NUMERICAL SIMULATION OF FLOW PATTERN AND HEAT TRANSFER MECHANISM FROM A HEATED CYLINDER IN CROSS FLOW

ABANG MA’ARUF BIN ABANG BUSSRI

BACHELOR OF MECHANICAL ENGINEERING
UNIVERSITI MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG

FACULTY OF MECHANICAL ENGINEERING

We certify that the project entitled *Numerical Simulation of Flow Pattern and Heat Transfer Mechanism from a Heated Cylinder in Cross Flow* is written by *Abang Ma'aruf Bin Abang Bussri*. We have examined the final copy of this project and in our opinion; it is fully adequate in terms of scope and quality for the award of the degree of Bachelor of Engineering. We herewith recommend that it be accepted in partial fulfilments of the requirements for the degree of Bachelor of Mechanical Engineering.

Mr. Devarajan Ramasamy	………………………….
Examiner	Signature

NUMERICAL SIMULATION OF FLOW PATTERN AND HEAT TRANSFER MECHANISM FROM A HEATED CYLINDER IN CROSS FLOW

ABANG MA'ARUF BIN ABANG BUSSRI

Report submitted in fulfilment of the requirements
for the award of the degree of
Bachelor of Mechanical Engineering

Faculty of Mechanical Engineering
UNIVERSITI MALAYSIA PAHANG

NOVEMBER 2009

SUPERVISOR'S DECLARATION

I hereby declare that I have checked this project and in my opinion, this project is adequate in terms of scope and quality for the award of the degree of Bachelor of Mechanical Engineering.

Signature
Name of Supervisor: Muhamad Zuhairi Sulaiman
Position: Lecturer
Date:

STUDENT'S DECLARATION

I hereby declare that the work in this project is my own except for quotations and summaries which have been duly acknowledged. The project has not been accepted for any degree and is not concurently submitted for award of other degree.

Signature
Name: Abang Ma'aruf Bin Abang Bussri
ID Number: MA06083
Date:

Dedicated to my parent

ACKNOWLEDGEMENTS

I would like express my sincere gratitude to my supervisor Muhamad Zuhairi Sulaiman for his bright ideas, valueless supports, helpful guidance, and continuous encouragement in making this research. He used to motivate me with his strong conviction for engineering, wide range of experience, professional conduct. I really appreciate his constant support from first time I met him. I am truly grateful for his progressive vision regarding to my learning in engineering, his tolerance of my naïve mistakes and his commitment to my future career. And I also would like to express my very special thanks to my ex-supervisor Mohd Rosdzimin Abdul Rahman for my final year project 1.

A million thanks to my lecturer who give me guides, valuables knowledge, and help us to face the engineering world which extremely challenged. My special thanks to all my supporting members who give me a useful advices, and recommendation.

I appreciate my sincere indebtedness and gratitude to my parents for their love, hope, dream and sacrifice throughout my whole life. 1 cannot find the appropriate words that could properly describe my appreciation for their devotion, support and faith in my ability to attain goals.

ABSTRACT

Theory of lattice Boltzmann method is introduced to simulate a fluid flow numerically. The lattice Boltzmann method has found to be useful in many application involving interfacial dynamics and complex boundaries. First, the introduction of this report is described the objective of the project. This project objective is to study flow pattern and heat transfer mechanism from a heated cylinder in cross flow. The problem statement is explained in further detailed. The problem is solving using the lattice Boltzmann method theory and some flow simulation. This background of the project is related to the lattice Boltzmann method equation that is involving the Navier-Stoke equation, the governing equation and Bhatnagar-Gross-Krook (BGK) approximation. The literature review explains and described further detail about the lattice Boltzmann method. The methodology discusses about the simulation of the isothermal and thermal of the lattice Boltzmann and how to implement mathematical equation into FORTRAN programming language. The isothermal flow includes the Poiseuille and flow past a cylinder in cross flow. The thermal flow includes the flow past a heated cylinder in cross flow. The isothermal and thermal of lattice Boltzmann equation have been derived from the Boltzmann equation by discretization in both time and phase space. The result obtained by numerical simulation, compared to analytical solution.

ABSTRAK

Teori kaedah kekisi Boltzmann diperkenalkan untuk membuat simulasi aliran bendalir secara berangka. Kaedah kekisi Boltzmann didapati amat berguna di dalam banyak aplikasi melibatkan dinamik antaramuka dan sempadan yang kompleks. Pertama, pengenalan bagi laporan ini dijelaskan dengan objektif laporan ini. Objektif laporan ini adalah untuk mengkaji corak aliran and mekanisma pemindahan haba daripada satu silinder yang dipanaskan dalam aliran rentas. Penyataan masalah dijelaskan dengan lebih mendalam. Masalah tersebut diselesaikan dengan teori kaedah kekisi Boltzmann yang melibatkan persamaan Navier-Stoke, persamaan utama and penghampiran Bhatnagar-Gross-Krook (BGK). Ulasan rencana menjelaskan dan membincangkan secara lebih terperinci mengenai kaedah kekisi Boltzmann. Metodologi membincangkan tentang simulasi kekisi Boltzmann dalam isothermal dan termal dan bagaimana kaedah untuk menukar persamaan-persamaan matematik ke dalam bahasa pengaturcaraan FORTRAN. Isothermal meliputi aliran Poiseuille dan aliran melalui silinder dalam aliran rentas. Manakala, termal meliputi aliran melalui silinder yang dipanaskan. Isothermal dan termal dalam persamaan kekisi Boltzmann telahpun diterbitkan daripada persamaan Boltzmann dengan pengdiskritan di kedua-dua ruang dan masa. Keputusan yang diperoleh dengan kaedah simulasi dibandingkan dengan hasiln penyelesaian secara analitik.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS

Ω	Collision operator
ρ	Density
ρf	Body force
υ	Kinematic viscosity
τ	Single relaxation time
χ	Thermal diffusity
c	Macroscopic velocity
t	Time step
f	Density distribution function
F	Forcing terms
f^{eq}	Equilibrium density distribution function
g	Internal energy distribution function
G	Pressure gradient
g^{eq}	Equilibrium internal energy distribution function
P	Pressure
Ra	Rayleigh Number
Re	Reynolds Number
Pt	Prandtl Number

LIST OF ABBREVIATIONS

BC	Boundary Condition
BGK	Bhatnagar-Gross-Krook
LBE	Lattice Boltzmann equation
LBM	Lattice Boltzmann Method
LGA	Lattice Gas Automatta
MD	Molecular Dynamics

CHAPTER 1

INTRODUCTION

1.1 Project Background

Flow pattern around a cylinder in cross flow and its heat transfer mechanism has fascinated among engineers, scientist and researchers for one or two decades. There are many practical cases exist involving flow around a cylinder in cross flow which has an extensive significant in engineering such as skyscrapers, cooling towers, offshore platform support, pipes, heat exchangers many more. Although there are large number of researches and studies discussed about flow around cylinder in cross flow existed in the past, this study focused on numerical simulation of the flow and heat transfer as well by using Lattice Boltzmann Method (LBM) algorithm.

1.1.1 Lattice Gas Automata

Lattice Boltzmann models were first based on Lattice Gas Automata (LGA) in that they used the same lattice and applied the same collision. Instead of particles, Lattice Boltzmann (LBM) deal with continuous distribution functions which interact locally (only distributions at a single node are involved) and which propagate after collision to the next neighbor node. This is the main advantage of LBM compare to LGA. The next step in the development was the simplification of the collision and the choice of different distribution functions. This gives LBM is more flexible than LGA.

1.1.2 Molecular Dynamics

In molecular dynamics (MD), one tries to simulate macroscopic behavior of real fluids by setting up the model which described the microscopic interaction as good as possible. This leads to realistic equation of states whereas LGA or LBM posses only isothermal relations between mass density and pressure. The complexity of the interactions in MD restricts the number of particles and the time of integration. A method somewhat in between MD and LGA is maximally discretized molecular dynamics.

1.1.3 Lattice Boltzmann Method

The lattice Boltzmann method (LBM) has developed into an alternative and promising numerical scheme for simulating fluid flows and modeling physics in fluids. Historically, the lattice Boltzmann approach developed from improvement of lattice gases, although it can also be derived directly from the simplified Bhatnagar-Gross-Krook (BGK) equation. The lattice Boltzmann method is based on microscopic models and macroscopic kinetic equations. The kinetic nature of the LBM introduces important features that distinguish it from other numerical methods. First, the streaming process of the LBM in velocity space is linear. Second, the incompressible Navier-Stokes (NS) equations can be obtained in the nearly incompressible limit of the LBM. The LBM originated from lattice gas approach (LGA), a discrete particle kinetics utilizing a discrete lattice and discrete time. The primary goal of LBM is to build a bridge between the microscopic and macroscopic dynamics rather than to deal with macroscopic dynamics directly. In other words, the goal is to derive macroscopic equations from microscopic dynamics by means of statistics rather than to solve macroscopic equation in Figure 1.1 below.

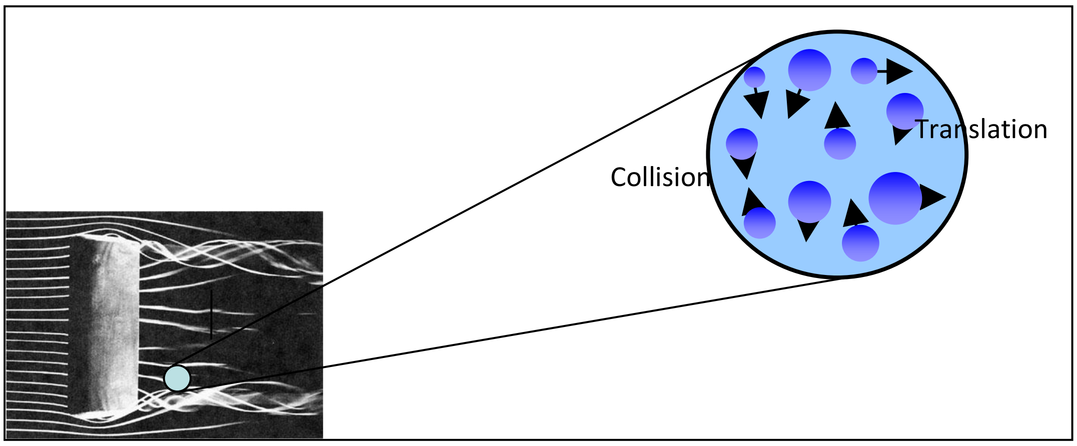


Figure 1.1: The relationship between macroscopic and microscopic.

Source: Azwadi 2007

1.2 Problem Statement

Understanding flow pattern and heat transfer mechanism from a heated cylinder in cross flow using Lattice Boltzmann Method.

1.3 Objective

(i) To study flow pattern and heat transfer mechanism from a heated cylinder in cross flow.
(ii) To study the flow pattern in a channel trough a cylinder in cross flow
(iii) To study the heat transfer mechanism of heated cylinder

1.4 Scope of Work

Low Rayleigh Number between 10^2 and 10^4 is used to study thermal flow in pressure driven parallel plate channel. In this case, Reynolds Number is fixed to 10^3. For isothermal flow the value of Reynolds Number is chosen between 10^2 and 10^4. Multi speed LBM D2Q9 microscopic velocity model is used to simulate the flow as it commonly and widely used yet simple to apply. Geometry of the cylinder is limited to square cylinder.

1.5 Process Flow Chart

Figure 1.2 shows the separation of information or processes in a step-by-step flow and easy to understand diagrams showing how steps in a process fit together. This makes useful tools for communicating how processes work and for clarity due to time limitation how a particular job is done in Final Year Project.

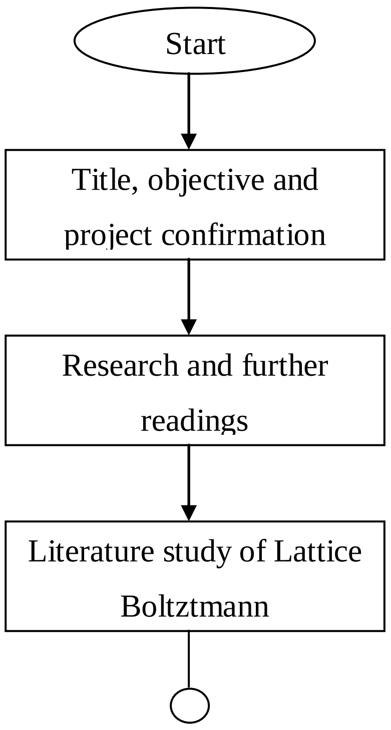

Start
Title, objective and project confirmation
Research and further readings
Literature study of Lattice Boltztmann

Figure 1.2: Project flowchart

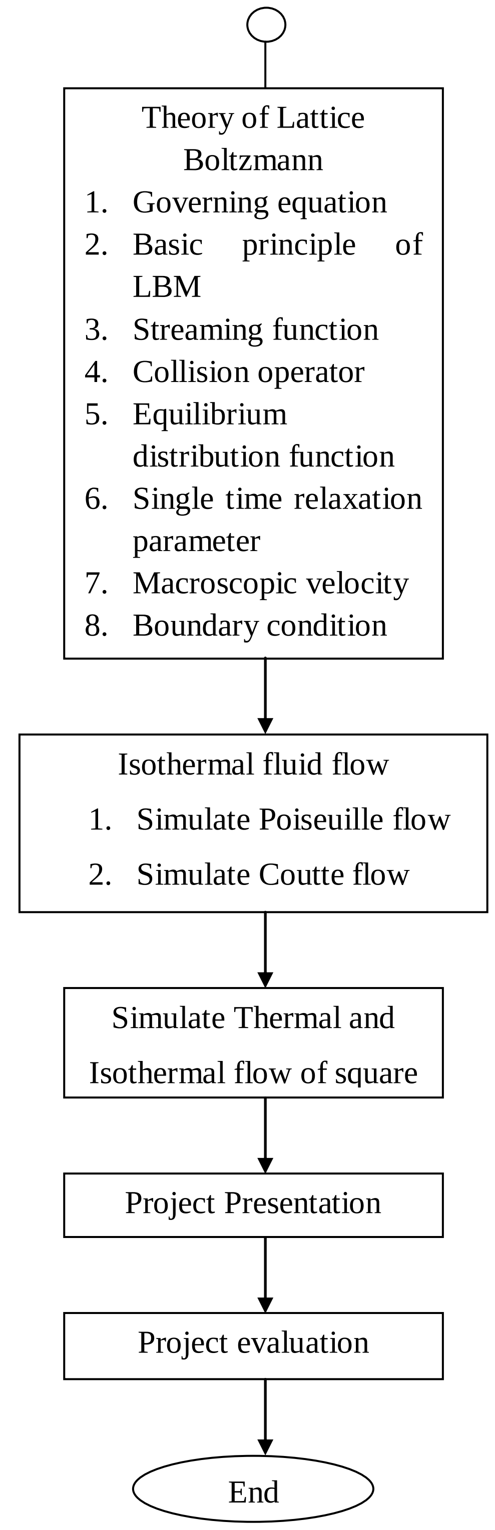

Theory of Lattice Boltzmann
1. Governing equation
2. Basic principle of LBM
3. Streaming function
4. Collision operator
5. Equilibrium distribution function
6. Single time relaxation parameter
7. Macroscopic velocity
8. Boundary condition
Isothermal fluid flow
1. Simulate Poiseuille flow
2. Simulate Coutte flow
Simulate Thermal and Isothermal flow of square
Project Presentation
Project evaluation
End

Figure 1.2: Project flowchart (continued)

CHAPTER 2

LITERATURE REVIEW

2.1 Governing Equation

$$\rho \frac{Du}{Dt} = -\nabla p + \rho f \tag{2.1}$$

$$\nabla \cdot u = 0 \tag{2.2}$$

Equation 2.1 shows the fluid velocity u of an inviscid (ideal) fluid of density ρ under the action of a body force ρf is determined and Equation 2.2 shows as Euler's equation. The scalar P is the pressure. This equation is supplemented by an equation describing the conservation of mass. For an incompressible fluid this is simply to get the continuity equation. Real fluids however are never truly inviscid. It must therefore see how Euler's equation is changed by the inclusion of viscous forces.

$$\frac{\partial u}{\partial t} + u\nabla \cdot u = -\nabla p + \left(\frac{2\tau_f - 1}{6} \right) \nabla^2 u \tag{2.3}$$

Equation 2.3 shows the flow of incompressible fluids can be described by the momentum equation. Derivation of continuity and momentum equation will get the Navier-Stokes equation by using Chapmann-Enskog expansion procedure. Chapmann-Enskog procedure is a method for obtaining an approximate solution. Chapmann-Enskog also manages to extract from the kinetic equation for the density distribution function, *F* a set of hydrodynamic equations for the particle number, the momentum and the energy per unit volume.

