Thesis for the Degree of Master of Science

DNS of Channel Flow with Finite Difference Method on a Staggered Grid

Mohammad Irannezhad

Division of Fluid Dynamics
Department of Applied Mechanics
Chalmers University of Technology

Göteborg, Sweden, 2006

DNS of Channel Flow with Finite Difference Method on a Staggered Grid

Mohammad Irannezhad

Diploma Work

Division of Fluid Dynamics, Department of Applied Mechanics,
Chalmers University of Technology
Se-412 96 Göteborg, Sweden
Phone +46-(0)31-7721400
Fax: +46-(0)31-180976

Printed at Chalmers Reproservice
Göteborg, Sweden 2006

DNS of Channel Flow with Finite Difference Method on a Staggered Grid

by

Mohammad Irannezhad

irannezh@student.chalmers.se

Division of Fluid Dynamics

Department of Applied Mechanics

Chalmers University of Technology

SE-412 96 Göteborg

Sweden

Abstract

An available code for solving the incompressible Navier-Stokes equations in a channel is studied and modified. The non-uniform grid in normal direction is transformed to a uniform grid by an analytical transformation. The continuous space is discretized with Finite Difference method on a staggered grid. The code employs the implicit Crank-Nicholson scheme for the linear term and explicit third order Runge-Kutta or Adams-Bashfort for the non-linear term. The modified fractional step method is used for applying the continuity equation into the velocity field in which the pressure gradient at the previous time step is used in the first step of the fractional step method. The resulting Poisson equation is solved by applying the Fourier series in two periodic directions while the velocity equations use finite differences in all directions.

The flow in the code is defined for Reynolds numbers based on half channel width and laminar Poiseuille centerline velocity and the initial laminar mass flow rate is kept constant during the iterations by setting the mean pressure gradient at each time step equal to the integrated wall shear stress. The code is modified to be run for Reynolds numbers based on friction velocity and it is shown that the mean velocity profiles are the same for two cases but the second order statistics are the same only at high Reynolds numbers.

Acknowledgments

I would like to take this opportunity to thank my supervisor Prof. Lars Davidson for his support, guidance and helpful discussions. I would like to thank him specially for his trust and patience during the first months. His short but deep comments helped me doing the thesis and without him it was impossible to get through it.

I also would like to thank PhD student Darioush Gohari Barhaghi whom I learned a lot from. Discussions with Darioush was always helpful and I do not remember him saying 'no' to me eventhough he was very busy.

My special thanks goes to all the people in the department for the friendly environment they had made making it possible to enjoy every moment of my studies. They are all kind, helpful and ready to help.

It is only due to my family that I am here. Expressing my feeling is impossible and I can just say thanks to them. Also, with equal importance to my family I say thanks to Oldooz, the girl who supported me with her never ending source of love and kindness.

Nomenclature

Upper-case Roman

A, B	Runge-Kutta scheme coefficients
H	non-linear term
L	linear function
N	non-linear function
Q	mass flow rate
R	known quantities
S	mean pressure gradient
U	streamwise mean velocity

Lower-case Roman

a, b, c	scheme coefficients
f	any function
k_1, k_2	scheme functions
p	pressure
t	time
u	velocity
u_b	bulk velocity
u_p	Poiseuille centerline velocity
u_τ	friction velocity
x	spanwise direction
x_i	space component
y	normal direction
z	streamwise direction

Upper-case Greek

Φ	scalar to find pressure
Δt	time step size
$\Delta x, \Delta y, \Delta z$	spanwise, normal and streamwise mesh spacings

Lower-case Greek

ϕ	any scalar or scalar function
$\alpha, \beta, \gamma, \sigma, \rho$	scheme coefficients
τ_w	wall shear stress
δ	half channel width

Abbreviations

CFL	Courant, Friedrichs and Lewy number
DNS	Direct Numerical Simulation
FD	Finite Difference
FFT	Fast Fourier Transform
Re	Reynolds number
SF	Scaling Factor

subscripts

b	bulk
i	direction, node number
k	substep
p	Poiseuille
w	wall

superscripts

k	intermediate
n	time step
$'$	first substep
$''$	second substep

symbols

∂	partial derivative
δ	discrete derivative
Ł$_{jj}$	Laplacian
<>	ensemble average

Contents

Chapter 1

Introduction

Fluid flow changes from an orderly and predictable state, laminar flow, to a chaotic and unpredictable one when a certain non-dimensional parameter, Reynolds number, exceeds a critical value. The chaotic state, *turbulence*, is the more common one in engineering and geophysical scales, and its practical significance, as well as the purely intellectual challenge of the problem, has attracted the attention of some of the best minds in the fields of physics, engineering and mathematics. Progress toward a rigorous analytic theory has been prevented by the fact that turbulence dynamics is *stochastic* (often having underlying organized structures) and strongly nonlinear. There are, however, rigorous kinematic results that stem from tensor analysis and the linear constraints of continuity, and these allow a reduction of variables in the statistical description of the velocity field in certain cases, especially for isotropic turbulence. Rigorous dynamical results are available only for limiting cases where the governing equations can be linearized which is seldom possible in practice.

1.1 DNS

The complex behavior of turbulence is the consequence of a fairly simple set of equations, the Navier-Stokes equations. However, as mentioned, analytical solutions to even the simplest turbulent flows do not exist. All attempts at a statistical theory of turbulence have ultimately been faced with the problem of closure, that is, the specification at some order of a statistical quantity for which no governing equation exists. When using these models some information about the flow field is lost due to the inherent averaging of this approach. A complete description of a turbulent flow, where the flow variables (e.g. velocity and pressure) are known as a function of space and time can therefore be obtained by numerically solving the Navier-Stokes equations without any models. These numerical solutions are termed *Direct Numerical Simulation*, or DNS in short.

The instantaneous range of scales in turbulent flows increases rapidly with the Reynolds number. To solve partial differential equations numerically, Navier-Stokes in this case, the continuous space should be discretized and the size of the resulting mesh should resemble the smallest scales of the flow. As a result, most engineering problems have too wide a range of scales to be directly computed using DNS, these flows are solved using turbulence models.

It should be stressed again that DNS is just a research tool, and not a brute-force solution to the Navier-Stokes equations for engineering problems. The power of this research tool is impressive and over the last two decades so much effort has been put on improving the techniques. The main power of DNS lies in the ability to calculate instantaneous fields of any term and any quantity, which is impossible in experiments and in modeled turbulence research.

1.2 Channel flow

As stated above DNS can only be applied to simple cases and computer science is far behind the scope of solving complex flows in complex geometries. One of these very simple cases is that of a fully developed flow between two plane channels, which can be considered as the second simple flow after isotropic decaying turbulence. The channel flow has proved to be an extremely useful framework for the study of wall-bounded turbulence. The first attempt to solve this flow using DNS was done by KIM *et al.* (1987) and since then there have been many modifications in the numerical method.

1.3 Aim of the thesis

The current thesis is aiming at first doing DNS of channel flow with finite difference method on a staggered grid by means of an existing code and then modify the code to be able to run it for new cases. To do this, first the code was to be thoroughly understood and the main part of the time devoted to the thesis was spent for this understanding. Also the code and the modified version are run and the results are compared with two available sets of data from other codes.

The code is included as a CD-ROM in Orlandi (2001). The book contains to some extent the numerical method used in the code but despite some explanations about the code, can not play a role as the manual for it. The book is divided into 13 different chapters corresponding to 13 different cases of which one is DNS of channel flow. The book and the code try to show that by the aid of recent developments in computers' speeds and memories, DNS of simple flows can be run on home workstations, which used to be possible only on huge facilities not available to everyone.

The code and the book have been set to a lot of modifications and many parts of them can be eliminated. This report and additional comments added to the body of the code try to serve as a manual for the code, not only to run it but also to give the necessary information to modify it. Analyzing the solvers are excluded since black-box treatment of the solver does not affect the process of modification.

Chapter 2

Numerical Method

In this chapter the numerical method to solve the Navier-Stokes equations is described. While most of the material presented in this chapter can be used to solve any differential equations, some are adapted to be used only for the case studied. Navier-Stokes equations for incompressible flows and in non-dimensional form read

$$\frac{\partial u_i}{\partial t} + \frac{\partial u_i u_j}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{1}{Re}\frac{\partial^2 u_i}{\partial x_j^2} \tag{2.1}$$

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{2.2}$$

Equation (2.1) is the momentum equation and equation (2.2) is the continuity equation. Navier-Stokes equations are unsteady and to solve them numerically, both space and time discretizations are needed. The aim is to find the velocities and pressures at every point and at all the times with the given boundary conditions. To do this, the momentum equation is first solved and a technique is then used to force the solution to satisfy the continuity equation.

2.1 Grid

To discretize the Navier-Stokes equations in space, the continuous space should be divided into a set of discrete points and the physical quantities should be stored at these points. When this passage to discrete space has been done,the continuous equations will change to a set of ordinary differential equations for each point. The Navier-Stokes equations are *elliptic* in space and the whole domain should be solved simultaneously, either the velocities are found for every point or non of them. The grid generation process is the dividing of the physical domain of the flow into smaller domains called *control volumes* or *cells*.

There are two major types of grids

- **Collocated** in which the scalars, pressure in this case, and the vector type quantities, velocity in this case, are both stored at the center of control volumes.
- **Staggered** in which the scalars are stored at the center of control volumes where as the vectors are stored at the *mesh faces*.

The pressure gradient appears in the momentum equation. In collocated grids this may result in unphysical pressure gradient predictions in case of a high-oscillatory pressure field such as the so called *checker-board* field. By using the staggered grids this problem is overcome. This code is based on staggered grids and this grid is explained here.

Figure (2.1) shows the 2-D staggered grid schematic, for 3-D case it is straightforward. In the picture the points where the different quantities are stored are shown, so the mesh is different when solving different components of velocity. We need a numbering convention to refer to the points. The mesh faces which actually divide the whole domain are numbered with integers 1,2,..., the control volume centers are named with fractional numbers $\frac{1}{2}, \frac{3}{2}, \ldots$, and the wall is called w. This convention is applied to all three directions.

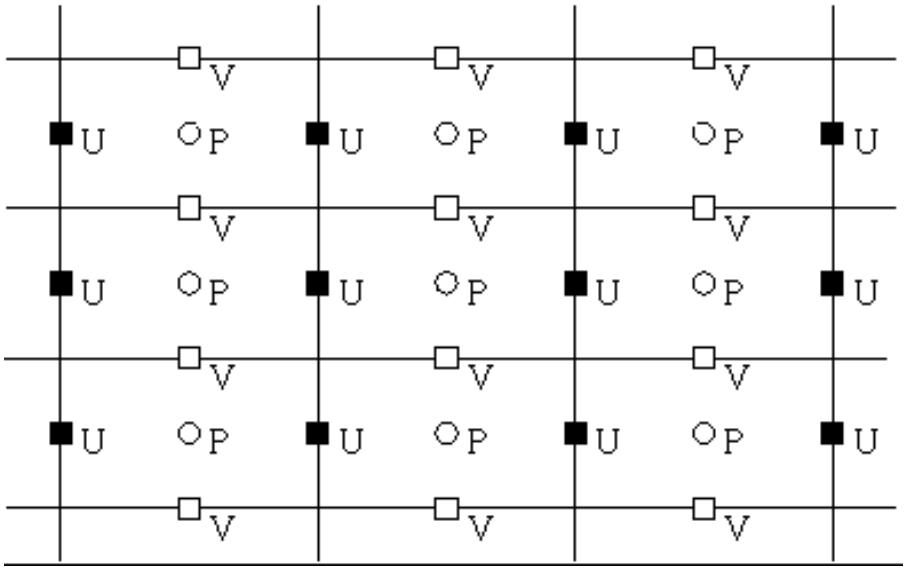

Figure 2.1: Schematic of a staggered grid, quantities are stored in different locations.

In a large number of applications high gradients are located in certain regions. The mesh at these locations should be finer to account for these rapid changes where as the other parts of the domain do not need such a fine mesh; to reduce the computational load, non-uniform meshes can be applied. The grid should be related to the physics of the flow and should be set in advance due to the experimental observations and physical arguments.

2.2 Spatial Discretization

When the domain is divided, the equations should be discretized in this domain. This will result in a set of ordinary differential equations for each grid point in space. There are different methods to discretize the equations and the code employs the *finite difference* method which is the most straightforward one. Finite difference method is simply the substitution of the continuous differential operators with corresponding discrete operators. As it is obvious from Navier-Stokes equations, operators of first and second orders are needed. Substituting the operators with discrete ones δ and δ^2 results in discrete Navier-Stokes equations

$$\frac{\delta u_i}{\delta t} + \frac{\delta u_i u_j}{\delta x_j} = -\frac{\delta p}{\delta x_i} + \frac{1}{Re}\frac{\delta^2 u_i}{\delta x_j^2} \tag{2.3}$$

$$\frac{\delta u_i}{\delta x_i} = 0 \tag{2.4}$$

Now we should define these operators at every point based on the quantities at other points. These operators are defined for uniform grids and later it will be shown how they can be adapted to apply to non-uniform grids too. Considering a quantity, say u, the operators are defined as

$$\left.\frac{\delta u}{\delta x}\right|_{i,j+\frac{1}{2},k+\frac{1}{2}} = \frac{1}{x|_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} - x|_{i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}}(u|_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} - u|_{i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}) \tag{2.5}$$

The u component of velocity is not defined at $(i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2})$ and $(i-\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2})$ and the velocities at these point are found by interpolation of velocities at two adjacent points. Considering a uniform mesh this leads to the simple *central difference* method and is second order accurate in space

$$\frac{\delta u}{\delta x}|_{i,j+\frac{1}{2},k+\frac{1}{2}} = \frac{1}{\Delta x}(u|_{i+1,j+\frac{1}{2},k+\frac{1}{2}} - u|_{i-1,j+\frac{1}{2},k+\frac{1}{2}}) \tag{2.6}$$

$$\frac{\partial u}{\partial x} = \frac{\delta u}{\delta x} + \mathcal{O}(\Delta x)^2 \tag{2.7}$$

Second order differential operator is defined to be two successive applications of the first order operator leading to

$$\frac{\delta^2 u}{\delta x^2} = \frac{\delta}{\delta x}(\frac{\delta u}{\delta x}) \tag{2.8}$$

and in case of a uniform grid to

$$\frac{\delta^2 u}{\delta x^2} = \frac{1}{(\Delta x)^2}(u|_{i-1,j+\frac{1}{2},k+\frac{1}{2}} - 2u|_{i,j+\frac{1}{2},k+\frac{1}{2}} + u|_{i+1,j+\frac{1}{2},k+\frac{1}{2}}) \tag{2.9}$$

which is also second order accurate in space. The other components and the other directions will be treated in the same way.

