(i,jj,k)**2+etazn(i,jj,k)**2)
     *         *vmuturb(i,j,k)/(tjacn(i,jj,k)*deletan(i,jj,k)*sigmk)
        diffb = (zetxb(i,j,k)**2+zetyb(i,j,k)**2+zetzb(i,j,k)**2)
     *         *vmuturb(i,j,k)/(tjacb(i,j,k)*delzetb(i,j,k)*sigmk)
        difff = (zetxb(i,j,kk)**2+zetyb(i,j,kk)**2+zetzb(i,j,kk)**2)
     *         *vmuturb(i,j,k)/(tjacb(i,j,kk)*delzetb(i,j,kk)*sigmk)

***   First calculate the source term coefficients

      ss1 = 0.0
      ss2 = 0.0
      ss3 = 0.0
      if(igrid.eq.1)goto 15

      bb1e = (exixe(i,j,k)*etaxe(i,j,k)
     *     +  exiye(i,j,k)*etaye(i,j,k)
     *     +  exize(i,j,k)*etaze(i,j,k))
     *      * vmuturb (i,j,k)/(tjace(i,j,k)*sigmk)
      bb1w = (exixe(ii,j,k)*etaxe(ii,j,k)
     *     +  exiye(ii,j,k)*etaye(ii,j,k)
     *     +  exize(ii,j,k)*etaze(ii,j,k))
     *      * vmuturb (i,j,k)/(tjace(ii,j,k)*sigmk)
      bb2e = (exixe(i,j,k)*zetxe(i,j,k)
     *     +  exiye(i,j,k)*zetye(i,j,k)
     *     +  exize(i,j,k)*zetze(i,j,k))
     *      * vmuturb (i,j,k)/(tjace(i,j,k)*sigmk)
      bb2w = (exixe(ii,j,k)*zetxe(ii,j,k)
     *     +  exiye(ii,j,k)*zetye(ii,j,k)
     *     +  exize(ii,j,k)*zetze(ii,j,k))
     *      * vmuturb (i,j,k)/(tjace(ii,j,k)*sigmk)

      bb3n = (etaxn(i,j,k)*exixn(i,j,k)
     *     +  etayn(i,j,k)*exiyn(i,j,k)
     *     +  etazn(i,j,k)*exizn(i,j,k))
     *      * vmuturb (i,j,k)/(tjacn(i,j,k)*sigmk)
      bb3s = (etaxn(i,jj,k)*exixn(i,jj,k)
     *     +  etayn(i,jj,k)*exiyn(i,jj,k)
     *     +  etazn(i,jj,k)*exizn(i,jj,k))
     *      * vmuturb (i,j,k)/(tjacn(i,jj,k)*sigmk)
      bb4n = (etaxn(i,j,k)*zetxn(i,j,k)
     *     +  etayn(i,j,k)*zetyn(i,j,k)
     *     +  etazn(i,j,k)*zetzn(i,j,k))
     *      * vmuturb (i,j,k)/(tjacn(i,j,k)*sigmk)
      bb4s = (etaxn(i,jj,k)*zetxn(i,jj,k)
     *     +  etayn(i,jj,k)*zetyn(i,jj,k)
     *     +  etazn(i,jj,k)*zetzn(i,jj,k))
     *      * vmuturb (i,j,k)/(tjacn(i,jj,k)*sigmk)

      bb5b = (zetxb(i,j,k)*exixb(i,j,k)
     *     +  zetyb(i,j,k)*exiyb(i,j,k)
     *     +  zetzb(i,j,k)*exizb(i,j,k))
     *      * vmuturb (i,j,k)/(tjacb(i,j,k)*sigmk)
      bb5f = (zetxb(i,j,kk)*exixb(i,j,kk)
     *     +  zetyb(i,j,kk)*exiyb(i,j,kk)
     *     +  zetzb(i,j,kk)*exizb(i,j,kk))
     *      * vmuturb (i,j,k)/(tjacb(i,j,kk)*sigmk)
      bb6b = (zetxb(i,j,k)*etaxb(i,j,k)
     *     +  zetyb(i,j,k)*etayb(i,j,k)
     *     +  zetzb(i,j,k)*etazb(i,j,k))
     *      * vmuturb (i,j,k)/(tjacb(i,j,k)*sigmk)
      bb6f = (zetxb(i,j,kk)*etaxb(i,j,kk)
     *     +  zetyb(i,j,kk)*etayb(i,j,kk)
     *     +  zetzb(i,j,kk)*etazb(i,j,kk))
     *      * vmuturb (i,j,k)/(tjacb(i,j,kk)*sigmk)

      ss1 = bb1e*(tkenne(i,j,k)  - tkenne(i,jj,k))
     *    - bb1w*(tkenne(ii,j,k) - tkenne(ii,jj,k))
     *    + bb2e*(tkenbe(i,j,k)  - tkenbe(i,j,kk))
     *    - bb2w*(tkenbe(ii,j,k) - tkenbe(ii,j,kk))

      ss2 = bb3n*(tkenne(i,j,k)  - tkenne(ii,j,k))
     *    - bb3s*(tkenne(i,jj,k) - tkenne(ii,jj,k))
     *    + bb4n*(tkennb(i,j,k)  - tkennb(i,j,kk))
     *    - bb4s*(tkennb(i,jj,k) - tkennb(i,jj,kk))

      ss3 = bb5b*(tkenbe(i,j,k)  - tkenbe(ii,j,k))
     *    - bb5f*(tkenbe(i,j,kk) - tkenbe(ii,j,kk))
     *    + bb6b*(tkennb(i,j,k)  - tkennb(i,jj,k))
     *    - bb6f*(tkennb(i,j,kk) - tkennb(i,jj,kk))

   15 ccp = (c2*eps(i,j,k)*rhop(i,j,k)
     *    + 0.5*Gamma(i,j,k)*vmuturb(i,j,k))/(tken(i,j,k)*tjac(i,j,k))
      bbb = 1.5*Gamma(i,j,k)*vmuturb(i,j,k)/tjac(i,j,k)
     *    + (c2-1)*rhop(i,j,k)*eps(i,j,k)/tjac(i,j,k)

***   Determine the main finite difference coefficients

***   Eastern boundaries

        if(jbound(i,j,k).eq.0.and.jbound(i+1,j,k).eq.nbnd)then
         ae(i,j,k) = 0.0
         tvel = (vx(i,j,k)**2+vy(i,j,k)**2+vz(i,j,k)**2)**0.5
         dwall = dabs(etay(i,j,k)*zetz(i,j,k)/(2*tjac(i,j,k)))
        exiss = (exix(i,j,k)+exiy(i,j,k)+exiz(i,j,k))/tjac(i,j,k)
        Reloc = (cmu**0.25)*rhop(i,j,k)*(dabs(tken(i,j,k))**0.5)
     *          *dwall/vmulam
         if(Reloc.gt.11.6)then
          tauw(i,j,k) = dabs(-cappa*tvel*Reloc*vmulam
     *           /(dwall*dlog(eps(i,j,k)*Reloc)))
         else
          tauw(i,j,k) = dabs(-vmulam*tvel/dwall)
         endif
        ccp = cmu*tvel*rhop(i,j,k)**2*tken(i,j,k)*exiss
     *        /(tauw(i,j,k)*tjac(i,j,k))
        bbb = tauw(i,j,k)*tvel*exiss/tjac(i,j,k)
       else
        AE(I,J,K) = DIFFE + DMAX1(-CONVE,ZERO)
       endif

***   Western boundaries

       if(jbound(i,j,k).eq.0.and.jbound(i-1,j,k).eq.nbnd)then
        aw(i,j,k) = 0.0
        tvel = (vx(i,j,k)**2+vy(i,j,k)**2+vz(i,j,k)**2)**0.5
        dwall = dabs(etay(i,j,k)*zetz(i,j,k)/(2*tjac(i,j,k)))
        exiss = (exix(i,j,k)+exiy(i,j,k)+exiz(i,j,k))/tjac(i,j,k)
        Reloc = (cmu**0.25)*rhop(i,j,k)*(dabs(tken(i,j,k))**0.5)
     *          *dwall/vmulam
         if(Reloc.gt.11.6)then
          tauw(i,j,k) = dabs(-cappa*tvel*Reloc*vmulam
     *           /(dwall*dlog(eps(i,j,k)*Reloc)))
         else
          tauw(i,j,k) = dabs(-vmulam*tvel/dwall)
         endif
        ccp = cmu*tvel*rhop(i,j,k)**2*tken(i,j,k)*exiss
     *        /(tauw(i,j,k)*tjac(i,j,k))
        bbb = tauw(i,j,k)*tvel*exiss/tjac(i,j,k)
       else
        AW(I,J,K) = DIFFW + DMAX1( CONVW,ZERO)
       endif

***   Northern boundaries

       if(jbound(i,j,k).eq.0.and.jbound(i,j+1,k).eq.nbnd)then
        an(i,j,k) = 0.0
        tvel = (vx(i,j,k)**2+vy(i,j,k)**2+vz(i,j,k)**2)**0.5
        dwall = dabs(exix(i,j,k)*zetz(i,j,k)/(2*tjac(i,j,k)))
        etass = (etax(i,j,k)+etay(i,j,k)+etaz(i,j,k))/tjac(i,j,k)
        Reloc = (cmu**0.25)*rhop(i,j,k)*(dabs(tken(i,j,k))**0.5)
     *          *dwall/vmulam
         if(Reloc.gt.11.6)then
          tauw(i,j,k) = dabs(-cappa*tvel*Reloc*vmulam
     *           /(dwall*dlog(eps(i,j,k)*Reloc)))
         else
          tauw(i,j,k) = dabs(-vmulam*tvel/dwall)
         endif
        ccp = cmu*tvel*rhop(i,j,k)**2*tken(i,j,k)*etass
     *        /(tauw(i,j,k)*tjac(i,j,k)+tiny)
```

```
          bbb = tauw(i,j,k)*tvel*etass/tjac(i,j,k)
         else
          AN(I,J,K) = DIFFN + DMAX1(-CONVN,ZERO)
         endif

***   Southern boundaries

         if(jbound(i,j,k).eq.0.and.jbound(i,j-1,k).eq.nbnd)then
          as(i,j,k) = 0.0
          tvel = (vx(i,j,k)**2+vy(i,j,k)**2+vz(i,j,k)**2)**0.5
          dwall = dabs(exix(i,j,k)*zetz(i,j,k)/(2*tjac(i,j,k)))
          etass = (etax(i,j,k)+etay(i,j,k)+etaz(i,j,k))/tjac(i,j,k)
          Reloc = (cmu**0.25)*rhop(i,j,k)*(dabs(tken(i,j,k))**0.5)
     *            *dwall/vmulam
           if(Reloc.gt.11.6)then
            tauw(i,j,k) = dabs(-cappa*tvel*Reloc*vmulam
     *             /(dwall*dlog(eps(i,j,k)*Reloc)))
           else
            tauw(i,j,k) = dabs(-vmulam*tvel/dwall)
           endif
          ccp = cmu*tvel*rhop(i,j,k)**2*tken(i,j,k)*etass
     *          /(tauw(i,j,k)*tjac(i,j,k)+tiny)
          bbb = tauw(i,j,k)*tvel*etass/tjac(i,j,k)
         else
          AS(I,J,K) = DIFFS + DMAX1( CONVS,ZERO)
         endif

***   Back boundaries

         if(jbound(i,j,k).eq.0.and.jbound(i,j,k+1).eq.nbnd)then
          ab(i,j,k) = 0.0
          tvel = (vx(i,j,k)**2+vy(i,j,k)**2)**0.5
          dwall = dabs(exix(i,j,k)*etay(i,j,k)/(2*tjac(i,j,k)))
          zetss = (zetx(i,j,k)+zety(i,j,k)+zetz(i,j,k))/tjac(i,j,k)
          Reloc = (cmu**0.25)*rhop(i,j,k)*(dabs(tken(i,j,k))**0.5)
     *            *dwall/vmulam
           if(Reloc.gt.11.6)then
            tauw(i,j,k) = dabs(-cappa*tvel*Reloc*vmulam
     *             /(dwall*dlog(eps(i,j,k)*Reloc)))
           else
            tauw(i,j,k) = dabs(-vmulam*tvel/dwall)
           endif
          ccp = cmu*tvel*rhop(i,j,k)**2*tken(i,j,k)*zetss
     *          /(tauw(i,j,k)*tjac(i,j,k))
          bbb = tauw(i,j,k)*tvel*zetss/tjac(i,j,k)
         else
          AB(I,J,K) = DIFFB + DMAX1(-CONVB,ZERO)
         endif

***   Front boundaries

         if(jbound(i,j,k).eq.0.and.jbound(i,j,k-1).eq.nbnd)then
          af(i,j,k) = 0.0
          tvel = (vx(i,j,k)**2+vy(i,j,k)**2)**0.5
          dwall = dabs(exix(i,j,k)*etay(i,j,k)/(2*tjac(i,j,k)))
```

```
           zetss = (zetx(i,j,k)+zety(i,j,k)+zetz(i,j,k))/tjac(i,j,k)
           Reloc = (cmu**0.25)*rhop(i,j,k)*(dabs(tken(i,j,k))**0.5)
     *            *dwall/vmulam
            if(Reloc.gt.11.6)then
             tauw(i,j,k) = dabs(-cappa*tvel*Reloc*vmulam
     *             /(dwall*dlog(eps(i,j,k)*Reloc)))
            else
             tauw(i,j,k) = dabs(-vmulam*tvel/dwall)
            endif
           ccp = cmu*tvel*rhop(i,j,k)**2*tken(i,j,k)*zetss
     *          /(tauw(i,j,k)*tjac(i,j,k)+tiny)
           bbb = tauw(i,j,k)*tvel*zetss/tjac(i,j,k)
          else
           AF(I,J,K) = DIFFF + DMAX1( CONVF,ZERO)
          endif

***

      apo(i,j,k) = rhoo(i,j,k)/(delt*tjac(i,j,k))
      apk(i,j,k) = ae(i,j,k) + aw(i,j,k) + an(i,j,k) + as(i,j,k)
     *           + ab(i,j,k) + af(i,j,k) + apo(i,j,k) + ccp

      source(i,j,k) = ss1 + ss2 + ss3 + bbb

   20 continue

      return

c-------------------------------------------------------------------
      Entry KPREPI (iii,kkk,jne,jnb)
c-------------------------------------------------------------------

***   This section prepares the TDMA coefficients
***   sweeping in I - direction

      jnb1 = jnb + 1
      jne1 = jne - 1

      jb = jbound(iii,jnb,kkk)
      aaj(jnb) = 1.0
      bbj(jnb) = bk(jb)
      ccj(jnb) = 0.0
      ddj(jnb) = ck(jb)

      do 30 j = jnb1,jne1
        aaj(j) = apk(iii,j,kkk)/relaxt
        bbj(j) = an(iii,j,kkk)
        ccj(j) = as(iii,j,kkk)
        ddj(j) = ae(iii,j,kkk)*tken(iii+1,j,kkk)
     *         + aw(iii,j,kkk)*tken(iii-1,j,kkk)
     *         + ab(iii,j,kkk)*tken(iii,j,kkk+1)
     *         + af(iii,j,kkk)*tken(iii,j,kkk-1)
     *         + apo(iii,j,kkk)*tkenold(iii,j,kkk) + source(iii,j,kkk)
     *         + apk(iii,j,kkk)*tken(iii,j,kkk)*(1-relaxt)/relaxt
   30 continue
```

```
      jb = jbound(iii,jne,kkk)
      aaj(jne) = 1.0
      bbj(jne) = 0.0
      ccj(jne) = bk(jb)
      ddj(jne) = ck(jb)

      return

c-----------------------------------------------------------------------
      Entry KPREPJ (jjj,kkk,ine,inb)
c-----------------------------------------------------------------------