2.1.1 Macroscopic Equation for Isothermal

$$\upsilon = \frac{2\tau - 1}{6} \tag{2.4}$$

Using Chapmann-Enskog expansion procedure, relation between the single time relaxation τ, in microscopic level and viscosity of fluid υ, in macroscopic level is shown as above.

2.1.2 Macroscopic Equation for Thermal

$$\frac{\partial T}{\partial t} + \nabla \cdot \left(\mathbf{u}T\right) = \left(\tau_g - \frac{1}{2}\right)\nabla^2 T \tag{2.5}$$

In thermal model, the energy equation is added. The energy equation is shown above.

$$\tau_f = 3\upsilon + \frac{1}{2} \tag{2.6}$$

$$\tau_g = \chi + \frac{1}{2} \tag{2.7}$$

Viscosity and thermal diffusivity will produce when using the Chapmann-Enskog expansion procedure where v is viscosity and χ is thermal diffusivity show in Equation 2.6 and Equation 2.7.

2.2 Bhatnagar-Gross-Krook (BGK)

The model of GGK was proposed in 1954 and became the most important model to solve the integral-differential Boltzmann equation (proposed by Boltzmann in 1872). In BGK model, the nonlinear collision term of the Boltzmann equation is replaced by a simpler term and the model makes the derivation of the transport equations for macroscopic variables much easier. A problem, which is easily solved by the BGK model, is that of relaxation of a state of a fluid to equilibrium. During the last 20 years or so, the BGK model has found an important new application - derivation of numerical schemes, namely kinetic schemes to solve hyperbolic conservation of laws. The collision process is mimicked by a distribution function rather than solving for collisions of every particle. Lattice-Boltzmann models are normally constructed using Fermi-Dirac or Maxwellian distributions as the collision process. However, solving these distribution functions is complicated and computationally expensive. The single relaxation time BGK operator approach allows us to solve this equilibrium distribution such that the microscopic equations are satisfied and the Navier-Stokes equations are recovered.

$$f(x + c\Delta t, t + \Delta t) - f(x,t) = \Omega(f) \tag{2.8}$$

The BGK equation was derived from the Boltzmann equation which shown in Equation 2.8

$$\frac{1}{\tau} f^{neq}(\mathbf{x},t) = \frac{1}{\tau}\left[f(\mathbf{x},t) - f^{eq}(\mathbf{x},t)\right] = \Omega(f) \tag{2.9}$$

Equation 2.9 show Bhatnagar-Gross-Krook equation where, *Ω*(*x t),* is the single relaxation time BGK operator and *f(x,t)* is the current distribution of particles on a lattice node. The *τ* symbol *is* the time relaxation parameter and *f^{eq}(x,t)* is the equilibrium function.

2.3 Lattice Boltzmann Equation

$$f(x+c\Delta t, t+\Delta t) - f(x,t) = \Omega(f) \tag{2.10}$$

The Boltzmann equation for any lattice model is an equation for the time evolution of *f(x,t),* the single-particle distribution at lattice site **x** where f is density distribution function, c is microscopic velocity, *Ω* is collision integral and *Δt* is a value of unity shown in Equation 2.10.

$$\Omega(f(x,t)) = -\frac{f(x,t) - f^{eq}(x,t)}{\tau} \tag{2.11}$$

Since the usual aim of lattice methods is to model macroscopic dynamics, the "exact" collision operator is unnecessarily complex and therefore numerically inefficient. The collision operator is approximated by a single-time-relaxation process in which relaxation to some appropriately chosen equilibrium distribution occurs at some constant rate. In particular the collision term, *Ω(f)* is replaced by the single-time-relaxation approximation shown in Equation 2.11.

The appropriately chosen equilibrium distribution, denoted by f^{eq}, depends on the local fluid variables, and $1/\tau$ is the rate of approach to this equilibrium. The relations $\Sigma\Omega=0$ and $\Sigma c\Delta T\Omega=0$ must be true to conserve mass and momentum, respectively. Here $1/\tau$ is the relaxation constant, the rate of change towards the equilibrium. Parameter τ is time relaxation. When τ = 1, the distribution functions are exactly set to the equilibrium distribution. When τ = 2, The distribution functions are midway between incoming distribution and the equilibrium distribution.

$$\Sigma f = \Sigma f^{eq} = \rho \text{ or } \int f dc = \int f^{eq} dc = \rho \tag{2.12}$$

$$\Sigma cf = \Sigma cf^{eq} = \rho u \text{ or } \int cf dc = \int cf^{eq} dc = \rho u \tag{2.13}$$

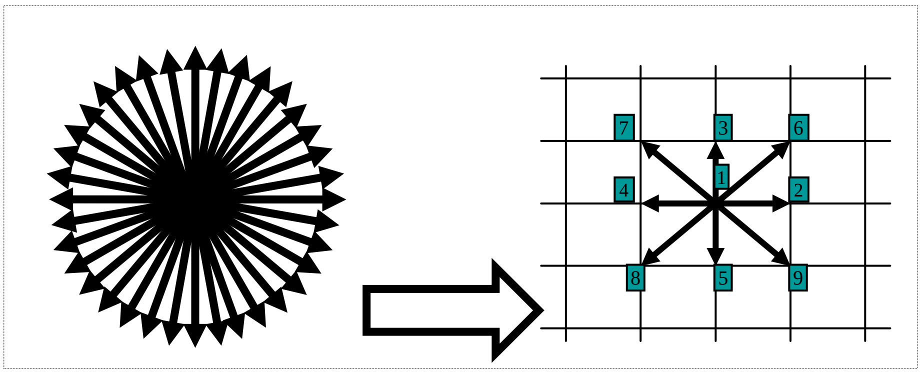
Deriving the macroscopic equations obeyed by this model, one performs a Taylor expansion in time (*t*) and space (*x*) and takes the long-wavelength and lowfrequency limit of the lattice-Boltzmann equation for the single-particle distribution. The result is a continuum form of the Boltzmann equation correct to second order in the lattice spacing and the time step. The macroscopic variables, such as the density, ρ and flow velocity, u can be evaluated as the moment to the distribution function as Equation 2.12 and Equation 2.13.

2.4 Discretization of microscopic velocity

Discretizing the Boltzmann-BGK equation at a set of velocities that correspond to the nodes of a Gauss-Hermite quadrature in velocity space, we effectively project and solve the Boltzmann equation in a subspace spanned by the leading Hermite polynomials.

2.4.1 Lattice Boltzmann Isothermal Model

From the continuous velocity, use the Gauss-Hermite integration to get the 9-discrete velocity model. Figure below will shown how from continuous velocity to get 9-discrete velocity model by using Gauss-Hermite integration.



Using Gauss-Hermite integration

Figure 2.1: 9-discrete velocity model

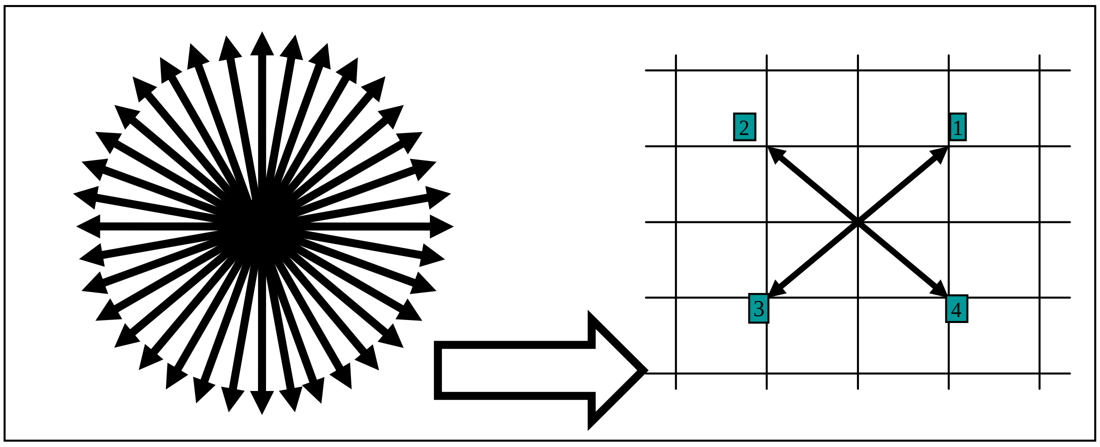
Source: Azwadi 2007

$$f(x + c\Delta t, t + \Delta t) - f(x, t) = -\frac{f - f^{eq}}{\tau} \tag{2.14}$$

Using 9-discrete velocity model means there are 9 directions that the particles will go after the collisions. The equation for 9-discrete velocity is shown in Equation 2.14 where *f(x+cΔ, t+Δt)* - *f(x,t)* is streaming process and f - f^{eq} / τ is collision process. There are only 9-discrete velocity for isothermal model because if use less than 9, it will not get the continuity and momentum equation.

2.4.2 Lattice Boltzmann Thermal Model

From the continuous velocity, use the Gauss-Hermite integration to get the 4-discrete velocity model. Figure below will shown how from continuous velocity to get 4-discrete velocity model by using Gauss-Hermite integration.



Using Gauss-Hermite integration

Figure 2.2: 4-discrete velocity model

Source: Azwadi 2007

$$g(x + c\Delta t, t + \Delta t) - g(x, t) = -\frac{g - g^{eq}}{\tau} \tag{2.15}$$

Using 4-discrete velocity model means there are 4 directions that the particles will go after the collisions. The equation for 4-discrete velocity is shown in equation 2.15 where $g(x+c\Delta, t+\Delta t) - g(x,t)$ is streaming process and $g - g^{eq} / \tau$ is collision process.

There are only 4-discrete velocity for isothermal model because if use less then 4, we will not get the continuity and momentum equation. If use more then 4, we will take long time to simulate.

2.5 Bounce-back Boundary Condition

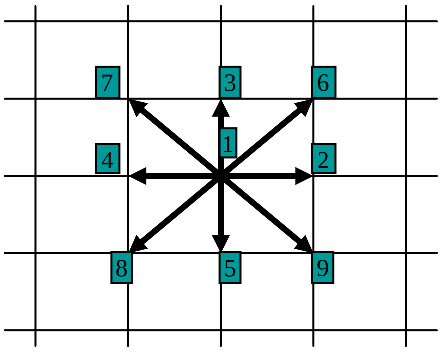


Figure 2.3: D2Q9 model

Source: Azwadi 2007

The initial approach to simulate boundary back was to follow the methods used in the lattice Gas Approach. The simplest boundary condition is the so-called bounceback boundary conditions where all the distribution functions at the boundaries back along to the link they arrived on as shown in Figure 2.4. This type of boundary condition can be shown by using 3x3 matrixes shown in Figure 2.5.

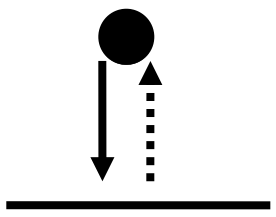


Figure 2.4: Bounceback boundary condition

Source: Succi 2001

$$\begin{bmatrix} f_3 \\ f_6 \\ f_7 \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} f_5 \\ f_8 \\ f_9 \end{bmatrix}$$

Figure 2.5: 3x3 matrixes boundary condition

Averaging the velocity at the boundary before and after the collision, give zero fluid velocity at wall.

CHAPTER 3

METHODOLOGY

3.1 Introduction

Although a large number of studies for the isothermal and thermal flow of square cylinder in cross flow in CFD, this study has introduced LBM to simulate a fluid flow. Each approach used to simulate the fluid flow will not give the identical result. The different are in term of accuracies, convergences, errors, efficiencies etc. It is impossible to give the exact value for numerical with analytical result. However, it would simulate pretty good to give the significant results. There a lot of numerical tools used to implement LBM in the simulation. But in this thesis, FORTRAN 90 programming language is used as the numerical analysis tools. Codes development is constructed using Compaq Visual FORTRAN Professional Edition 6.6.0.

3.2 LBM Algorithm

A typical LBM algorithm schemes are executed for each time step in the sequenced order as shown in Figure 3.1. All parameters and global variables are declared in lattice unit (*lu*) as it is the fundamental measure of length in the LBM models and time step (*ts*) are time the unit Sukop et al. Therefore, physical unit are not required and it just enough to give the appropriate value for physical quantities but, it must carefully chosen to avoid the code instability. Some reports give the stable value for the simulation Sterling and Chen et al. So, this study will use the provided value to run the simulation.

Variable prescription must be set up in initialization step before running the LBM algorithm. Generally, it is given based on stability analysis. Its value either fixed or in range of stable allowable value as introduced by Succi et al and Chen et al. One crucial think need to be treated carefully in fluid flow is boundary condition (BC). Good BC treatment will give the expected result. In this study, the simulation uses some BC which is selected suit with the case study.

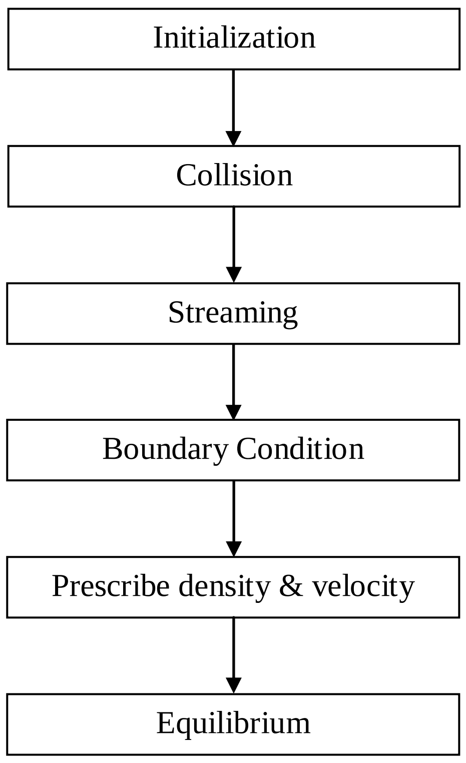


Figure 3.1: A typical sequence of LBM algorithm

3.2.1 Initialization

Program code is constructed with main program and list of subroutines. In the main program, global parameter such as meshing sizes, boundary sizes, initial condition, controlled and fixed variables are declared. The meshing sizes set to be 200x40 nodes isothermal and 22x50 nodes for thermal. Where the horizontal boundary, *x* start at point 1st till 200th nodes whereas the vertical boundary, *y* start at point 1st as well till 40th nodes. The nodes cannot start with 0th nodes due to array bounce exceed while streaming step take place. The remaining constant and variables declared as the real variables. However, to achieve higher accuracy, double precision used. Figure 3.2 shows FORTRAN code implementation for initialization in main program.