2.2.1 Non-uniform grid

In case of a non-uniform grid the accuracy of both operators will formally reduce to one and the direct equations of (2.6) and (2.9) are not applicable. There are two different ways of generating a non-uniform grid, in the first method the non-uniform grid is generated at first and all the calculations are done by means of the operators given above. The second way, which is used in the code, uses two different domains, a *physical domain* in which the grid is non-uniform and a *computational domain* in which the grid is uniform. The uniform grid is transformed to a non-uniform grid by means of an analytical function. If x is the physical domain and ξ is the computational domain the function is

$$x = f(\xi) \tag{2.10}$$

so the grid is generated uniformly in the computational domain and then the final results are transformed to the corresponding points in the physical domain. In the discretized Navier-Stokes equations the derivatives are in physical domain so we should adapt the operators to apply for the computational domain, the key is the simple *chain rule* of differentiation.

$$\frac{\partial u}{\partial x} = \frac{\partial \xi}{\partial x}\frac{\partial u}{\partial \xi} \tag{2.11}$$

The resulting discrete operator is first order accurate in physical domain despite it is second order in computational domain. In case of the second order operator there are two different options, the first is simply the application of two successive first order operators

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial}{\partial \xi}(\frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial x})\frac{\partial \xi}{\partial x} \tag{2.12}$$

the second explicits the first order derivatives in (2.12), that is

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial}{\partial \xi}\Big(\frac{\frac{\partial u}{\partial \xi}}{\frac{\partial x}{\partial \xi}}\Big)\frac{\partial \xi}{\partial x} = \frac{\partial \xi}{\partial x}\left(\frac{\frac{\partial}{\partial \xi}(\frac{\partial u}{\partial \xi})\frac{\partial x}{\partial \xi} - \frac{\partial u}{\partial \xi}\frac{\partial}{\partial \xi}(\frac{\partial x}{\partial \xi})}{(\frac{\partial x}{\partial \xi})^2}\right) \tag{2.13}$$

and this gives

$$\frac{\partial^2 u}{\partial x^2} = \frac{\frac{\partial^2 u}{\partial \xi^2}\frac{\partial x}{\partial \xi} - \frac{\partial u}{\partial \xi}\frac{\partial^2 x}{\partial \xi^2}}{(\frac{\partial x}{\partial \xi})^3} \tag{2.14}$$

Immediately it turns out that even in 1-D case the second approach is more cumbersome requiring a greater number of operations and even a large memory occupancy to store the transformation metric quantities.

As a first remark, it is important to point out that it is convenient to look for smooth transformations, to avoid discontinuities that could affect the accuracy of the solution. Actually, the final results are sensitive to the stretching factor in physical domain. A smooth analytical transformation fulfills this requirement but it leads to several options. When the analytic form of f is known it is tempting to calculate the metrics of transformation analytically and use them to evaluate the discrete derivatives by aforementioned expressions. This is a dangerous choice especially in presence of enhanced grid clustering. The evaluation of truncation errors by Taylor series expansions shows that error cancellations occur when the metrics are evaluated numerically. There is a code included which shows the higher accuracy of the second order derivative of a sample function by numerical evaluation of metrics than analytic evaluation, this code is in the file CODESCH2/SECD of the diskette.

Even when numerical evaluation of the metrics is chosen it is possible to choose between (2.12) and (2.14). It is possible to manipulate the truncation expressions of the two expressions but it is easier to write a code and prove by examples that (2.12) gives better result. This is the expression used all the way in the code.

2.2.2 Boundaries

The expressions used for discrete derivatives can not be applied to the boundaries directly and some modifications are needed. The boundary condition treatment presented in this section addresses only those boundary condition types which are used in the code. Two different kinds of boundary conditions are assigned in the code, *periodic* in streamwise and spanwise directions and *no-slip* in the direction normal to the wall. In case of periodic boundary condition, the quantity for which the periodic boundary condition is to be applied is set equal at the first and last mesh faces. However, the no-slip condition is a bit more difficult to assign. In this section it will be shown that how it is possible to find an expression for the second derivative at the first node from the wall, $(i, w+\frac{1}{2}, k)$, to account for the no-slip boundary condition. It is worth to note that the first derivative expression can be used without any elaboration for the boundary nodes and only some extrapolations are needed.

To find an expression for second derivative of the first node near the wall we first write the Taylor expansion series of the nodes on the wall and the node next to the first node

$$u_w = u_{w+\frac{1}{2}} - \Delta y_1 \frac{\partial u}{\partial y}|_{w+\frac{1}{2}} + \frac{\Delta y_1^2}{2}\frac{\partial^2 u}{\partial y^2}|_{w+\frac{1}{2}} + \mathcal{O}(\Delta y)^3 \tag{2.15}$$

$$u_{w+\frac{3}{2}} = u_{w+\frac{1}{2}} + \Delta y_2 \frac{\partial u}{\partial y}|_{w+\frac{1}{2}} + \frac{\Delta y_2^2}{2}\frac{\partial^2 u}{\partial y^2}|_{w+\frac{1}{2}} + \mathcal{O}(\Delta y)^3 \tag{2.16}$$

where

$$\Delta y_1 = y|_{i,w+\frac{1}{2},k} - y|_{i,w,k}, \qquad \Delta y_2 = y|_{i,w+\frac{3}{2},k} - y|_{i,w+\frac{1}{2},k} \tag{2.17}$$

by eliminating $\frac{\partial u}{\partial y}|_{w+\frac{1}{2}}$ we get the final expression

$$\frac{\partial^2 u}{\partial y^2}|_{w+\frac{1}{2}} = \frac{2u_{w+\frac{3}{2}}}{\Delta y_1(\Delta y_1 + \Delta y_2)} + \frac{2u_w}{\Delta y_2(\Delta y_1 + \Delta y_2)} - \frac{2u_{w+\frac{1}{2}}}{\Delta y_1 \Delta y_2} + \mathcal{O}(\Delta y) \tag{2.18}$$

The mesh is non-uniform in normal direction and $y = f(\xi)$ is the transformation. We adopt a notation, where a subscript denotes the derivative of the base variable with respect to the subscript e.g. $y_\xi = \frac{dy}{d\xi}$ or alternatively $y_\xi = \frac{\partial y}{\partial \xi}$. Expressing the above equation in computational domain gives the final expression used in the code

$$\frac{\partial^2 u}{\partial y^2}|_{w+\frac{1}{2}} = \frac{1}{(\Delta\xi)^2}\left[\frac{2u_{w+\frac{3}{2}}}{y_\xi|_{w+1}\left(\frac{y_\xi|_w}{2} + y_\xi|_{w+1}\right)} + \frac{4u_w}{y_\xi|_w\left(\frac{y_\xi|_w}{2} + y_\xi|_{w+1}\right)} - \frac{4u_{w+\frac{1}{2}}}{y_\xi|_w y_\xi|_{w+1}}\right] \tag{2.19}$$

Using this expression, the spatial discretization of the Navier-Stokes equations is complete. It is noted that the velocity at the wall has appeared in this expression which shows how the no-slip boundary condition, including those with moving walls, can be applied to the discretized equations. This expression is used for both streamwise and spanwise velocities and in case of normal velocity no problem appears since in a staggered grid the normal velocities are calculated at the faces leading to enough information for the central

difference formulas to be used. This expression is used to assign the no-slip boundary condition in all the codes in the book. It is to be noted that even for Navier-Stokes equations casted in streamline coordinates the same expression is needed to assign the proper boundary conditions for the vorticity at the wall.

2.3 Time discretization

Unlike in space where Navier-Stokes equations are elliptic, they are *parabolic* in time. Due to this parabolic behavior, marching in time is done to solve the unsteady Navier-Stokes equations. This means that, to have the solution at a certain time, only the information up to that time is needed, usually information at the previous time step is enough. The spatial discretization of parabolic partial differential equations leads to a system of ordinary differential equations for each grid point. There are four terms in Navier-stokes equations

- $\frac{\partial u}{\partial t}$ which is calculated with central differences and contains the unknown u_i^{n+1}

$$\frac{\partial u}{\partial t} = \frac{u_i^{n+1} - u_i^n}{\Delta t} \tag{2.20}$$

- $\frac{\partial u_i u_j}{\partial x_j}$ is called the *advective* term. This term is non-linear and needs special care if we do not want to work with non-linear equations. From now on we call it the non-linear term.

- $-\frac{\partial p}{\partial x_i}$ is the pressure gradient term. This term is treated based on the method used to satisfy the continuity equation. In the next section when explaining the *fractional step* method, it will be clarified how this term is treated.

- $\frac{1}{Re}\frac{\partial^2 u_i}{\partial x_j^2}$ is the viscous loss term. This is a linear term and as stated before the no-slip condition is applied through this term, from now on we call it the linear term.

There are two different types of treating the terms in time discretization

- **Implicit** in which the terms are expressed by the data at past and present times. Treating the non-linear terms implicitly leads to non-linear sets of ordinary differential equations which need iterative schemes to be solved.

- **Explicit** in which the terms are expressed by the data using only the previous times. All the terms treated with these kind of schemes will lead to linear sets of ordinary differential equations.

The simplest way is to treat both terms explicitly even if severe restrictions for the time step occurs. For simplicity let's consider the equation below which serves as a model for more complicated cases

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} = \frac{1}{Re}\frac{\partial^2 \phi}{\partial x^2} \tag{2.21}$$

All the arguments made here can be used for the Navier-Stokes equations. Treating both terms explicitly leads to

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} + u\frac{\phi_i^n - \phi_{i-1}^n}{\Delta x} = \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{\Delta x^2} + \mathcal{O}(\Delta t) \tag{2.22}$$

This scheme is referred to as explicit forward Euler and gives a straightforward procedure to advance in time. A serious drawback of this scheme is its instability. A standard stability analysis leads to satisfy the following inequalities

$$\frac{\Delta t}{\Delta x} < 1 \tag{2.23}$$

$$\frac{\Delta t}{\Delta x^2 Re} < \frac{1}{2} \tag{2.24}$$

The first inequality is referred to as the Courant, Friedrichs and Lewy, *CFL*, condition and is due to the explicit treatment of the advective term of equation (2.21). In this simple example the condition could be bypassed by computing implicitly the advective term (by evaluating the second term of equation (2.22) at the time level n+1) and this would yield an unconditionally stable scheme provided that the second inequality is neglected. In the Navier-Stokes equations, however, the advective terms are non-linear and the implicit treatment is impractical, since the unknowns appear also as multiplicative coefficients of the derivative, leading to non-linear sets of differential equations. The linearision of the non-linear term permits the implicit treatment, however, by neglecting iterations this reduces the accuracy of the scheme.

The second inequality is the viscous stability condition, usually more restrictive than the previous one. This is particularly true for low Reynolds number flows and in the presence of fine spatial discretization (near the walls in channel flow). In addition it turns out that while the CFL condition remains unchanged when two or three dimensional flows are considered, the viscous stability condition becomes more restrictive since the right hand side of the inequality reads $\frac{1}{2n}$ with n equal to the number of spatial dimensions of the problem. On the other hand, the viscous term in the Navier-Stokes equations is linear and an implicit treatment does not require any particular effort, except that to invert tridiagonal matrices.

In the following of this section, an implicit scheme for the linear term and two explicit schemes for the non-linear term are described. In choosing between scheme it is preferable to use schemes with lower memory requirements, since any dummy array is large when the mesh is fine. Remembering that this is a code to be run on a desktop workstation this requirement is better understood.

2.3.1 Implicit Crank-Nicholson scheme

Implicit Crank-Nicholson scheme is widely used to solve partial differential equations due to its simplicity and stability. To describe this scheme let's consider equation (2.21) without the non-linear advective term

$$\frac{\partial \phi}{\partial t} = \frac{1}{Re}\frac{\partial^2 \phi}{\partial x^2} \tag{2.25}$$

Crank-Nicholson scheme evaluates the linear term at the time level $n + \frac{1}{2}$, leading to

$$\phi_i^{n+1} = \phi_i^n + \frac{\Delta t}{2Re}\Big[\delta^2\phi_i^{n+1} + \delta^2\phi_i^n\Big] \tag{2.26}$$

for the 1D case. As before δ is the discrete differential operator. This scheme is second order accurate in time and is unconditionally stable meaning that time steps of arbitrary size are permitted. However, even if stability does not restrict the size of the time step, the accuracy does. In fact the error $\mathcal{O}(\Delta t)^2$ implies that large time steps result in inaccurate solutions.

Using the second order accurate central difference formulation in space for the discrete operator and using the variable $\Delta\phi_i = \phi_i^{n+1} - \phi_i^n$ we get

$$-\frac{\Delta t}{2Re\Delta x^2}\Delta\phi_{i+1} + \Big(1 + \frac{\Delta t}{Re\Delta x^2}\Big)\Delta\phi_i - \frac{\Delta t}{2Re\Delta x^2}\Delta\phi_i = R_i^n \tag{2.27}$$

where Δx is the spatial step and R_i^n contains all the known quantities. This equation written for all the nodes and completed for the boundary conditions can be solved by inverting a tri-diagonal matrix. The procedure will be much more complicated in 2-D and 3-D cases. At the end, it is worth to note that the Crank-Nicholson scheme is simply the trapezoidal rule of integration.

2.3.2 Explicit Adams-Bashfort

Let' consider equation (2.21) again, this time we neglect the diffusive (viscous, i.e the linear) part of it

$$\frac{d\phi}{dt} = f(\phi, t) \tag{2.28}$$

which can represent a linear or non-linear ordinary differential equation. The simplest way to solve this equation is to calculate it explicitly by evaluating the right hand side at the previous time step, that is

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = f_i^n \tag{2.29}$$

This is not a very effective way of solving the equation. A better way is to use more information to evaluate the right hand side of the equation. Again using the trapezoidal rule we get

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = f_i^{n+\frac{1}{2}} + \mathcal{O}(\Delta t)^2 \tag{2.30}$$

The only difference with implicit schemes such as Crank-Nicholson is that we calculate $f_i^{n+\frac{1}{2}}$ by means of the information at the times n and $n-1$. Using the linear interpolation for this purpose leads to

$$f_i^{n+\frac{1}{2}} = \frac{1}{2}(3f_i^n - f_i^{n-1}) + \mathcal{O}(\Delta t)^2 \tag{2.31}$$

This scheme needs an extra array in comparison to Crank-Nicholson scheme to store the f_i^{n-1} term and this increases the memory requirements. On the other hand, it is very easy to apply since it does not need any matrix inversion and the advancement in time is very straightforward. This is an explicit scheme and conditionally stable for $CFL < 1$. A drawback of this scheme is that it needs constant time steps during the integration to keep the second order accuracy. If the time step changes during the integration both the order of accuracy and the coefficients multiplying f_i^n and f_i^{n-1} change.