***   This section prepares the TDMA coefficients
***   sweeping in J - direction

      inb1 = inb + 1
      ine1 = ine - 1

      jb = jbound(inb,jjj,kkk)
      aai(inb) = 1.0
      bbi(inb) = bk(jb)
      cci(inb) = 0.0
      ddi(inb) = ck(jb)

      do 40 i = inb1,ine1
        aai(i) = apk(i,jjj,kkk)/relaxt
        bbi(i) = ae(i,jjj,kkk)
        cci(i) = aw(i,jjj,kkk)
        ddi(i) = an(i,jjj,kkk)*tken(i,jjj+1,kkk)
     *         + as(i,jjj,kkk)*tken(i,jjj-1,kkk)
     *         + ab(i,jjj,kkk)*tken(i,jjj,kkk+1)
     *         + af(i,jjj,kkk)*tken(i,jjj,kkk-1)
     *         + apo(i,jjj,kkk)*tkenold(i,jjj,kkk) + source(i,jjj,kkk)
     *         + apk(i,jjj,kkk)*tken(i,jjj,kkk)*(1-relaxt)/relaxt
   40 continue

      jb = jbound(ine,jjj,kkk)
      aai(ine) = 1.0
      bbi(ine) = 0.0
      cci(ine) = bk(jb)
      ddi(ine) = ck(jb)

      return
      end

***********************************************************************
      Subroutine ECOEFF
***********************************************************************

***   This subroutine calculates the coefficients to solve the
***   epsilon turbulence equation

      INCLUDE 'COMM'
```

```
***   Set all the constants equal to zero

      do 10 k = 2,knml
       do 10 j = 2,jnml
        do 10 i = 2,inml
         an(i,j,k) = 0.0
         as(i,j,k) = 0.0
         ae(i,j,k) = 0.0
         aw(i,j,k) = 0.0
         ab(i,j,k) = 0.0
         af(i,j,k) = 0.0
         ape(i,j,k) = 0.0
         apo(i,j,k) = 0.0
         source(i,j,k) = 0.0
   10 continue

***   Calculate the coefficients

      do 20 k = 2,knml
       do 20 i = 2,inml
        do 20 j = 2,jnml

        if(jbound(i,j,k).gt.0)goto 20
        ii = i-1
        jj = j-1
        kk = k-1

        conve = (exixe(i,j,k)*ue(i,j,k)
     *        +  exiye(i,j,k)*ve(i,j,k)
     *        +  exize(i,j,k)*we(i,j,k))*rhoe(i,j,k)/tjace(i,j,k)
        convw = (exixe(ii,j,k)*ue(ii,j,k)
     *        +  exiye(ii,j,k)*ve(ii,j,k)
     *        +  exize(ii,j,k)*we(ii,j,k))*rhoe(ii,j,k)/tjace(ii,j,k)
        convn = (etaxn(i,j,k)*un(i,j,k)
     *        +  etayn(i,j,k)*vn(i,j,k)
     *        +  etazn(i,j,k)*wn(i,j,k))*rhon(i,j,k)/tjacn(i,j,k)
        convs = (etaxn(i,jj,k)*un(i,jj,k)
     *        +  etayn(i,jj,k)*vn(i,jj,k)
     *        +  etazn(i,jj,k)*wn(i,jj,k))*rhon(i,jj,k)/tjacn(i,jj,k)
        convb = (zetxb(i,j,k)*ub(i,j,k)
     *        +  zetyb(i,j,k)*vb(i,j,k)
     *        +  zetzb(i,j,k)*wb(i,j,k))*rhob(i,j,k)/tjacb(i,j,k)
        convf = (zetxb(i,j,kk)*ub(i,j,kk)
     *        +  zetyb(i,j,kk)*vb(i,j,kk)
     *        +  zetzb(i,j,kk)*wb(i,j,kk))*rhob(i,j,kk)/tjacb(i,j,kk)

        diffe = (exixe(i,j,k)**2+exiye(i,j,k)**2+exize(i,j,k)**2)
     *          *vmuturb(i,j,k)/(tjace(i,j,k)*delexie(i,j,k)*sigme)
        diffw = (exixe(ii,j,k)**2+exiye(ii,j,k)**2+exize(ii,j,k)**2)
     *          *vmuturb(i,j,k)/(tjace(ii,j,k)*delexie(ii,j,k)*sigme)
        diffn = (etaxn(i,j,k)**2+etayn(i,j,k)**2+etazn(i,j,k)**2)
     *          *vmuturb(i,j,k)/(tjacn(i,j,k)*deletan(i,j,k)*sigme)
        diffs = (etaxn(i,jj,k)**2+etayn(i,jj,k)**2+etazn(i,jj,k)**2)
     *          *vmuturb(i,j,k)/(tjacn(i,jj,k)*deletan(i,jj,k)*sigme)
        diffb = (zetxb(i,j,k)**2+zetyb(i,j,k)**2+zetzb(i,j,k)**2)
     *         *vmuturb(i,j,k)/(tjacb(i,j,k)*delzetb(i,j,k)*sigme)
        difff = (zetxb(i,j,kk)**2+zetyb(i,j,kk)**2+zetzb(i,j,kk)**2)
     *         *vmuturb(i,j,k)/(tjacb(i,j,kk)*delzetb(i,j,kk)*sigme)

***   First calculate the source term coefficients

      ss1 = 0.0
      ss2 = 0.0
      ss3 = 0.0
      if(igrid.eq.1)goto 15

      bb1e = (exixe(i,j,k)*etaxe(i,j,k)
     *     +  exiye(i,j,k)*etaye(i,j,k)
     *     +  exize(i,j,k)*etaze(i,j,k))
     *      * vmuturb(i,j,k)/(tjace(i,j,k)*sigme)
      bb1w = (exixe(ii,j,k)*etaxe(ii,j,k)
     *     +  exiye(ii,j,k)*etaye(ii,j,k)
     *     +  exize(ii,j,k)*etaze(ii,j,k))
     *      * vmuturb(i,j,k)/(tjace(ii,j,k)*sigme)
      bb2e = (exixe(i,j,k)*zetxe(i,j,k)
     *     +  exiye(i,j,k)*zetye(i,j,k)
     *     +  exize(i,j,k)*zetze(i,j,k))
     *      * vmuturb(i,j,k)/(tjace(i,j,k)*sigme)
      bb2w = (exixe(ii,j,k)*zetxe(ii,j,k)
     *     +  exiye(ii,j,k)*zetye(ii,j,k)
     *     +  exize(ii,j,k)*zetze(ii,j,k))
     *      * vmuturb(i,j,k)/(tjace(ii,j,k)*sigme)

      bb3n = (etaxn(i,j,k)*exixn(i,j,k)
     *     +  etayn(i,j,k)*exiyn(i,j,k)
     *     +  etazn(i,j,k)*exizn(i,j,k))
     *      * vmuturb(i,j,k)/(tjacn(i,j,k)*sigme)
      bb3s = (etaxn(i,jj,k)*exixn(i,jj,k)
     *     +  etayn(i,jj,k)*exiyn(i,jj,k)
     *     +  etazn(i,jj,k)*exizn(i,jj,k))
     *      * vmuturb(i,j,k)/(tjacn(i,jj,k)*sigme)
      bb4n = (etaxn(i,j,k)*zetxn(i,j,k)
     *     +  etayn(i,j,k)*zetyn(i,j,k)
     *     +  etazn(i,j,k)*zetzn(i,j,k))
     *      * vmuturb(i,j,k)/(tjacn(i,j,k)*sigme)
      bb4s = (etaxn(i,jj,k)*zetxn(i,jj,k)
     *     +  etayn(i,jj,k)*zetyn(i,jj,k)
     *     +  etazn(i,jj,k)*zetzn(i,jj,k))
     *      * vmuturb(i,j,k)/(tjacn(i,jj,k)*sigme)

      bb5b = (zetxb(i,j,k)*exixb(i,j,k)
     *     +  zetyb(i,j,k)*exiyb(i,j,k)
     *     +  zetzb(i,j,k)*exizb(i,j,k))
     *      * vmuturb(i,j,k)/(tjacb(i,j,k)*sigme)
      bb5f = (zetxb(i,j,kk)*exixb(i,j,kk)
     *     +  zetyb(i,j,kk)*exiyb(i,j,kk)
     *     +  zetzb(i,j,kk)*exizb(i,j,kk))
     *      * vmuturb(i,j,k)/(tjacb(i,j,kk)*sigme)
      bb6b = (zetxb(i,j,k)*etaxb(i,j,k)
     *     +  zetyb(i,j,k)*etayb(i,j,k)
     *    +  zetzb(i,j,k)*etazb(i,j,k))
     *     * vmuturb(i,j,k)/(tjacb(i,j,k)*sigme)
      bb6f = (zetxb(i,j,kk)*etaxb(i,j,kk)
     *    +  zetyb(i,j,kk)*etayb(i,j,kk)
     *    +  zetzb(i,j,kk)*etazb(i,j,kk))
     *     * vmuturb(i,j,k)/(tjacb(i,j,kk)*sigme)

      ss1 = bb1e*(epsne(i,j,k)  - epsne(i,jj,k))
     *    - bb1w*(epsne(ii,j,k) - epsne(ii,jj,k))
     *    + bb2e*(epsbe(i,j,k)  - epsbe(i,j,kk))
     *    - bb2w*(epsbe(ii,j,k) - epsbe(ii,j,kk))

      ss2 = bb3n*(epsne(i,j,k)   - epsne(ii,j,k))
     *    - bb3s*(epsne(i,jj,k)  - epsne(ii,jj,k))
     *    + bb4n*(epsnb(i,j,k)   - epsnb(i,j,kk))
     *    - bb4s*(epsnb(i,jj,k)  - epsnb(i,jj,kk))

      ss3 = bb5b*(epsbe(i,j,k)  - epsbe(ii,j,k))
     *    - bb5f*(epsbe(i,j,kk) - epsbe(ii,j,kk))
     *    + bb6b*(epsnb(i,j,k)  - epsnb(i,jj,k))
     *    - bb6f*(epsnb(i,j,kk) - epsnb(i,jj,kk))

   15 ccp = (2*c2-1)*eps(i,j,k)*rhop(i,j,k)/(tken(i,j,k)*tjac(i,j,k))
      bbb = (c1*Gamma(i,j,k)*vmuturb(i,j,k)
     *    + (c2-1)*rhop(i,j,k)*eps(i,j,k))*eps(i,j,k)
     *    /(tken(i,j,k)*tjac(i,j,k))

***   Determine the main finite difference coefficients

***   Eastern boundaries

        if(jbound(i,j,k).eq.0.and.jbound(i+1,j,k).eq.nbnd)then
         ae(i,j,k) = 0.0
         dwall = dabs(etay(i,j,k)*zetz(i,j,k)/(2*tjac(i,j,k)))
         epsmu = (cmu**0.75)*(dabs(tken(i,j,k))**1.5)/(cappa*dwall)
         ccp = great
         bbb = great*epsmu
        else
         AE(I,J,K) = DIFFE + DMAX1(-CONVE,ZERO)
        endif

***   Western boundaries

        if(jbound(i,j,k).eq.0.and.jbound(i-1,j,k).eq.nbnd)then
         aw(i,j,k) = 0.0
         dwall = dabs(etay(i,j,k)*zetz(i,j,k)/(2*tjac(i,j,k)))
         epsmu = (cmu**0.75)*(dabs(tken(i,j,k))**1.5)/(cappa*dwall)
         ccp = great
         bbb = great*epsmu
        else
         AW(I,J,K) = DIFFW + DMAX1( CONVW,ZERO)
        endif

***   Northern boundaries
```

```
      if(jbound(i,j,k).eq.0.and.jbound(i,j+1,k).eq.nbnd)then
       an(i,j,k) = 0.0
       dwall = dabs(exix(i,j,k)*zetz(i,j,k)/(2*tjac(i,j,k)))
       epsmu = (cmu**0.75)*(dabs(tken(i,j,k))**1.5)/(cappa*dwall)
       ccp = great
       bbb = great*epsmu
      else
       AN(I,J,K) = DIFFN + DMAX1(-CONVN,ZERO)
      endif

***   Southern boundaries

      if(jbound(i,j,k).eq.0.and.jbound(i,j-1,k).eq.nbnd)then
       as(i,j,k) = 0.0
       dwall = dabs(exix(i,j,k)*zetz(i,j,k)/(2*tjac(i,j,k)))
       epsmu = (cmu**0.75)*(dabs(tken(i,j,k))**1.5)/(cappa*dwall)
       ccp = great
       bbb = great*epsmu
      else
       AS(I,J,K) = DIFFS + DMAX1( CONVS,ZERO)
      endif

***   Back boundaries

      if(jbound(i,j,k).eq.0.and.jbound(i,j,k+1).eq.nbnd)then
       ab(i,j,k) = 0.0
       dwall = dabs(exix(i,j,k)*etay(i,j,k)/(2*tjac(i,j,k)))
       epsmu = (cmu**0.75)*(dabs(tken(i,j,k))**1.5)/(cappa*dwall)
       ccp = great
       bbb = great*epsmu
      else
       AB(I,J,K) = DIFFB + DMAX1(-CONVB,ZERO)
      endif

***   Front boundaries

      if(jbound(i,j,k).eq.0.and.jbound(i,j,k-1).eq.nbnd)then
       af(i,j,k) = 0.0
       dwall = dabs(exix(i,j,k)*etay(i,j,k)/(2*tjac(i,j,k)))
       epsmu = (cmu**0.75)*(dabs(tken(i,j,k))**1.5)/(cappa*dwall)
       ccp = great
       bbb = great*epsmu
      else
       AF(I,J,K) = DIFFF + DMAX1( CONVF,ZERO)
      endif

      apo(i,j,k) = rhoo(i,j,k)/(delt*tjac(i,j,k))
      ape(i,j,k) = ae(i,j,k) + aw(i,j,k) + an(i,j,k) + as(i,j,k)
     *           + ab(i,j,k) + af(i,j,k) + apo(i,j,k) + ccp

      source(i,j,k) = ss1 + ss2 + ss3 + bbb

   20 continue

      return
```

```
c-----------------------------------------------------------------------
      Entry EPREPI (iii,kkk,jne,jnb)
c-----------------------------------------------------------------------

***   This section prepares the TDMA coefficients
***   sweeping in I - direction

      jnb1 = jnb + 1
      jne1 = jne - 1

      jb = jbound(iii,jnb,kkk)
      aaj(jnb) = 1.0
      bbj(jnb) = be(jb)
      ccj(jnb) = 0.0
      ddj(jnb) = ce(jb)

      do 30 j = jnb1,jne1
        aaj(j) = ape(iii,j,kkk)/relaxt
        bbj(j) = an(iii,j,kkk)
        ccj(j) = as(iii,j,kkk)
        ddj(j) = ae(iii,j,kkk)*eps(iii+1,j,kkk)
     *          + aw(iii,j,kkk)*eps(iii-1,j,kkk)
     *          + ab(iii,j,kkk)*eps(iii,j,kkk+1)
     *          + af(iii,j,kkk)*eps(iii,j,kkk-1)
     *          + apo(iii,j,kkk)*epsold(iii,j,kkk) + source(iii,j,kkk)
     *          + ape(iii,j,kkk)*eps(iii,j,kkk)*(1-relaxt)/relaxt
   30 continue

      jb = jbound(iii,jne,kkk)
      aaj(jne) = 1.0
      bbj(jne) = 0.0
      ccj(jne) = be(jb)
      ddj(jne) = ce(jb)

      return

c-----------------------------------------------------------------------
      Entry EPREPJ (jjj,kkk,ine,inb)
c-----------------------------------------------------------------------