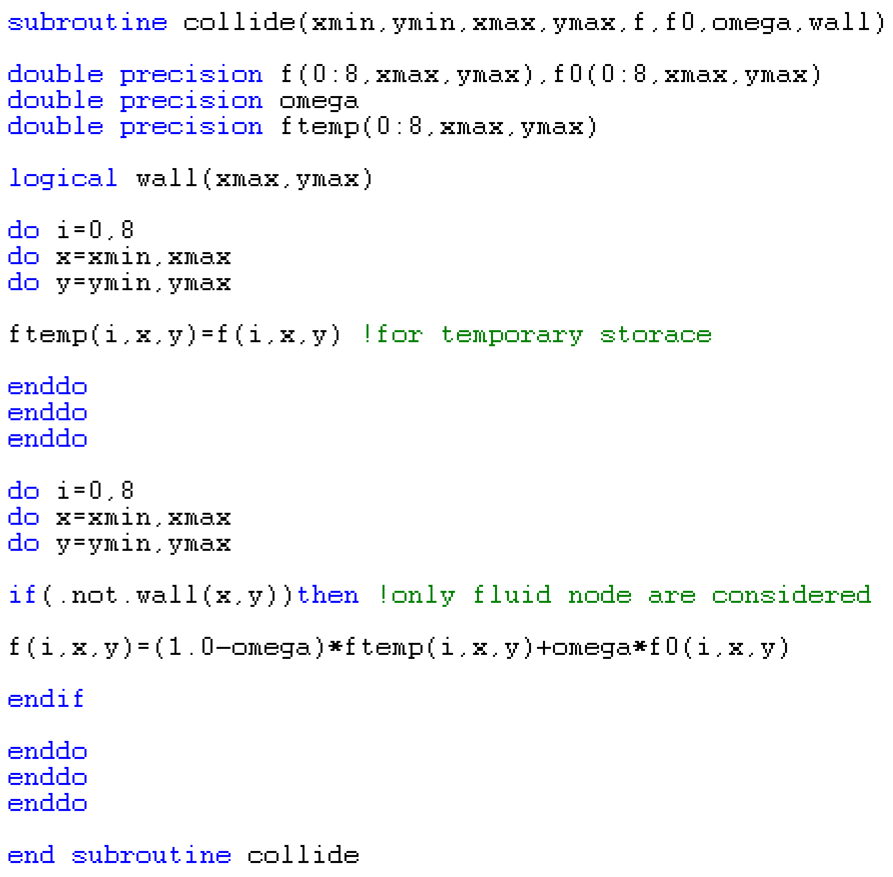
```
double precision f(0:8,xmax,ymax),f0(0:8,xmax,ymax)
double precision rho(xmax,ymax),u(xmax,ymax),v(xmax,ymax)
double precision zeta(xmax,ymax),p(xmax,ymax)
double precision cx(0:8),cy(0:8),w(0:8)
double precision u0,v0,rho0
double precision cs2,tau,omega,nu,lc,l,c,re,a,umax,g

double precision ut(xmax,ymax),vt(xmax,ymax) !analytical velocity
double precision err(ymax),error !error

logical wall(xmax,ymax)

integer step,maxstep

u0=0.0
v0=0.0
rho0=1.0
cs2=(1.0/3.0) !sound speed square (lattice constant)
tau=0.55 !single time relaxation
omega=1.0/tau
nu=cs2*(tau-1.0/2.0) !kinematic viscosity
lc=(ymax-ymin)/2.0+1
l=lc-1
c=(xmax-ymin)/2.0
re=1000.0 !reynolds number
umax=re*nu/(2*l) !center velocity
g=2*rho0*nu*umax/l**2.0 !pressure gradient
```

Figure 3.2: Initialization in main program

Before that, all operation variables must declare in the main program before LBM algorithm start. In this study, all the variables uses consistently. All the variable representation is shown in Appendix A. The code shown above is a part of isothermal Poiseuille flow simulation.

3.2.2 Collision Step

When the iteration starts, collision step take place within the fluids. Therefore, never apply the collision step at solid boundaries. Solid node boundary has a special treatment that is so called no-slip BC. This BC will be discussed later. Furthermore, in programming arithmetic, the old value will be replaced with new value after one single loop or time step. So, a temporary storage for the value is necessary to avoid errors. Figure 3.3 shows the code for collision step along with temporary storage treatment.



```
subroutine collide(xmin,ymin,xmax,ymax,f,f0,omega,wall)

double precision f(0:8,xmax,ymax),f0(0:8,xmax,ymax)
double precision omega
double precision ftemp(0:8,xmax,ymax)

logical wall(xmax,ymax)

do i=0,8
do x=xmin,xmax
do y=ymin,ymax

ftemp(i,x,y)=f(i,x,y) !for temporary storage

enddo
enddo
enddo

do i=0,8
do x=xmin,xmax
do y=ymin,ymax

if(.not.wall(x,y))then !only fluid node are considered

f(i,x,y)=(1.0-omega)*ftemp(i,x,y)+omega*f0(i,x,y)

endif

enddo
enddo
enddo

end subroutine collide
```

Figure 3.3: Collision step in LBM algorithm

3.2.3 Streaming Step

Streaming step is the propagation of fluid particle in fluid flow. Since, there are nine velocities model used, nine type of propagation step constructed. Streaming step deals crucially with array size of density distribution function and might result array result array bounce exceed errors if did not built carefully. Just like collision step, this step needs a temporary storage for density distribution function. Figure 3.4 shows the code for rest and right motion of fluid propagation.

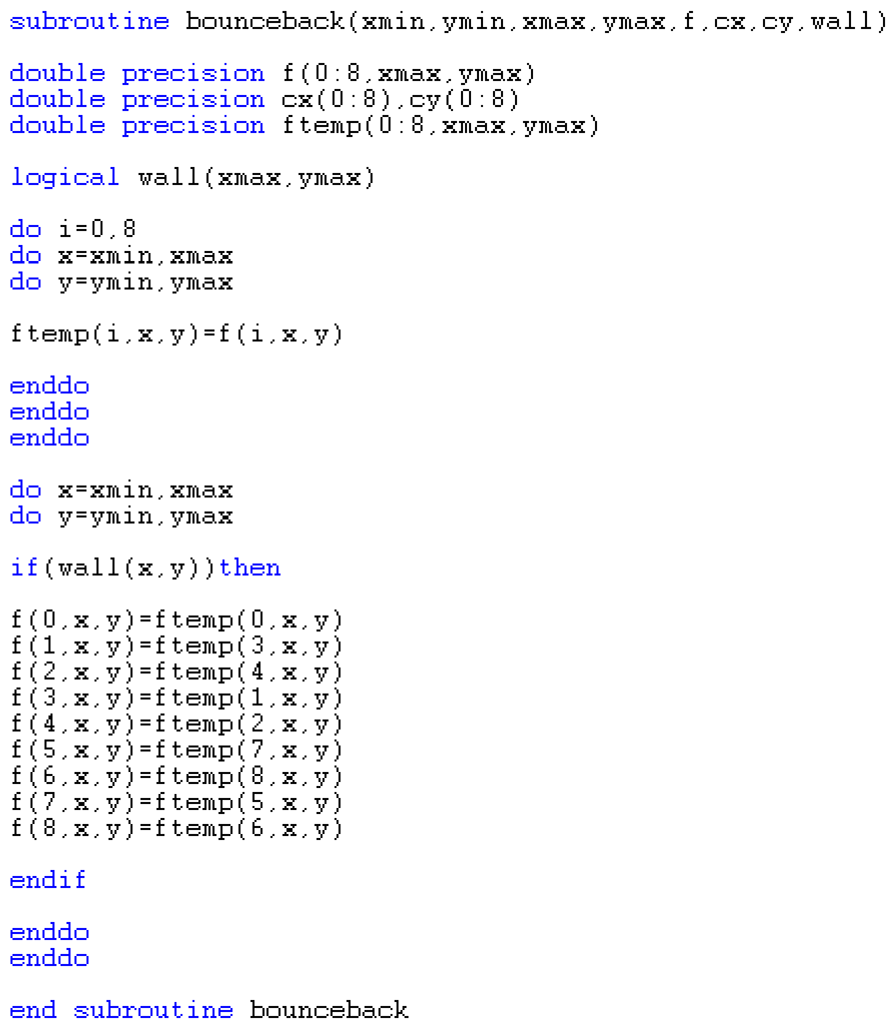
```
subroutine stream(xmin,ymin,xmax,ymax,f,cx,cy)

double precision f(0:8,xmax,ymax)
double precision cx(0:8),cy(0:8)
double precision ftemp(0:8,xmax,ymax)

do i=0,8
do x=xmin,xmax
do y=ymin,ymax

ftemp(i,x,y)=f(i,x,y)!temporary storage

enddo
enddo
enddo

do i=0,8
if(i.eq.0)then
do x=xmin,xmax
do y=ymin,ymax

f(i,x,y)=ftemp(i,x,y)

enddo
enddo

else if(i.eq.1)then
do x=xmin,xmax-1
do y=ymin,ymax

f(i,x+1,y)=ftemp(i,x,y) !to the right motion

f(i,xmin,y)=ftemp(i,xmax,y)!periodic

enddo
enddo
```

Figure 3.4: Streaming step for fluid propagation for rest and right motion.

This streaming step code shows only the propagation of the rest particle, and right motion. Periodic BC is applied during propagation. The four corners of the boundary, south-west, north-west, south-east and north-east treated with special treatment. The further details of the code implementation for streaming step are shown in Appendix B.

3.2.4 Boundary Condition

Generally, there are two types of BC used in the simulation. The so called bounce back BC or sometimes called as no slip BC and periodic BC. In this case study, the cylinder placed in a parallel plat channel. Hence, this is an open flow where both sides at left and right are opened. Whereas, at top and bottom of the channel are solid nodes. The bounce back occurs at top and bottom of the channel. Figure 3.5 shows how the implementation of bounce back rule in FORTRAN code.



```
subroutine bounceback(xmin,ymin,xmax,ymax,f,cx,cy,wall)

double precision f(0:8,xmax,ymax)
double precision cx(0:8),cy(0:8)
double precision ftemp(0:8,xmax,ymax)

logical wall(xmax,ymax)

do i=0,8
do x=xmin,xmax
do y=ymin,ymax

ftemp(i,x,y)=f(i,x,y)

enddo
enddo
enddo

do x=xmin,xmax
do y=ymin,ymax

if(wall(x,y))then

f(0,x,y)=ftemp(0,x,y)
f(1,x,y)=ftemp(3,x,y)
f(2,x,y)=ftemp(4,x,y)
f(3,x,y)=ftemp(1,x,y)
f(4,x,y)=ftemp(2,x,y)
f(5,x,y)=ftemp(7,x,y)
f(6,x,y)=ftemp(8,x,y)
f(7,x,y)=ftemp(5,x,y)
f(8,x,y)=ftemp(6,x,y)

endif

enddo
enddo

end subroutine bounceback
```

Figure 3.5: Bounce back rule in FORTRAN code

Remind that, the bounce back BC need the temporary storage as well to avoid value replacement. The above code shows only the 2, 4, 5, 6, 7 and 8 macroscopic velocities since the isothermal Poiseuille flow consist only top and bottom solid nodes. There two ways to make the coding i.e. using logical within the wall boundary or state the velocities at particular boundary. Next BC is periodic BC. Periodic BC are the simplest instance of BC one can think of, the joy of programmer. They are typically intended to isolate bulk phenomena from the actual boundaries of the real physical system and consequently they are adequate for physical phenomena where surface effects play a negligible role Succi et al. Figure 3.6 picture how the periodic boundary condition works.

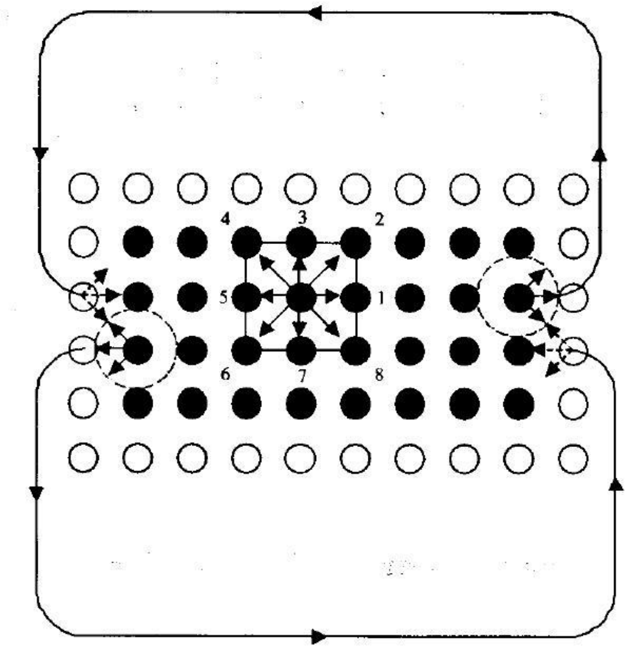


Figure 3.6: Periodic BC. Particle leaving from the left side re-enter at right or inlet. Source: Succi 2001

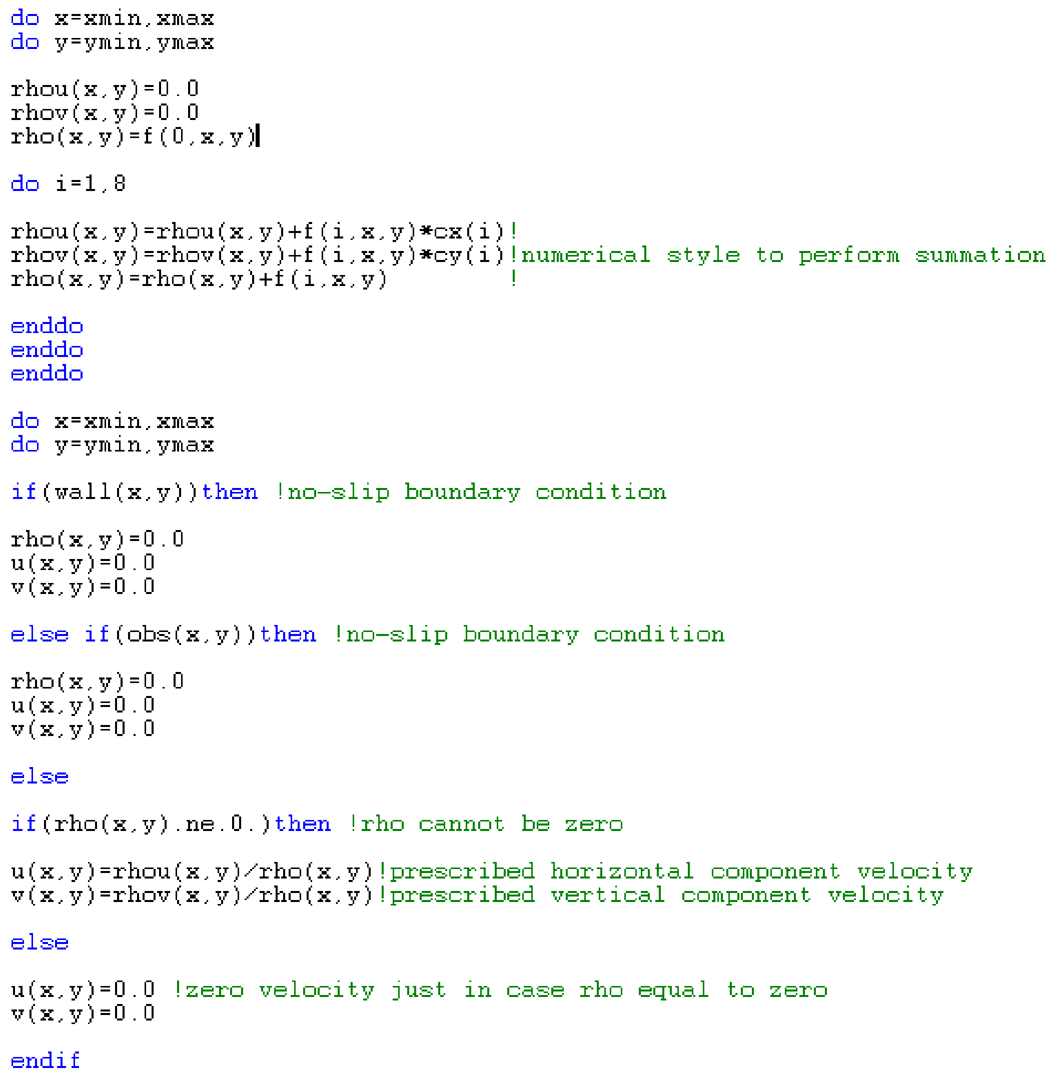
Periodic BC implemented while at streaming step. When the fluid particle propagate to the left most of the boundary, *xmax*, it will fill the left buffer and vice of versa. So on with the left most will fill the right most of the end boundary. Figure 3.7 show how to put periodic BC in streaming step.

```
!periodic boundary condition
do y=ymin,ymax

f(1,xmin,y+cy(1))=ftemp(1,xmax,y)!
f(5,xmin,y+cy(5))=ftemp(5,xmax,y)!west
f(8,xmin,y+cy(8))=ftemp(8,xmax,y)!

f(3,xmax,y+cy(3))=ftemp(3,xmin,y)!
f(6,xmax,y+cy(6))=ftemp(6,xmin,y)!east
f(7,xmax,y+cy(7))=ftemp(7,xmin,y)!

enddo

do x=xmin,xmax

f(2,x+cx(2),ymin)=ftemp(2,x,ymax)!
f(5,x+cx(5),ymin)=ftemp(5,x,ymax)!north
f(6,x+cx(6),ymin)=ftemp(6,x,ymax)!

f(4,x+cx(4),ymax)=ftemp(4,x,ymin)!
f(7,x+cx(7),ymax)=ftemp(7,x,ymin)!south
f(8,x+cx(8),ymax)=ftemp(8,x,ymin)!

enddo

f(5,xmin,ymin)=ftemp(5,xmax,ymax)!south-west
f(8,xmin,ymax)=ftemp(8,xmax,ymin)!north-west
f(6,xmax,ymin)=ftemp(6,xmin,ymax)!south-east
f(7,xmax,ymax)=ftemp(7,xmin,ymin)!north-east
```

Figure 3.7: Code for periodic BC. It implemented along with streaming step.