2.3.3 Explicit Runge-Kutta scheme

The order of accuracy of a method increases by including more terms in the Taylor expansion representation of the quantity at the next time step. There are different methods for providing additional information to increase the order of accuracy. The Runge-Kutta method introduces intermediate points between t^n and t^{n+1} and evaluates f at these intermediate points. The additional function evaluations, of course, result in higher cost per time step; but the accuracy is increased, and it turns out that better stability properties are also obtained. In this section first the second order Runge-Kutta scheme is described which leads to better understanding of the more complicated third order scheme used in the code.

Second order Runge-Kutta

Consider equation (2.28) again, the solution at time step t^{n+1} is obtained from

$$\phi^{n+1} = \phi^n + \gamma_1 k_1 + \gamma_2 k_2 \tag{2.32}$$

where the functions k_1 and k_2 are defined sequentially

$$k_1 = \Delta t f(\phi^n, t^n) \tag{2.33}$$

$$k_2 = \Delta t f(\phi^n + \beta k_1, t^n + \alpha \Delta t) \tag{2.34}$$

and α, β, γ_1, γ_2 are constants to be determined. These constants are determined to ensure the highest order of accuracy. To establish the highest order of accuracy, let's consider the Taylor series expansion for ϕ^{n+1}

$$\phi^{n+1} = \phi^n + \Delta t \frac{\partial \phi^n}{\partial t} + \frac{(\Delta t)^2}{2} \frac{\partial^2 \phi^n}{\partial t^2} + \mathcal{O}(\Delta t)^3 \tag{2.35}$$

but we know

$$\frac{\partial \phi}{\partial t} = f(\phi^n, t^n) \tag{2.36}$$

Now we use the chain rule of differentiation to obtain an expression for the second derivative of ϕ with respect to t expressed in terms of f and its derivatives. It should be noted that ϕ is a function of t and f is a function of both t and ϕ

$$\frac{\partial^2\phi}{\partial t^2} = \frac{\partial}{\partial t}\Big(\frac{\partial\phi}{\partial t}\Big) = \frac{\partial}{\partial t}\Big(f(\phi,t)\Big) = \frac{\partial f}{\partial t}\frac{\partial t}{\partial t} + \frac{\partial f}{\partial \phi}\frac{\partial \phi}{\partial t} \tag{2.37}$$

and the final expression is

$$\frac{\partial^2\phi}{\partial t^2} = f_t + f f_\phi \tag{2.38}$$

as defined before f_t and f_ϕ are the partial derivatives of f with respect to t and ϕ. Thus,

$$\phi^{n+1} = \phi^n + \Delta t f(\phi^n, t^n) + \frac{(\Delta t)^2}{2}\Big(f_t^n + f^n f_\phi^n\Big) \tag{2.39}$$

To establish the order of accuracy of Runge-Kutta method as given by equation (2.32), we must compare its estimate for ϕ_{n+1} to that of the Taylor series formula (2.35). For this comparison to be useful we should convert the various terms in these expressions into common forms. Two dimensional Taylor series expansion of k_2 leads to

$$k_2 = \Delta t\Big[f(\phi^n, t^n) + \beta k_1 f_\phi^n + \alpha\Delta t f_t^n + \mathcal{O}(\Delta t)^2\Big] \tag{2.40}$$

Noting that $k_1 = \Delta t f(\phi^n, t^n)$ and substituting in (2.32) yields

$$\phi^{n+1} = \phi^n + (\gamma_1+\gamma_2)\Delta t f^n + \gamma_2\beta(\Delta t)^2 f^n f_\phi^n + \gamma_2\alpha(\Delta t)^2 f_t^n + ... \tag{2.41}$$

Comparison of equations (2.41) and (2.39) and matching the coefficients of similar terms leads to

$$\gamma_1 + \gamma_2 = 1$$

$$\gamma_2\alpha = \frac{1}{2}$$

$$\gamma_2\beta = \frac{1}{2}$$

These are three non-linear equations for four unknowns. Using α as a free parameter, we have

$$\gamma_1 = 1 - \frac{1}{2\alpha} \qquad \beta = \alpha \qquad \gamma_2 = \frac{1}{2\alpha}$$

With three out of the four constants chosen, we have a one-parameter family of second-order Runge-Kutta formulas:

$$k_1 = \Delta t f(\phi^n, t^n) \tag{2.42}$$

$$k_2 = \Delta t f(\phi^n + \alpha k_1, t^n + \alpha\Delta t) \tag{2.43}$$

$$\phi^{n+1} = \phi^n + (1 - \frac{1}{2\alpha})k_1 + \frac{1}{2\alpha}k_2 \tag{2.44}$$

Thus, we have a second-order Runge-Kutta formula for each value of α chosen. The choice of $\alpha = \frac{1}{2}$ is made frequently. In actual computations, one calculates k_1 using (2.42); this value is then used to compute k_2 using (2.43) followed by the calculation of ϕ^{n+1} using (2.44).

Third order Runge-Kutta

Third-order Runge-Kutta is exactly the same as the second-order version except that in this case the advancement in time is done in three substeps instead of two, or equally two intermediate points between t^n and t^{n+1} are evaluated to improve the order of accuracy of the scheme. The third-order version is presented here with the same notations as the book to help better understanding of the scheme.

$$\phi' = \phi^n + a\Delta t^n f^n \tag{2.45}$$

$$\phi'' = \phi' + b\Delta t^n f(\phi^A, t^n + A\Delta t^n) \tag{2.46}$$

$$\phi^A = \phi^n + A\Delta t^n f^n \tag{2.47}$$

$$\phi^{n+1} = \phi'' + c\Delta t^n f(\phi^B, t^n + (a+B)\Delta t^n) \tag{2.48}$$

$$\phi^B = \phi' + B\Delta t^n f(\phi^A, t^n + A\Delta t^n) \tag{2.49}$$

We follow the same procedure we did for the second-order version to evaluate the order of accuracy of the scheme. This consists of substituting in the expression for ϕ^{n+1} all the terms given above and comparing the results with those of the Taylor expansion series.

$$\phi^{n+1} = \phi^n + a\Delta t^n f^n + b\Delta t^n f(\phi^n + A\Delta t^n f^n, t^n + A\Delta t^n) +$$

$$c\Delta t^n f(\phi^n + a\Delta t^n f^n + B\Delta t^n f(\phi^n + A\Delta t^n f^n, t^n + A\Delta t^n), t^n + (a+B)\Delta t^n) \tag{2.50}$$

Then we should express the third derivative of ϕ with respect to t with the other variables, as we did for the second derivative when studying the second-order scheme.

$$\frac{\partial^3\phi}{\partial t^3} = \frac{\partial}{\partial t}(\frac{\partial^2\phi}{\partial t^2}) = \frac{\partial}{\partial t}(f_t + f f_\phi) = \frac{\partial f_t}{\partial t}\frac{\partial t}{\partial t} + \frac{\partial f_t}{\partial \phi}\frac{\partial \phi}{\partial t} + f_\phi\Big[\frac{\partial f}{\partial t}\frac{\partial t}{\partial t} + \frac{\partial f}{\partial \phi}\frac{\partial \phi}{\partial t}\Big] + f\Big[\frac{\partial f_\phi}{\partial t}\frac{\partial t}{\partial t} + \frac{\partial f_\phi}{\partial \phi}\frac{\partial \phi}{\partial t}\Big] \tag{2.51}$$

which is

$$\frac{\partial^3\phi}{\partial t^3} = f_{tt} + 2f f_{\phi t} + f_\phi f_t + f f_\phi^2 + f^2 f_{\phi\phi} \tag{2.52}$$

Taylor series expansion of ϕ^{n+1} is

$$\phi^{n+1} = \phi^n + \Delta t^n \frac{\partial \phi_n}{\partial t} + \frac{(\Delta t^n)^2}{2}\frac{\partial^2\phi^n}{\partial t^2} + \frac{(\Delta t^n)^3}{6} + \mathcal{O}(\Delta t^n)^4 \tag{2.53}$$

so

$$\phi^{n+1} = \phi^n + \Delta t^n f + \frac{(\Delta t^n)^2}{2(f_t + f f_\phi} + \frac{(\Delta t^n)^3}{6}[f_{tt} + 2f f_{\phi t} + f_\phi f_t + f f_\phi^2 + f^2 f_{\phi\phi}] + \mathcal{O}(\Delta t^n)^4 \tag{2.54}$$

Now we substitute in (2.50) the Taylor series expansion of f at different point appeared there which yields

$$\phi^{n+1} = \phi^n + a\Delta t^n f^n + b\Delta t^n \Big[f^n + A\Delta t^n f^n f^n_\phi + A\Delta t^n f^n_t + A^2(\Delta t^n)^2 f^n f^n_{\phi t}$$

$$+\frac{1}{2}A^2(\Delta t^n)^2 (f^n)^2 f^n_{\phi\phi} + \frac{1}{2}A^2(\Delta t^n)^2 f^2_{tt}\Big] + c\Delta t^n \Big[f^n + f^n_\phi \Big(a\Delta t^n f^n +$$

$$B\Delta t^n (f^n + A\Delta t^n f^n f^n_\phi + A\Delta t^n f^n_t\Big) + (a+B)\Delta t^n f^n_t + \frac{1}{2}(a+B)^2(\Delta t^n)^2 f^n f^n_{\phi t} + \frac{1}{2}(a+B)^2(\Delta t^n)^2 f^n_{tt} +$$

$$\frac{1}{2}(a+B)^2(\Delta t^n)^2 (f^n)^2 f^n_{\phi\phi}\Big] \tag{2.55}$$

When writing the Taylor series in the above expression, terms of orders higher than three are eliminated. Arranging this expression and equating the coefficients for Δt^n,$(\Delta t^n)^2$ and $(\Delta t^n)^3$ we find the following equations

$$a + b + c = 1$$

$$(a + B)c + Ab = \frac{1}{2}$$

$$(a + B)^2 c + A^2 b = \frac{1}{3}$$

$$ABc = \frac{1}{6}$$

This is a system of four equations for five unknowns. A commonly used solution is found by setting $b = 0$. This gives

$$a = \tfrac{1}{4}, \qquad A = \tfrac{8}{15}, \qquad B = \tfrac{5}{12}, \qquad c = \tfrac{3}{4}$$

It is obvious that the third-order Runge-Kutta is advancing in three substeps $a\Delta t^n$, $b\Delta t^n$ and $c\Delta t^n$ and the total advancement is Δt^n.

2.3.4 Hybrid third-order Runge-Kutta/Crank-Nicholson

The coefficients found for the third-order Runge-Kutta in the previous section are not directly usable for solving the Navier-Stokes equations since the linear term was neglected. To find the equations for this case we consider a more general equation

$$\frac{\partial \phi}{\partial t} = N(\phi) + L\phi \tag{2.56}$$

where L is the linear operator and N is the non-linear function. Similar to the previous section, these substeps are done

$$\phi' = \phi^n + \gamma_1 \Delta t^n N^n + \rho_1 \Delta t^n N^{-'} + \alpha_1 \Delta t^n \frac{L\phi' + L\phi^n}{2} \tag{2.57}$$

$$\phi'' = \phi' + \gamma_2 \Delta t^n N' + \rho_2 \Delta t^n N^n + \alpha_2 \Delta t^n \frac{L\phi'' + L\phi'}{2} \tag{2.58}$$

$$\phi^{n+1} = \phi'' + \gamma_3 \Delta t^n N'' + \rho_3 \Delta t^n N' + \alpha_3 \Delta t^n \frac{L\phi'' + L\phi^{n+1}}{2} \tag{2.59}$$

The term $N^{-'}$ is the counterpart of N'' at the time level $n-1$. Since no previous information is provided at the time $t = 0$, it is desirable that $\rho_1 = 0$. This gives an interesting property to the scheme, every time step is self-starting and does not need any information from past.

The linear term at each time substep is evaluated by implicit Crank-Nicholson scheme. The coefficients α_i are determined by imposing that nonlinear and linear terms advance at the same time step fraction. This gives

$$\alpha_k = \rho_k + \gamma_k \tag{2.60}$$

This can be understood better if it is reminded that the Runge-Kutta scheme is actually advancing in three substeps to fulfill one time step. The size of these three steps are obvious by looking at equations (2.57) to (2.59). From these equations it is seen that by imposing $\rho_1 = 0$, ϕ' is evaluated at time $t^n + \gamma_1 \Delta t^n$. This is seen if we take a look at the Taylor series expansion

$$\phi^{n+1} = \phi^n + h\frac{\partial \phi^n}{\partial t} + \mathcal{O}(h^2) \tag{2.61}$$

h being the time advancement. Comparing this to equation (2.57) shows that the first substep is $h = \gamma_1 \Delta t^n$, and to have the same time advancement for the linear term we should set $\alpha_1 = \gamma_1$ or $\alpha_1 = \gamma_1 + \rho_1$ since $\rho_1 = 0$.

The second substep size is found when we substitute ϕ' from (2.57) into (2.58). Neglecting the linear term and higher order terms in Taylor series expansion of N' we get

$$\phi'' = \phi^n + (\gamma_1 + \gamma_2 + \rho_2)\Delta t^n N^n \tag{2.62}$$

which shows an advancement of $\gamma_1 + \gamma_2 + \rho_2$ from beginning or $\gamma_2 + \rho_2$ from the last substep, leading to $\alpha_2 = \gamma_2 + \rho_2$.

Following the same procedure we get $\alpha_3 = \gamma_3 + \rho_3$. We could use the information from the last subsection to find the time advancement substep sizes. By neglecting the linear terms we get

$$\phi^{n+1} = \phi^n + \Delta t^n[(\gamma_1 + \rho_2)N^n + (\gamma_2 + \rho_3)N' + \gamma_3 N''] \tag{2.63}$$

and from previous subsection we have

$$\phi^{n+1} = \phi^n + a\Delta t^n f(\phi^n, t^n) + b\Delta t^n f(\phi^A, t^n + A\Delta t^n) + c\Delta t^n f(\phi^B, t^n + (a+B)\Delta t^n) \tag{2.64}$$

comparison of these two equations gives

$$a = \gamma_1 + \rho_2$$

$$b = \gamma_2 + \rho_3$$

$$c = \rho_3$$

Taking into account the expressions for ϕ^A and ϕ^B we have

$$A = \gamma_1$$

$$B = \gamma_2$$

and finally we find the coefficients used in the code,

$$\alpha_1 = \frac{8}{15}, \alpha_2 = \frac{2}{15}, \alpha_3 = \frac{1}{3}$$

$$\gamma_1 = \frac{8}{15}, \gamma_2 = \frac{5}{12}, \gamma_3 = \frac{3}{4}$$

$$\rho_1 = 0, \rho_2 = -\frac{17}{60}, \rho_3 = -\frac{5}{12} \tag{2.65}$$

The third order accuracy is achieved only if the linear term is neglected and the hybrid scheme is only second order accurate. This may be strange that we use third order Runge-Kutta when the easier and faster second-order can be used without missing any accuracy. However, the key is the stability, the third order Runge-Kutta is stable at higher CFL numbers than those for second order making the use of it reasonable.