***   This section prepares the TDMA coefficients
***   sweeping in J - direction

      inb1 = inb + 1
      ine1 = ine - 1

      jb = jbound(inb,jjj,kkk)
      aai(inb) = 1.0
      bbi(inb) = be(jb)
      cci(inb) = 0.0
      ddi(inb) = ce(jb)

      do 40 i = inb1,ine1
        aai(i) = ape(i,jjj,kkk)/relaxt
```

```
         bbi(i) = ae(i,jjj,kkk)
         cci(i) = aw(i,jjj,kkk)
         ddi(i) = an(i,jjj,kkk)*eps(i,jjj+1,kkk)
     *          + as(i,jjj,kkk)*eps(i,jjj-1,kkk)
     *          + ab(i,jjj,kkk)*eps(i,jjj,kkk+1)
     *          + af(i,jjj,kkk)*eps(i,jjj,kkk-1)
     *          + apo(i,jjj,kkk)*epsold(i,jjj,kkk) + source(i,jjj,kkk)
     *          + ape(i,jjj,kkk)*eps(i,jjj,kkk)*(1-relaxt)/relaxt
   40 continue

      jb = jbound(ine,jjj,kkk)
      aai(ine) = 1.0
      bbi(ine) = 0.0
      cci(ine) = be(jb)
      ddi(ine) = ce(jb)

      return
      end

*************************************************************************
      Subroutine SOLVE
*************************************************************************

***   This subroutine solves the flow with TDMA

      INCLUDE 'COMM'

c------------------------------------------------------------------------
      Entry TDMAI (ii,kk,fi,nje,njb)
c------------------------------------------------------------------------

***   TDMA along a North-South strip

      njb1 = njb + 1
      nje1 = nje - 1

      ppj(njb) = bbj(njb)/aaj(njb)
      qqj(njb) = ddj(njb)/aaj(njb)

      do 10 j = njb1,nje
       denom = aaj(j) - ccj(j)*ppj(j-1)
       ppj(j) = bbj(j)/denom
       qqj(j) = (ddj(j) + ccj(j)*qqj(j-1))/denom
   10 continue

      fi(ii,nje,kk) = qqj(nje)
      do 20 j = nje1,njb,-1
       fi(ii,j,kk) = ppj(j)*fi(ii,j+1,kk) + qqj(j)
   20 continue

      return

c------------------------------------------------------------------------
      Entry TDMAJ (jj,kk,fi,nie,nib)
c------------------------------------------------------------------------
```

```
***    TDMA along a East-West strip

       nib1 = nib + 1
       nie1 = nie - 1

       ppi(nib) = bbi(nib)/aai(nib)
       qqi(nib) = ddi(nib)/aai(nib)

       do 30 i = nib1,nie
        denom = aai(i) - cci(i)*ppi(i-1)
        ppi(i) = bbi(i)/denom
        qqi(i) = (ddi(i) + cci(i)*qqi(i-1))/denom
    30 continue

       fi(nie,jj,kk) = qqi(nie)
       do 40 i = nie1,nib,-1
        fi(i,jj,kk) = ppi(i)*fi(i+1,jj,kk) + qqi(i)
    40 continue

       return
       end

************************************************************************
       Subroutine ADJUST
************************************************************************

***    This subroutine corrects the pressures and the velocities
***    for cellwise continuity

       INCLUDE 'COMM'

       do 10 k = 2,knm1
        do 10 i = 2,inm1
         do 20 j = 2,jnm1
          if(jbound(i,j,k).gt.0)goto 20

***    Correct the pressures

          pp(i,j,k) = pp(i,j,k) + pcor(i,j,k)

***    Correct the velocities

         vxcor =  (exix(i-1,j,k)*pcor(i-1,j,k)/tjac(i-1,j,k)
     *          - exix(i+1,j,k)*pcor(i+1,j,k)/tjac(i+1,j,k)
     *        + etax(i,j-1,k)*pcor(i,j-1,k)/tjac(i,j-1,k)
     *       - etax(i,j+1,k)*pcor(i,j+1,k)/tjac(i,j+1,k)
     *      + zetx(i,j,k-1)*pcor(i,j,k-1)/tjac(i,j,k-1)
     *     - zetx(i,j,k+1)*pcor(i,j,k+1)/tjac(i,j,k+1))/(2*apu(i,j,k)
     *  *(1/relax - 1))

         vxstar(i,j,k) = vxstar(i,j,k) + vxcor

         vycor =  (exiy(i-1,j,k)*pcor(i-1,j,k)/tjac(i-1,j,k)
     *          - exiy(i+1,j,k)*pcor(i+1,j,k)/tjac(i+1,j,k)
     *         + etay(i,j-1,k)*pcor(i,j-1,k)/tjac(i,j-1,k)
     *        - etay(i,j+1,k)*pcor(i,j+1,k)/tjac(i,j+1,k)
     *       + zety(i,j,k-1)*pcor(i,j,k-1)/tjac(i,j,k-1)
     *      - zety(i,j,k+1)*pcor(i,j,k+1)/tjac(i,j,k+1))/(2*apv(i,j,k)
     *  *(1/relax - 1))

        vystar(i,j,k) = vystar(i,j,k) + vycor

        vzcor =  (exiz(i-1,j,k)*pcor(i-1,j,k)/tjac(i-1,j,k)
     *          - exiz(i+1,j,k)*pcor(i+1,j,k)/tjac(i+1,j,k)
     *         + etaz(i,j-1,k)*pcor(i,j-1,k)/tjac(i,j-1,k)
     *        - etaz(i,j+1,k)*pcor(i,j+1,k)/tjac(i,j+1,k)
     *       + zetz(i,j,k-1)*pcor(i,j,k-1)/tjac(i,j,k-1)
     *      - zetz(i,j,k+1)*pcor(i,j,k+1)/tjac(i,j,k+1))/(2*apw(i,j,k)
     *  *(1/relax - 1))

        vzstar(i,j,k) = vzstar(i,j,k) + vzcor

   20  continue
   10 continue

      do 40 k = 2,knm1
       do 40 j = 2,jnm1
        do 40 i = 2,inm1
        if(jbound(i,j,k).gt.0)goto 40
        vx(i,j,k) = vxstar(i,j,k)
        vy(i,j,k) = vystar(i,j,k)
        vz(i,j,k) = vzstar(i,j,k)
   40 continue

      return
      end

*************************************************************************
      Function APECL (peclet)
*************************************************************************

***   This function selects the type of differencing

      INCLUDE 'COMM'

***   Central difference
      if(kapecl.eq.1) apecl = 1.0-0.5*peclet
***   Upwind difference
      if(kapecl.eq.2) apecl = 1.0
***   Hybrid difference
      if(kapecl.eq.3) apecl = dmax1(zero,1.0-0.5*peclet)
***   Power law difference
      if(kapecl.eq.4) apecl = dmax1(zero,(1.0-0.1*peclet)**5)
***   Exponential difference
      if(kapecl.eq.5)then
      if(peclet.eq.zero)peclet = 0.00000001
      apecl = peclet/(exp(peclet)-1.0)
      endif
      end
************************************************************************
      Subroutine INTPOLV
************************************************************************

***   This subroutine interpolates the velocities at the cell walls

      INCLUDE 'COMM'

***   (u,v,w) - Velocities represents interpolated velocities between
***   major grid points and (vx,vy,vz) - velocities are the
***   calculated velocities

***   Interpolate velocities on Eastern cell walls

      do 10 k = 2,knml
       do 10 j = 2,jnml
        do 10 ll = 1,lobs(j,k)
         nie2 = iiend(ll,j,k) - 2
         nib1 = iibeg(ll,j,k) + 1
         do 10 i = nib1,nie2
          ue(i,j,k) = vxstar(i+1,j,k)*f1(i,j,k)
     *              + vxstar(i,j,k)*(1-f1(i,j,k))
          ve(i,j,k) = vystar(i+1,j,k)*f1(i,j,k)
     *              + vystar(i,j,k)*(1-f1(i,j,k))
          we(i,j,k) = vzstar(i+1,j,k)*f1(i,j,k)
     *              + vzstar(i,j,k)*(1-f1(i,j,k))
   10 continue

***   Interpolate velocities on Northern cell walls

      do 20 k = 2,knml
       do 20 i = 2,inml
        do 20 mm = 1,mobs(i,k)
         nje2 = jjend(mm,i,k) - 2
         njb1 = jjbeg(mm,i,k) + 1
         do 20 j = njb1,nje2
          un(i,j,k) = vxstar(i,j+1,k)*f2(i,j,k)
     *              + vxstar(i,j,k)*(1-f2(i,j,k))
          vn(i,j,k) = vystar(i,j+1,k)*f2(i,j,k)
     *              + vystar(i,j,k)*(1-f2(i,j,k))
          wn(i,j,k) = vzstar(i,j+1,k)*f2(i,j,k)
     *              + vzstar(i,j,k)*(1-f2(i,j,k))
   20 continue

***   Interpolate velocities on Backward facing cell walls

      do 30 j = 2,jnml
       do 30 i = 2,inml
        do 30 nn = 1,nobs(i,j)
         nke2 = kkend(nn,i,j) - 2
         nkb1 = kkbeg(nn,i,j) + 1
         do 30 k = nkb1,nke2
          ub(i,j,k) = vxstar(i,j,k+1)*f3(i,j,k)
     *              + vxstar(i,j,k)*(1-f3(i,j,k))
          vb(i,j,k) = vystar(i,j,k+1)*f3(i,j,k)
     *              + vystar(i,j,k)*(1-f3(i,j,k))
          wb(i,j,k) = vzstar(i,j,k+1)*f3(i,j,k)
     *              + vzstar(i,j,k)*(1-f3(i,j,k))
   30 continue

***   Interpolation at Eastern cell walls for strong
***   pressure-velocity coupling

      do 40 k = 2,knm1
       do 40 j = 2,jnm1
        do 50 ll = 1,lobs(j,k)
         nie2 = iiend(ll,j,k) - 2
         nie3 = iiend(ll,j,k) - 3
         nib1 = iibeg(ll,j,k) + 1
         nib2 = iibeg(ll,j,k) + 2
         if(nie3.lt.nib2)goto 40
         do 60 i = nib2,nie3
          l = i-1
          ue(i,j,k) = ue(i,j,k)
     * -  (exixe(l,j,k)*pie(l,j,k)/tjace(l,j,k)
     * -   exixe(i,j,k)*pie(i,j,k)/tjace(i,j,k))/(2*apu(i,j,k))
     * -  (exixe(i,j,k)*pie(i,j,k)/tjace(i,j,k)
     * -   exixe(i+1,j,k)*pie(i+1,j,k)/tjace(i+1,j,k))/(2*apu(i+1,j,k))
     * +  (exix(i,j,k)*pp(i,j,k)/tjac(i,j,k)
     * -   exix(i+1,j,k)*pp(i+1,j,k)/tjac(i+1,j,k))
     *     /((apu(i,j,k)+apu(i+1,j,k))/2)
          ve(i,j,k) = ve(i,j,k)
     * -  (exiye(l,j,k)*pie(l,j,k)/tjace(l,j,k)
     * -   exiye(i,j,k)*pie(i,j,k)/tjace(i,j,k))/(2*apv(i,j,k))
     * -  (exiye(i,j,k)*pie(i,j,k)/tjace(i,j,k)
     * -   exiye(i+1,j,k)*pie(i+1,j,k)/tjace(i+1,j,k))/(2*apv(i+1,j,k))
     * +  (exiy(i,j,k)*pp(i,j,k)/tjac(i,j,k)
     * -   exiy(i+1,j,k)*pp(i+1,j,k)/tjac(i+1,j,k))
     *     /((apv(i,j,k)+apv(i+1,j,k))/2)
          we(i,j,k) = we(i,j,k)
     * -  (exize(l,j,k)*pie(l,j,k)/tjace(l,j,k)
     * -   exize(i,j,k)*pie(i,j,k)/tjace(i,j,k))/(2*apw(i,j,k))
     * -  (exize(i,j,k)*pie(i,j,k)/tjace(i,j,k)
     * -   exize(i+1,j,k)*pie(i+1,j,k)/tjace(i+1,j,k))/(2*apw(i+1,j,k))
     * +  (exiz(i,j,k)*pp(i,j,k)/tjac(i,j,k)
     * -   exiz(i+1,j,k)*pp(i+1,j,k)/tjac(i+1,j,k))
     *     /((apw(i,j,k)+apw(i+1,j,k))/2)
   60    continue
   50   continue
   40 continue

***   Interpolation at Northern cell walls for strong
***   pressure-velocity coupling

      do 70 k = 2,knm1
       do 70 i = 2,inm1
        do 80 mm = 1,mobs(i,k)
         nje2 = jjend(mm,i,k) - 2
        nje3 = jjend(mm,i,k) - 3
        njb1 = jjbeg(mm,i,k) + 1
        njb2 = jjbeg(mm,i,k) + 2
        if(nje3.lt.njb2)goto 70
        do 90 j = njb2,nje3
         m = j-1
         un(i,j,k) = un(i,j,k)
     * -  (etaxn(i,m,k)*pin(i,m,k)/tjacn(i,m,k)
     * -   etaxn(i,j,k)*pin(i,j,k)/tjacn(i,j,k))/(2*apu(i,j,k))
     * -  (etaxn(i,j,k)*pin(i,j,k)/tjacn(i,j,k)
     * -   etaxn(i,j+1,k)*pin(i,j+1,k)/tjacn(i,j+1,k))/(2*apu(i,j+1,k))
     * +  (etax(i,j,k)*pp(i,j,k)/tjac(i,j,k)
     * -   etax(i,j+1,k)*pp(i,j+1,k)/tjac(i,j+1,k))
     *     /((apu(i,j+1,k)+apu(i,j,k))/2)
         vn(i,j,k) = vn(i,j,k)
     * -  (etayn(i,m,k)*pin(i,m,k)/tjacn(i,m,k)
     * -   etayn(i,j,k)*pin(i,j,k)/tjacn(i,j,k))/(2*apv(i,j,k))
     * -  (etayn(i,j,k)*pin(i,j,k)/tjacn(i,j,k)
     * -   etayn(i,j+1,k)*pin(i,j+1,k)/tjacn(i,j+1,k))/(2*apv(i,j+1,k))
     * +  (etay(i,j,k)*pp(i,j,k)/tjac(i,j,k)
     * -   etay(i,j+1,k)*pp(i,j+1,k)/tjac(i,j+1,k))
     *     /((apv(i,j+1,k)+apv(i,j,k))/2)
         wn(i,j,k) = wn(i,j,k)
     * -  (etazn(i,m,k)*pin(i,m,k)/tjacn(i,m,k)
     * -   etazn(i,j,k)*pin(i,j,k)/tjacn(i,j,k))/(2*apw(i,j,k))
     * -  (etazn(i,j,k)*pin(i,j,k)/tjacn(i,j,k)
     * -   etazn(i,j+1,k)*pin(i,j+1,k)/tjacn(i,j+1,k))/(2*apw(i,j+1,k))
     * +  (etaz(i,j,k)*pp(i,j,k)/tjac(i,j,k)
     * -   etaz(i,j+1,k)*pp(i,j+1,k)/tjac(i,j+1,k))
     *     /((apw(i,j+1,k)+apw(i,j,k))/2)
   90   continue
   80  continue
   70 continue