3.2.5 Prescribe Density and Velocity

Density and velocity is the primary output result of the simulation. The derivation of other physical quantity such as pressure, vorticity and average velocity are depending on this particular value. Prescribe density and velocity is the local density and velocity of each node every time step. Prescribe density and velocity effect the density distribution function while the fluid particles achieve the equilibrium distribution function. The equilibrium distribution function will be discussed later. Figure 3.8 shows the code for prescribing the value of density and velocity.



```fortran
do x=xmin,xmax
do y=ymin,ymax

rhou(x,y)=0.0
rhov(x,y)=0.0
rho(x,y)=f(0,x,y)

do i=1,8

rhou(x,y)=rhou(x,y)+f(i,x,y)*cx(i)!
rhov(x,y)=rhov(x,y)+f(i,x,y)*cy(i)!numerical style to perform summation
rho(x,y)=rho(x,y)+f(i,x,y)        !

enddo
enddo
enddo

do x=xmin,xmax
do y=ymin,ymax

if(wall(x,y))then !no-slip boundary condition

rho(x,y)=0.0
u(x,y)=0.0
v(x,y)=0.0

else if(obs(x,y))then !no-slip boundary condition

rho(x,y)=0.0
u(x,y)=0.0
v(x,y)=0.0

else

if(rho(x,y).ne.0.)then !rho cannot be zero

u(x,y)=rhou(x,y)/rho(x,y)!prescribed horizontal component velocity
v(x,y)=rhov(x,y)/rho(x,y)!prescribed vertical component velocity

else

u(x,y)=0.0 !zero velocity just in case rho equal to zero
v(x,y)=0.0

endif
```

Figure 3.8: Prescribe density and velocity

Density defines as total of density distribution function of all macroscopic velocities.

$$\rho = \sum_{i=0}^{8} f_i \qquad (3.1)$$

However, there are no such functions for summation in FORTRAN programming language. So, need to convert the above equation to numerical style of summation. Then, the Equation 3.1 becomes,

For $i = 0$

$$\rho_i = f_i \tag{3.2}$$

Thus,

$$\rho_0 = f_0 \tag{3.3}$$

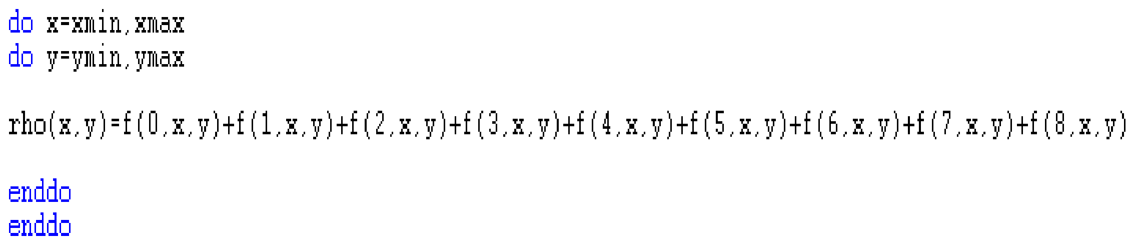
Then, for $1 \leq i \leq 8$

$$\rho_i = \rho_{i-1} + f_{i-1} \tag{3.4}$$

Instead of using the above numerical style, there is other way to do the summation function. Well, the computation could be performing by manually summing all the density distribution function. From equation 3.1,

$$\rho = f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 \tag{3.5}$$

Figure 3.9 shows the numerical implementation for manual summation. The first method is suitable for performing a whole summation of a large amount of variables. Whereas, the second method is suitable for performing a certain set of variables.



```
do x=xmin,xmax
do y=ymin,ymax

rho(x,y)=f(0,x,y)+f(1,x,y)+f(2,x,y)+f(3,x,y)+f(4,x,y)+f(5,x,y)+f(6,x,y)+f(7,x,y)+f(8,x,y)

enddo
enddo
```

Figure 3.9: Numerical implementation for manual summation

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

Back to the case study, simulating both isothermal and thermal flow of cylinder in cross flow is based on Poiseuille flow since the cylinder is placed axis symmetrically in the parallel plate channel. First of all, Poiseuille flow code needs to be validated. Therefore, numerical result for velocity profile of flow in the parallel plate channel compare to the analytical result for Poiseuille flow. After all, the cylinder placed in the channel. Here, square cylinder is used as an obstacle.

4.2 Isothermal Poiseuille Flow

Isothermal Poiseuille can be simulate with many ways such as pressure driven, body force like gravity force, specify pressure at both end of the channel and apply incoming velocity at inlet channel. But, in this study, pressure driven Poiseuille flow of parallel plate channel in two dimensions is used. In pressure driven Poiseuille flow, body force is added to the fluid particle. Body force is depend on pressure gradient, $g = -dp/dx$. Pressure gradient are refer to the Reynolds number. Generally, Reynolds number is chosen in low value i.e. in range of 10 till 1000. The Reynolds number cannot be too high due to turbulent flow. Then, instability will take placed thus will not get the expected result.

Post processing tool to visualize the flow is chosen using MATLAB programming language since it has an attractive and powerful features. The data obtained from the *.dat* file created by FORTRAN code imported directly to MATLAB. Figure 4.1 shows the flow visualization of Poiseuille flow. There are three different feature of MATLAB function can be use to visualize. From the comparison of the function, *streamslice* function is used to visualize flow as in Figure 4.3. Velocity vector plot only show the direction of average local velocity. The magnitude of the velocity is represented by size of the arrow. Streamline function in Figure 4.1, is plotting the streamline of the flow. It visualizes the streamline smoothly. However, it is unable to plot the vortex of the flow. Its result is clearer in the simulation of flow pass the cylinder in cross flow. Moreover, *streamslice* function provides the arrow to indicate the velocity direction preciously in the mean time plot the streamline.

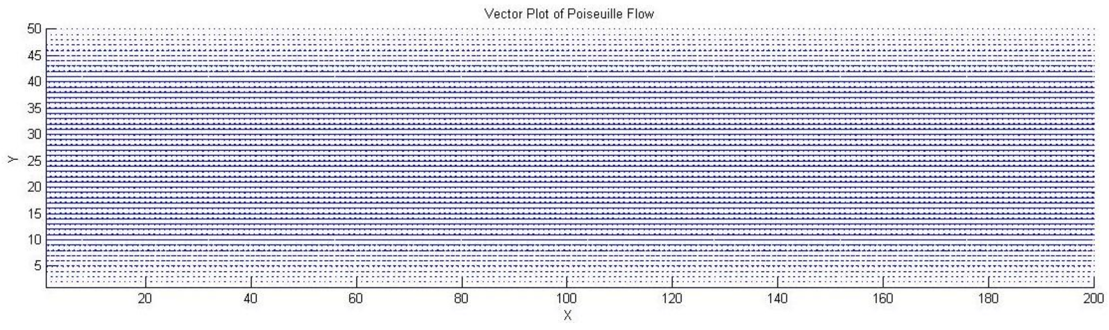


Figure 4.1 (a): Vector plot of Poiseuille flow

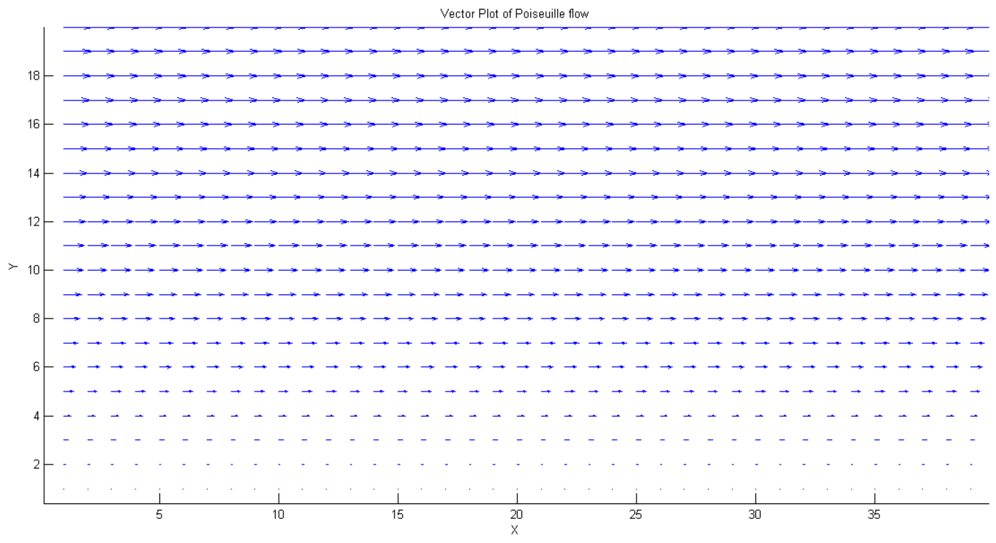


Figure 4.1 (b): Closer look of Vector Plot of Poiseuille flow

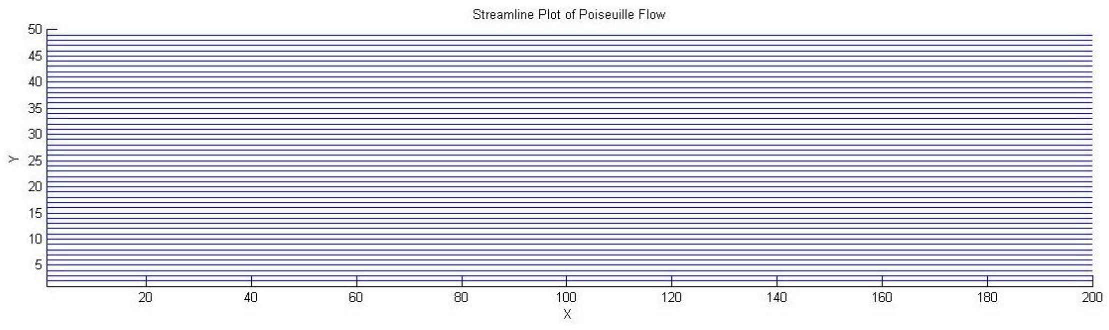


Figure 4.2 (a): Streamline function plot for the streamline

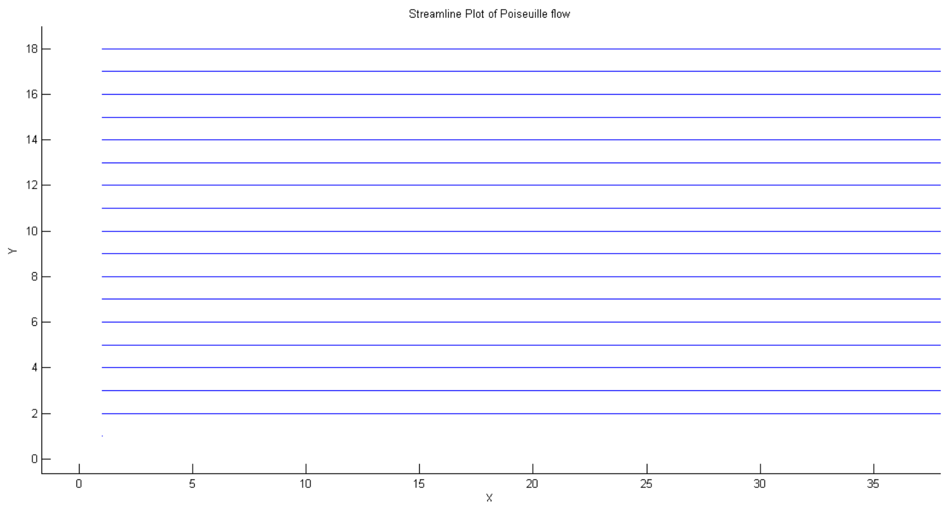


Figure 4.2(b): Closer look of Streamline function plot for the streamline

No matter what the visualization used, the exact value for Poiseuille flow is impossible to obtain. Even thought, the result is closer enough with very small error. Figure 4.4 shows the developing of Poiseuille flow by increasing of time step. Time step is in unit of *ts*.

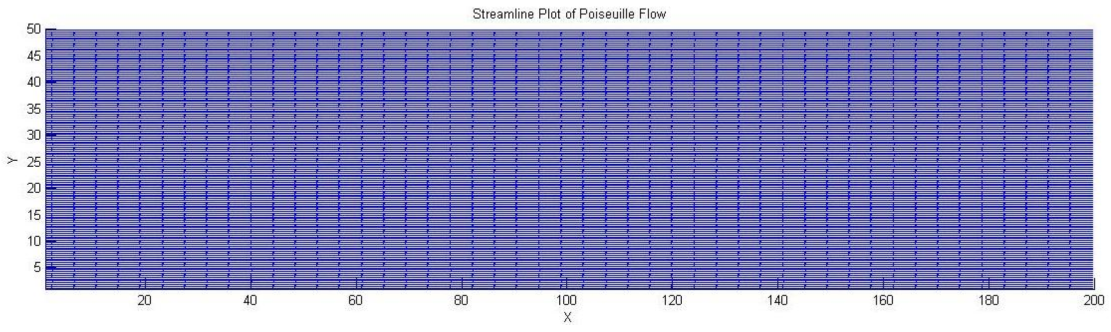


Figure 4.3 (a): Streamslice function plot for Poiseuille flow.

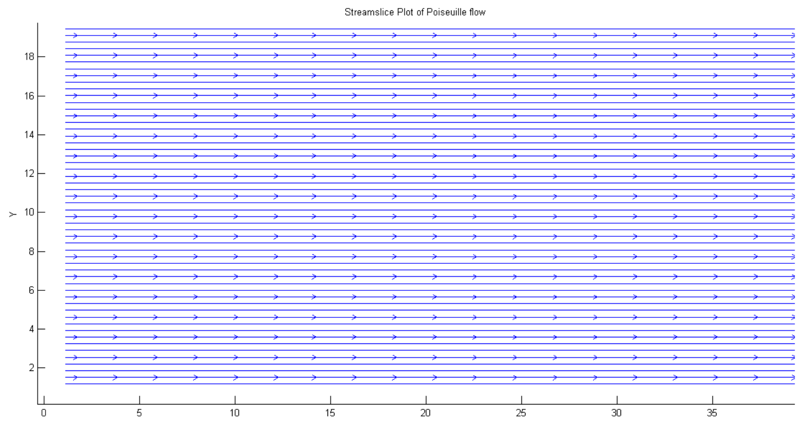
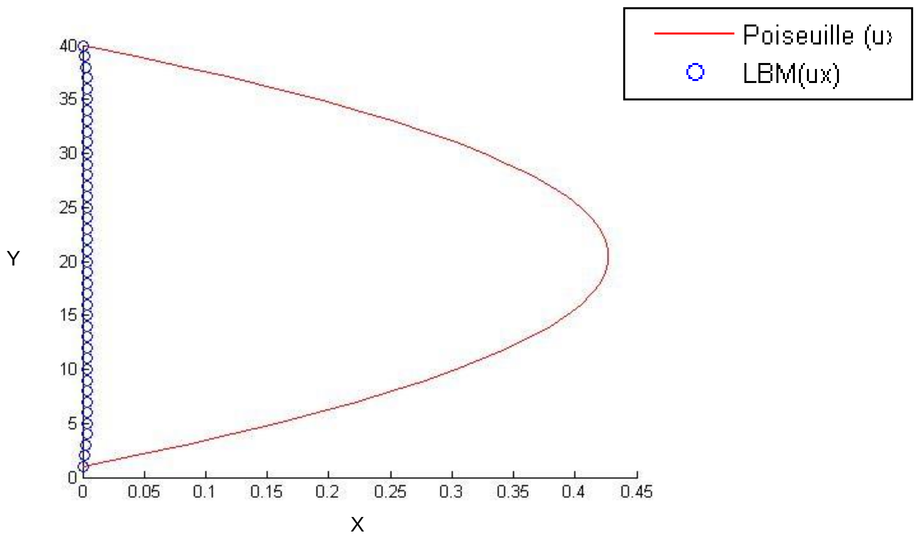
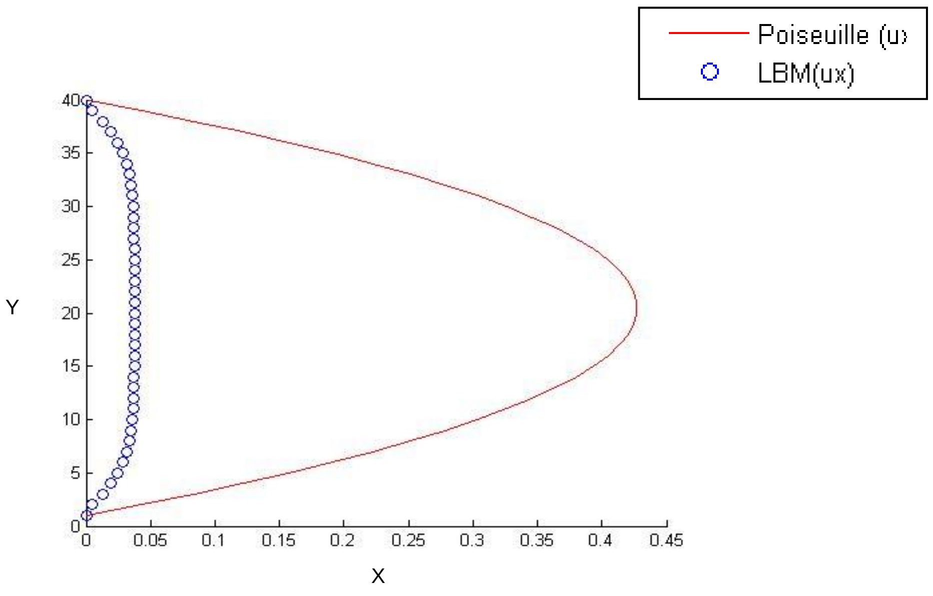