2.3.5 Adams-Bashfort *vs* third-order Runge-Kutta

When running a case with the code we should choose between Adams-Bashfort and third-order Runge-Kutta schemes for the non-linear term, so it is essential to know the advantages and disadvantages of these schemes.

- Adams-Bashfort is second order accurate where as Runge-Kutta is third-order, but considering the linear term the accuracy of the whole equation is second order and the third order accuracy of the scheme is useless.
- Adams-Bashfort is stable for $CFL < 1$ while Runge-Kutta is stable for $CFL < \sqrt{3}$ leading to the possibility of having large time steps, hence decreasing the computations. This is the main advantage of using third-order Runge-Kutta. However, even if stability permits $CFL = \sqrt{3}$, accuracy may not.
- although Runge-Kutta is stable for $CFL < \sqrt{3}$ and Adams-Bashfort for $CFL < 1$ but experience reveals that calculations with Adams-Bashfort should be run for $CFL < 0.6$ while with the aid of the viscous term Runge-Kutta can be run for $CFL = 2$ and even higher. This is an interesting property if the calculations are to be run for constant time step, since Runge-Kutta can manage severe situations where the magnitude of velocity is high but is followed by slow dynamics.

- Runge-Kutta advances in three substeps and needs a lot more function evaluations.
- Adams-Bashfort is second order accurate only in constant time step mode, also the coefficients of this scheme should be changed when the time step size changes. This case is not used in the code and the Adams-Bashfort should be used only with constant time step mode.
- each time step is self-starting in Runge-Kutta method but not in Adams-Bashfort, this allows to perform the calculations always at the maximum time step without decreasing the accuracy of the scheme or using any interpolations.
- Adams-Bashfort needs an extra array compared to Runge-Kutta to store the non-linear term at the previous time step.
- Runge-Kutta can be used in both constant time step and constant CFL number modes.

2.4 Fractional Step method

The Navier-Stokes equations are a set of four equations, three momentum and one continuity, with four variables, three velocity components and the pressure. What we did till now was to neglect the continuity and to solve the momentum equations assuming that the pressure is known. But there are two problems, first the velocities may not satisfy the continuity equation, and second the pressure at the next time step is not calculated yet. There are iterative methods by which we can satisfy the continuity equation by solving the pressure and correcting it iteratively to have the correct value at each time step making the velocity field satisfy the continuity equation. This is an effective but slow method. Here another method called *fractional step methods* is described which forces the continuity equation to be satisfied. In this method first an intermediate velocity field is found which does not satisfy continuity and then by aid of this velocity field an equation for a virtual scalar quantity is found which is related to the pressure. From this scalar both the solenoidal velocity field and the pressure are calculated.

There are various versions of this method slightly differing from each other. In the early versions, KIM & MOAN (1985), the pressure was neglected in the first step of the method and there was some difficulty to assign the correct boundary conditions to the intermediate velocity field. The method described here and used in the code uses the pressure at the previous time step explicitly in the first step and by this, the boundary conditions of the intermediate velocity are the same as the real velocity field. The Navier-Stokes momentum equations discretized by Crank-Nicholson for the linear term and Adams-Bashfort or third-order Runge-Kutta schemes for the non-linear term reads

$$\frac{u_i^k - u_i^n}{\Delta t} = \gamma_k H_i^n + \rho_k H_i^{n-1} + \frac{\alpha_k}{2Re} L_{jj}(u_i^k + u_i^n) - \alpha_k \sigma_1 \frac{\delta p^n}{\delta x_i} + S_{i3} \tag{2.66}$$

where H represents the non-linear term(s), L_{jj} is the Laplacian operator and u_i^k is the intermediate velocity field. ρ_k, γ_k and α_k are the scheme coefficients found in the previous section. The coefficient σ_1 is found later and its role and importance will be discussed.

In a fully developed channel flow, friction forces are compensated by a mean pressure gradient. Actually, unlike the velocity, the pressure can not be periodic in the streamwise direction and a mean pressure gradient is needed to drive the flow. In the equation above the pressure gradient term is split into two parts. The first, S_{i3} is the mean pressure gradient which acts in the streamwise direction, third direction in the code as presented in this equation. The other one is the fluctuating pressure gradient which is periodic. The calculation of the fluctuations of pressure is explained here but the calculation of the mean pressure gradient referred to as the *source term* is explained in the next chapter. Both terms are explicited in the equation above and are available from previous time step.

There are three equations and three unknowns which can be solved numerically by inverting a tri-diagonal matrix with the same boundary conditions as the real velocity field. From these equations an intermediate velocity field is found which does not satisfy the continuity equation locally. However, the intermediate velocity field satisfies the continuity equation globally because of the boundary conditions used, periodic in streamwise and spanwise and no-slip or slip in normal direction.

The code is written for $\Delta u_i = u_i^k - {u_i}^n$ quantity so equation (2.66) is recasted in this form which gives

$$\Delta u_i - \beta_k L_{jj} \Delta u_i = (\gamma_k H_i^n + \rho_k H_i^{n-1} - \alpha_k \sigma_1 \frac{\delta}{\delta x_i} p^n)\Delta t + 2\beta_k L_{jj} u_i^n \tag{2.67}$$

where $\beta_k = \frac{\alpha_k \Delta t}{2Re}$.

Now the continuity should be satisfied. To do this we define a scalar quantity Φ as follows

$$\frac{u_i^{n+1} - u_i^k}{\Delta t} = -\alpha_k \sigma_2 \frac{\delta \Phi}{\delta x_i} \tag{2.68}$$

The real velocity field is solenoidal $\frac{\delta u_i^{n+1}}{\delta x_i} = 0$, so if we take the divergence of (2.68) we get

$$L_{jj}\Phi = \frac{1}{\alpha_k \sigma_2 \Delta t} \frac{\delta u_i^k}{\delta x_i} \tag{2.69}$$

The right hand side of the above equation is known and scalar Φ can be calculated. The real velocity field which satisfies the continuity equation is then calculated from (2.68). The equation solving for Φ is a *Poisson* equation and pseudospectral methods are used in the code to solve it. If we determine the pressure at the next time step then the velocity and pressure fields satisfy continuity and momentum equations and we can march a time step. The pressure at the next time step is defined to be

$$p^{n+1} = p^n + \Phi - \beta_k L_{jj} \Phi \tag{2.70}$$

For Adams-Bashfort this process is done once to advance one step in time but in case of Runge-Kutta all this is done three times corresponding to three substeps to advance one time step in time. If we substitute back the intermediate velocity field from (2.68) into (2.66) we get the real equation which has been solved after two successive steps of the fractional step method. This is done for Adams-Bashfort here and for Runge-Kutta it is straightforward. The resulted equation is

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = H_i^{n+\frac{1}{2}} + \frac{1}{2Re} L_{jj} u_i^{n+\frac{1}{2}} - \frac{\delta}{\delta x_i}[\sigma_1 p^n + \sigma_2(\Phi - \beta_k L_{jj}\Phi)] \tag{2.71}$$

The constants σ_1 and σ_2 can be chosen to have the pressure gradient at the desired time and location. For example in the first versions $\sigma_1 = 0$ and the pressure was fully implicit. If there was any necessity to analyze the pressure field, it was calculated from

$$p^{n+1} = \Phi - \beta_k L_{jj}\Phi \tag{2.72}$$

As explained before, in the version used here, the pressure at the next time step is evaluated from (2.70), so the actual pressure gradient appearing in (2.66) is

$$-\frac{\delta}{\delta x_i}[p^n(\sigma_1 - \sigma_2) + \sigma_2 p^{n+1}] \tag{2.73}$$

If $\sigma_1 = \sigma_2 = 1$ is chosen then the pressure gradient is evaluated at $n+1$; on the other hand for $\sigma_1 = 1$ and $\sigma_2 = \frac{1}{2}$ the pressure gradient is evaluated at $n + \frac{1}{2}$. The accuracy is different in the two cases; when the term is evaluated at $n+1$ the accuracy is of first order in time and when evaluated at $n+\frac{1}{2}$ it is second order accurate in time. A check has been done and it has been found that the reduced accuracy gives a more stable scheme which is more appropriate in time dependent simulations where stability is an important issue. In this code the first choice is made.

2.4.1 Factorization

The successive two steps of fractional step method allow the solution of the Navier-Stokes equations. For three dimensional flows we need to solve the equations with a fast and, less important, low-storage method. The implicit treatment of the linear term helps to reach a fast solution but the resulting matrix on the left hand side of (2.67) has a large band and hence can not be directly inverted. The key to the problem is factorization approximation. Factorization is approximating the left hand side matrix of (2.67) with the product of three other matrices. This can be done within the accuracy of the schemes as it is discussed in Moin (2001).

Chapter 3

Code description

In this chapter the code presented in the book to solve the channel flow using DNS is described. The schemes used are the ones explained in the previous chapter. There are some issues related to the initial conditions and Reynolds number which are explained here. Here it is explained how to run the code and how the results are saved. There are additional comments added to the code which try to make the code understandable.

3.1 General description

The code is written in Fortran and consists of some 60 subroutines in about 4400 lines. This code solves the fully developed channel flow in a computational box at node numbers given by the user. The code solves for a Reynolds number based on half channel width and laminar centerline Poiseuille flow. The grid transformation technique is used to go from a non-uniform grid in normal direction to a uniform grid. The Crank-Nicholson scheme is used for the linear term and Adams-Bashfort and Runge-Kutta schemes can be used alternatively for the non-linear term. The fractional step method leads to a Poisson equation for a scalar which gives the pressure field. This pressure is used to correct the velocity field so that continuity is satisfied. The Poisson equation is solved with pseudospectral methods whereas factorization is used to solve the momentum equation. The flow is periodic in streamwise and spanwise directions and no-slip or slip conditions can be applied to the walls alternatively. There is the possibility to have moving walls and regions of transpiration on the walls which are not covered in this report; only the way to avoid their effect is addressed here. In the code the direction referred to as 1 is spanwise, 2 is normal and 3 is streamwise.

3.2 Getting started

The CD-ROM contains a compressed file: *diskette.tar.gz*, the decompression of this file leads to a directory: *diskette*, containing 12 directories of which *CODESCH9* and *NCARFFT* are used to run the channel flow. Directory *NCARFFT* contains a bunch of Fast Fourier Transform routines used in the code to solve the Poisson equation for the scalar Φ. This is not covered in this report more since it is part of the solver. Directory

CODESCH9 contains five subdirectories for running the channel flow and post processing the results. The files of the code are in *CHA* subdirectory and are following files: *chadsma.f*, *chadshn.f*, *chadsnn.f*, *chadstn.f*, *chadstinv.f*, *chadsphnc.f*, *chadsou.f*, *chadsin.f*, *chadstr.f*, *chadsru.f* and *parameter.f* and *parameter.f*. The code can be run without any other file in this directory. There are of course some make-files which can be used to run the code of which the file *chadsnc.mak* can be used for the available FFT routines. The make-files are not necessary and the commands to run the code are given here. In addition to the files explained, files *chapn.d* and *nfchcapn* given in the subroutine *prova* are necessary too. These are input files which are explained later.

To run the code first a modification is needed in the file *chadsou.f*, the following line should be added to the subroutine *outpf*

```
real :: nsp1,nsp2,nsp3
```

This is because Fortran takes the variables starting with `n` as integers but the variables nsp1, nsp2 and nsp3 are real numbers and they should be defined explicitly with the above command. After this modification we can compile the code, with the following command

```
pgf90 -c chadsma.f chadshn.f chadsnn.f chadstn.f chadstinv.f chadsphnc.f
chadsou.f chadsin.f chadstr.f chadsru.f .../NCARFFT/ffts.f
```

This command compiles the above files and makes object files with same names. Then we make an executable file with command

```
pgf90 -o chadsnc chadsma.o chadshn.o chadsnn.o chadstn.o chadstinv.o
chadsphnc.o chadsou.o chadsin.o chadstr.o chadsru.o .../NCARFFT/ffts.o
```

Now there is an executable file *chadsnc* which solves for the flow. This files needs two input files mentioned before to be run correctly. These two files and the file parameter.f are explained below.

3.2.1 Input files

parameter.f

This is probably the most unimportant file, it only declares five constants *m1,m2,m3,ndv* and *ndd*. The variable *ndd* is never used in the code, the variable *ndv* is the dimensions of the flow which is three in this case. The other variables are the size of arrays in three directions. These should be at least twice as large as the real mesh faces in the spanwise and streamwise directions to avoid any problems in case of mesh refining. The mesh refining process will be explained later.

nfchcapn

This file contains file names in which some of the results are written. As will be explained the best way to save the results is to save it in your own way.

chapn.d

This is the important input file which declares the case which we want to run. There are some variables which are given value in this file, an example of this file is given below

```
33     151  65   3                n1  n2   n3  nsst
1      0      0   1                nwrit   nread   iav   iprfi
4.0      .5     2.80               alx3  alx1   str2
4200.   1.5E-01                    Re  vper
.08  24001    2000   1    10       dt   ntst   nprint  npin  npstf
11                                 nstop
1    1.7  .2   10.    380.         icfl   cflc    tpin    tprin   tfin
5   37  4   0    0                 jri  jrf   djr   irejr   iruuca
220.                               timeav
0    0    0    0                   islv1s,islv1n,islv3s,islv3n
1.        0.                       tosc  uosc
.00   10.    0.                    flowq2     tau2    t0sl    t0sl
.5   0.  0.6    0.4                y1gsd    y1ssd     y3gsd    y3ssd
0.      1.                         y1disd   y3disd
1                                  ifield
```

Some of these variable are not important in calculations. The important variables are

- **n1** is the number of mesh faces in spanwise direction, direction one.
- **n2** is the number of mesh faces in normal direction, direction two.
- **n3** is the number of mesh faces in streamwise direction, direction three.
- **nsst** declares the scheme used for the non-linear term, $nsst = 1$ implies Adams-Bashfort and $nsst = 3$ implies Runge-Kutta.
- **nwrit** declares if the results should be written in files.
- **nread** declares if the the calculations should start from scratch or a continuation field is read and continued.
- **iav** declares if averaging the instantaneous fields should be done during calculations.
- **iprfi** declares if any instantaneous fields should be saved on file.
- **a1x3** is the size of computational box in streamwise direction divided by π.
- **a1x1** is the size of computational box in spanwise direction divided by π.
- **str2** is the constant coefficient in grid transformation function which will be explained in this chapter.
- **Re** is the Reynolds number of the flow.
- **vper** is the perturbation velocity given as part of the initial conditions when the calculations are to start from scratch.
- **dt** is the time step size when the constant time step mode is chosen.