***   Interpolation at Backward facing cell walls for strong
***   pressure-velocity coupling

      do 100 j = 2,jnm1
       do 100 i = 2,inm1
        do 110 nn = 1,nobs(i,j)
         nke2 = kkend(nn,i,j) - 2
         nke3 = kkend(nn,i,j) - 3
         nkb1 = kkbeg(nn,i,j) + 1
         nkb2 = kkbeg(nn,i,j) + 2
         if(nke3.lt.nkb2)goto 100
         do 120 k = nkb2,nke3
          n = k-1
          ub(i,j,k) = ub(i,j,k)
     * -  (zetxb(i,j,n)*pib(i,j,n)/tjacb(i,j,n)
     * -   zetxb(i,j,k)*pib(i,j,k)/tjacb(i,j,k))/(2*apu(i,j,k))
     * -  (zetxb(i,j,k)*pib(i,j,k)/tjacb(i,j,k)
     * -   zetxb(i,j,k+1)*pib(i,j,k+1)/tjacb(i,j,k+1))/(2*apu(i,j,k+1))
     * +  (zetx(i,j,k)*pp(i,j,k)/tjac(i,j,k)
     * -   zetx(i,j,k+1)*pp(i,j,k+1)/tjac(i,j,k+1))
     *     /((apu(i,j,k+1)+apu(i,j,k))/2)
```

```
           vb(i,j,k) = vb(i,j,k)
     * -   (zetyb(i,j,n)*pib(i,j,n)/tjacb(i,j,n)
     * -    zetyb(i,j,k)*pib(i,j,k)/tjacb(i,j,k))/(2*apv(i,j,k))
     * -   (zetyb(i,j,k)*pib(i,j,k)/tjacb(i,j,k)
     * -    zetyb(i,j,k+1)*pib(i,j,k+1)/tjacb(i,j,k+1))/(2*apv(i,j,k+1))
     * +   (zety(i,j,k)*pp(i,j,k)/tjac(i,j,k)
     * -    zety(i,j,k+1)*pp(i,j,k+1)/tjac(i,j,k+1))
     *      /((apv(i,j,k+1)+apv(i,j,k))/2)
           wb(i,j,k) = wb(i,j,k)
     * -   (zetzb(i,j,n)*pib(i,j,n)/tjacb(i,j,n)
     * -    zetzb(i,j,k)*pib(i,j,k)/tjacb(i,j,k))/(2*apw(i,j,k))
     * -   (zetzb(i,j,k)*pib(i,j,k)/tjacb(i,j,k)
     * -    zetzb(i,j,k+1)*pib(i,j,k+1)/tjacb(i,j,k+1))/(2*apw(i,j,k+1))
     * +   (zetz(i,j,k)*pp(i,j,k)/tjac(i,j,k)
     * -    zetz(i,j,k+1)*pp(i,j,k+1)/tjac(i,j,k+1))
     *      /((apw(i,j,k+1)+apw(i,j,k))/2)
  120    continue
  110   continue
  100 continue

  130 return
      end

*************************************************************************
      Subroutine INTPOL
*************************************************************************

***   This subroutine interpolates the pressures and the velocities

      INCLUDE 'COMM'

***   Interpolate velocities and pressure on Eastern cell walls

      do 10 k = 2,knm1
       do 10 j = 2,jnm1
        do 10 ll = 1,lobs(j,k)
         nie1 = iiend(ll,j,k) - 1
         nie2 = iiend(ll,j,k) - 2
         nib = iibeg(ll,j,k)
         nib1 = iibeg(ll,j,k) + 1
         pie(nib,j,k) = pp(nib1,j,k)
         pie(nie1,j,k) = pp(nie1,j,k)
         do 10 i = nib1,nie2
          ue(i,j,k) = vx(i+1,j,k)*f1(i,j,k)
     *              + vx(i,j,k)*(1-f1(i,j,k))
          ve(i,j,k) = vy(i+1,j,k)*f1(i,j,k)
     *              + vy(i,j,k)*(1-f1(i,j,k))
          we(i,j,k) = vz(i+1,j,k)*f1(i,j,k)
     *              + vz(i,j,k)*(1-f1(i,j,k))
          pie(i,j,k) = pp(i+1,j,k)*f1(i,j,k)
     *               + pp(i,j,k)*(1-f1(i,j,k))
         if(j.eq.jnm1)goto 10
         ffa = (1-f1(i,j+1,k))*((f2(i,j,k)+f2(i+1,j,k))/2)
         ffb = f1(i,j+1,k)*((f2(i,j,k)+f2(i+1,j,k))/2)
         ffc = f1(i,j,k)*(1-(f2(i,j,k)+f2(i+1,j,k))/2)
        ffd = (1-f1(i,j,k))*(1-(f2(i,j,k)+f2(i+1,j,k))/2)
        une(i,j,k) = ffa*vx(i,j+1,k) + ffb*vx(i+1,j+1,k)
     *               + ffc*vx(i+1,j,k) + ffd*vx(i,j,k)
        vne(i,j,k) = ffa*vy(i,j+1,k) + ffb*vy(i+1,j+1,k)
     *               + ffc*vy(i+1,j,k) + ffd*vy(i,j,k)
        wne(i,j,k) = ffa*vz(i,j+1,k) + ffb*vz(i+1,j+1,k)
     *               + ffc*vz(i+1,j,k) + ffd*vz(i,j,k)
   10 continue

***   Interpolate velocities and pressure on Northern cell walls

      do 20 k = 2,knm1
       do 20 i = 2,inm1
        do 20 mm = 1,mobs(i,k)
         nje1 = jjend(mm,i,k) - 1
         nje2 = jjend(mm,i,k) - 2
         njb = jjbeg(mm,i,k)
         njb1 = jjbeg(mm,i,k) + 1
         pin(i,njb,k) = pp(i,njb1,k)
         pin(i,nje1,k) = pp(i,nje1,k)
         do 20 j = njb1,nje2
          un(i,j,k) = vx(i,j+1,k)*f2(i,j,k)
     *               + vx(i,j,k)*(1-f2(i,j,k))
          vn(i,j,k) = vy(i,j+1,k)*f2(i,j,k)
     *               + vy(i,j,k)*(1-f2(i,j,k))
          wn(i,j,k) = vz(i,j+1,k)*f2(i,j,k)
     *               + vz(i,j,k)*(1-f2(i,j,k))
          pin(i,j,k) = pp(i,j+1,k)*f2(i,j,k)
     *                + pp(i,j,k)*(1-f2(i,j,k))
         if(k.eq.knm1)goto 20
         ffa = (1-f3(i,j+1,k))*((f2(i,j,k)+f2(i,j,k+1))/2)
         ffb = f3(i,j+1,k)*((f2(i,j,k)+f2(i,j,k+1))/2)
         ffc = f3(i,j,k)*(1-(f2(i,j,k)+f2(i,j,k+1))/2)
         ffd = (1-f3(i,j,k))*(1-(f2(i,j,k)+f2(i,j,k+1))/2)
         unb(i,j,k) = ffa*vx(i,j+1,k) + ffb*vx(i,j+1,k+1)
     *               + ffc*vx(i,j,k+1) + ffd*vx(i,j,k)
         vnb(i,j,k) = ffa*vy(i,j+1,k) + ffb*vy(i,j+1,k+1)
     *               + ffc*vy(i,j,k+1) + ffd*vy(i,j,k)
         wnb(i,j,k) = ffa*vz(i,j+1,k) + ffb*vz(i,j+1,k+1)
     *               + ffc*vz(i,j,k+1) + ffd*vz(i,j,k)
   20 continue

***   Interpolate velocities and pressure on Backward facing cell walls

      do 30 j = 2,jnm1
       do 30 i = 2,inm1
        do 30 nn = 1,nobs(i,j)
         nke1 = kkend(nn,i,j) - 1
         nke2 = kkend(nn,i,j) - 2
         nkb = kkbeg(nn,i,j)
         nkb1 = kkbeg(nn,i,j) + 1
         pib(i,j,nkb) = pp(i,j,nkb1)
         pib(i,j,nke1) = pp(i,j,nke1)
         do 30 k = nkb1,nke2
          ub(i,j,k) = vx(i,j,k+1)*f3(i,j,k)
     *             + vx(i,j,k)*(1-f3(i,j,k))
          vb(i,j,k) = vy(i,j,k+1)*f3(i,j,k)
     *             + vy(i,j,k)*(1-f3(i,j,k))
          wb(i,j,k) = vz(i,j,k+1)*f3(i,j,k)
     *             + vz(i,j,k)*(1-f3(i,j,k))
          pib(i,j,k) = pp(i,j,k+1)*f3(i,j,k)
     *              + pp(i,j,k)*(1-f3(i,j,k))
         if(i.eq.inml)goto 30
         ffa = (1-f1(i,j,k+1))*((f3(i,j,k)+f3(i+1,j,k))/2)
         ffb = f1(i,j,k+1)*((f3(i,j,k)+f3(i+1,j,k))/2)
         ffc = f1(i,j,k)*(1-(f3(i,j,k)+f3(i+1,j,k))/2)
         ffd = (1-f1(i,j,k))*(1-(f3(i,j,k)+f3(i+1,j,k))/2)
         ube(i,j,k) = ffa*vx(i,j,k+1) + ffb*vx(i+1,j,k+1)
     *             + ffc*vx(i+1,j,k) + ffd*vx(i,j,k)
         vbe(i,j,k) = ffa*vy(i,j,k+1) + ffb*vy(i+1,j,k+1)
     *             + ffc*vy(i+1,j,k) + ffd*vy(i,j,k)
         wbe(i,j,k) = ffa*vz(i,j,k+1) + ffb*vz(i+1,j,k+1)
     *             + ffc*vz(i+1,j,k) + ffd*vz(i,j,k)
   30 continue

      return
      end

*************************************************************************
      Subroutine INTPOLKE
*************************************************************************

***   This subroutine interpolates for k and epsilon

      INCLUDE 'COMM'

***   Set the k and epsilon values on boundary edges

      do 10 i = 1,inml
       eps(i,1,1) = eps(i,2,2)
       eps(i,1,kn) = eps(i,2,knml)
       eps(i,jn,1) = eps(i,jnml,2)
       eps(i,jn,kn) = eps(i,jnml,knml)
       tken(i,1,1) = tken(i,2,2)
       tken(i,1,kn) = tken(i,2,knml)
       tken(i,jn,1) = tken(i,jnml,2)
       tken(i,jn,kn) = tken(i,jnml,knml)
   10 continue

      do 20 j = 1,jnml
       eps(1,j,1) = eps(2,j,2)
       eps(1,j,kn) = eps(2,j,knml)
       eps(in,j,1) = eps(inml,j,2)
       eps(in,j,kn) = eps(inml,j,knml)
       tken(1,j,1) = tken(2,j,2)
       tken(1,j,kn) = tken(2,j,knml)
       tken(in,j,1) = tken(inml,j,2)
       tken(in,j,kn) = tken(inml,j,knml)
   20 continue
```

```
      do 30 k = 1,knml
       eps(1,1,k) = eps(2,2,k)
       eps(in,1,k) = eps(inml,2,k)
       eps(1,jn,k) = eps(2,jnml,k)
       eps(in,jn,k) = eps(inml,jnml,k)
       tken(1,1,k) = tken(2,2,k)
       tken(in,1,k) = tken(inml,2,k)
       tken(1,jn,k) = tken(2,jnml,k)
       tken(in,jn,k) = tken(inml,jnml,k)
   30 continue

***   Interpolate for k and epsilon

      do 40 k = 1,knml
       do 40 j = 1,jnml
        do 40 i = 1,inml
         ffa = (1-f1(i,j+1,k))*((f2(i,j,k)+f2(i+1,j,k))/2)
         ffb = f1(i,j+1,k)*((f2(i,j,k)+f2(i+1,j,k))/2)
         ffc = f1(i,j,k)*(1-(f2(i,j,k)+f2(i+1,j,k))/2)
         ffd = (1-f1(i,j,k))*(1-(f2(i,j,k)+f2(i+1,j,k))/2)
         tkenne(i,j,k) = ffa*tken(i,j+1,k) + ffb*tken(i+1,j+1,k)
     *                 + ffc*tken(i+1,j,k) + ffd*tken(i,j,k)
         epsne(i,j,k) = ffa*eps(i,j+1,k) + ffb*eps(i+1,j+1,k)
     *                + ffc*eps(i+1,j,k) + ffd*eps(i,j,k)
         ffa = (1-f3(i,j+1,k))*((f2(i,j,k)+f2(i,j,k+1))/2)
         ffb = f3(i,j+1,k)*((f2(i,j,k)+f2(i,j,k+1))/2)
         ffc = f3(i,j,k)*(1-(f2(i,j,k)+f2(i,j,k+1))/2)
         ffd = (1-f3(i,j,k))*(1-(f2(i,j,k)+f2(i,j,k+1))/2)
         tkennb(i,j,k) = ffa*tken(i,j+1,k) + ffb*tken(i,j+1,k+1)
     *                 + ffc*tken(i,j,k+1) + ffd*tken(i,j,k)
         epsnb(i,j,k) = ffa*eps(i,j+1,k) + ffb*eps(i,j+1,k+1)
     *                + ffc*eps(i,j,k+1) + ffd*eps(i,j,k)
         ffa = (1-f1(i,j,k+1))*((f3(i,j,k)+f3(i+1,j,k))/2)
         ffb = f1(i,j,k+1)*((f3(i,j,k)+f3(i+1,j,k))/2)
         ffc = f1(i,j,k)*(1-(f3(i,j,k)+f3(i+1,j,k))/2)
         ffd = (1-f1(i,j,k))*(1-(f3(i,j,k)+f3(i+1,j,k))/2)
         tkenbe(i,j,k) = ffa*tken(i,j,k+1) + ffb*tken(i+1,j,k+1)
     *                 + ffc*tken(i+1,j,k) + ffd*tken(i,j,k)
         epsbe(i,j,k) = ffa*eps(i,j,k+1) + ffb*eps(i+1,j,k+1)
     *                + ffc*eps(i+1,j,k) + ffd*eps(i,j,k)
   40 continue

      return
      end

**********************************************************************
      Subroutine EFFVISC
**********************************************************************

***   This subroutine calculates the effective viscosity

      INCLUDE 'COMM'

      difftken = 0.0
      diffeps = 0.0
```

```
***   Calculate the effective viscosity and the
***   turbulence convergence parameters

      do 10 k = 2,knm1
       do 10 j = 2,jnm1
        do 10 i = 2,inm1
         if(jbound(i,j,k).ne.0)goto 10
         vmu = vmulam + vmuturb(i,j,k)
         viscp(i,j,k) = viscp(i,j,k) + (vmu-viscp(i,j,k))*0.5
         difftken = difftken + tken(i,j,k)
         diffeps = diffeps + eps(i,j,k)
   10 continue

      difftken = dabs(difftken - restken)/restken
      diffeps = dabs(diffeps -reseps)/reseps

***   Apply the outflow boundary condition to viscosity

      k = kn
      do 20 j = 2,jnm1
       do 20 i = 2,inm1
        jb = jbound(i,j,k)
        viscp(i,j,k) = be(jb)*viscp(i,j,k-1)
   20 continue

***   Interpolate viscosities on Eastern cell walls

      do 30 k = 2,knm1
       do 30 j = 2,jnm1
        do 30 ll = 1,lobs(j,k)
         nie1 = iiend(ll,j,k) - 1
         nie2 = iiend(ll,j,k) - 2
         nib = iibeg(ll,j,k)
         nib1 = iibeg(ll,j,k) + 1
         visce(nib,j,k) = viscp(nib1,j,k)
         visce(nie1,j,k) = viscp(nie1,j,k)
         do 30 i = nib1,nie2
          visce(i,j,k) = viscp(i+1,j,k)*f1(i,j,k)
     *                 + viscp(i,j,k)*(1-f1(i,j,k))
   30 continue

***   Interpolate viscosities on Northern cell walls

      do 40 k = 2,knm1
       do 40 i = 2,inm1
        do 40 mm = 1,mobs(i,k)
         nje1 = jjend(mm,i,k) - 1
         nje2 = jjend(mm,i,k) - 2
         njb = jjbeg(mm,i,k)
         njb1 = jjbeg(mm,i,k) + 1
         viscn(i,njb,k) = viscp(i,njb1,k)
         viscn(i,nje1,k) = viscp(i,nje1,k)
         do 40 j = njb1,nje2
          viscn(i,j,k) = viscp(i,j+1,k)*f2(i,j,k)
     *                 + viscp(i,j,k)*(1-f2(i,j,k))
   40 continue