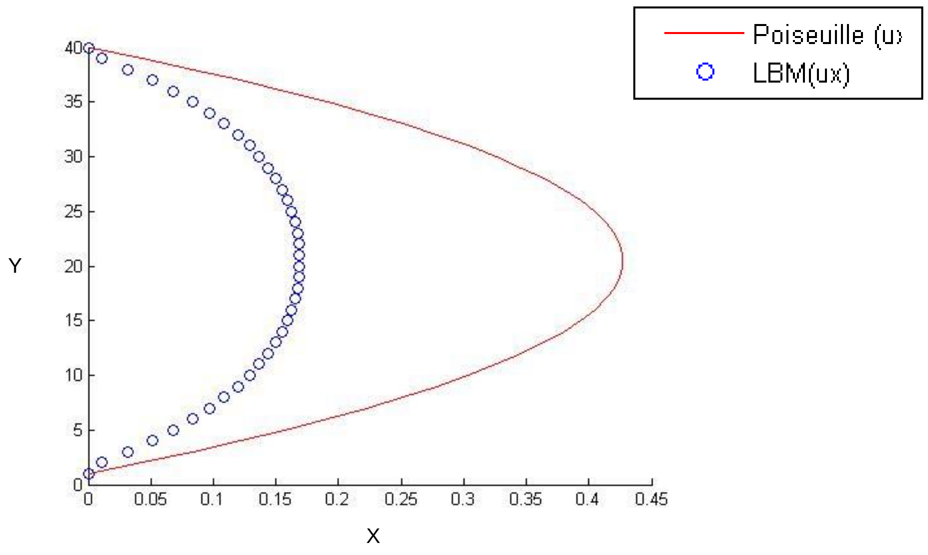
Figure 4.3 (b): A closer look of Streamslice function plot which has arrows to indicate flow directions



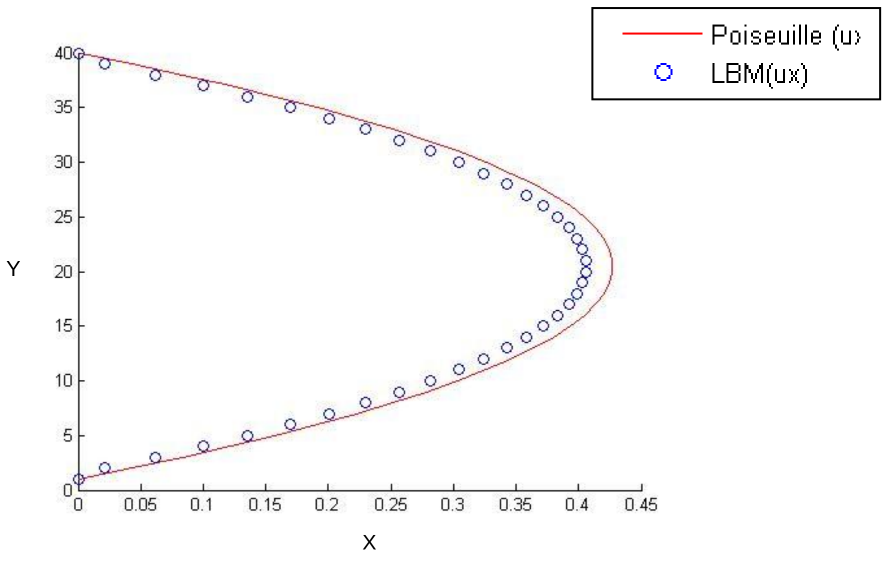
(a) t = 100



(b) t = 1000



(c) t = 5000



(d) t = 1000000

Figure 4.4: Graph of channel width versus velocity of every time step.

The fully developed Poiseuille flow obtained at time step t = 1000000. At this moment, the convergence of the velocity profile is approximately 1×10^{8}. It is fine to consider the flow experienced fully developed Poiseuille flows at this condition. The errors for the Poiseuille flow is define in equation 4.1 as used by Zuo et al. It implies that, no possibility to get exact value. In equation 4.2, the convergence of velocity profile reaches

$$error \equiv \max \frac{\sqrt{\left(u_x^t - v_x\right)^2 + \left(u_y^t - v_y\right)^2}}{u_0} \tag{4.1}$$

Where u_x^t, v_y^t is the analytical velocity, whereas, u_0 is the peak velocity.

$$\frac{\sum_i \sum_j | v_x(i,j,t+\delta) - v_x(i,j,t) | + | v_y(i,j,t+\delta) - v_y(i,j,t) |}{\sum_i \sum_j | v_x(i,j,t) | + | v_y(i,j,t) |} \tag{4.2}$$

4.3 Flow Past a Square Cylinder in Cross Flow

In general, a square cylinder is placed in the channel which has the Poiseuille flow velocity profile. The schematic diagram of the square cylinder placed in the channel is shown in Figure 4.5.

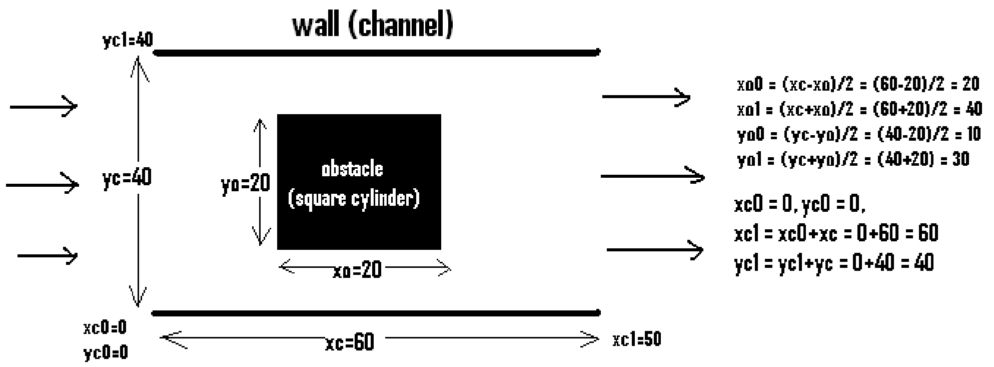
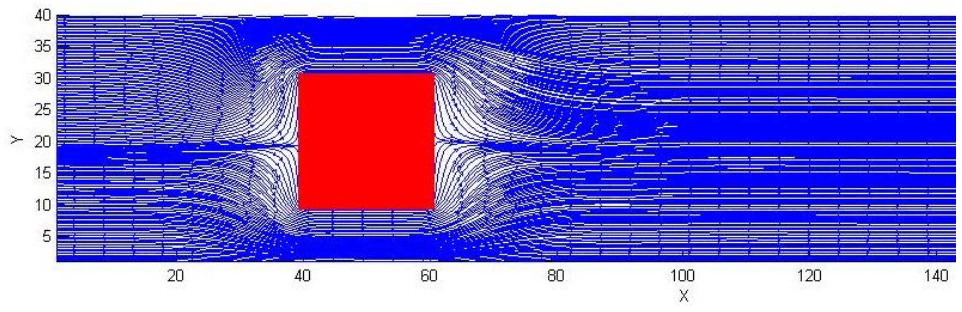


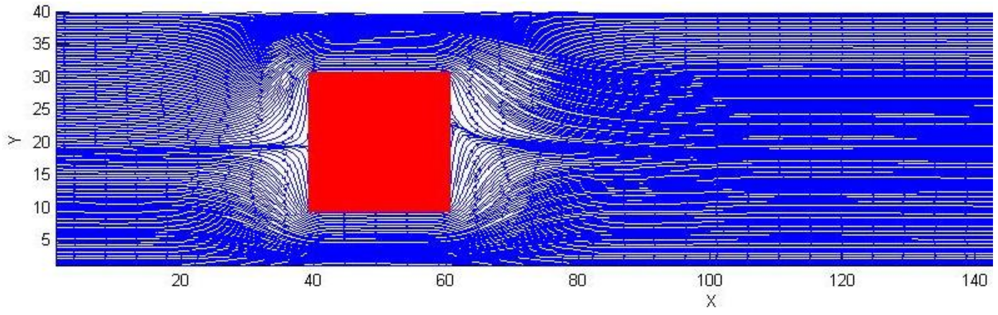
Figure 4.5: Schematic diagram of square cylinder

While running the codes, the value of time relaxation parameter τ, is fixed set to be 0.55. I was reported that from the past investigation by some researchers, the value for τ cannot less than 0.506. Besides, the value for pressure gradient has to be chosen carefully which commonly set less than 0.1. It depends on Reynolds number. Reynolds number is varies from 10 to 1000.

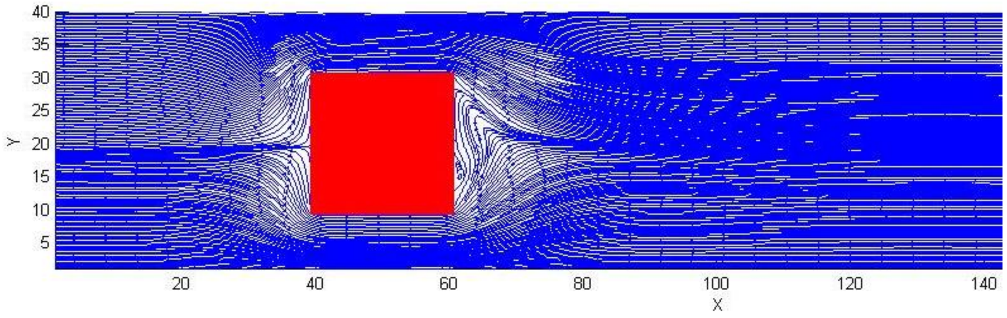
The controlled variable now is Reynolds number. Hence, flow simulates with different Reynolds number. The Reynolds number starts from 50 to 1000. Figure 4.6 show the formation of vortex shedding behind square cylinder with varies Reynolds number.



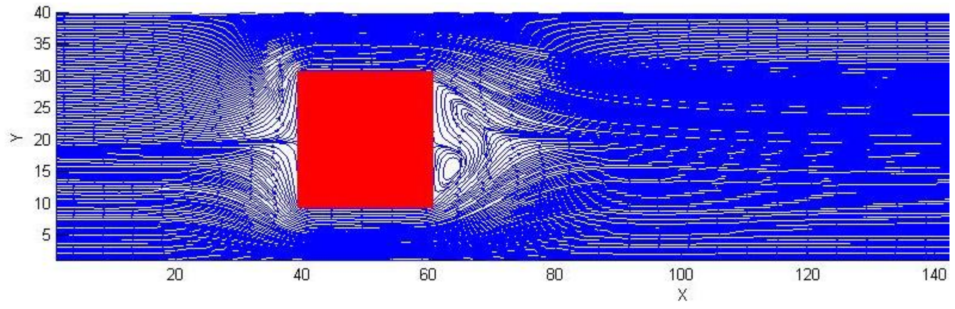
(a) Re = 50



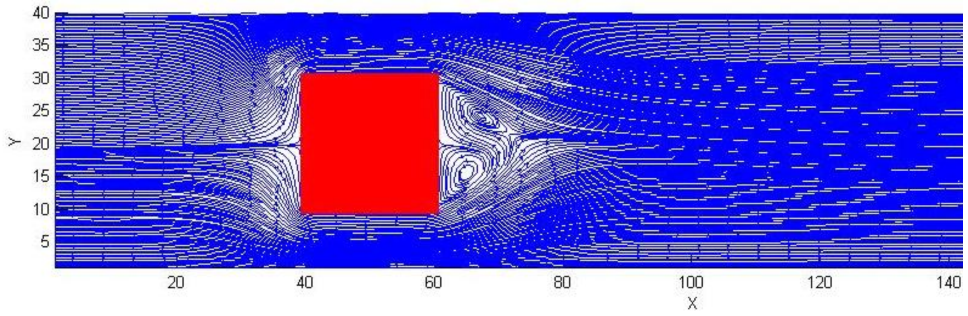
(b) Re = 100



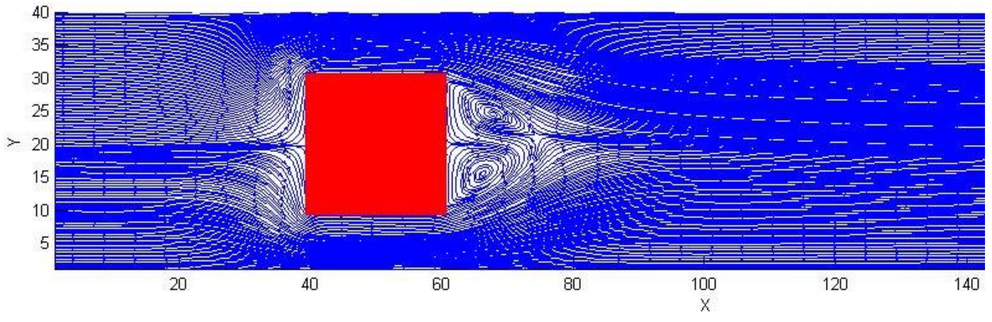
(c) Re = 200



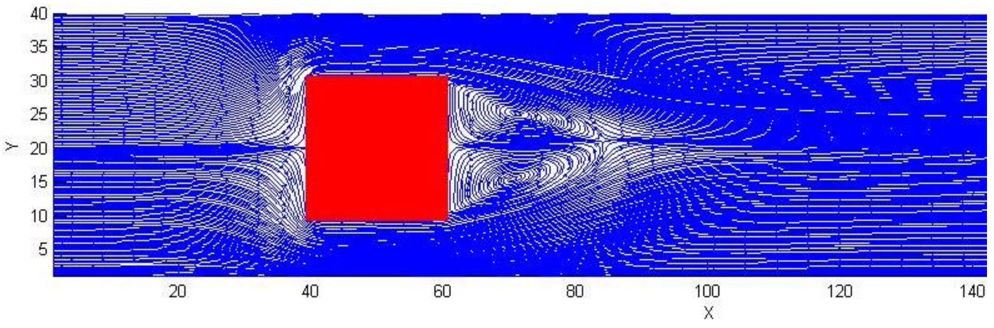
(d) Re = 300



(e) Re = 400



(f) Re = 500



(g) Re = 1000

Figure 4.6: The vortex shedding behind a square cylinder

An observation can be made from above figure that, vortex start to form behind the square cylinder at Re = 300. And it also implies that the higher the Reynolds number, the better vortex formation would be and the longer length of the vortex. However, this simulation can be continuing to higher Reynolds number. The simulation tries to achieve Re = 2000 and the result is in Figure 4.7. The vortex is not only formed behind the square cylinder, it also formed close at wall channel.

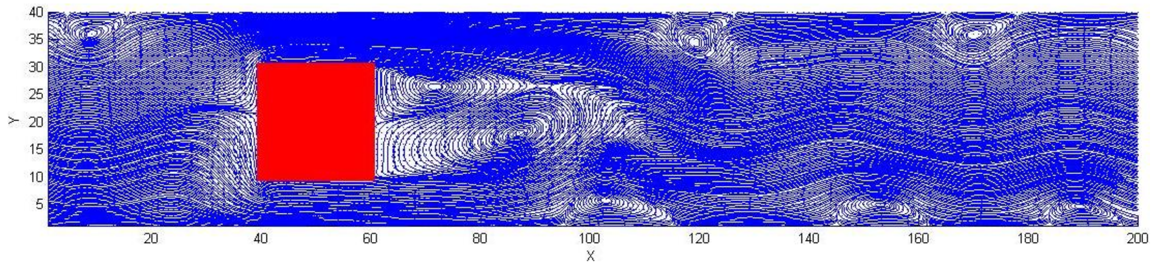


Figure 4.7: Vortex shedding at Re = 2000

Then, this simulation might continue further to higher Reynolds number. But, it would result instability and produced the errors in output file or unexpected result. To prove that statement, Reynolds number increase to Re = 10000. It shown in Figure 4.8.