- **icfl** declares if the calculations should be done in constant time step mode or constant CFL number mode.
- **cflc** is the CFL number for constant CFL number calculations.
- **tpin** declares the time between two successive fields being averaged.
- **tprin** declares the time between two successive fields being saved on file.
- **tfin** declares when to stop calculations.
- **timeav** declares when to start averaging the instantaneous fields.
- **islv1s** declares the velocity slip condition in spanwise direction on the lower wall, one stands for free-slip and zero for no-slip.
- **islv1n** declares the velocity slip condition in spanwise direction on the upper wall.
- **islv3s** declares the velocity slip condition in streamwise direction on the lower wall.
- **islv1n** declares the velocity slip condition in streamwise direction on the upper wall.
- **ifield** declares if the velocities at the wall should be saved on file too.

Four variables (lines 5 and 6) *ntst, nprint, npin* and *nstop* are the counterparts of *timeav, tprin, tpin* and *tfin* for the cases run in constant time step size mode. The variable *npstf* is only used once in the code to define another variable which is never used again, so this variable has no effect in the code and is most probably a remainder of some modifications in the code. Variables *jri, jrf, djr, irejr* and *iruuca* (line 8) are the variables declaring if correlations should be computed inside the code and in which points to be calculated. They are not of importance as we want to calculate and save what we want and in our own format.
Variables *tosc, uosc, flowq2, tau2, t0sl, t0sl, y1gsd, y1ssd, y3gsd, y3ssd, y1disd* and *y3disd* (lines 11 14) are relate to declaration of the wall movements and transpiration slots which are not the issue of the report. To eliminate the effect of them in computing the channel flow it is enough to set *flowq2* to zero.

The size of the computational box in normal direction is $2\delta = 2$. In the beginning of the code there are three variables defined as $n1m = n1 - 1$, $n2m = n2 - 1$ and $n3m = n3 - 1$ which are the number of cells in each direction.

3.2.2 Output files

There are many files in the code where data for post processing are saved, but the best way is to write the necessary data in our own format and in our own files. The best way is to save a number of statistically independent instantaneous fields and to calculate all statistical quantities from these data. This is done in the code and the instantaneous fields are saved with the names *fieldN.dat* where N is a number equal to $n.tpin$ at the time $n.tpin$. This saving process starts at *timeav* and ends at *tfin*. Any of these fields can be used as a continuation field. At times before *timeav* the fields at times $n.tpin$ are written

in a file named *cha.wri*. The original code has some problems in reading the restart field in. This is modified and whenever a restart field is to be fed as the initial field it should be renamed to *cha.wri*(in the original code it is supposed to read the file *cha.rea*). The best way to calculate averages or other quantities is to add them after the subroutine *tschem* in file *chadstn.f*, which is the subroutine to advance one step in time.

3.3 Grid transformation

As explained in the previous chapter a grid transformation is used in the code. The analytical form of the function is

$$y = \frac{a.tanh(\xi - 0.5)}{tanh(\frac{a}{2})} \tag{3.1}$$

where a is a coefficient which gives the ability to make the mesh finer near the wall, the higher the a, the finer the mesh near the wall. Figure (3.1) shows this function for two values of a which are later used to run two test cases.

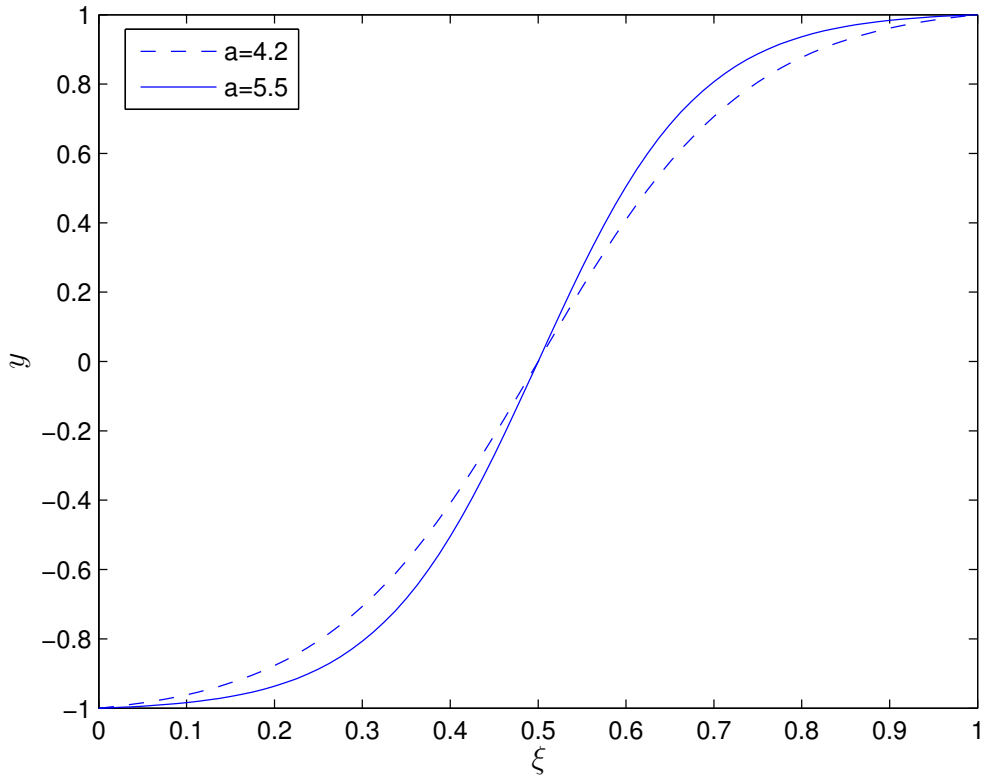

Figure 3.1: grid transformation function for a=5.5 and a=4.2

The physical space which is from -1 to 1 is mapped onto a computational domain from zero to 1 by this function. The computational box is divided into cells in subroutine *meshes* in the file *chadsnn.f*. In subroutine *indic* in file *chadsnn.f* some variables are defined which make it easy to apply the boundary conditions. Some comments are added in the code

which make this more clear. In subroutine *coordi* in file *chadsnn.f* the physical coordinate is calculated from the grid made in subroutine *meshes* by using the above analytical function. In subroutine *metric* in file *chadsnn.f* two arrays are defined: *caj(1-n2m)* which is $\frac{\partial y}{\partial \xi}$ at the cell centers($j+\frac{1}{2}$), and *cac(1-n2)* which is $\frac{\partial y}{\partial \xi}$ at the faces. These metrics make it possible to calculate the second derivatives and to discretize the equations in space.

3.4 The flow to solve

It was mentioned before that the code is to solve the fully developed channel flow with two homogeneous and one inhomogeneous directions. Also, it was mentioned that the flow is specified by a Reynolds number based on half channel width and laminar Poiseuille centerline velocity, Re_p. In this section it is explained what this choice of Reynolds number implies. The arguments in this section are the basis for the modification explained in the next chapter.

3.4.1 Laminar Poiseuille flow

Laminar Poiseuille flow is a steady one-dimensional flow between two planes. The resulting flow is a laminar parabolic velocity profile. The governing equation reads

$$-\frac{\partial p}{\partial x} + \mu \frac{\partial^2 u}{\partial y^2} = 0 \tag{3.2}$$

where μ is the viscosity. In non-dimensional form the governing equation reads

$$-\frac{\partial p}{\partial x} + \frac{1}{Re}\frac{\partial^2 u}{\partial y^2} = 0 \tag{3.3}$$

This is one equation for two unknowns. If we consider a constant pressure drop along the flow direction then we get

$$-\frac{\Delta p}{\Delta x} + \frac{1}{Re}\frac{\partial^2 u}{\partial y^2} = 0 \tag{3.4}$$

Solving this ordinary differential equation with no-slip boundary conditions, $u(y=-\delta) = 0$ and $u(y=\delta) = 0$ leads to

$$u(y) = \frac{1}{2Re}\frac{\Delta p}{\Delta x}(y^2 - \delta^2) \tag{3.5}$$

If Re in this equation is considered to be Re_p, Reynolds number based on laminar Poiseuille centerline velocity, then $\delta = 1$ and $u(0) = 1$ leads to

$$\frac{\Delta p}{\Delta x} = -\frac{2}{Re} \tag{3.6}$$

3.4.2 Initial fields

As noted before, the calculations can start either from scratch or as continuation of an available field. When the calculations start from scratch we need to specify an initial field. The initial field here is the aforementioned laminar flow. To trigger turbulence, a random disturbance is added to this flow. This random disturbance is zero-mean and the reason behind this will be explained soon. The initial velocity field is given in subroutine *inqpr* in file *chadsin.f*, which is called within subroutine *initia* in the file *chadsin.f*. The initial velocity field is

$$u_{3,initial} = (1 - y^2) \tag{3.7}$$

while the other components of velocity are given zero mean values. The disturbances added in three directions are of the order of *vper* given in the input file *chapn.d*. To avoid large fluctuations near the wall that could require small time steps for CFL restrictions, the random disturbances in the region near the boundary is multiplied by 0.2. The initial velocity field formed in this way is not solenoidal and does not satisfy the continuity equation, but this is not important since the fractional step method has the property to make it solenoidal at the first time step. The initial pressure fluctuations are set to zero in *initia* subroutine and a value of $\frac{2}{Re}$ is given to the mean pressure gradient, the source term which is called *dp3ns* in the code, to make the initial pressure field consistent with that of velocity. It will soon be obvious that this is a very important point.

The other way is to start the calculations from an available field. This is done in subroutine *inirea* in file *chadsou.f*. The file should have the name *cha.wri* and should have been written in the same format as it is read, the original format of the code is kept in the modified code. This velocity and pressure fields should be scaled first. The velocity field should be scaled in such a way that the mass flow rate of the scaled field is equal to the mass flow rate of the laminar Poiseuille profile given in (3.7). The pressure should be scaled with square of the scaling factor for velocity. This is a very important point and missing it leads to calculation of a flow with a Reynolds number other than the one specified in *chapn.d* file.

When the calculations start from scratch it is better to start with a coarse mesh and after some time when the flow is rather fully developed make the mesh finer. The continuation property makes this easy. In file *chadsou.f* there are interpolation and extrapolation subroutines that make it possible to build velocity and pressure fields from available fields at meshes which are two times finer or coarser than the mesh used for the available field from a previous run. These subroutines are called in the subroutine *inirea* if they are needed. This is done automatically since the number of mesh points in the available field is written at the beginning of it and comparison to the mesh points given in *chapn.d* shows in which directions the field should be interpolated or extrapolated. These subroutines are only for streamwise and spanwise directions since it is better not to run for meshes which are coarse in the normal direction. In other words, it is better to keep the mesh fine in normal direction from the beginning. If this kind of continuation fields are used there is no need to scale it because they have the same mass flow rate as laminar flow of (3.7).

3.5 Mean pressure gradient

Discretization of all the terms in Navier-Stokes equations has been described in sections 2.2 and 2.3 except for the source term. When explaining the fractional step method we took it as to be known and here it is explained how to find it. In the code, the source term is set equal to viscous forces at each time step

$$s_{33} = -\frac{1}{Vol.}\int_x\int_y\int_z \frac{1}{Re}\frac{\partial^2 u_3}{\partial x_j^2} \tag{3.8}$$

This quantity is used explicitly in the first step of the fractional step method. This choice of mean pressure gradient leads to a very interesting property, since by this choice the mass flow rate is kept constant which means that the time rate of change of mass flow rate is zero. This can be proved if we take a look at the mass flow rate relation

$$Q = \sum_{i=1}^{n1m}\sum_{j=1}^{n2m} u_{3,j}\Delta y_j \Delta x_i \tag{3.9}$$

where Q represents the mass flow rate and direction of x is the spanwise direction.
Now let's integrate the Navier-Stokes momentum equation in streamwise direction, which leads to the following relation for each time step

$$\int\int\int \frac{\partial u_3}{\partial t}\Delta x\Delta y\Delta z + \int\int\int \frac{\partial u_3 u_j}{\partial x_j}\Delta x\Delta y\Delta z =$$

$$-\int\int\int \frac{\partial p}{\partial x_3}\Delta x\Delta y\Delta z + \int\int\int \frac{\partial^2 u_3}{\partial x_j^2}\Delta x\Delta y\Delta z + \int\int\int s_{33}\Delta x\Delta y\Delta z \tag{3.10}$$

The non-linear term is zero due to the boundary conditions. The fluctuating pressure term is zero due to periodicity in the third direction. The final equation is

$$\frac{\partial}{\partial t}\int\int\int u_3\Delta x\Delta y\Delta z = \int\int\int \frac{\partial^2 u_3}{\partial x_j^2}\Delta x\Delta y\Delta z + s_{33}\int\int\int \Delta x\Delta y\Delta z \tag{3.11}$$

If at each time step we set

$$\int\int\int \frac{\partial^2 u_3}{\partial x_j^2}\Delta x\Delta y\Delta z + s_{33}\int\int\int \Delta x\Delta y\Delta z = 0 \tag{3.12}$$

Then we get

$$\frac{\partial Q}{\partial t} = 0 \tag{3.13}$$

This means that the mass flow rate is constant and does not change. This is the reason behind scaling the available fields first to use as initial conditions. The calculation of the pressure gradient is done in subroutine *invtr3* in file *chadstn.f*.

Chapter 4

Modifications and Results

The code described in the previous chapter is modified to solve flows with specified Reynolds numbers based on friction velocity. These modifications and their effect are described in this chapter. Two test cases are run with the code and the results are compared with two sets of data obtained from other codes.