***   Interpolate viscosities on Backward facing cell walls

      do 50 j = 2,jnm1
       do 50 i = 2,inm1
        do 50 nn = 1,nobs(i,j)
         nke1 = kkend(nn,i,j) - 1
         nke2 = kkend(nn,i,j) - 2
         nkb = kkbeg(nn,i,j)
         nkb1 = kkbeg(nn,i,j) + 1
         viscb(i,j,nkb) = viscp(i,j,nkb1)
         viscb(i,j,nke1) = viscp(i,j,nke1)
         do 50 k = nkb1,nke2
          viscb(i,j,k) = viscp(i,j,k+1)*f3(i,j,k)
     *                 + viscp(i,j,k)*(1-f3(i,j,k))
   50 continue

      return
      end

*************************************************************************
      Subroutine SETBND
*************************************************************************

***   This subroutine provides the boundary conditions

      INCLUDE 'COMM'

      Dimension ibeg(20),iend(20),jbeg(20),jend(20),kbeg(20),kend(20)

***------------------------------------------------------------------------
***   Part 1 : Read boundary input data
***------------------------------------------------------------------------

      do 10 k = 1,kn
       do 10 j = 1,jn
        do 10 i = 1,in
         jbound(i,j,k) = 0
   10 continue

      read(4,'(a)')dummy
      read(4,'(a)')dummy
      read(4,*)nbnd

***   Define the different boundaries - read input

      do 20 l = 1,nbnd
       read(4,'(a)')dummy
       read(4,'(a)')dummy
       read(4,'(a)')dummy
       read(4,*)bu(l),cu(l),bv(l),cv(l),bw(l),cw(l),bp(l),cp(l),
     *          be(l),ce(l),bk(l),ck(l)
       read(4,'(a)')dummy
      read(4,*)nfunc

      do 30 n = 1,nfunc
       read(4,'(a)')dummy
       read(4,*)ibeg(n),iend(n),jbeg(n),jend(n),kbeg(n),kend(n)
       do 40 k = 1,kn
        do 40 j = 1,jn
         do 40 i = 1,in
          if(i.ge.ibeg(n).and.i.le.iend(n).and.
     *       j.ge.jbeg(n).and.j.le.jend(n).and.
     *       k.ge.kbeg(n).and.k.le.kend(n))then
           jbound(i,j,k) = 1
          endif
   40   continue
   30  continue
   20 continue

***-----------------------------------------------------------------------
***   Part 2 : Initial velocity and turbulence distribution
***-----------------------------------------------------------------------

***   Inflow profiles

      k = 1
      do 50 j = 1,jn
       READ(4,*)JCH,VZZ,TKENN,EPSS
       do 50 i = 1,in
c        eps(i,j,k) = ce(1)
c        tken(i,j,k) = ck(1)
        eps(i,j,k) = epss
        tken(i,j,k) = tkenn
        if(iturb.eq.1)then
         vmuturb(i,j,k) = (cmu*rhop(i,j,k)*tken(i,j,k)**2)/eps(i,j,k)
         viscp(i,j,k) = vmulam + vmuturb(i,j,k)
        endif
        viscb(i,j,k) = viscp(i,j,k)
        if(jbound(i,j,k).eq.1)then
         jb = jbound(i,j,k)
         vx(i,j,k) = cu(jb)
         vy(i,j,k) = cv(jb)
c         vz(i,j,k) = cw(jb)
         vz(i,j,k) = vzz
         ub(i,j,k) = vx(i,j,k)
         vb(i,j,k) = vy(i,j,k)
         wb(i,j,k) = vz(i,j,k)
         delzetb(i,j,k) = 0.5
        endif
   50 continue

***   Specify initial velocity and turbulence distribution

      do 60 k = 1,kn
       do 60 j = 1,jn
        do 60 i = 1,in
c         if(jbound(i,j,k).eq.1)goto 60
```

```
           if(k.eq.1)goto 60
c            eps(i,j,k) = epsinit
c            tken(i,j,k) = tkeninit
           eps(i,j,k) = eps(i,j,1)
           tken(i,j,k) = tken(i,j,1)
           if(jbound(i,j,k).ne.0)goto 60
c            vx(i,j,k) = vxinit
c            vy(i,j,k) = vyinit
c            vz(i,j,k) = vzinit
           vx(i,j,k) = vx(i,j,1)
           vy(i,j,k) = vy(i,j,1)
           vz(i,j,k) = vz(i,j,1)
   60 continue

***   OUTFLOW BOUNDARY (2)

      do 70 k = 1,kn
       do 70 j = 1,jn
        do 70 i = 1,in
         if(jbound(i,j,k).eq.2)then
          jb = jbound(i,j,k)
          vx(i,j,k) = bu(jb)*vx(i,j,k-1)
          vy(i,j,k) = bv(jb)*vy(i,j,k-1)
          vz(i,j,k) = bw(jb)*vz(i,j,k-1)
          ub(i,j,k-1) = vx(i,j,k)
          vb(i,j,k-1) = vy(i,j,k)
          wb(i,j,k-1) = vz(i,j,k)
          delzetb(i,j,k-1) = 0.5
         endif
   70 continue

***   EASTERN (3) AND WESTERN (4) BOUNDARIES

      do 80 k = 1,kn
       do 80 j = 1,jn
        do 80 i = 1,in
         if(jbound(i,j,k).eq.0)goto 80
         jb = jbound(i,j,k)
         goto(80,80,3,4,80,80),jb

    3    vx(i,j,k) = bu(jb)*vx(i-1,j,k) + cu(jb)
         vy(i,j,k) = bv(jb)*vy(i-1,j,k) + cv(jb)
         vz(i,j,k) = bw(jb)*vz(i-1,j,k) + cw(jb)
         ue(i-1,j,k) = vx(i,j,k)
         ve(i-1,j,k) = vy(i,j,k)
         we(i-1,j,k) = vz(i,j,k)
         delexie(i-1,j,k) = 0.5
         goto 80

    4    vx(i,j,k) = bu(jb)*vx(i+1,j,k) + cu(jb)
         vy(i,j,k) = bv(jb)*vy(i+1,j,k) + cv(jb)
         vz(i,j,k) = bw(jb)*vz(i+1,j,k) + cw(jb)
         ue(i,j,k) = vx(i,j,k)
         ve(i,j,k) = vy(i,j,k)
         we(i,j,k) = vz(i,j,k)
        delexie(i,j,k) = 0.5

   80 continue

***   NORTHERN (5) AND SOUTHERN (6) BOUNDARIES

      do 90 k = 1,kn
       do 90 j = 1,jn
        do 90 i = 1,in
         if(jbound(i,j,k).eq.0)goto 90
         jb = jbound(i,j,k)
         goto (90,90,90,90,5,6),jb

    5    vx(i,j,k) = bu(jb)*vx(i,j-1,k) + cu(jb)
         vy(i,j,k) = bv(jb)*vy(i,j-1,k) + cv(jb)
         vz(i,j,k) = bw(jb)*vz(i,j-1,k) + cw(jb)
         un(i,j-1,k) = vx(i,j,k)
         vn(i,j-1,k) = vy(i,j,k)
         wn(i,j-1,k) = vz(i,j,k)
         deletan(i,j,k) = 0.5
         goto 90

    6    vx(i,j,k) = bu(jb)*vx(i,j+1,k) + cu(jb)
         vy(i,j,k) = bv(jb)*vy(i,j+1,k) + cv(jb)
         vz(i,j,k) = bw(jb)*vz(i,j+1,k) + cw(jb)
         un(i,j,k) = vx(i,j,k)
         vn(i,j,k) = vy(i,j,k)
         wn(i,j,k) = vz(i,j,k)
         deletan(i,j,k) = 0.5

   90 continue

***   FIXED WALL BOUNDARIES (7)

      do 100 k = 1,kn
       do 100 j = 1,jn
        do 100 i = 1,in
         if(jbound(i,j,k).ne.nbnd)goto 100
         vx(i,j,k) = 0.0
         vy(i,j,k) = 0.0
         vz(i,j,k) = 0.0
         ue(i,j,k) = 0.0
         ve(i,j,k) = 0.0
         we(i,j,k) = 0.0
         un(i,j,k) = 0.0
         vn(i,j,k) = 0.0
         wn(i,j,k) = 0.0
         ub(i,j,k) = 0.0
         vb(i,j,k) = 0.0
         wb(i,j,k) = 0.0
         ue(i-1,j,k) = 0.0
         ve(i-1,j,k) = 0.0
         we(i-1,j,k) = 0.0
         un(i,j-1,k) = 0.0
         vn(i,j-1,k) = 0.0
```

```
          wn(i,j-1,k) = 0.0
          ub(i,j,k-1) = 0.0
          vb(i,j,k-1) = 0.0
          wb(i,j,k-1) = 0.0
          delexie(i,j,k) = 0.5
          delexie(i-1,j,k) = 0.5
          deletan(i,j,k) = 0.5
          deletan(i,j-1,k) = 0.5
          delzetb(i,j,k) = 0.5
          delzetb(i,j,k-1) = 0.5

  100 continue

***-----------------------------------------------------------------------
***   Produce an output of the boundary conditions
***-----------------------------------------------------------------------

      kx = 1
      jx = -1
      if(kn.gt.18) kx = 2
      if(kn.gt.36) kx = 3
      if(kn.gt.54) kx = 4
      if(kn.gt.72) kx = 5

      write(9,22)

      do 110 i = 1,in
       write(9,25)i
       do 110 kkk = 1,kx
        write(9,*)' '
        nkend = 18*kkk
        nkbeg = nkend-17
        if(nkend.gt.kn) nkend = kn
        write(9,24) (kk,kk = nkbeg,nkend)
        write(9,21)

        do 190 j = jn,1,jx
         write(9,23)j,(jbound(i,j,kk),kk = nkbeg,nkend)
  190   continue
  110 continue

   21 format(6x,73('-'))
   22 format(//1x,'BOUNDARY CONDITIONS (JBOUND)'/)
   23 format(1x,I2,'  |',23(1x,I3))
   24 format(1x,' J/K = ',23(I2,2x))
   25 format(1x,'I-SURFACE NO. ',I3)

      return
      end

*************************************************************************
      Subroutine STEP
*************************************************************************

***   This subroutine adjusts velocities before and behind steps
```

```
      INCLUDE 'COMM'

***------------------------------------------------------------------
***   Part 1 : Forward facing step
***------------------------------------------------------------------

      if(kfst.eq.777)goto 35

      kfst1 = kfst - 1
      jstep2 = jstep + 1
      kfst2 = kfst + 1
      gvoor = 0.0
      gna = 0.0

      do 10 i = 2,istep
       do 10 j = 2,jnm1
        gvoor = gvoor + vz(i,j,kfst1)*zetz(i,j,kfst1)/tjac(i,j,kfst1)
   10 continue

      do 11 i = 2,istep
       do 11 j = jstep2,jnm1
         gna = gna + vz(i,j,kfst)*zetz(i,j,kfst)/tjac(i,j,kfst)
   11 continue

      ffs = gvoor/gna

      do 20 i = 2,istep
       do 20 j = jstep2,jnm1
        vz(i,j,kfst) = vz(i,j,kfst)*ffs
   20 continue

      do 25 k = kfst2,kbst
       do 25 i = 2,istep
        do 25 j = jstep2,jnm1
         vz(i,j,k) = vz(i,j,k-1)
   25 continue

***   Adjust vy-velocities

      k = kfst1
      do 30 i = 2,istep
       do 30 j = 2,jnm1

        gout = dabs(zetzb(i,j,k)
     *        *(vz(i,j,k)+vz(i,j,k+1))/(2*tjacb(i,j,k)))
     *       + dabs(etayn(i,j,k)
     *         *vn(i,j,k)/tjacn(i,j,k))

        gin = dabs(zetzb(i,j,k-1)
     *        *wb(i,j,k-1)/tjacb(i,j,k-1))
     *      + dabs(etayn(i,j-1,k)
     *        *vn(i,j-1,k)/tjacn(i,j-1,k))

        ggain = gin - gout
        vn(i,j,k) = ggain*tjacn(i,j,k)/etayn(i,j,k)
        vy(i,j,k) = (vn(i,j-1,k)+vn(i,j,k))/2
   30 continue

***-----------------------------------------------------------------------
***   Part 2 : Backward facing step
***-----------------------------------------------------------------------

   35 if(kbst.eq.777)goto 65

       kbst1 = kbst + 1
       jstep2 = jstep + 1

***   Adjust vy-velocities

      k = kbst1
      do 60 i = 2,istep
       do 60 j = 2,jnm1

        gout = dabs(zetz(i,j,k)*wb(i,j,k)/tjac(i,j,k))
     *       + dabs(etay(i,j,k)*vn(i,j-1,k)/tjac(i,j,k))

        gin = dabs(zetz(i,j,k)*(vz(i,j,k-1)+vz(i,j,k))/(2*tjac(i,j,k)))
     *      + dabs(etay(i,j,k)*vn(i,j,k)/tjac(i,j,k))
        ggain = gin - gout
        vn(i,j,k) = ggain*tjac(i,j,k)/etay(i,j,k)
        vy(i,j,k) = (vn(i,j-1,k)+vn(i,j,k))/2
   60 continue

   65 return
      end

*************************************************************************
      Subroutine BOUND
*************************************************************************

***   This subroutine adjusts velocities at outflow boundary conditions

      INCLUDE 'COMM'

***-----------------------------------------------------------------------
***   Part 1 : Main velocities on dependant boundaries
***-----------------------------------------------------------------------

      do 10 k = 1,kn
       do 10 j = 1,jn
        do 10 i = 1,in
        jb = jbound(i,j,k)
        if(jb.eq.0)goto 10
        goto (10,2,3,4,5,6,10)jb

***   Outflow

    2   vx(i,j,k) = bu(jb)*vx(i,j,k-1)
        vy(i,j,k) = bv(jb)*vy(i,j,k-1)
      vz(i,j,k) = bw(jb)*(vz(i,j,k-2)+vz(i,j,k-1))/2
      ub(i,j,k-1) = vx(i,j,k)
      vb(i,j,k-1) = vy(i,j,k)
      wb(i,j,k-1) = vz(i,j,k)
      goto 10

***   East

    3 vx(i,j,k) = bu(jb)*vx(i-1,j,k) + cu(jb)
      vy(i,j,k) = bv(jb)*vy(i-1,j,k) + cv(jb)
      vz(i,j,k) = bw(jb)*vz(i-1,j,k) + cw(jb)
      ue(i-1,j,k) = vx(i,j,k)
      ve(i-1,j,k) = vy(i,j,k)
      we(i-1,j,k) = vz(i,j,k)
      goto 10

***   West

    4 vx(i,j,k) = bu(jb)*vx(i+1,j,k) + cu(jb)
      vy(i,j,k) = bv(jb)*vy(i+1,j,k) + cv(jb)
      vz(i,j,k) = bw(jb)*vz(i+1,j,k) + cw(jb)
      ue(i,j,k) = vx(i,j,k)
      ve(i,j,k) = vy(i,j,k)
      we(i,j,k) = vz(i,j,k)
      goto 10

    5 vx(i,j,k) = bu(jb)*vx(i,j-1,k) + cu(jb)
      vy(i,j,k) = bv(jb)*vy(i,j-1,k) + cv(jb)
      vz(i,j,k) = bw(jb)*vz(i,j-1,k) + cw(jb)
      un(i,j-1,k) = vx(i,j,k)
      vn(i,j-1,k) = vy(i,j,k)
      wn(i,j-1,k) = vz(i,j,k)
      goto 10