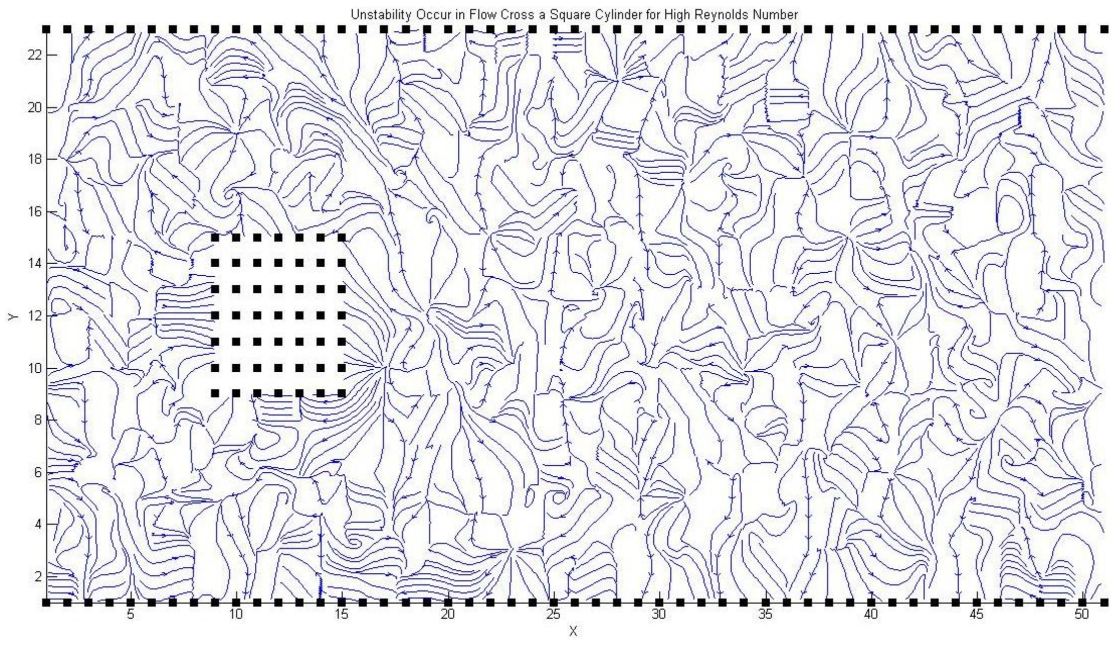


Figure 4.8: Instability occur such Re = 100000

The primary output result obtained in numerical simulation is velocities and densities. The velocities are plotted using streamline function. Cubic interpolated approach is introduced to get the series of streamline. Other particular variable such as pressure and vorticity can be calculated using derivation of primary output. Figure 4.9 shows the density contour.

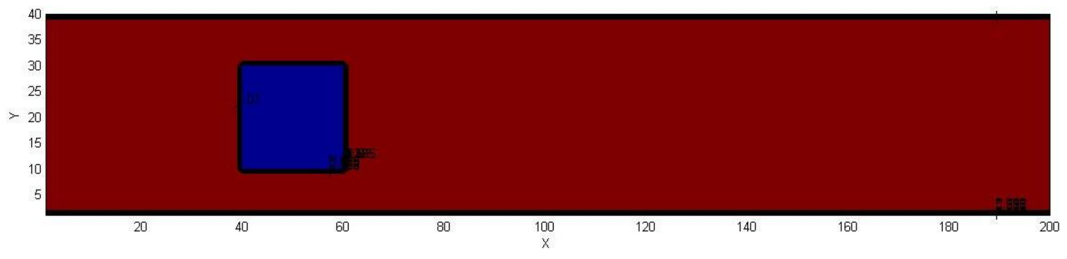


Figure 4.9: Density contour all range of Reynolds number

A significant observation could be made here where there were small variances of density distribution in fluid region. It nearly said to be that, the entire region have the same density. It implies that, these simulations are in incompressible limit. Where, the densities are nearly constant. The pressure experienced the same manner as the pressure relates linearly with density. Figure 4.10 shows the pressure contour.

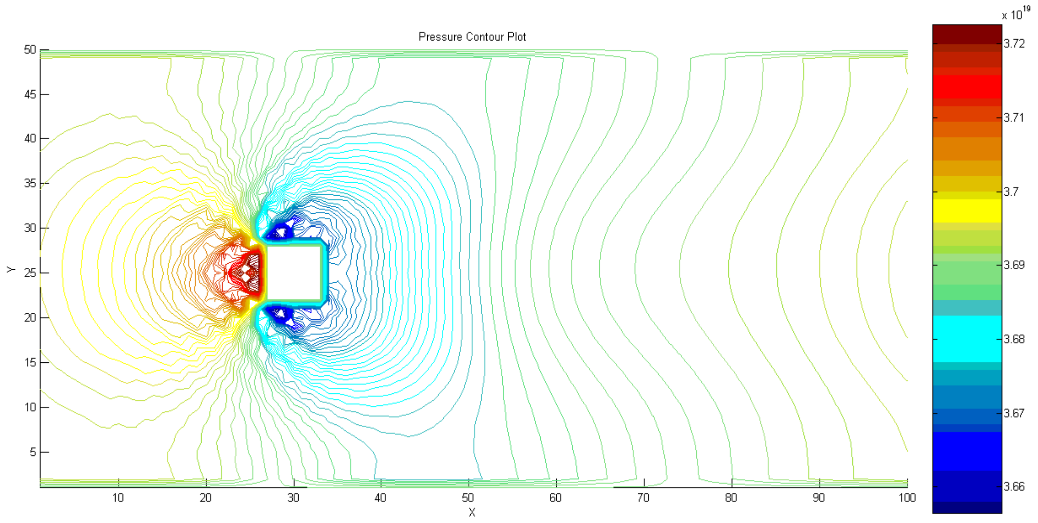
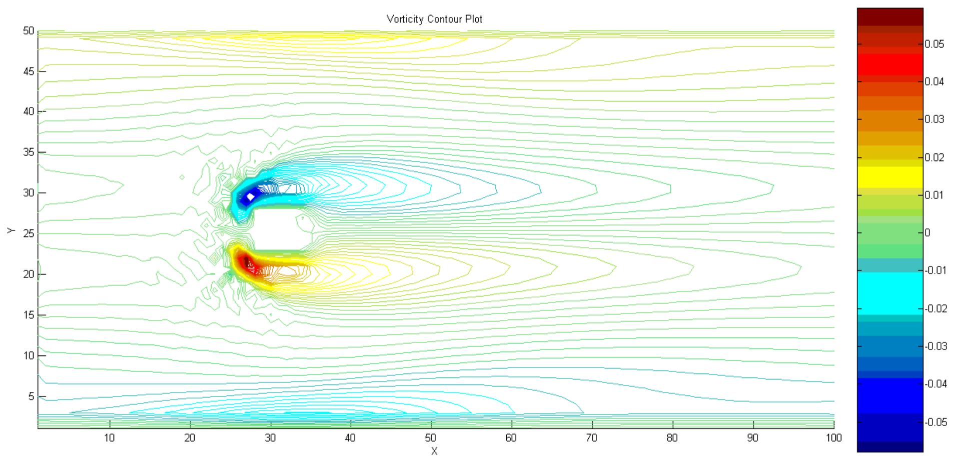
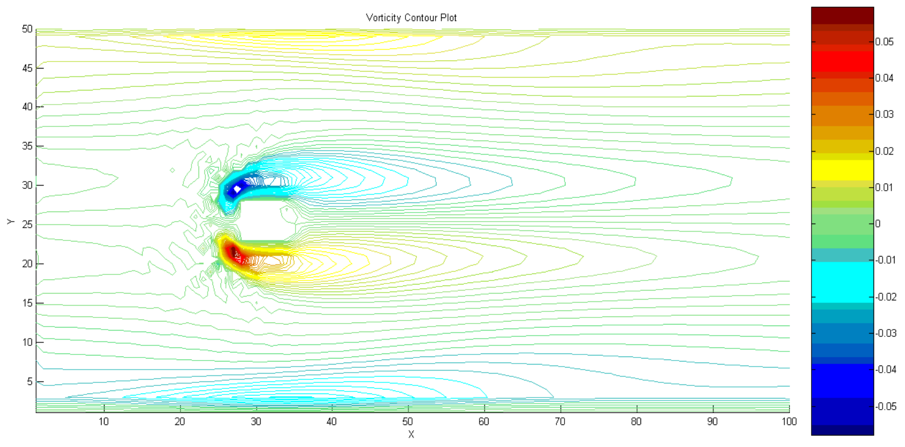


Figure 4.10: Pressure contour for all range of Reynolds number.

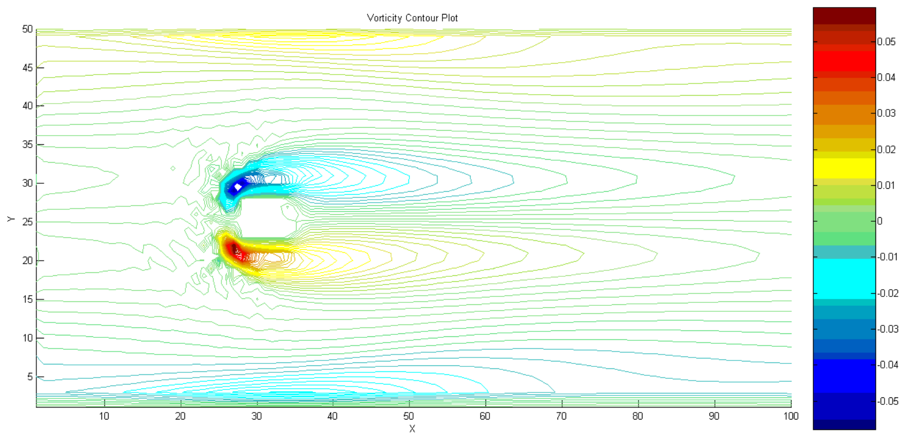
Finally, the vorticity contour for two dimensional flow can be interoperate as follow in Figure 4.11. Since the vorticity function depending on local velocity profiles, it varies with different Reynolds number.