4.1 Modification

The fully developed channel flow can be run for two different situations. The first preserves the mass flow rate as it is done in the code described, referred to as *Original code* from now on, and the second preserves the pressure force. When preserving the mass flow rate it is usual to specify the channel flow with its Reynolds number based on half channel width and *bulk velocity* defined as

$$u_b = \frac{1}{2\delta}\int_{-\delta}^{+\delta} u dy \tag{4.1}$$

$$Re_b = \frac{u_b \delta}{\nu} \tag{4.2}$$

On the other hand, when preserving the pressure force, the flow is specified by the Reynolds number based on half channel width and *friction velocity* defined as

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}} \tag{4.3}$$

$$Re_\tau = \frac{u_\tau \delta}{\nu} \tag{4.4}$$

The flows corresponding to these two cases are different. The first flow is a constant mass flow rate flow with fluctuating mean pressure gradient as source term while the second flow is a constant force flow with fluctuating mass flow rate.

The original code, as described before, solves the first flow. This code is modified to solve for the second flow. To do this the mean pressure gradient term is replaced by constant number $dp3ns = 1$ and a few more simple modifications are made to be consistent with this change. These simple modifications are described inside the code as comments.

By preserving the source term the Reynolds number appearing in the input file should be set as Re_τ.

4.1.1 Reynolds numbers

There are three different Reynolds numbers appearing in this report, and here the relations between them are explained. To find a relation between Re_b and Re_p we calculate the mass flow rate of Poiseuille flow first. Writing the Poiseuille flow in terms of its centerline velocity leads to

$$u_p(y) = u_p(0)(1 - (\frac{y}{\delta})^2) = u_p(1 - (\frac{y}{\delta})^2) \tag{4.5}$$

$$Q_p = \int_x \int_y u_p(y) dy dx = \int_0^{l_x} \int_{-\delta}^{+\delta} u_p(1 - (\frac{y}{\delta})^2) = \frac{4}{3} u_p \delta l_x \tag{4.6}$$

where x is in the spanwise direction. Comparison of bulk and Poiseuille centerline velocities gives

$$u_p = \frac{3}{2} u_b \tag{4.7}$$

and this leads to

$$Re_p = \frac{3}{2} Re_b \tag{4.8}$$

This implies that if we want to run the code with any Re_b we should run the original code with corresponding Re_p given by the above relation.

There are no direct relations between Re_τ and the others, but it is worth to mention that to feed the modified code with available fields no scaling is necessary except those helping the profile to reach the expected final values sooner.

4.1.2 Mean velocity profiles

In the previous section it was mentioned that the modified code solves a rather different flow than the original one. Here it will be shown that in spite of this difference the mean velocity profiles should be the same. Consider the equation for the mean velocity, which is obtained by averaging instantaneous velocity field in homogeneous directions and time

$$< U_3 > (j) = \int_t \int_x \int_z u_3(i, j, k, t) dz dx dt \tag{4.9}$$

The equation for this quantity is found by integrating the Navier-Stokes in streamwise direction (z direction due to the code notation), spanwise direction and time

$$\int_t \int_x \int_z \frac{\partial u_3}{\partial t} dz dx dt + \int_t \int_x \int_z \frac{\partial u_3 u_j}{\partial x_j} dz dx dt = - \int_t \int_x \int_z \frac{\partial p}{\partial x_i} dz dx dt + \int_t \int_x \int_z \frac{1}{Re} \frac{\partial^2 u_3}{\partial x_j^2} dz dx dt$$
$$+ \int_t \int_x \int_z s_{33} dz dx dt \tag{4.10}$$

The first term is the time rate of the mean streamwise velocity and is zero due to fully developed flow condition. The second term is zero in x and z directions due to periodicity in these directions. The fluctuating pressure is zero due to periodicity in streamwise direction. Also, the linear term is zero in x and z directions due to periodicity. The remaining terms are

$$\int_t \int_x \int_z \frac{\partial u_3 u_2}{\partial x_2} dz dx dt = \int_t \int_x \int_z \frac{1}{Re} \frac{\partial^2 u_3}{\partial x_2^2} dz dx dt + \int_t \int_x \int_z s_{33} dz dx dt \tag{4.11}$$

If equation (4.11) is scaled with

$$SF = \frac{\int_t \int_x \int_z s_{33} dz dx dt}{Vol.} \tag{4.12}$$

Then the resulting equation is

$$\frac{\int_t \int_x \int_z \frac{\partial u_3 u_2}{\partial x_2} dz dx dt}{SF} = \frac{\int_t \int_x \int_z \frac{1}{Re} \frac{\partial^2 u_3}{\partial x_2^2} dz dx dt}{SF} + Vol. \tag{4.13}$$

Comparing this equation with that written for constant force flow we find that the equations are exactly the same. This means that, as far as the mean velocity is concerned, the flows are exactly the same.

4.2 Results

4.2.1 Test case 1: $Re_\tau = 500$

The first case is the fully developed channel flow for a Reynolds number of 500 based on friction velocity. This case is run with the modified code and in a computational box of 0.5π in spanwise, 2 in normal and 4π in streamwise directions. The mesh is $64 \times 80 \times 64$ and the grid transformation coefficient is $a = 5.5$. This grid gives a maximum stretching factor of 14.6% and gives the first cell center at $y^+ = 0.301$. The Runge-Kutta scheme is used for the non-linear term and the calculations are done with constant CFL number. The $CFL = 1$ is chosen for the sake of accuracy though it could have been chosen up to 1.7 considering stability. There are statistically independent fields saved on disk and the statistical quantities are calculated from these data.

After running for this case the bulk velocity is computed for the resulting flow and Re_b and consequently Re_p are calculated. The original code is run for this resulting Re_p on the same mesh and CFL conditions. The resulting flow from original code corresponds to $Re_\tau = 498$ which is within the numerical accuracy of the target value 500. The results show that the mean velocities are the same for the original and modified cases as was predicted. The results are shown in figures (4.1) to (4.6) and are compared to a set of data referred to as *DNS2006* in the figures. These data are from Davidson (2006) corresponding to $Re_\tau = 500$ on exactly the same mesh as the test case. The code giving the *DNS2006* data is a finite volume code while grid transformation is not implemented and the non-uniform mesh is generated at first.

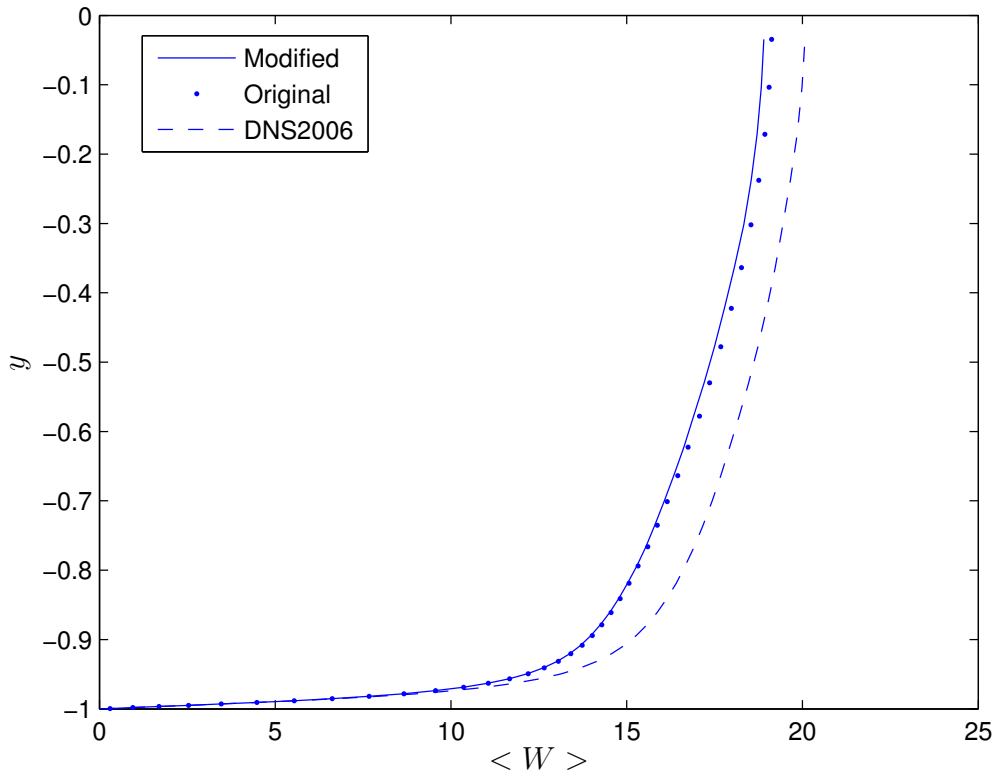

Figure 4.1: streamwise mean velocity, $Re_\tau = 500$

As it is seen from the results, the near wall behavior is rather the same for all codes but toward the middle of the channel they begin to deviate. The mean velocity is about 6% less for the code described here compared to that of *DNS2006*. The flow resulted from this code is less turbulent than the *DNS2006* data. The normal stresses and turbulence kinetic energies are rather equal for the original and modified codes.

4.2.2 Test case 2: $Re_\tau = 180$

The second case is run on the same computational box and same cell numbers but for Reynolds 180 based on friction velocity. The coefficient of transformation function is $a = 4.2$ in this case, which results in a maximum stretching factor of 10.6% and the first node is at $y^+ = 0.298$. The results are shown in figures (4.7) to (4.12). In addition to *DNS2006* data, there are data referred to as *DNS1999*, these are data from MOSER *et al.* (1999). The results show the same behavior for mean velocity but the second order statistics for modified and original codes are more deviated in this case. As for the previous case the mean velocities coincide with other data near the wall but deviate toward mid-channel. The flow resulted from this code is less turbulent than the other codes.