    6 vx(i,j,k) = bu(jb)*vx(i,j+1,k) + cu(jb)
      vy(i,j,k) = bv(jb)*vy(i,j+1,k) + cv(jb)
      vz(i,j,k) = bw(jb)*vz(i,j+1,k) + cw(jb)
      un(i,j,k) = vx(i,j,k)
      vn(i,j,k) = vy(i,j,k)
      wn(i,j,k) = vz(i,j,k)

   10 continue

***-------------------------------------------------------------------
***   Part 2 : Interpolated boundary velocities
***-------------------------------------------------------------------

      do 20 k = 1,kn
       do 20 j = 1,jn
        do 20 i = 1,in
         jb = jbound(i,j,k)
         if(jb.eq.0)goto 20
         goto (20,12,13,14,15,16,20)jb

   12    kk = k-1
```

```
      unb(i,j,kk)=(1-f2(i,j,kk))*vx(i,j,k)+f2(i,j,kk)*vx(i,j+1,k)
      vnb(i,j,kk)=(1-f2(i,j,kk))*vy(i,j,k)+f2(i,j,kk)*vy(i,j+1,k)
      wnb(i,j,kk)=(1-f2(i,j,kk))*vz(i,j,k)+f2(i,j,kk)*vz(i,j+1,k)
      ube(i,j,kk)=(1-f1(i,j,kk))*vx(i,j,k)+f1(i,j,kk)*vx(i+1,j,k)
      vbe(i,j,kk)=(1-f1(i,j,kk))*vy(i,j,k)+f1(i,j,kk)*vy(i+1,j,k)
      wbe(i,j,kk)=(1-f1(i,j,kk))*vz(i,j,k)+f1(i,j,kk)*vz(i+1,j,k)
      goto 20

13    ii = i-1
      une(ii,j,k)=(1-f2(ii,j,k))*vx(i,j,k)+f2(ii,j,k)*vx(i,j+1,k)
      vne(ii,j,k)=(1-f2(ii,j,k))*vy(i,j,k)+f2(ii,j,k)*vy(i,j+1,k)
      wne(ii,j,k)=(1-f2(ii,j,k))*vz(i,j,k)+f2(ii,j,k)*vz(i,j+1,k)
      ube(ii,j,k)=(1-f3(ii,j,k))*vx(i,j,k)+f3(ii,j,k)*vx(i,j,k+1)
      vbe(ii,j,k)=(1-f3(ii,j,k))*vy(i,j,k)+f3(ii,j,k)*vy(i,j,k+1)
      wbe(ii,j,k)=(1-f3(ii,j,k))*vz(i,j,k)+f3(ii,j,k)*vz(i,j,k+1)
      goto 20

14    ii = i+1
      ube(i,j,k)=(1-f3(ii,j,k))*vx(i,j,k)+f3(ii,j,k)*vx(i,j,k+1)
      vbe(i,j,k)=(1-f3(ii,j,k))*vy(i,j,k)+f3(ii,j,k)*vy(i,j,k+1)
      wbe(i,j,k)=(1-f3(ii,j,k))*vz(i,j,k)+f3(ii,j,k)*vz(i,j,k+1)
      une(i,j,k)=(1-f2(ii,j,k))*vx(i,j,k)+f2(ii,j,k)*vx(i,j+1,k)
      vne(i,j,k)=(1-f2(ii,j,k))*vy(i,j,k)+f2(ii,j,k)*vy(i,j+1,k)
      wne(i,j,k)=(1-f2(ii,j,k))*vz(i,j,k)+f2(ii,j,k)*vz(i,j+1,k)
      goto 20

15    jj = j-1
      unb(i,jj,k)=(1-f3(i,jj,k))*vx(i,j,k)+f3(i,jj,k)*vx(i,j,k+1)
      vnb(i,jj,k)=(1-f3(i,jj,k))*vy(i,j,k)+f3(i,jj,k)*vy(i,j,k+1)
      wnb(i,jj,k)=(1-f3(i,jj,k))*vz(i,j,k)+f3(i,jj,k)*vz(i,j,k+1)
      une(i,jj,k)=(1-f1(i,jj,k))*vx(i,j,k)+f1(i,jj,k)*vx(i+1,j,k)
      vne(i,jj,k)=(1-f1(i,jj,k))*vy(i,j,k)+f1(i,jj,k)*vy(i+1,j,k)
      wne(i,jj,k)=(1-f1(i,jj,k))*vz(i,j,k)+f1(i,jj,k)*vz(i+1,j,k)
      goto 20

16    jj = j+1
      unb(i,j,k)=(1-f3(i,jj,k))*vx(i,j,k)+f3(i,jj,k)*vx(i,j,k+1)
      vnb(i,j,k)=(1-f3(i,jj,k))*vy(i,j,k)+f3(i,jj,k)*vy(i,j,k+1)
      wnb(i,j,k)=(1-f3(i,jj,k))*vz(i,j,k)+f3(i,jj,k)*vz(i,j,k+1)
      une(i,j,k)=(1-f1(i,jj,k))*vx(i,j,k)+f1(i,jj,k)*vx(i+1,j,k)
      vne(i,j,k)=(1-f1(i,jj,k))*vy(i,j,k)+f1(i,jj,k)*vy(i+1,j,k)
      wne(i,j,k)=(1-f1(i,jj,k))*vz(i,j,k)+f1(i,jj,k)*vz(i+1,j,k)
      goto 20

20 continue

***-----------------------------------------------------------------
***   Part 3 : Fixed wall velocities
***-----------------------------------------------------------------

   do 30 k = 1,kn
    do 30 j = 1,jn
     do 30 i = 1,in
      jb = jbound(i,j,k)
      if(jb.ne.nbnd)goto 30
```

```
         goto 17

   17    une(i,j,k) = 0.0
         une(i-1,j,k) = 0.0
         une(i,j-1,k) = 0.0
         une(i-1,j-1,k) = 0.0
         vne(i,j,k) = 0.0
         vne(i-1,j,k) = 0.0
         vne(i,j-1,k) = 0.0
         vne(i-1,j-1,k) = 0.0
         wne(i,j,k) = 0.0
         wne(i-1,j,k) = 0.0
         wne(i,j-1,k) = 0.0
         wne(i-1,j-1,k) = 0.0
         ube(i,j,k) = 0.0
         ube(i-1,j,k) = 0.0
         ube(i,j,k-1) = 0.0
         ube(i-1,j,k-1) = 0.0
         vbe(i,j,k) = 0.0
         vbe(i-1,j,k) = 0.0
         vbe(i,j,k-1) = 0.0
         vbe(i-1,j,k-1) = 0.0
         wbe(i,j,k) = 0.0
         wbe(i-1,j,k) = 0.0
         wbe(i,j,k-1) = 0.0
         wbe(i-1,j,k-1) = 0.0
         unb(i,j,k) = 0.0
         unb(i,j,k-1) = 0.0
         unb(i,j-1,k) = 0.0
         unb(i,j-1,k-1) = 0.0
         vnb(i,j,k) = 0.0
         vnb(i,j,k-1) = 0.0
         vnb(i,j-1,k) = 0.0
         vnb(i,j-1,k-1) = 0.0
         wnb(i,j,k) = 0.0
         wnb(i,j,k-1) = 0.0
         wnb(i,j-1,k) = 0.0
         wnb(i,j-1,k-1) = 0.0

   30 continue

***   Calculate the massflow at the outflow boundary

      fmout = 0.0

      do 40 j = 2,jnml
       do 40 i = 2,inml
        fmo(i,j) = rhob(i,j,knml)*(zetxb(i,j,knml)*ub(i,j,knml)
     *                         + zetyb(i,j,knml)*vb(i,j,knml)
     *                    + zetzb(i,j,knml)*wb(i,j,knml))/tjacb(i,j,knml)
        fmout = fmout + fmo(i,j)
   40 continue

      fcont = fmin/fmout
      do 50 j = 2,jnml
       do 50 i = 2,inml
        vz(i,j,kn) = vz(i,j,kn)*fcont
        wb(i,j,knml) = vz(i,j,kn)
   50 continue

      return

c----------------------------------------------------------------------
      Entry MASSIN
c----------------------------------------------------------------------

***   Calculate the massflow at the inflow boudary

      fmin = 0.0

      do 60 j = 2,jnml
       do 60 i = 2,inml
        if(jbound(i,j,1).ne.1)goto 60
        fmi(i,j) = rhob(i,j,1)*(zetxb(i,j,1)*ub(i,j,1)
     *                        + zetyb(i,j,1)*vb(i,j,1)
     *                        + zetzb(i,j,1)*wb(i,j,1))/tjacb(i,j,1)
        fmin = fmin + fmi(i,j)
   60 continue

      return
      end

***********************************************************************
      Subroutine LOGO
***********************************************************************

***   This subroutine creates the solving logo

      write(*,*)
      write(*,*)'    **********************************************'
      write(*,*)'    *                                            *'
      write(*,*)'    *       3333  DDDD   FFFFF L       000       *'
      write(*,*)'    *      3    3 D   D  F     L      0   0      *'
      write(*,*)'    *         33  D    D FFF   L     0     0     *'
      write(*,*)'    *      3    3 D   D  F     L      0   0      *'
      write(*,*)'    *       3333  DDDD   F     LLLLL   000       *'
      write(*,*)'    *                                            *'
      write(*,*)'    *     THIS PROGRAM SOLVES 3D COMPRESSIBLE    *'
      write(*,*)'    *       LAMINAR OR TURBULENT FLOWS IN        *'
      write(*,*)'    *         CURVILINEAR CO-ORDINATES           *'
      write(*,*)'    *                                            *'
      WRITE(*,*)'    *               DEVELOPED BY                 *'
      write(*,*)'    *                                            *'
      WRITE(*,*)'    *              HERMANN ROLFES                *'
      write(*,*)'    *                                            *'
      write(*,*)'    **********************************************'
      write(*,*)
      write(*,*)
      return
      end

***********************************************************************
      Subroutine OUTPUT
***********************************************************************

***   This subroutine generates the velocity and pressure output data

      INCLUDE 'COMM'

      write(*,*)'**********  WRITING OUTPUT  **********'

      kx = 1
      jx = -1
      if(kn.gt.7) kx = 2
      if(kn.gt.14) kx = 3
      if(kn.gt.21) kx = 4
      if(kn.gt.28) kx = 5
      if(kn.gt.35) kx = 6
      if(kn.gt.42) kx = 7
      if(kn.gt.49) kx = 8
      if(kn.gt.56) kx = 9
      if(kn.gt.63) kx = 10
      if(kn.gt.72) kx = 11

      do 100 i = 1,in
      write(9,2)i
    2 format(//1x,'SURFACE NO. ',I3,3X,
     *'Z - DIRECTION VELOCITY DISTRIBUTION (VZ)'/)

      do 100 kkk = 1,kx

       write(9,*)' '
       kend = 7*kkk
       kbeg = kend-6
       if(kend.gt.kn) kend = kn
       write(9,20) (kk,kk = kbeg,kend)

       do 3 j = jn,1,jx
        write(9,23)j,(vz(i,j,kk),kk = kbeg,kend)
    3  continue

  100 continue

      do 200 i = 1,in
      write(9,4)i
    4 format(//1x,'SURFACE NO. ',I3,3X
     *,'Y - DIRECTION VELOCITY DISTRIBUTION (VY)'/)

      do 200 kkk = 1,kx

       write(9,*)' '
       kend = 7*kkk
       kbeg = kend-6
       if(kend.gt.kn) kend = kn
       write(9,20) (kk,kk = kbeg,kend)

       do 5 j = jn,1,jx
        write(9,23)j,(vy(i,j,kk),kk = kbeg,kend)
    5  continue

  200 continue

      do 300 i = 1,in
      write(9,6)i
    6 format(//1x,'SURFACE NO. ',I4,3X
     *,'X - DIRECTION VELOCITY DISTRIBUTION (VX)'/)

      do 300 kkk = 1,kx

       write(9,*)' '
       kend = 7*kkk
       kbeg = kend-6
       if(kend.gt.kn) kend = kn
       write(9,20) (kk,kk = kbeg,kend)

       do 7 j = jn,1,jx
        write(9,23)j,(vx(i,j,kk),kk = kbeg,kend)
    7  continue

  300 continue

      do 400 i = 1,in
       write(9,8)i
    8 format(//1x,'SURFACE NO. ',I4,3X
     *,'PRESSURE DISTRIBUTION (P)'/)

      do 400 kkk = 1,kx

       write(9,*)' '
       kend = 7*kkk
       kbeg = kend-6
       if(kend.gt.kn) kend = kn
       write(9,20) (kk,kk = kbeg,kend)

       do 9 j = jn,1,jx
        write(9,23)j,(pp(i,j,kk),kk = kbeg,kend)
    9  continue

  400 continue

      do 500 i = 1,in
       write(9,10)i
   10 format(//1x,'SURFACE NO. ',I4,3X
     *,'PRESSURE CORRECTIONS (PCOR)'/)

      do 500 kkk = 1,kx

       write(9,*)' '
```

```
        kend = 7*kkk
        kbeg = kend-6
        if(kend.gt.kn) kend = kn
        write(9,20) (kk,kk = kbeg,kend)

        do 11 j = jn,1,jx
         write(9,23)j,(pcor(i,j,kk),kk = kbeg,kend)
   11   continue

  500 continue

      do 600 i = 1,in
       write(9,12)i
   12 format(//1x,'SURFACE NO. ',I4,3X
     *,'DISSIPATION RATE - EPSILON'/)

      do 600 kkk = 1,kx

        write(9,*)' '
        kend = 7*kkk
        kbeg = kend-6
        if(kend.gt.kn) kend = kn
        write(9,20) (kk,kk = kbeg,kend)

        do 13 j = jn,1,jx
         write(9,23)j,(eps(i,j,kk),kk = kbeg,kend)
   13   continue

  600 continue

      do 700 i = 1,in
       write(9,14)i
   14 format(//1x,'SURFACE NO. ',I4,3X
     *,'TURBULENT KINETIC ENERGY'/)

      do 700 kkk = 1,kx

        write(9,*)' '
        kend = 7*kkk
        kbeg = kend-6
        if(kend.gt.kn) kend = kn
        write(9,20) (kk,kk = kbeg,kend)

        do 15 j = jn,1,jx
         write(9,23)j,(tken(i,j,kk),kk = kbeg,kend)
   15   continue

  700 continue

      do 800 i = 1,in
       write(9,16)i
   16 format(//1x,'SURFACE NO. ',I4,3X
     *,'EFFECTIVE VISCOCITY'/)

      do 800 kkk = 1,kx
```

```
        write(9,*)' '
        kend = 7*kkk
        kbeg = kend-6
        if(kend.gt.kn) kend = kn
        write(9,20) (kk,kk = kbeg,kend)

        do 17 j = jn,1,jx
         write(9,23)j,(viscp(i,j,kk),kk = kbeg,kend)
   17   continue

  800 continue

***   Define the format expressions

   20 format(1x,' J/K =',9(I2,9x))
   23 format(1x,I2,9(1x,F10.5))

      return
      end

**************************************************************************
      Subroutine PLOTOUT
**************************************************************************

***   This subroutine generates the velocity and pressure plot data

      INCLUDE 'COMM'

      write(13,2)inml,jnml,knml,isweep

      if(iturb.eq.1)then
      do 20 k = 2,knml
       do 20 j = 2,jnml
        do 20 i = 2,inml
        write(13,1)x(i,j,k),y(i,j,k),z(i,j,k),tken(i,j,k),eps(i,j,k)
        write(13,1)vx(i,j,k),vy(i,j,k),vz(i,j,k),pp(i,j,k),viscp(i,j,k)
   20 continue
      endif          .