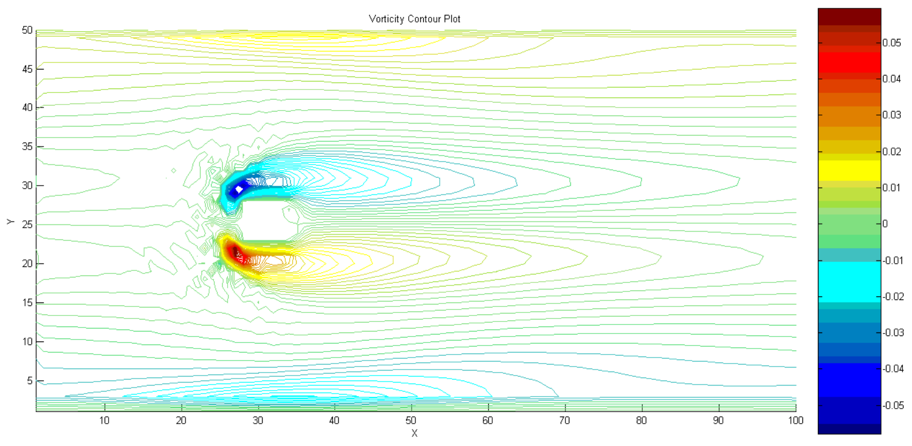
(a) Re = 50



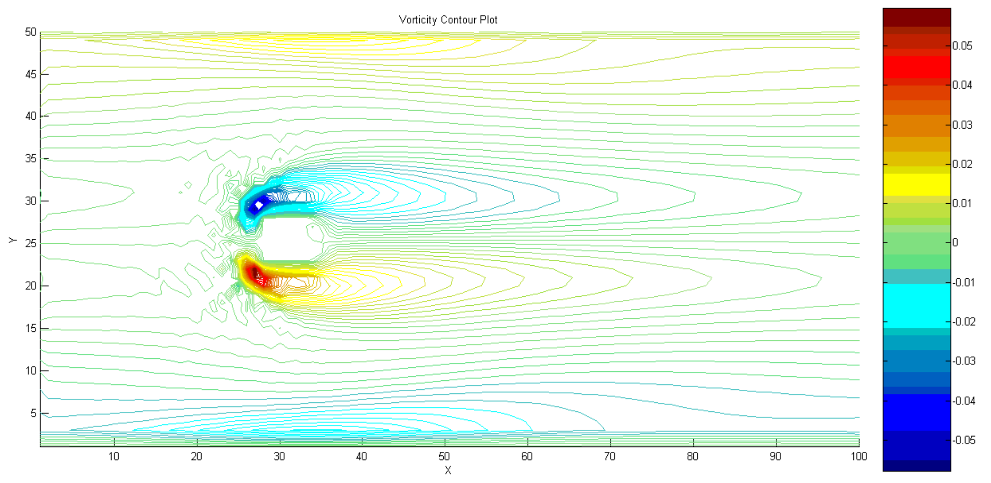
(b) Re = 100



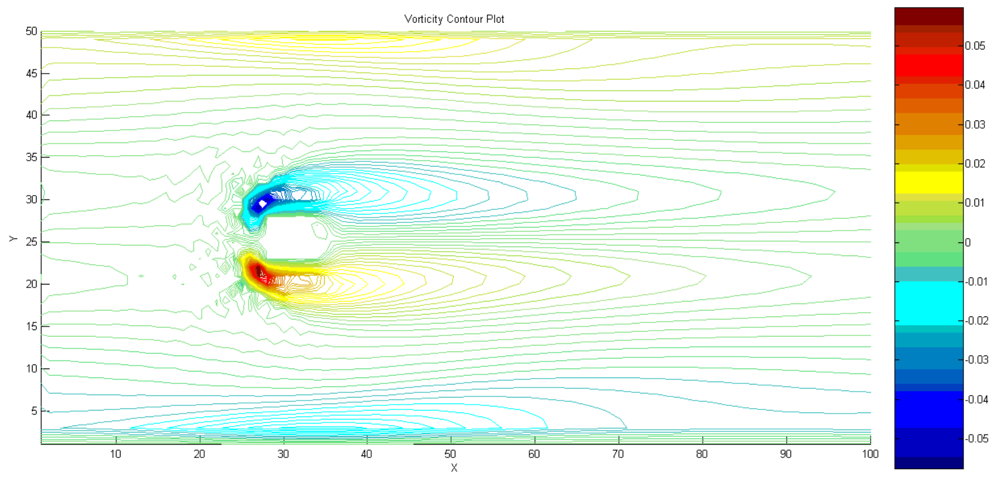
(c) Re = 300



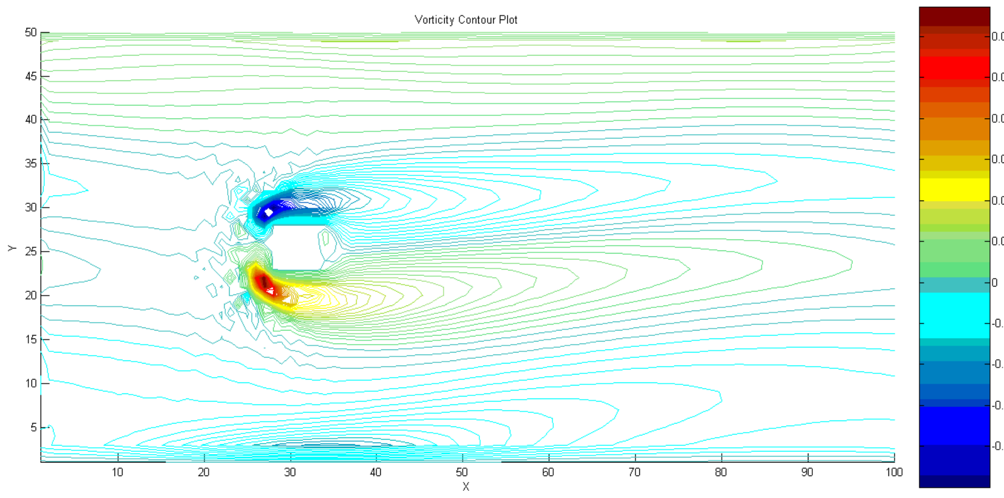
(d) Re = 400



(e) Re = 500



(f) Re = 1000



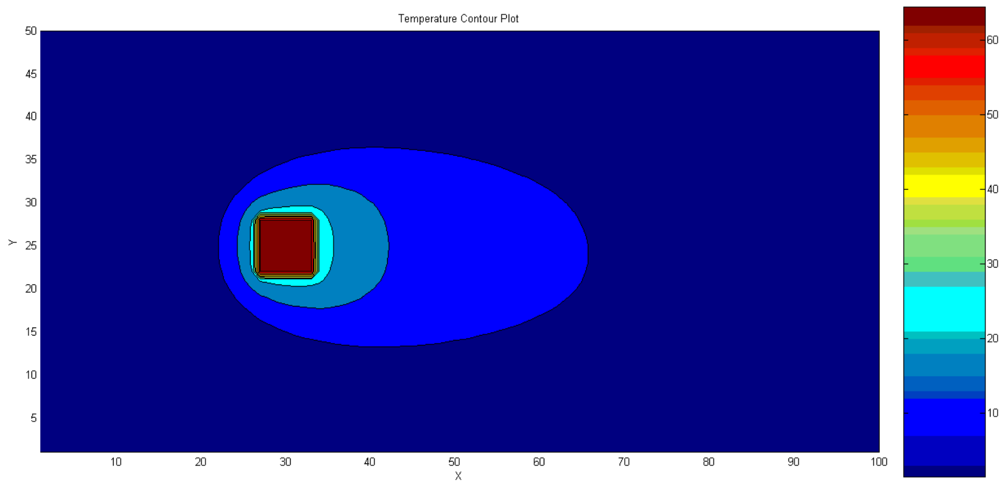
(g) Re = 5000

Figure 4.11: Vorticity contour plot

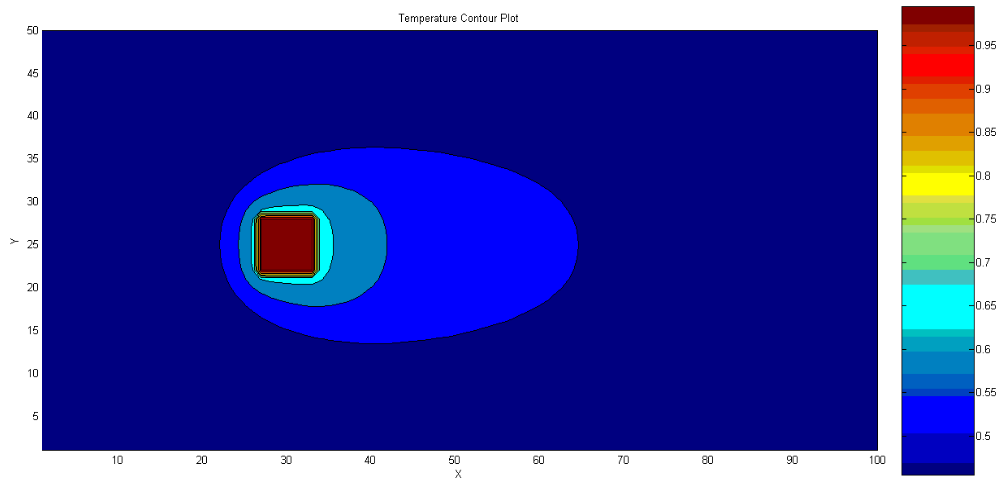
At Reynolds number 5000, the vorticity contour is so huge. It implies that the vortex formed frequently inside the channel. In the figure, the yellow or orange color represents positive vorticity i.e. positive rotation whereas blue color represents the negative value.

4.4 Thermal Flow of a Heated Square Cylinder

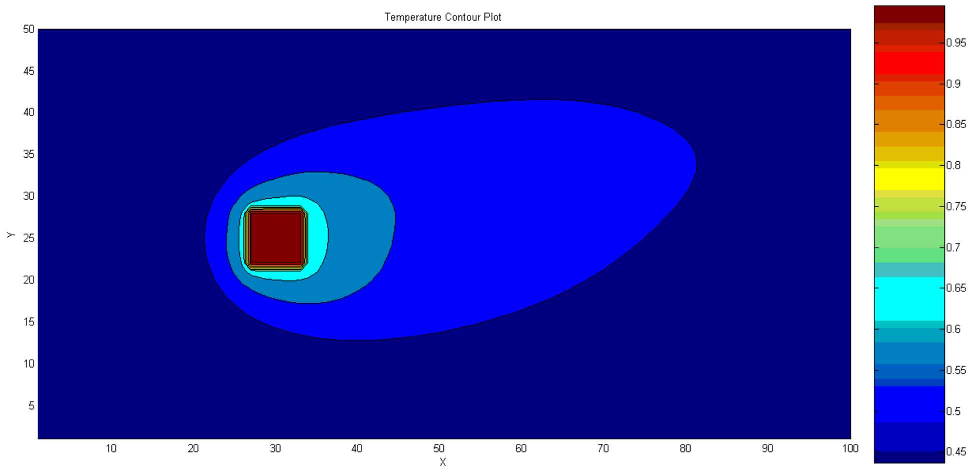
The thermal flow is quite similar to the isothermal but the different here is, isothermal flow is flow at constant temperature and thermal flow is flow with appearance of temperature. Unlike the isothermal flow, the controlled dimensionless parameter now is Rayleigh number. Rayleigh no depends on kinematic viscosity and diffusity therefore it also relates with Reynolds number. Figure 4.12 picture the simulation of temperature during the flows with different Rayleigh number.



(a) Ra = 500



(b) Ra = 1000



(c) Ra = 2000

Figure 4.12: Temperature contour plot

The fluid flows transfer the temperature along with flow directions. This study simulates the thermal flow between Ra = 100 to 1000. However, this simulation code can go further for higher Rayleigh number. For example, in Figure 4.13 the instability take place such a high Rayleigh number

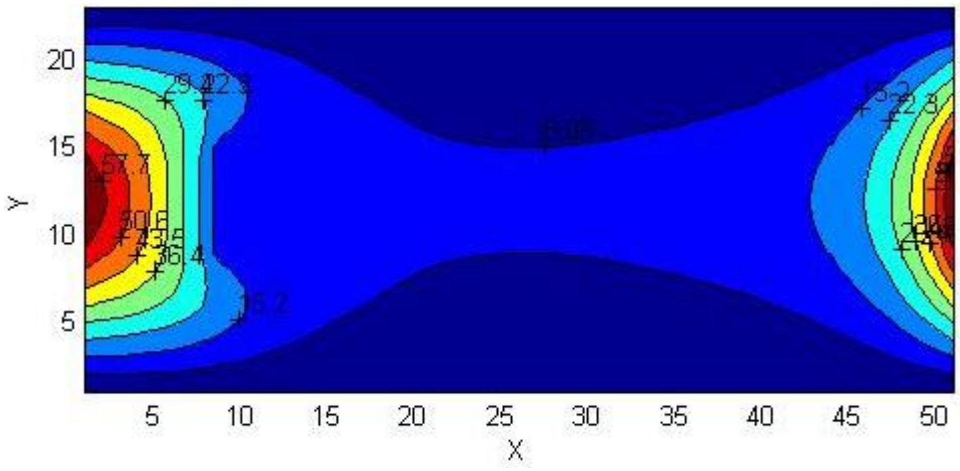


Figure 4.13: Instability in thermal flow such at high Rayleigh number

CHAPTER 5

CONCLUSION AND RECOMENDATION

1.1 Conclusion

LBM had simulated the fluid flow both for isothermal and thermal flow. LBM has a high ability and very powerful to simulate flow in complex geometry and the result is very close to analytical value and yet very simple to implement in parallel programming even though it does not gives the exact value. In fact, there were some errors occurred. However those errors can be reduced by many ways. Some treatment such as a good choice of BC, refining grids and stability analysis, could reduce the errors. The boundary treatment that used in this simulation is an empirical rule which is can be improve to get second order of accuracy. This simulation could be better if more boundary condition introduced to get the second order of accuracy.

In some case, the LBM simulation experienced instability during the simulation. This limitation has been studied by many researchers and found much allowable value for each variable. But, in this study, the value only covers for time relaxation time parameter and velocity ranges or Reynolds number. Recent study discovered by many researchers still not perfect to simulate a flow accurately. Especially in thermal flow, the equation of the LB model for thermal flow nowadays needs to be improved.

1.2 Recommendation

Efficiency of the code for numerical simulation is very important. It should not be too complicated and as simple as possible to protect the simplicity in LBM. The simplicity of LBM should not lost too far. If that so, more storage consumed. This will be automatically, reduce the efficiency of the LBM code. The limit of the LB model such as hydrodinamically limit, incompressible limit resists the research scope of study. Therefore, new LB model with versatile implementation is a need.

Some BC which can be proposed is curvilinear BC. It commonly used for curve side or surface of the boundary. The bounce back BC will behave differently at every point of nodes. Therefore, it improves the numerical accuracies. Further improvement in LBM could be done by using multi-relaxation times. Where, every collision in lattice experience different collision terms. In this study, all collision terms multiplied with same relaxation parameters (single relaxation times).

Performance of the simulation is depending on the computer processor. The faster the computer, the simulation would be more efficient. This study performs using average computer efficiency. Thus, the result take a little bit longer to obtain. Plus, the graphical quality or visualization is not enough to visualize the flow clearly. Nowadays, there is so-called supercomputer used to perform a massive computation including numerical analysis or simulation.

REFFERENCES

S. V. Patankar, “Numerical Heat Transfer and Fluid Flow” Hemisphere Publishing, 1980.

Dieter Wolf-Gladrow, “A Lattice Boltzmann Equation for Diffusion”, Journal of Statistical Physics, 1995.

Xiaoyi He and Li-Shi Luo “A priori derivation of the lattice Boltzmann equation”, Center for Nonlinear Studies, 1997.

Li-Shi Luo, “Analytic Solutions of Linearized Lattice Boltzmann Equation for Simple Flows”, Journal of Statistical Physics, 1997.

Qisu Zou, Shuling Hou and Gary D. Doolen, “Analytical Solutions of the Lattice Boltzmann BGK Model”, Journal of Statistical Physics,1995.

Wolfgang Wagner, “Approximation of the Boltzmann Equation by Discrete Velocity Models”, Journal of Statistical Physics, 1995.

N. G. van Kampen, “Chapman-Enskog as an Application of the Method for Eliminating Fast Variables”, Journal of Statistical Physics, 1987.

Takashi Abe, “Derivation of the Lattice Boltzmann Method by Means of the Discrete Ordinate Method for the Boltzmann Equation”, Academic Press, 1997.

Jonas Tölke, Manfred Krafczyk, Manuel Schulz and Ernst Rank, “Discretization of the Boltzmann equation in velocity space using a Galerkin approach” Technische Universität München, Elsevier Science B.V., 2000.

Xiaowen Shan and Xiaoyi He, “Discretization of the Velocity Space in the Solution of the Boltzmann Equation”, The American Physical Society, 1997.

Shiyi Chen, Gary D. Doolen, and Kenneth G. Eggert, “Lattice-Boltzmann Fluid Dynamics”, Los Alamos Science, 1994.

APPENDIX A
VARIABLES REPRESENTATION

`chi`	Thermal diffusity
`f(i,x,y)`	Density distribution function
`fbuoy(i,x,y)`	Buoyancy force
`fpoi(i,x,y)`	Poiseuille force
`f0(i,x,y)`	Equilibrium density distribution function
`g(i,x,y)`	Internal energy distribution function
`g0(i,x,y)`	Equilibrium internal energy distribution function
`grav`	Gravity force
`grad`	Pressure gradient
`nu`	Kinematic viscosity
`omega`	Collision operator
`p(x,y)`	Pressure
`rho(x,y)`	Density
`t(x,y)`	temperature
`tauv`	Single relaxation time for isothermal
`tauc`	Single relaxation time for thermal
`u(x,y)`	Horizontal component of velocity. Initially set to 0.0
`v(x,y)`	Vertical component of velocity. Initially set to 0.0
`zeta(i,x,y)`	Vorticity function