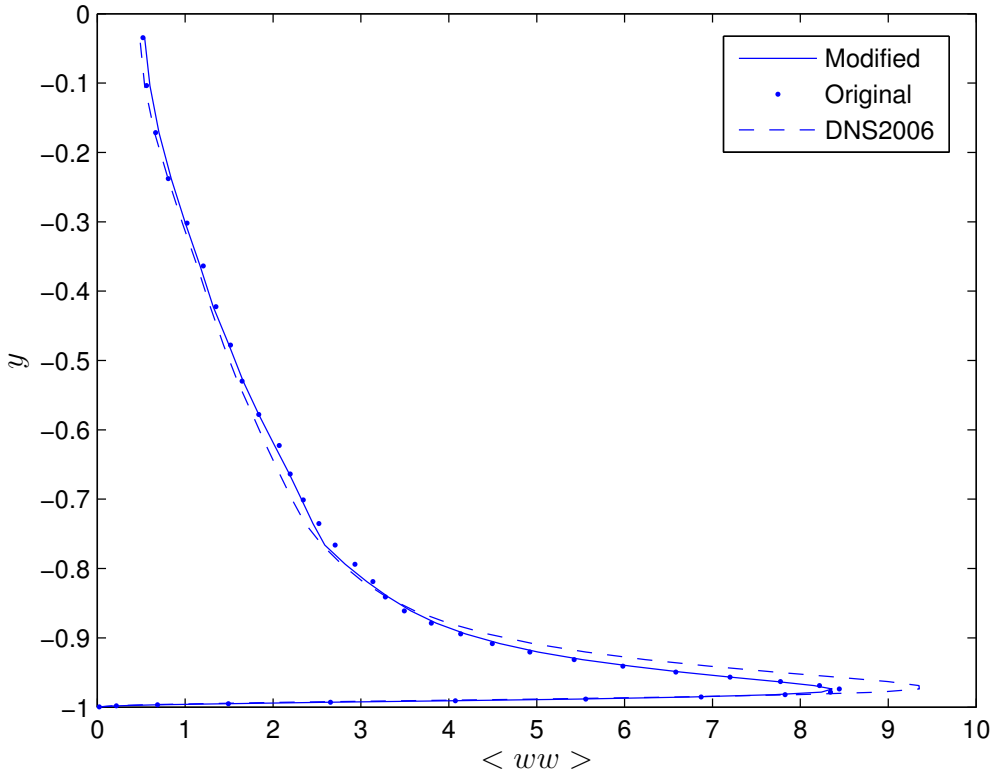

Figure 4.2: normal Reynolds stress in streamwise direction, $Re_\tau = 500$

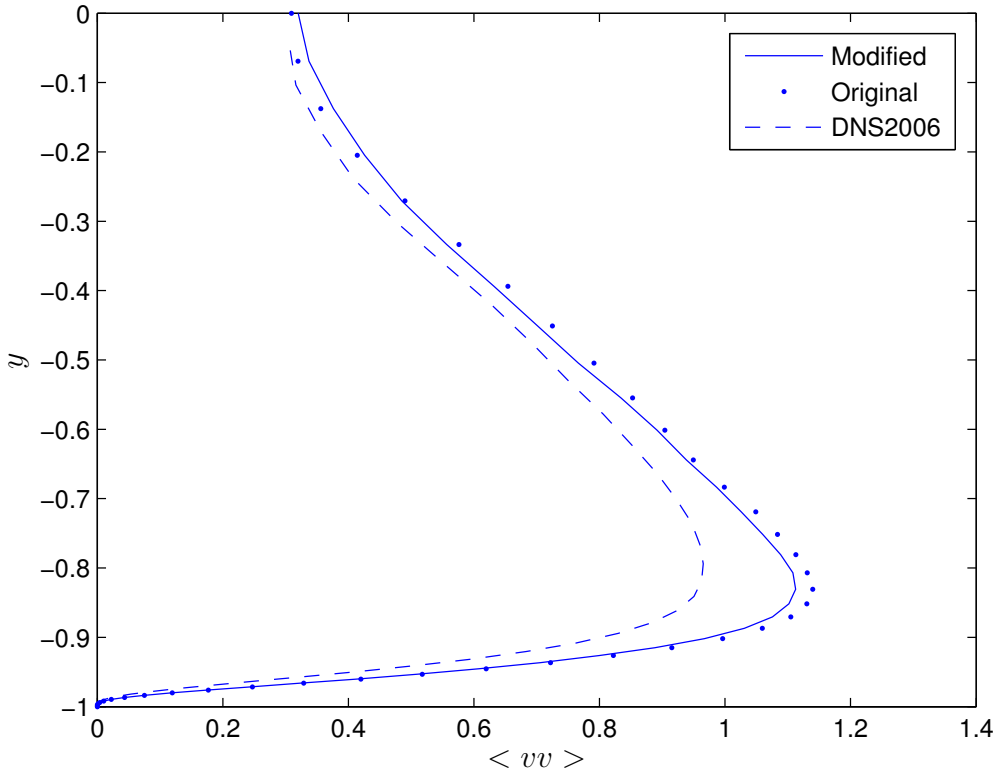

Figure 4.3: normal Reynolds stress in normal direction, $Re_\tau = 500$

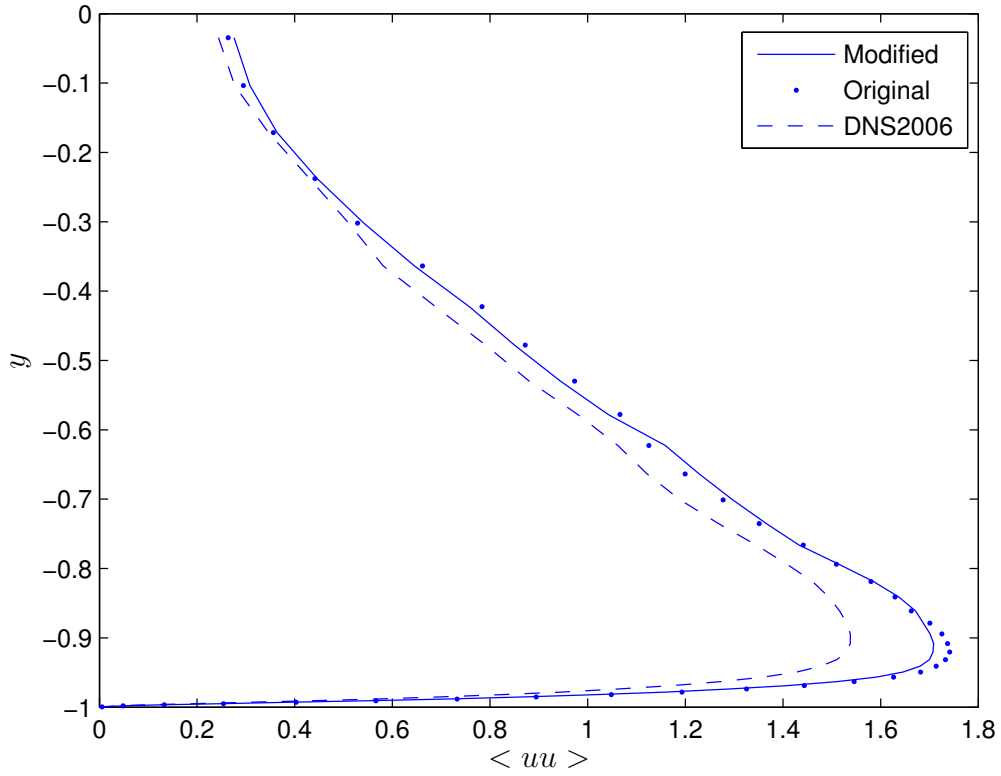

Figure 4.4: normal Reynolds stress in spanwise direction, $Re_\tau = 500$

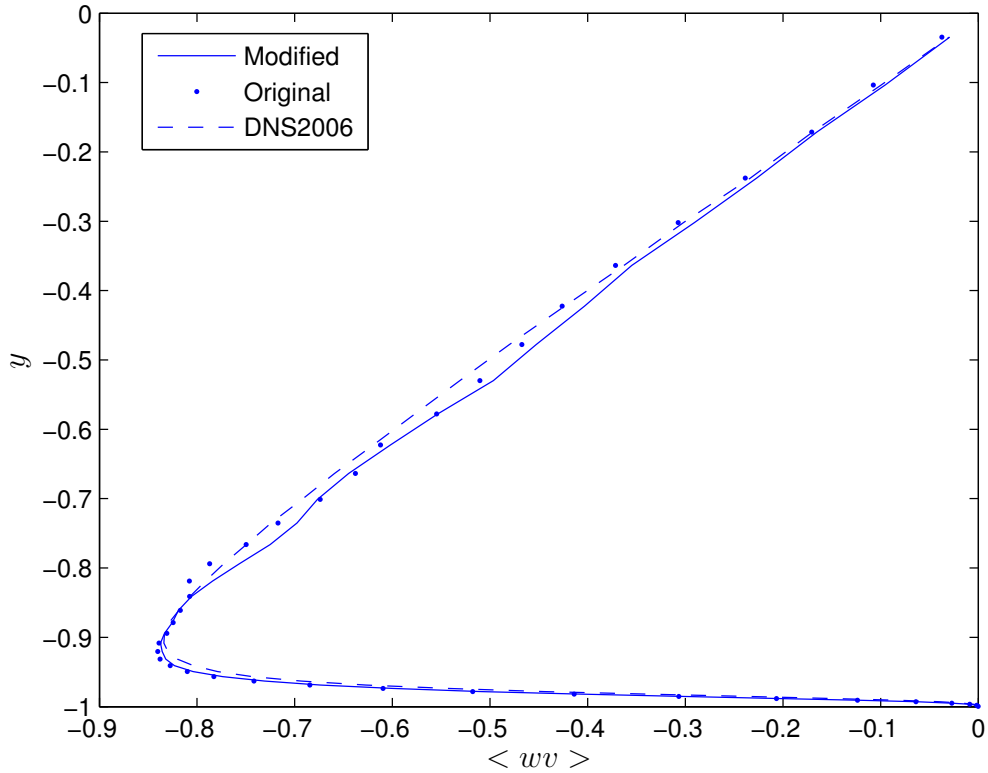

Figure 4.5: Reynolds shear stress, $Re_\tau = 500$

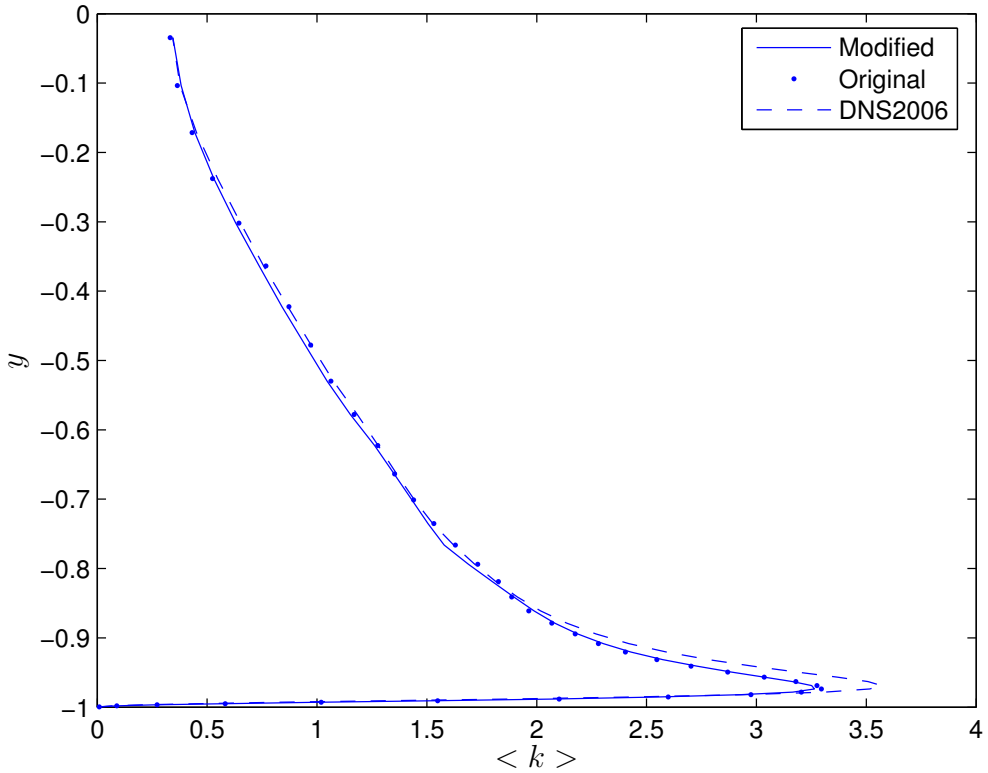

Figure 4.6: turbulent kinetic energy, $Re_\tau = 500$

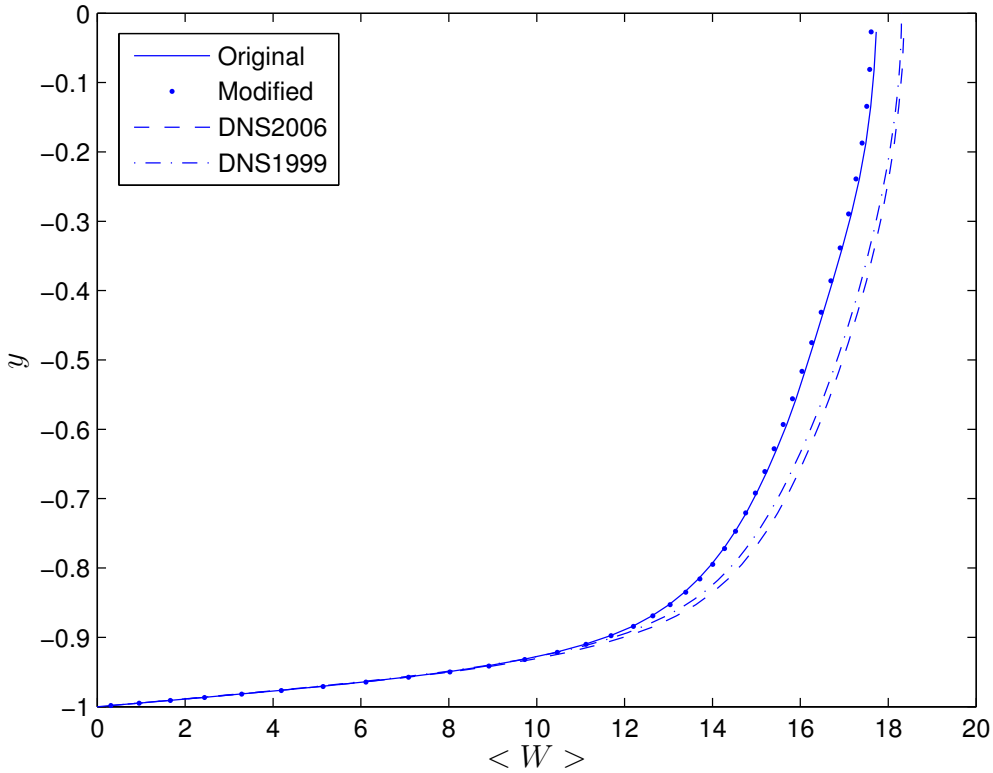

Figure 4.7: streamwise mean velocity, $Re_\tau = 180$

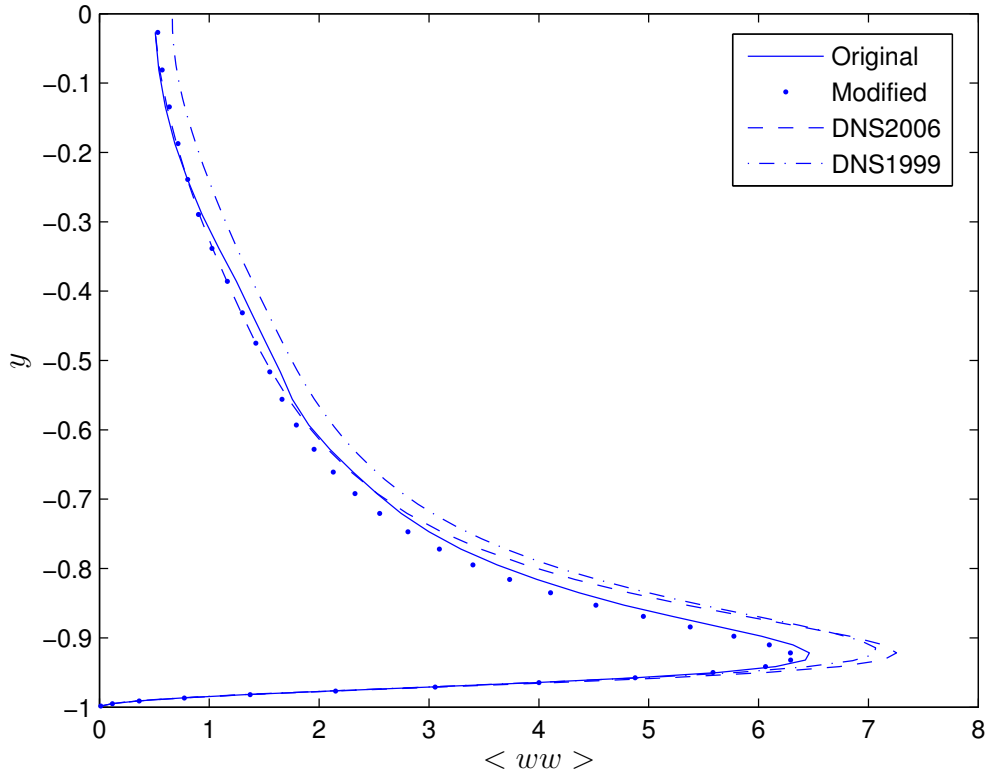

Figure 4.8: normal Reynolds stress in streamwise direction, $Re_\tau = 180$

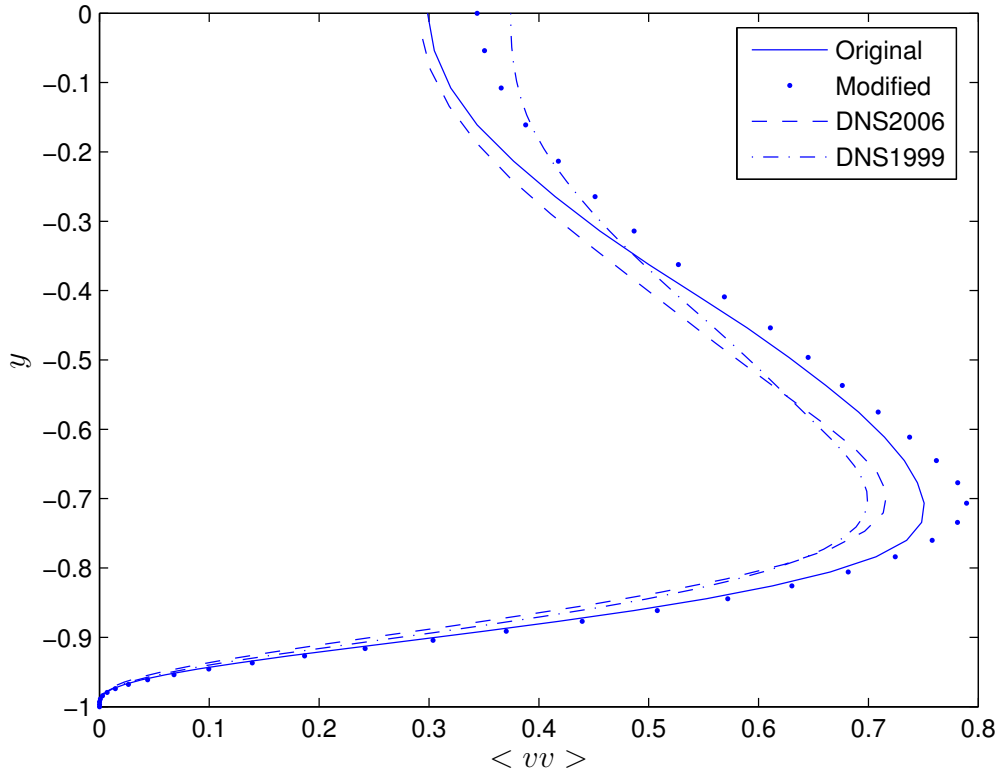

Figure 4.9: normal Reynolds stress in normal direction, $Re_\tau = 180$

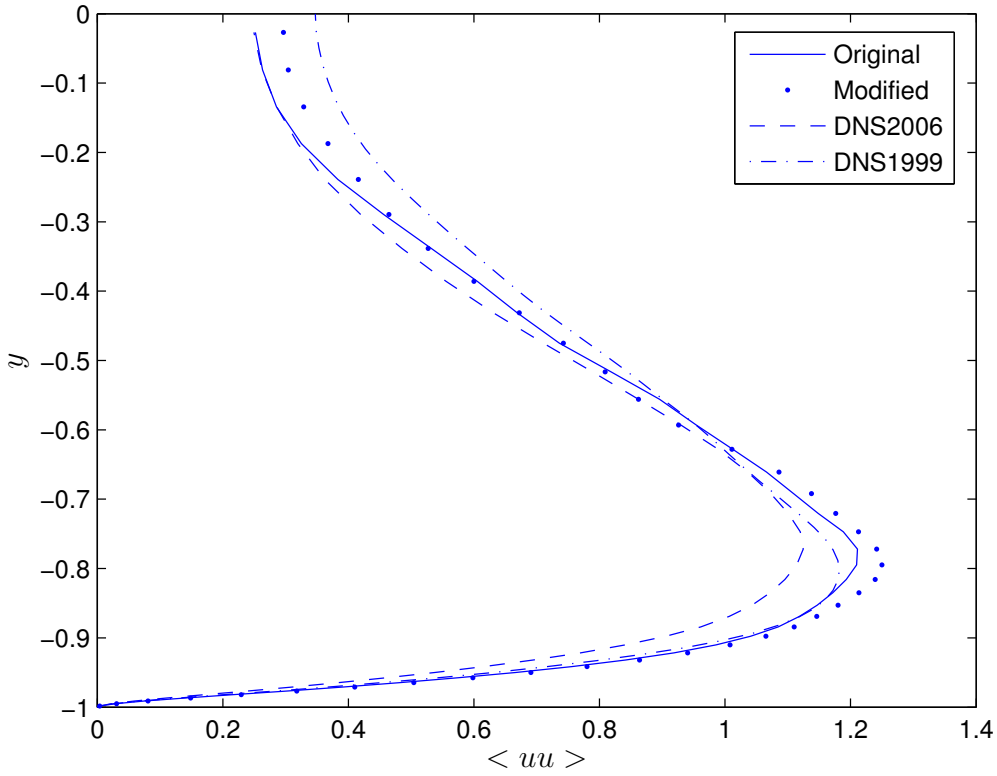

Figure 4.10: normal Reynolds stress in spanwise direction, $Re_\tau = 180$

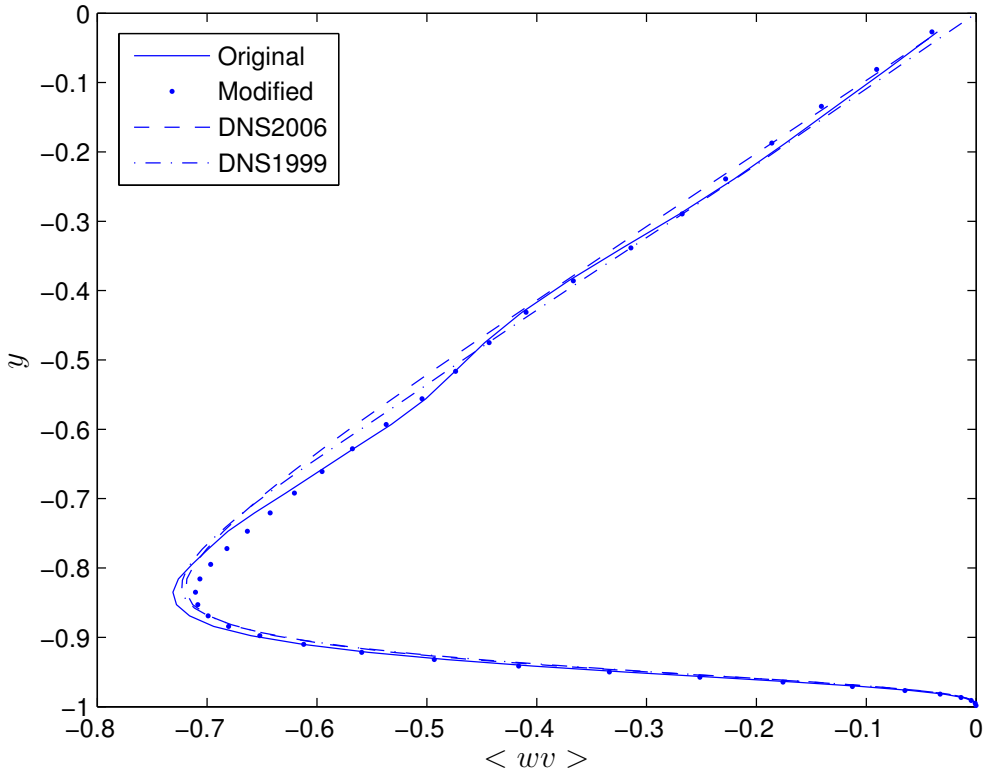

Figure 4.11: Reynolds shear stress, $Re_\tau = 180$

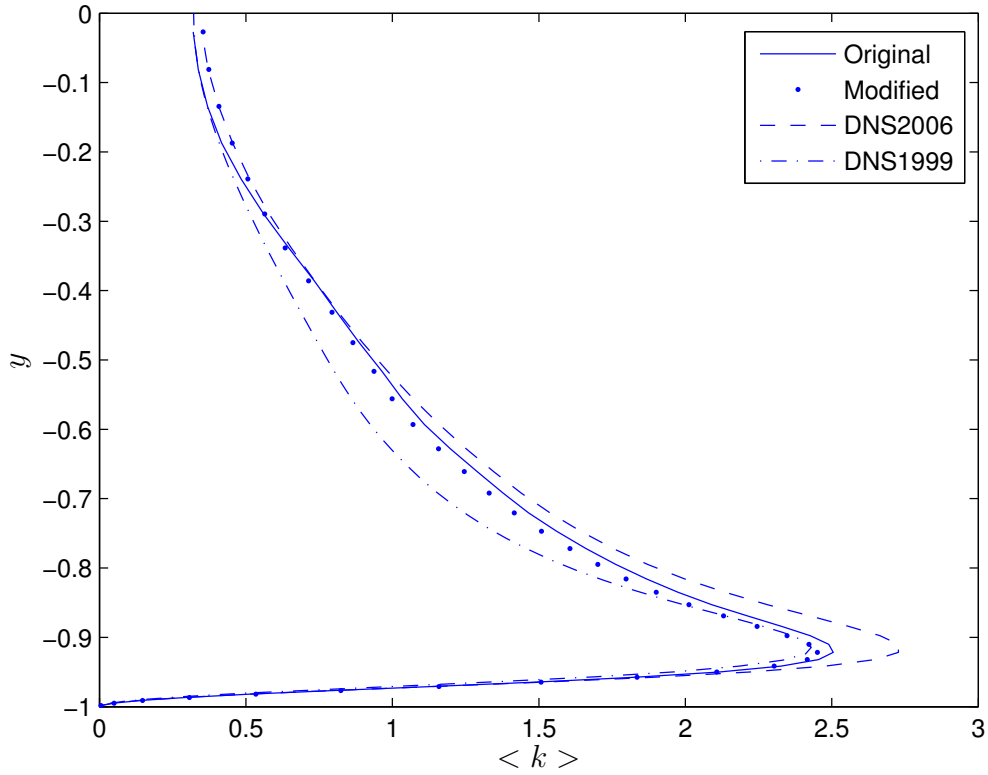

Figure 4.12: turbulent kinetic energy, $Re_\tau = 180$

Chapter 5

Conclusions

- while using the Adams-Bashfort scheme, the calculations should be run at constant time step size mode.
- while using Runge-Kutta scheme, the calculations can be run in both constant time step size and constant CFL number modes.
- the original code solves fully developed channel flow with constant mass flow rate.
- the modified code solves fully developed channel flow with constant pressure drop.
- mean velocity predictions of the original and modified codes are the same for corresponding Reynolds numbers.
- second order statistics become the same for original and modified codes only at high Reynolds numbers.
- the flow resulted from this code coincides with that of DNS1999 and DNS2006 near the wall but deviates toward the mid-channel.
- the flow resulted from this code is less turbulent than that of DNS2006 and DNS1999.

Chapter 6

Future work

The code is capable of solving the channel flow with moving walls and walls with regions of transpiration. This can be analyzed more inside the code and cases relating to drag control can be run with the same code and with minimum modifications. Also, it is possible to have modifications in the code to run for totally new cases, examples can be vertical channel flow, horizontal channel with walls at different temperatures, or even vertical channel with buoyancy effects.

Bibliography

Chorin, A. J. 1968 Numerical solution of the Navier-Stokes equations. *MATHEMATICS OF COMPUTATION* **22** (104), 745–765.

Davidson, L. 2006 DNS of channel flow with finite volume method. *Private Communications* .

KIM, J. & MOAN, P. 1985 Application of a fractional-step method to incompressible Navier-Stokes equations. *J. Comp. Phys.* **59**, 308–323.

KIM, J., MOIN, P. & MOSER, R. 1987 Turbulence statistics in fully developed channel flow at low Reynolds number. *J. Fluid Mech.* **177**, 133–166.

Moin, P. 2001 *FUNDAMENTALS OF ENGINEERING NUMERICAL ANALYSIS*. Cambridge University Press, Cambridge, UK.

Moin, P. & Mahesh, K. 1998 Direct Numerical Simulation: A tool in turbulence research. *Annual Review of Fluid Mechanics* **30**, 539–578.

MOSER, R., KIM, J. & MANSOUR, N. 1999 DNS of turbulent channel flow up to Re=590. *Physics of Fluids* **11**, 943–945.

Orlandi, P. 2001 *Fluid Flow Phenomena A Numerical Toolkit*. KLUWER ACADEMIC PUBLISHERS.

S.Rogallo, R. & Moin, P. 1984 Numerical simulation of turbulent flows. *Annual Review of Fluid Mechanics* **16**, 99–137.

VERSTEEG, H. K. & MALALASEKERA, W. 1995 *An Introduction to Computational Fluid Dynamics The Finite Volume Method*. Longman Group Ltd.

Appendix A

Procedures

In this appendix are presented some very short comments on the procedures used in the code.

- **chadshn.f**
  - **hdnl1,hdnl2,hdnl3** in these subroutines the nonlinear terms for x, y and z momentum equations are calculated, because of the staggered grid used the other components of velocity should be interpolated to find their value at the desired point.