      if(iturb.eq.0)then
      do 30 k = 2,knml
       do 30 j = 2,jnml
        do 30 i = 2,inml
        write(13,3)x(i,j,k),y(i,j,k),z(i,j,k)
        write(13,4)vx(i,j,k),vy(i,j,k),vz(i,j,k),pp(i,j,k)
   30 continue
      endif

    1 format(1x,5f12.5)
    2 format(1x,4i5)
    3 format(1x,3f12.5)
    4 format(1x,4f12.5)

      goto 10000
```

```
 9000 write(*,*)'!!! UNABLE TO OPEN PLOT FILE !!!'
      stop

10000 return
      end

*******************************************************************************
      Subroutine LINEPARAM
*******************************************************************************

***   This subroutine writes lineparameters

      INCLUDE 'COMM'

      write(9,*)' '
      write(9,*)'Residuals of u,v,w on k - planes'
      do k = 1,kn
       write(9,1)k,resuz(k),resvz(k),reswz(k)
      enddo

      write(9,*)' '
      write(9,*)'Residuals of u,v,w on j - planes'
      do j = 1,jn
       write(9,1)j,resuy(j),resvy(j),reswy(j)
      enddo

      write(9,*)' '
      write(9,*)'Residuals of u,v,w on i - planes'
      do i = 1,in
       write(9,1)i,resux(i),resvx(i),reswx(i)
      enddo

    1 format(2x,i4,4x,e11.5,4x,e11.5,4x,e11.5)

      return
      end

*******************************************************************************
      Subroutine GEOM
*******************************************************************************

***   This subroutine reads the output file from the grid generation
***   package and controls and processes the grid information

      INCLUDE 'COMM'

      dimension xx(160000),yy(160000),zz(160000)

***   Read input grid data

      write(*,*)'**********  READING GRID DATA  **********'
      read(2,*)ing,jng,kng

      do 10 i = 1,ing
       do 10 j = 1,jng
```

```
          do 10 k = 1,kng
           ng = (i-1)*jng*kng + (j-1)*kng + k
           read(2,*)nng,xx(ng),yy(ng),zz(ng)
   10 continue

      ingm1 = ing - 1
      jngm1 = jng - 1
      kngm1 = kng - 1
      in = 2 + ingm1/2
      jn = 2 + jngm1/2
      kn = 2 + kngm1/2
      inm1 = in - 1
      jnm1 = jn - 1
      knm1 = kn - 1
      inm2 = in - 2
      jnm2 = jn - 2
      knm2 = kn - 2

      write(*,*)in,jn,kn

      do 15 i = 1,in
       do 15 j = 1,jn
        do 15 k = 1,kn
         x(i,j,k) = 777.77777
         y(i,j,k) = 777.77777
         z(i,j,k) = 777.77777
   15 continue

***   Calculate transformation values

      do 20 i = 2,ingm1,2
       do 20 j = 2,jngm1,2
        do 20 k = 2,kngm1,2

        ii = i/2 + 1
        jj = j/2 + 1
        kk = k/2 + 1
        istot = jng*kng
        ng = (i-1)*istot + (j-1)*kng + k

***   Award x,y,z co-ordinates on boundaries

       if(i.eq.2)then
        x(ii-1,jj,kk) = xx(ng-(jng*kng))
        y(ii-1,jj,kk) = yy(ng-(jng*kng))
        z(ii-1,jj,kk) = zz(ng-(jng*kng))
       endif
       if(i.eq.ingm1)then
        x(ii+1,jj,kk) = xx(ng+(jng*kng))
        y(ii+1,jj,kk) = yy(ng+(jng*kng))
        z(ii+1,jj,kk) = zz(ng+(jng*kng))
       endif
       if(j.eq.2)then
        x(ii,jj-1,kk) = xx(ng-kng)
        y(ii,jj-1,kk) = yy(ng-kng)
```

```
         z(ii,jj-1,kk) = zz(ng-kng)
        endif
        if(j.eq.jngm1)then
         x(ii,jj+1,kk) = xx(ng+kng)
         y(ii,jj+1,kk) = yy(ng+kng)
         z(ii,jj+1,kk) = zz(ng+kng)
        endif
        if(k.eq.2)then
         x(ii,jj,kk-1) = xx(ng-1)
         y(ii,jj,kk-1) = yy(ng-1)
         z(ii,jj,kk-1) = zz(ng-1)
        endif
        if(k.eq.kngm1)then
         x(ii,jj,kk+1) = xx(ng+1)
         y(ii,jj,kk+1) = yy(ng+1)
         z(ii,jj,kk+1) = zz(ng+1)
        endif

***    Set the co-ordinates at each major node

        x(ii,jj,kk) = xx(ng)
        y(ii,jj,kk) = yy(ng)
        z(ii,jj,kk) = zz(ng)

***    The numerical values of the first order derivations of the
***    specific transformation can be calculated from the
***    carthesian co-ordinates of each point; de1,de2,de3 = 1

         dxdex = xx(ng+istot)-xx(ng-istot)
         dydex = yy(ng+istot)-yy(ng-istot)
         dzdex = zz(ng+istot)-zz(ng-istot)
         dxdet = xx(ng+kng)-xx(ng-kng)
         dydet = yy(ng+kng)-yy(ng-kng)
         dzdet = zz(ng+kng)-zz(ng-kng)
         dxdzt = xx(ng+1)-xx(ng-1)
         dydzt = yy(ng+1)-yy(ng-1)
         dzdzt = zz(ng+1)-zz(ng-1)

***    Eastern cell walls

         dxdete = xx(ng+istot+kng)-xx(ng+istot-kng)
         dydete = yy(ng+istot+kng)-yy(ng+istot-kng)
         dzdete = zz(ng+istot+kng)-zz(ng+istot-kng)
         dxdzte = xx(ng+istot+1)-xx(ng+istot-1)
         dydzte = yy(ng+istot+1)-yy(ng+istot-1)
         dzdzte = zz(ng+istot+1)-zz(ng+istot-1)
         if(ii.eq.inm1)then
          dxdexe = 2*(xx(ng+istot)-xx(ng))
          dydexe = 2*(yy(ng+istot)-yy(ng))
          dzdexe = 2*(zz(ng+istot)-zz(ng))
         else
          dxdexe = xx(ng+2*istot)-xx(ng)
          dydexe = yy(ng+2*istot)-yy(ng)
          dzdexe = zz(ng+2*istot)-zz(ng)
         endif
```

```
***   Western cell walls

        dxdetw = xx(ng-istot+kng)-xx(ng-istot-kng)
        dydetw = yy(ng-istot+kng)-yy(ng-istot-kng)
        dzdetw = zz(ng-istot+kng)-zz(ng-istot-kng)
        dxdztw = xx(ng-istot+1)-xx(ng-istot-1)
        dydztw = yy(ng-istot+1)-yy(ng-istot-1)
        dzdztw = zz(ng-istot+1)-zz(ng-istot-1)
        if(ii.eq.2)then
         dxdexw = 2*(xx(ng)-xx(ng-istot))
         dydexw = 2*(yy(ng)-yy(ng-istot))
         dzdexw = 2*(zz(ng)-zz(ng-istot))
        else
         dxdexw = xx(ng)-xx(ng-2*istot)
         dydexw = yy(ng)-yy(ng-2*istot)
         dzdexw = zz(ng)-zz(ng-2*istot)
        endif

***   Northern cell walls

        dxdexn = xx(ng+kng+istot)-xx(ng+kng-istot)
        dydexn = yy(ng+kng+istot)-yy(ng+kng-istot)
        dzdexn = zz(ng+kng+istot)-zz(ng+kng-istot)
        dxdztn = xx(ng+kng+1)-xx(ng+kng-1)
        dydztn = yy(ng+kng+1)-yy(ng+kng-1)
        dzdztn = zz(ng+kng+1)-zz(ng+kng-1)
        if(jj.eq.jnml)then
         dxdetn = 2*(xx(ng+kng)-xx(ng))
         dydetn = 2*(yy(ng+kng)-yy(ng))
         dzdetn = 2*(zz(ng+kng)-zz(ng))
        else
         dxdetn = xx(ng+2*kng)-xx(ng)
         dydetn = yy(ng+2*kng)-yy(ng)
         dzdetn = zz(ng+2*kng)-zz(ng)
        endif

***   Southern cell walls

        dxdexs = xx(ng-kng+istot)-xx(ng-kng-istot)
        dydexs = yy(ng-kng+istot)-yy(ng-kng-istot)
        dzdexs = zz(ng-kng+istot)-zz(ng-kng-istot)
        dxdzts = xx(ng-kng+1)-xx(ng-kng-1)
        dydzts = yy(ng-kng+1)-yy(ng-kng-1)
        dzdzts = zz(ng-kng+1)-zz(ng-kng-1)
        if(jj.eq.2)then
         dxdets = 2*(xx(ng)-xx(ng-kng))
         dydets = 2*(yy(ng)-yy(ng-kng))
         dzdets = 2*(zz(ng)-zz(ng-kng))
        else
         dxdets = xx(ng)-xx(ng-2*kng)
         dydets = yy(ng)-yy(ng-2*kng)
         dzdets = zz(ng)-zz(ng-2*kng)
        endif
```

```
***   Backward facing cell walls

        dxdexb = xx(ng+1+istot)-xx(ng+1-istot)
        dydexb = yy(ng+1+istot)-yy(ng+1-istot)
        dzdexb = zz(ng+1+istot)-zz(ng+1-istot)
        dxdetb = xx(ng+1+kng)-xx(ng+1-kng)
        dydetb = yy(ng+1+kng)-yy(ng+1-kng)
        dzdetb = zz(ng+1+kng)-zz(ng+1-kng)
        if(kk.eq.knm1)then
         dxdztb = 2*(xx(ng+1)-xx(ng))
         dydztb = 2*(yy(ng+1)-yy(ng))
         dzdztb = 2*(zz(ng+1)-zz(ng))
        else
         dxdztb = xx(ng+2)-xx(ng)
         dydztb = yy(ng+2)-yy(ng)
         dzdztb = zz(ng+2)-zz(ng)
        endif

***   Forward facing cell walls

        dxdexf = xx(ng-1+istot)-xx(ng-1-istot)
        dydexf = yy(ng-1+istot)-yy(ng-1-istot)
        dzdexf = zz(ng-1+istot)-zz(ng-1-istot)
        dxdetf = xx(ng-1+kng)-xx(ng-1-kng)
        dydetf = yy(ng-1+kng)-yy(ng-1-kng)
        dzdetf = zz(ng-1+kng)-zz(ng-1-kng)
        if(kk.eq.2)then
         dxdztf = 2*(xx(ng)-xx(ng-1))
         dydztf = 2*(yy(ng)-yy(ng-1))
         dzdztf = 2*(zz(ng)-zz(ng-1))
        else
         dxdztf = xx(ng)-xx(ng-2)
         dydztf = yy(ng)-yy(ng-2)
         dzdztf = zz(ng)-zz(ng-2)
        endif

***   Calculate the jacobian of the transformations

        tjac(ii,jj,kk) = 1/
     *  ( dxdex*( dydet*dzdzt
     *           -dydzt*dzdet )
     *   -dxdet*( dydex*dzdzt
     *           -dydzt*dzdex )
     *   +dxdzt*( dydex*dzdet
     *           -dydet*dzdex ) )
        tjace(ii,jj,kk) = 1/
     *  ( dxdexe*( dydete*dzdzte
     *            -dydzte*dzdete )
     *   -dxdete*( dydexe*dzdzte
     *            -dydzte*dzdexe )
     *   +dxdzte*( dydexe*dzdete
     *            -dydete*dzdexe ) )
        tw = 1/
     *  ( dxdexw*( dydetw*dzdztw
     *            -dydztw*dzdetw )
     *   -dxdetw*( dydexw*dzdztw
     *            -dydztw*dzdexw )
     *   +dxdztw*( dydexw*dzdetw
     *            -dydetw*dzdexw ) )
        tjacn(ii,jj,kk) = 1/
     *  ( dxdexn*( dydetn*dzdztn
     *            -dydztn*dzdetn )
     *   -dxdetn*( dydexn*dzdztn
     *            -dydztn*dzdexn )
     *   +dxdztn*( dydexn*dzdetn
     *            -dydetn*dzdexn ) )
        ts = 1/
     *  ( dxdexs*( dydets*dzdzts
     *            -dydzts*dzdets )
     *   -dxdets*( dydexs*dzdzts
     *            -dydzts*dzdexs )
     *   +dxdzts*( dydexs*dzdets
     *            -dydets*dzdexs ) )
        tjacb(ii,jj,kk) = 1/
     *  ( dxdexb*( dydetb*dzdztb
     *            -dydztb*dzdetb )
     *   -dxdetb*( dydexb*dzdztb
     *            -dydztb*dzdexb )
     *   +dxdztb*( dydexb*dzdetb
     *            -dydetb*dzdexb ) )
        tf = 1/
     *  ( dxdexf*( dydetf*dzdztf
     *            -dydztf*dzdetf )
     *   -dxdetf*( dydexf*dzdztf
     *            -dydztf*dzdexf )
     *   +dxdztf*( dydexf*dzdetf
     *            -dydetf*dzdexf ) )

***   Generate the transformation expressions

        exix(ii,jj,kk) = tjac(ii,jj,kk)*
     *                           ( dydet*dzdzt
     *                           - dydzt*dzdet )
        exiy(ii,jj,kk) = -tjac(ii,jj,kk)*
     *                           ( dxdet*dzdzt
     *                           - dxdzt*dzdet )
        exiz(ii,jj,kk) = tjac(ii,jj,kk)*
     *                           ( dxdet*dydzt
     *                           - dxdzt*dydet )
        etax(ii,jj,kk) = -tjac(ii,jj,kk)*
     *                           ( dydex*dzdzt
     *                           - dydzt*dzdex )
        etay(ii,jj,kk) = tjac(ii,jj,kk)*
     *                           ( dxdex*dzdzt
     *                           - dxdzt*dzdex )
        etaz(ii,jj,kk) = -tjac(ii,jj,kk)*
     *                           ( dxdex*dydzt
     *                           - dxdzt*dydex )
        zetx(ii,jj,kk) = tjac(ii,jj,kk)*
     *                           ( dydex*dzdet
     *                          - dydet*dzdex )
        zety(ii,jj,kk) = -tjac(ii,jj,kk)*
     *                          ( dxdex*dzdet
     *                          - dxdet*dzdex )
        zetz(ii,jj,kk) = tjac(ii,jj,kk)*
     *                          ( dxdex*dydet
     *                          - dxdet*dydex )

***   Eastern cell walls

        exixe(ii,jj,kk) = tjace(ii,jj,kk)*
     *                          ( dydete*dzdzte
     *                          - dydzte*dzdete )
        exiye(ii,jj,kk) = -tjace(ii,jj,kk)*
     *                          ( dxdete*dzdzte
     *                          - dxdzte*dzdete )
        exize(ii,jj,kk) = tjace(ii,jj,kk)*
     *                          ( dxdete*dydzte
     *                          - dxdzte*dydete )
        etaxe(ii,jj,kk) = -tjace(ii,jj,kk)*
     *                          ( dydexe*dzdzte
     *                          - dydzte*dzdexe )
        etaye(ii,jj,kk) = tjace(ii,jj,kk)*
     *                          ( dxdexe*dzdzte
     *                          - dxdzte*dzdexe )
        etaze(ii,jj,kk) = -tjace(ii,jj,kk)*
     *                          ( dxdexe*dydzte
     *                          - dxdzte*dydexe )
        zetxe(ii,jj,kk) = tjace(ii,jj,kk)*
     *                          ( dydexe*dzdete
     *                          - dydete*dzdexe )
        zetye(ii,jj,kk) = -tjace(ii,jj,kk)*
     *                          ( dxdexe*dzdete
     *                          - dxdete*dzdexe )
        zetze(ii,jj,kk) = tjace(ii,jj,kk)*
     *                          ( dxdexe*dydete
     *                          - dxdete*dydexe )