- **chadsin.f**
  - **initia** the initial pressure field is given here.
  - **inqpr** the initial velocity field is given here.
  - **slotin** the regions of transpiration on the wall are calculated here.
- **chadsma.f** the main body of the code is here along with the main procedure. The input files are read and necessary files are created here.
  - **solve** this is the main procedure in which the other ones are called.
- **chadsnn.f**
  - **cfl** CFL number is calculated here. This number is used to calculate the time step size in constant CFL mode.
  - **coordi** the coordinate transformation function is used here to calculate the mesh in physical space. Also are selected some points to store certain quantities at, this is done for post processing purposes.
  - **divg** divergence of velocity is calculated in this subroutine.
  - **divgck** this subroutine perform the calculation of divergence of velocity and checks whether the mass is conserved locally.
  - **indic** some variables are defined here which make it simple to assign the boundary conditions.

  - **meshes** the computational mesh is computed here.
  - **metric** the metrics of grid transformation are calculated here.

- **chadsou.f**

  - **openfi** some files are opened here to be used later, either to read from or write on them.
  - **timeseq** some quantities are written on file here to be used later to start the calculations from. These are written on the points given in subroutine *coordi*.
  - **contwr** the file which will be used as a restarting file is written here, this is the main subroutine for this purpose.
  - **enerca** some quantities like total energy and enstrophy are calculated here.
  - **inirea** here is read the continuation field if the calculations are not to start from scratch.
  - **inx1x3** this subroutine and subroutines starting with *in* and *ex* are the interpolation and extrapolation procedures when making the mesh finer or coarser.
  - **q2div** here the normal velocity component is calculated from continuity equation when a file is read which has only streamwise and spanwise velocity component.
  - **oldqua** reading in the old time mean quantities is done here when using continuation fields and averaging during computations.
  - **outh** saves some information on files.
  - **oldqua** saves some information on files.
  - **taver** calculates the time average of velocity and stresses during computations.
  - **tavwri** writes quantities to evaluate averages in time
  - **tavrea** read quantities to evaluate averages in time when continuing a previous field.
  - **vmaxv** computes the maximum velocity in the field for convergence check.
  - **wstre** calculates the wall shear.

- **chadsphnc.f** here are subroutines to solve the Poisson equation.

- **chadru.f**

  - **rearuu** this subroutine reads the velocity correlations evaluated in a previous run to restart the time averaging process.
  - **ruucal** this subroutine performs the calculation of velocity correlations.
  - **ruphkp** this subroutine performs the calculation of the correlations by using the FFT routines. The direct calculations in the physical space are much more time consuming.

- **chadstin.f**
  - **updvp** this subroutine calculates the solenoidal velocity field by fractional step method.
  - **coeinv** coefficients of the tri-diagonal matrix are calculated here to be used in the solver.
  - **tschem** a time advancement is done here by calling proper subroutines, including all steps.
  - **invtr1** this subroutine performs the inversion of the momentum equation in spanwise direction.
  - **invtr2** this subroutine performs the inversion of the momentum equation in normal direction.
  - **invtr3** this subroutine performs the inversion of the momentum equation in streamwise direction, also the mean pressure gradient is calculated and assigned here.
  - **prcalc** calculates the pressure field.
  - **boucdq** the velocity at the wall is calculated here when working with moving walls.
- **chadstinv.f** here are the counterparts of the subroutines in *chadstn.f* for slip walls.
- **chadstr.f** here are a bunch of tri-diagonal solvers.