***   Western cell walls

      if(ii.eq.2)then
        tjac(ii-1,jj,kk) = tjac(ii,jj,kk)
        tjace(ii-1,jj,kk) = tw
        exixe(ii-1,jj,kk) = tw*
     *                          ( dydetw*dzdztw
     *                          - dydztw*dzdetw )
        exiye(ii-1,jj,kk) = -tw*
     *                          ( dxdetw*dzdztw
     *                          - dxdztw*dzdetw )
        exize(ii-1,jj,kk) = tw*
     *                          ( dxdetw*dydztw
     *                          - dxdztw*dydetw )
        etaxe(ii-1,jj,kk) = -tw*
     *                          ( dydexw*dzdztw
     *                          - dydztw*dzdexw )
```

```
            etaye(ii-1,jj,kk) = tw*
     *                          ( dxdexw*dzdztw
     *                          - dxdztw*dzdexw )
            etaze(ii-1,jj,kk) = -tw*
     *                          ( dxdexw*dydztw
     *                          - dxdztw*dydexw )
            zetxe(ii-1,jj,kk) = tw*
     *                          ( dydexw*dzdetw
     *                          - dydetw*dzdexw )
            zetye(ii-1,jj,kk) = -tw*
     *                          ( dxdexw*dzdetw
     *                          - dxdetw*dzdexw )
            zetze(ii-1,jj,kk) = tw*
     *                          ( dxdexw*dydetw
     *                          - dxdetw*dydexw )
          endif

***     Northern cell walls

            exixn(ii,jj,kk) = tjacn(ii,jj,kk)*
     *                          ( dydetn*dzdztn
     *                          - dydztn*dzdetn )
            exiyn(ii,jj,kk) = -tjacn(ii,jj,kk)*
     *                          ( dxdetn*dzdztn
     *                          - dxdztn*dzdetn )
            exizn(ii,jj,kk) = tjacn(ii,jj,kk)*
     *                          ( dxdetn*dydztn
     *                          - dxdztn*dydetn )
            etaxn(ii,jj,kk) = -tjacn(ii,jj,kk)*
     *                          ( dydexn*dzdztn
     *                          - dydztn*dzdexn )
            etayn(ii,jj,kk) = tjacn(ii,jj,kk)*
     *                          ( dxdexn*dzdztn
     *                          - dxdztn*dzdexn )
            etazn(ii,jj,kk) = -tjacn(ii,jj,kk)*
     *                          ( dxdexn*dydztn
     *                          - dxdztn*dydexn )
            zetxn(ii,jj,kk) = tjacn(ii,jj,kk)*
     *                          ( dydexn*dzdetn
     *                          - dydetn*dzdexn )
            zetyn(ii,jj,kk) = -tjacn(ii,jj,kk)*
     *                          ( dxdexn*dzdetn
     *                          - dxdetn*dzdexn )
            zetzn(ii,jj,kk) = tjacn(ii,jj,kk)*
     *                          ( dxdexn*dydetn
     *                          - dxdetn*dydexn )

***     Southern cell walls

          if(jj.eq.2)then
            tjac(ii,jj-1,kk) = tjac(ii,jj,kk)
            tjacn(ii,jj-1,kk) = ts
            exixn(ii,jj-1,kk) = ts*
     *                          ( dydets*dzdzts
     *                          - dydzts*dzdets )
```

```
         exiyn(ii,jj-1,kk) = -ts*
     *                        ( dxdets*dzdzts
     *                        - dxdzts*dzdets )
         exizn(ii,jj-1,kk) = ts*
     *                        ( dxdets*dydzts
     *                        - dxdzts*dydets )
         etaxn(ii,jj-1,kk) = -ts*
     *                        ( dydexs*dzdzts
     *                        - dydzts*dzdexs )
         etayn(ii,jj-1,kk) = ts*
     *                        ( dxdexs*dzdzts
     *                        - dxdzts*dzdexs )
         etazn(ii,jj-1,kk) = -ts*
     *                        ( dxdexs*dydzts
     *                        - dxdzts*dydexs )
         zetxn(ii,jj-1,kk) = ts*
     *                        ( dydexs*dzdets
     *                        - dydets*dzdexs )
         zetyn(ii,jj-1,kk) = -ts*
     *                        ( dxdexs*dzdets
     *                        - dxdets*dzdexs )
         zetzn(ii,jj-1,kk) = ts*
     *                        ( dxdexs*dydets
     *                        - dxdets*dydexs )
      endif

***   Backward facing cell walls

         exixb(ii,jj,kk) = tjacb(ii,jj,kk)*
     *                        ( dydetb*dzdztb
     *                        - dydztb*dzdetb )
         exiyb(ii,jj,kk) = -tjacb(ii,jj,kk)*
     *                        ( dxdetb*dzdztb
     *                        - dxdztb*dzdetb )
         exizb(ii,jj,kk) = tjacb(ii,jj,kk)*
     *                        ( dxdetb*dydztb
     *                        - dxdztb*dydetb )
         etaxb(ii,jj,kk) = -tjacb(ii,jj,kk)*
     *                        ( dydexb*dzdztb
     *                        - dydztb*dzdexb )
         etayb(ii,jj,kk) = tjacb(ii,jj,kk)*
     *                        ( dxdexb*dzdztb
     *                        - dxdztb*dzdexb )
         etazb(ii,jj,kk) = -tjacb(ii,jj,kk)*
     *                        ( dxdexb*dydztb
     *                        - dxdztb*dydexb )
         zetxb(ii,jj,kk) = tjacb(ii,jj,kk)*
     *                        ( dydexb*dzdetb
     *                        - dydetb*dzdexb )
         zetyb(ii,jj,kk) = -tjacb(ii,jj,kk)*
     *                        ( dxdexb*dzdetb
     *                        - dxdetb*dzdexb )
         zetzb(ii,jj,kk) = tjacb(ii,jj,kk)*
     *                        ( dxdexb*dydetb
     *                        - dxdetb*dydexb )
```

```
***   Forward facing cell walls

      if(kk.eq.2)then
        tjac(ii,jj,kk-1) = tjac(ii,jj,kk)
        tjacb(ii,jj,kk-1) = tf
        exixb(ii,jj,kk-1) = tf*
     *                      ( dydetf*dzdztf
     *                      - dydztf*dzdetf )
        exiyb(ii,jj,kk-1) = -tf*
     *                      ( dxdetf*dzdztf
     *                      - dxdztf*dzdetf )
        exizb(ii,jj,kk-1) = tf*
     *                      ( dxdetf*dydztf
     *                      - dxdztf*dydetf )
        etaxb(ii,jj,kk-1) = -tf*
     *                      ( dydexf*dzdztf
     *                      - dydztf*dzdexf )
        etayb(ii,jj,kk-1) = tf*
     *                      ( dxdexf*dzdztf
     *                      - dxdztf*dzdexf )
        etazb(ii,jj,kk-1) = -tf*
     *                      ( dxdexf*dydztf
     *                      - dxdztf*dydexf )
        zetxb(ii,jj,kk-1) = tf*
     *                      ( dydexf*dzdetf
     *                      - dydetf*dzdexf )
        zetyb(ii,jj,kk-1) = -tf*
     *                      ( dxdexf*dzdetf
     *                      - dxdetf*dzdexf )
        zetzb(ii,jj,kk-1) = tf*
     *                      ( dxdexf*dydetf
     *                      - dxdetf*dydexf )
      endif

   20 continue

***   Transformation expressions on boundary walls

***   East and West boundaries

      do 30 j = 2,jnm1
       do 30 k = 2,knm1

        etax(1,j,k) = etax(2,j,k)
        etay(1,j,k) = etay(2,j,k)
        etaz(1,j,k) = etaz(2,j,k)
        exix(1,j,k) = exix(2,j,k)
        exiy(1,j,k) = exiy(2,j,k)
        exiz(1,j,k) = exiz(2,j,k)
        zetx(1,j,k) = zetx(2,j,k)
        zety(1,j,k) = zety(2,j,k)
        zetz(1,j,k) = zetz(2,j,k)

        etax(in,j,k) = etax(inm1,j,k)
        etay(in,j,k) = etay(inml,j,k)
        etaz(in,j,k) = etaz(inml,j,k)
        exix(in,j,k) = exix(inml,j,k)
        exiy(in,j,k) = exiy(inml,j,k)
        exiz(in,j,k) = exiz(inml,j,k)
        zetx(in,j,k) = zetx(inml,j,k)
        zety(in,j,k) = zety(inml,j,k)
        zetz(in,j,k) = zetz(inml,j,k)
        tjac(in,j,k) = tjac(inml,j,k)

   30 continue

***   North and South boundaries

      do 40 i = 2,inml
       do 40 k = 2,knml

        etax(i,1,k) = etax(i,2,k)
        etay(i,1,k) = etay(i,2,k)
        etaz(i,1,k) = etaz(i,2,k)
        exix(i,1,k) = exix(i,2,k)
        exiy(i,1,k) = exiy(i,2,k)
        exiz(i,1,k) = exiz(i,2,k)
        zetx(i,1,k) = zetx(i,2,k)
        zety(i,1,k) = zety(i,2,k)
        zetz(i,1,k) = zetz(i,2,k)

        etax(i,jn,k) = etax(i,jnml,k)
        etay(i,jn,k) = etay(i,jnml,k)
        etaz(i,jn,k) = etaz(i,jnml,k)
        exix(i,jn,k) = exix(i,jnml,k)
        exiy(i,jn,k) = exiy(i,jnml,k)
        exiz(i,jn,k) = exiz(i,jnml,k)
        zetx(i,jn,k) = zetx(i,jnml,k)
        zety(i,jn,k) = zety(i,jnml,k)
        zetz(i,jn,k) = zetz(i,jnml,k)
        tjac(i,jn,k) = tjac(i,jnml,k)

   40 continue

***   Back and Front boundaries

      do 50 i = 2,inml
       do 50 j = 2,jnml

        etax(i,j,1) = etax(i,j,2)
        etay(i,j,1) = etay(i,j,2)
        etaz(i,j,1) = etaz(i,j,2)
        exix(i,j,1) = exix(i,j,2)
        exiy(i,j,1) = exiy(i,j,2)
        exiz(i,j,1) = exiz(i,j,2)
        zetx(i,j,1) = zetx(i,j,2)
        zety(i,j,1) = zety(i,j,2)
        zetz(i,j,1) = zetz(i,j,2)
```

```
         etax(i,j,kn) = etax(i,j,knml)
         etay(i,j,kn) = etay(i,j,knml)
         etaz(i,j,kn) = etaz(i,j,knml)
         exix(i,j,kn) = exix(i,j,knml)
         exiy(i,j,kn) = exiy(i,j,knml)
         exiz(i,j,kn) = exiz(i,j,knml)
         zetx(i,j,kn) = zetx(i,j,knml)
         zety(i,j,kn) = zety(i,j,knml)
         zetz(i,j,kn) = zetz(i,j,knml)
         tjac(i,j,kn) = tjac(i,j,knml)

   50 continue

***   Calculate the interpolation factors

      do 60 i = 2,inm2
       do 60 j = 2,jnml
        do 60 k = 2,knml
         ii = 2*(i-1)
         jj = 2*(j-1)
         kk = 2*(k-1)
         istot = jng*kng
         ng = (ii-1)*istot + (jj-1)*kng + kk

         dls1 = sqrt( (xx(ng+istot) - xx(ng))**2
     *               +(yy(ng+istot) - yy(ng))**2
     *               +(zz(ng+istot) - zz(ng))**2 )
         dls2 = sqrt( (xx(ng+2*istot) - xx(ng+istot))**2
     *               +(yy(ng+2*istot) - yy(ng+istot))**2
     *               +(zz(ng+2*istot) - zz(ng+istot))**2 )
         fl(i,j,k) = dlsl/(dlsl+dls2)

   60 continue

      do 70 i = 2,inml
       do 70 j = 2,jnm2
        do 70 k = 2,knml
         ii = 2*(i-1)
         jj = 2*(j-1)
         kk = 2*(k-1)
         istot = jng*kng
         ng = (ii-1)*istot + (jj-1)*kng + kk

         d2sl = sqrt( (xx(ng+kng) - xx(ng))**2
     *               +(yy(ng+kng) - yy(ng))**2
     *               +(zz(ng+kng) - zz(ng))**2 )
         d2s2 = sqrt( (xx(ng+2*kng) - xx(ng+kng))**2
     *               +(yy(ng+2*kng) - yy(ng+kng))**2
     *               +(zz(ng+2*kng) - zz(ng+kng))**2 )
         f2(i,j,k) = d2sl/(d2sl+d2s2)

   70 continue

      do 80 i = 2,inml
       do 80 j = 2,jnml
```

```
        do 80 k = 2,knm2
         ii = 2*(i-1)
         jj = 2*(j-1)
         kk = 2*(k-1)
         istot = jng*kng
         ng = (ii-1)*istot + (jj-1)*kng + kk

         d3s1 = sqrt( (xx(ng+1) - xx(ng))**2
     *             +(yy(ng+1) - yy(ng))**2
     *             +(zz(ng+1) - zz(ng))**2 )
         d3s2 = sqrt( (xx(ng+2) - xx(ng+1))**2
     *             +(yy(ng+2) - yy(ng+1))**2
     *             +(zz(ng+2) - zz(ng+1))**2 )
         f3(i,j,k) = d3s1/(d3s1+d3s2)

   80 continue

       do 90 k = 1,kn
        do 90 j = 1,jn
         do 90 i = 1,in
          delexie(i,j,k) = 1.0
          deletan(i,j,k) = 1.0
          delzetb(i,j,k) = 1.0
   90 continue

***-----------------------------------------------------------------
***  Part 3 : Write grid data file for flow simulation
***-----------------------------------------------------------------

       if(isetup.eq.1)then

       write(7,*)in,inm1
       write(7,*)jn,jnm1
       write(7,*)kn,knm1

       do 100 k = 1,kn
        do 100 j = 1,jn
         do 100 i = 1,in

          write(7,1)k,j,i,f1(i,j,k),f2(i,j,k),
     *              f3(i,j,k),x(i,j,k),y(i,j,k),z(i,j,k)
          write(7,2)exix(i,j,k),exiy(i,j,k),exiz(i,j,k),
     *              etax(i,j,k),etay(i,j,k),etaz(i,j,k)
          write(7,2)zetx(i,j,k),zety(i,j,k),zetz(i,j,k),
     *              exixe(i,j,k),exiye(i,j,k),exize(i,j,k)
          write(7,2)etaxe(i,j,k),etaye(i,j,k),etaze(i,j,k),
     *              zetxe(i,j,k),zetye(i,j,k),zetze(i,j,k)
          write(7,2)exixn(i,j,k),exiyn(i,j,k),exizn(i,j,k),
     *              etaxn(i,j,k),etayn(i,j,k),etazn(i,j,k)
          write(7,2)zetxn(i,j,k),zetyn(i,j,k),zetzn(i,j,k),
     *              exixb(i,j,k),exiyb(i,j,k),exizb(i,j,k)
          write(7,2)etaxb(i,j,k),etayb(i,j,k),etazb(i,j,k),
     *              zetxb(i,j,k),zetyb(i,j,k),zetzb(i,j,k)
          write(7,3)tjac(i,j,k),tjace(i,j,k),tjacn(i,j,k),tjacb(i,j,k)
```

```
  100 continue

      endif

    1 format(1x,3i4,2x,3f6.3,2x,3f10.5)
    2 format(1x,6f12.5)
    3 format(1x,4f15.7)

      return
      end
```