APPENDIX B

ISOTHERMAL POISEUILLE FLOW SOURCE CODE

```
!Created By:
!Abang Ma'aruf Bin Abang Bussri
!Faculty of Mechanical Engineering
!Universiti Malaysia Pahang
!abang.maaruf@gmail.com
!
!**********************************************************
**************
!Main Program
!**********************************************************
**************
     program poiseuilleiso

     parameter(xmin=1,ymin=1,xmax=100,ymax=50,tmax=10000)

     double precision f(0:8,xmax,ymax),f0(0:8,xmax,ymax)
     double precision
rho(xmax,ymax),u(xmax,ymax),v(xmax,ymax)
     double precision zeta(xmax,ymax),p(xmax,ymax)
     double precision cx(0:8),cy(0:8),w(0:8)
     double precision u0,v0,rho0
     double precision cs2,tau,omega,nu,lc,l,c,re,umax,g

     double precision ut(xmax,ymax),vt(xmax,ymax)
!analytical velocity
     double precision vel(xmax,ymax),vmax,deltav
     double precision err(ymax),error(tmax) !error

     logical wall(xmax,ymax)

     integer t,deltat

     u0=0.0
     v0=0.0
     rho0=1.0
     cs2=(1.0/3.0)
     tau=0.55
     omega=1.0/tau
     nu=cs2*(tau-1.0/2.0)
     lc=(ymax-ymin)/2.0+1
     l=lc-1
     c=(xmax-ymin)/2.0
     re=10.0
     umax=re*nu/(2*l)
     g=2*rho0*nu*umax/l**2.0

     write(*,*)'xmax =',xmax
```

```
      write(*,*)'ymax =',ymax
      write(*,*)'u0 =',u0
      write(*,*)'v0 =',v0
      write(*,*)'rho0 =',rho0
      write(*,*)'cs2 =',cs2
      write(*,*)'tau =',tau
      write(*,*)'omega =',omega
      write(*,*)'nu =',nu
      write(*,*)'l =',l
      write(*,*)'re =',re
      write(*,*)'umax =',umax
      write(*,*)'g =',g
      write(*,*)
      write(*,*)'maxstep =',tmax

      call channel(xmin,ymin,xmax,ymax,wall)
      call macro(cx,cy,w)
      call
init(xmin,ymin,xmax,ymax,f,f0,rho,u,v,rho0,u0,v0,cx,cy,w,
     &wall)

      write(*,*)
      write(*,*) 'enter time step interval'
      read(*,*) deltat

      do t=1,tmax

      call
equilibrium(xmin,ymin,xmax,ymax,f0,rho,u,v,cx,cy,w)
      call collide(xmin,ymin,xmax,ymax,f,f0,omega,wall)
      call stream(xmin,ymin,xmax,ymax,f,cx,cy)
      call bounceback(xmin,ymin,xmax,ymax,f,cx,cy,wall)
      call force(xmin,ymin,xmax,ymax,f,rho,cx,w,g)
      call
calculate(xmin,ymin,xmax,ymax,f,rho,u,v,zeta,p,cx,cy,
     &cs2,lc,l,c,wall,ut,vt,umax,vel,err)

      error(t)=err(ymin)

      do y=ymin,ymax

      if(abs(err(y)) .gt. error(t))then

      error(t) = err(y)

      endif

      enddo

      vmax=vel(xmin,ymin)

      do x=xmin,xmax
      do y=ymin,ymax

      if(abs(vel(x,y)) .gt. vmax)then
      vmax = vel(x,y)

      endif

      enddo
      enddo

      deltav=abs(vmax-deltav)

      if(mod(t,deltat).eq.0)then

      write(*,'(I7,E14.6,E14.6,E14.6)')
 t,vmax,error(t),deltav

      endif

      deltav=vmax

      enddo

      open(unit=70,file='velocity.dat')

      do x=xmin,xmax
      do y=ymin,ymax

      write(70,*) x,y,u(x,y),v(x,y)

      enddo
      enddo

      close(70)

      open(unit=80,file='density.dat')

      do x=xmin,xmax
      do y=ymin,ymax

      write(80,*) x,y,rho(x,y)

      enddo
      enddo

      close(80)

      open(unit=90,file='pressure.dat')

      do x=xmin,xmax
      do y=ymin,ymax

      write(90,*) x,y,p(x,y)

      enddo
      enddo

      close(90)
      open(unit=100,file='vorticity.dat')

      do x=xmin,xmax
      do y=ymin,ymax

      write(100,*) x,y,zeta(x,y)

      enddo
      enddo

      close(110)

      open(unit=110,file='numerical.dat')

      do y=ymin,ymax

      write(110,*) x,y,u(c,y),v(c,y)

      enddo

      close(120)

      open(unit=120,file='analytical.dat')

      do y=ymin,ymax

      write(120,*) x,y,ut(c,y),vt(c,y)

      enddo

      close(120)

      open(unit=130,file='error.dat')

      do t=1,tmax

      write(130,*) t,error(t)

      enddo

      close(130)

      end program poiseuilleiso
 !**********************************************************
 **************
 !Parallel Plate Channel
 !**********************************************************
 **************
      subroutine channel(xmin,ymin,xmax,ymax,wall)

      logical wall(xmax,ymax)

      do x=xmin,xmax
      do y=ymin,ymax

      wall(x,y)=.false.
```

```
      enddo
      enddo

      do x=xmin,xmax

      wall(x,ymin)=.true.
      wall(x,ymax)=.true.

      enddo

      end subroutine channel

!*********************************************************
**************
!Macroscopic Velocity
!*********************************************************
**************
      subroutine macro(cx,cy,w)

      double precision cx(0:8),cy(0:8),w(0:8)
!     double precision theta(0:8),pi

!     pi=4.0*atan(1.0)

      cx(0)= 0.0;cy(0)= 0.0
      cx(1)= 1.0;cy(1)= 0.0
      cx(2)= 0.0;cy(2)= 1.0
      cx(3)=-1.0;cy(3)= 0.0
      cx(4)= 0.0;cy(4)=-1.0
      cx(5)= 1.0;cy(5)= 1.0
      cx(6)=-1.0;cy(6)= 1.0
      cx(7)=-1.0;cy(7)=-1.0
      cx(8)= 1.0;cy(8)=-1.0

      w(0)=4.0/9.0

      do i=1,4

      w(i)=1.0/9.0
!     theta(i)=(i-1.0)*pi/2.0
!     cx(i)=cos(theta(i))
!     cy(i)=sin(theta(i))

      enddo

      do i=5,8

      w(i)=1.0/36.0
!     theta(i)=(i-5.0)*pi/2.0+pi/4.0
!     cx(i)=sqrt(2.0)*cos(theta(i))
!     cy(i)=sqrt(2.0)*sin(theta(i))

      enddo

      end subroutine macro
```

```
!**********************************************************
**************
!Initial Condition
!**********************************************************
**************
     subroutine
init(xmin,ymin,xmax,ymax,f,f0,rho,u,v,rho0,u0,v0,cx
     &,cy,w,wall)

     double precision f(0:8,xmax,ymax),f0(0:8,xmax,ymax)
     double precision
rho(xmax,ymax),u(xmax,ymax),v(xmax,ymax)
     double precision rho0,u0,v0
     double precision cx(0:8),cy(0:8),w(0:8)

     logical wall(xmax,ymax)

     do x=xmin,xmax
     do y=ymin,ymax

     rho(x,y)=rho0
     u(x,y)=u0
     v(x,y)=v0

     enddo
     enddo

     call
equilibrium(xmin,ymin,xmax,ymax,f0,rho,u,v,cx,cy,w)

     do i=0,8
     do x=xmin,xmax
     do y=ymin,ymax

     f(i,x,y)=f0(i,x,y)

     enddo
     enddo
     enddo

     end subroutine init
!**********************************************************
**************
!Equilibrium Density Distribution Function
!**********************************************************
**************
     subroutine
equilibrium(xmin,ymin,xmax,ymax,f0,rho,u,v,cx,cy,w)

     double precision f0(0:8,xmax,ymax)
     double precision
rho(xmax,ymax),u(xmax,ymax),v(xmax,ymax)
     double precision cx(0:8),cy(0:8),w(0:8)

     do x=xmin,xmax
     do y=ymin,ymax
     do i=0,8

     f0(i,x,y)=rho(x,y)*w(i)*(1.0+3.0*(cx(i)*u(x,y)+cy(i)*
v(x,y))+9.0
     &          /2.0*(cx(i)*u(x,y)+cy(i)*v(x,y))**2.0-
3.0/2.0*(u(x,y)
     &          **2.0+v(x,y)**2.0))

     enddo
     enddo
     enddo

     end subroutine equilibrium
!**********************************************************
**************
!Collision Step
!**********************************************************
**************
     subroutine
collide(xmin,ymin,xmax,ymax,f,f0,omega,wall)

     double precision f(0:8,xmax,ymax),f0(0:8,xmax,ymax)
     double precision omega
     double precision ftemp(0:8,xmax,ymax)

     logical wall(xmax,ymax)

     do i=0,8
     do x=xmin,xmax
     do y=ymin,ymax

     ftemp(i,x,y)=f(i,x,y)

     enddo
     enddo
     enddo

     do i=0,8
     do x=xmin,xmax
     do y=ymin,ymax

     if(.not.wall(x,y))then

     f(i,x,y)=(1.0-omega)*ftemp(i,x,y)+omega*f0(i,x,y)

     endif

     enddo
     enddo
     enddo


     end subroutine collide
!**********************************************************
**************
!Streaming Step
```

```
!*********************************************************
**************
      subroutine stream(xmin,ymin,xmax,ymax,f,cx,cy)

      double precision f(0:8,xmax,ymax)
      double precision cx(0:8),cy(0:8)
      double precision ftemp(0:8,xmax,ymax)

      do i=0,8
      do x=xmin,xmax
      do y=ymin,ymax

      ftemp(i,x,y)=f(i,x,y)

      enddo
      enddo
      enddo

      do i=0,8
      if(i.eq.0)then
      do x=xmin,xmax
      do y=ymin,ymax

      f(i,x,y)=ftemp(i,x,y)

      enddo
      enddo

      else if(i.eq.1)then
      do x=xmin,xmax-1
      do y=ymin,ymax

      f(i,x+1,y)=ftemp(i,x,y)

      f(i,xmin,y)=ftemp(i,xmax,y)!periodic

      enddo
      enddo

      else if(i.eq.2)then
      do x=xmin,xmax
      do y=ymin,ymax-1

      f(i,x,y+1)=ftemp(i,x,y)

!     f(i,x,ymin)=ftemp(4,x,ymin+1)!bounceback

      enddo
      enddo

      else if(i.eq.3)then
      do x=xmin+1,xmax
      do y=ymin,ymax

      f(i,x-1,y)=ftemp(i,x,y)
```

```
        f(i,xmax,y)=ftemp(i,xmin,y)!periodic

        enddo
        enddo

        else if(i.eq.4)then
        do x=xmin,xmax
        do y=ymin+1,ymax

        f(i,x,y-1)=ftemp(i,x,y)

!       f(i,x,ymax)=ftemp(2,x,ymax-1)!bounceback

        enddo
        enddo

        else if(i.eq.5)then
        do x=xmin,xmax-1
        do y=ymin,ymax-1

        f(i,x+1,y+1)=ftemp(i,x,y)

        f(i,xmin,y+1)=ftemp(i,xmax,y)!periodic

!       f(i,x,ymin)=ftemp(7,x+1,ymin+1)!bounceback

!
        f(i,xmin,ymin)=ftemp(7,xmax,ymin+1)!periodic+bounceba
ck

        enddo
        enddo

        else if(i.eq.6)then
        do x=xmin+1,xmax
        do y=ymin,ymax-1

        f(i,x-1,y+1)=ftemp(i,x,y)

        f(i,xmax,y+1)=ftemp(i,xmin,y)!periodic

!       f(i,x,ymin)=ftemp(8,x-1,ymin+1)!bounceback

!
        f(i,xmax,ymin)=ftemp(8,xmin,ymin+1)!periodic+bounceba
ck

        enddo
        enddo

        else if(i.eq.7)then
        do x=xmin+1,xmax
        do y=ymin+1,ymax

        f(i,x-1,y-1)=ftemp(i,x,y)
```

```
     f(i,xmax,y-1)=ftemp(i,xmin,y)!periodic

!    f(i,x,ymax)=ftemp(5,x-1,ymax-1)!bounceback

!    f(i,xmax,ymax)=ftemp(5,xmin,ymax-
1)!periodic+bounceback

     enddo
     enddo

     else if(i.eq.8)then
     do x=xmin,xmax-1
     do y=ymin+1,ymax

     f(i,x+1,y-1)=ftemp(i,x,y)

     f(i,xmin,y-1)=ftemp(i,xmax,y)!periodic

!    f(i,x+1,ymax)=ftemp(6,x,ymax-1)!bounceback

!    f(i,xmin,ymax)=ftemp(6,xmax,ymax-
1)!periodic+bounceback

     enddo
     enddo

     endif
     enddo

     f(5,xmin,ymin)=ftemp(5,xmax,ymax)!inlet bottom corner
     f(8,xmin,ymax)=ftemp(8,xmax,ymin)!inlet top corner
     f(6,xmax,ymin)=ftemp(6,xmin,ymax)!outlet bottom
corner
     f(7,xmax,ymax)=ftemp(7,xmin,ymin)!outlet top corner

     end subroutine stream
!*********************************************************
**************
!No-Slip Boundary Condition
!*********************************************************
**************
     subroutine
bounceback(xmin,ymin,xmax,ymax,f,cx,cy,wall)

     double precision f(0:8,xmax,ymax)
     double precision cx(0:8),cy(0:8)
     double precision ftemp(0:8,xmax,ymax)

     logical wall(xmax,ymax)

     do i=0,8
     do x=xmin,xmax
     do y=ymin,ymax

     ftemp(i,x,y)=f(i,x,y)
```

```
      enddo
      enddo
      enddo

      do x=xmin,xmax

      f(2,x,ymin)=ftemp(4,x,ymin)
      f(4,x,ymax)=ftemp(2,x,ymax)
      f(5,x,ymin)=ftemp(7,x,ymin)
      f(6,x,ymin)=ftemp(8,x,ymin)
      f(7,x,ymax)=ftemp(5,x,ymax)
      f(8,x,ymax)=ftemp(6,x,ymax)

      enddo

!     do x=xmin,xmax
!     do y=ymin,ymax

!     if(wall(x,y))then

!     f(0,x,y)=ftemp(0,x,y)
!     f(1,x,y)=ftemp(3,x,y)
!     f(2,x,y)=ftemp(4,x,y)
!     f(3,x,y)=ftemp(1,x,y)
!     f(4,x,y)=ftemp(2,x,y)
!     f(5,x,y)=ftemp(7,x,y)
!     f(6,x,y)=ftemp(8,x,y)
!     f(7,x,y)=ftemp(5,x,y)
!     f(8,x,y)=ftemp(6,x,y)
!
!     endif

!     enddo
!     enddo

      end subroutine bounceback
!*********************************************************
**************
!Forcing Terms
!*********************************************************
**************
      subroutine force(xmin,ymin,xmax,ymax,f,rho,cx,w,g)

      double precision f(0:8,xmax,ymax)
      double precision rho(xmax,ymax)
      double precision cx(0:8),w(0:8)
      double precision g
      double precision ftemp(0:8,xmax,ymax)

      do i=0,8
      do x=xmin,xmax
      do y=ymin,ymax

      ftemp(i,x,y)=f(i,x,y)

      enddo
```

```
     enddo
     enddo

     do i=0,8
     do x=xmin,xmax
     do y=ymin,ymax

     f(i,x,y)=ftemp(i,x,y)+3.0*w(i)*rho(x,y)*cx(i)*g

     enddo
     enddo
     enddo

     end subroutine force
!***********************************************************
**************
!Calculate Density,Velocity,Vorticity & Pressure
!***********************************************************
**************
     subroutine
calculate(xmin,ymin,xmax,ymax,f,rho,u,v,zeta,p,cx,cy,
     &cs2,lc,l,c,wall,ut,vt,umax,vel,err)

     double precision f(0:8,xmax,ymax)
     double precision
rho(xmax,ymax),u(xmax,ymax),v(xmax,ymax)
     double precision zeta(xmax,ymax),p(xmax,ymax)
     double precision cx(0:8),cy(0:8)
     double precision rhou(xmax,ymax),rhov(xmax,ymax)
     double precision cs2,lc,l,c,umax

     double precision ut(xmax,ymax),vt(xmax,ymax)
!analytical velocity
     double precision vel(xmax,ymax)
     double precision err(ymax),errtemp(ymax)

     logical wall(xmax,ymax)

     do x=xmin,xmax
     do y=ymin,ymax

     rhou(x,y)=0.0
     rhov(x,y)=0.0
     rho(x,y)=f(0,x,y)

     do i=1,8

     rhou(x,y)=rhou(x,y)+f(i,x,y)*cx(i)
     rhov(x,y)=rhov(x,y)+f(i,x,y)*cy(i)
     rho(x,y)=rho(x,y)+f(i,x,y)

     enddo
     enddo
     enddo

     do x=xmin,xmax
     do y=ymin,ymax

     if(wall(x,y))then

     rho(x,y)=0.0
     u(x,y)=0.0
     v(x,y)=0.0

     else

     if(rho(x,y).ne.0.)then

     u(x,y)=rhou(x,y)/rho(x,y)
     v(x,y)=rhov(x,y)/rho(x,y)

     else

     u(x,y)=0.0
     v(x,y)=0.0

     endif

     endif

     enddo
     enddo

     do x=xmin+1,xmax-1
     do y=ymin+1,ymax-1

     if(wall(x,y).or.wall(xmin,y).or.wall(xmax,y).or.wall(
x,ymin).or.wa

&ll(x,ymax).or.wall(xmin,ymin).or.wall(xmax,ymin).or.wall(
xmin,ymax
     &).or.wall(xmax,ymax))then

     zeta(x,y)=((v(x+1,y)-v(x-1,y))-(u(x,y+1)-u(x,y-
1)))/xmax

     zeta(xmin,y)=((v(xmin+1,y)-v(xmax,y))-
     &             (u(xmin,y+1)-u(xmin,y-1)))/xmax

     zeta(xmax,y)=((v(xmin,y)-v(xmax-1,y))-
     &             (u(xmax,y+1)-u(xmax,y-1)))/xmax

     zeta(x,ymin)=((v(x+1,ymin)-v(x-1,ymin))-
     &             (u(x,ymin+1)-u(x,ymax)))/xmax

     zeta(x,ymax)=((v(x+1,ymax)-v(x-1,ymax))-
     &             (u(x,ymin)-u(x,ymax-1)))/xmax

     zeta(xmin,ymin)=((v(xmin+1,ymin)-v(xmax,ymin))-
     &                (u(xmin,ymin+1)-u(xmin,ymax)))/xmax

     zeta(xmax,ymin)=((v(xmin,ymin)-v(xmax-1,ymin))-
     &                (u(xmax,ymin+1)-u(xmax,ymax)))/xmax

     zeta(xmin,ymax)=((v(xmin+1,ymax)-v(xmax,ymax))-
     &                (u(xmin,ymin)-u(xmin,ymax-1)))/xmax

     zeta(xmax,ymax)=((v(xmin,ymax)-v(xmax-1,ymax))-
     &                (u(xmax,ymin)-u(xmax,ymax-1)))/xmax

     else

     zeta(x,y)=0.5*((v(x+1,y)-v(x-1,y))-(u(x,y+1)-u(x,y-
1)))/xmax

     zeta(xmin,y)=0.5*((v(xmin+1,y)-v(xmax,y))-
     &             (u(xmin,y+1)-u(xmin,y-1)))/xmax

     zeta(xmax,y)=0.5*((v(xmin,y)-v(xmax-1,y))-
     &             (u(xmax,y+1)-u(xmax,y-1)))/xmax

     zeta(x,ymin)=0.5*((v(x+1,ymin)-v(x-1,ymin))-
     &             (u(x,ymin+1)-u(x,ymax)))/xmax

     zeta(x,ymax)=0.5*((v(x+1,ymax)-v(x-1,ymax))-
     &             (u(x,ymin)-u(x,ymax-1)))/xmax

     zeta(xmin,ymin)=0.5*((v(xmin+1,ymin)-v(xmax,ymin))-
     &                (u(xmin,ymin+1)-u(xmin,ymax)))/xmax

     zeta(xmax,ymin)=0.5*((v(xmin,ymin)-v(xmax-1,ymin))-
     &                (u(xmax,ymin+1)-u(xmax,ymax)))/xmax

     zeta(xmin,ymax)=0.5*((v(xmin+1,ymax)-v(xmax,ymax))-
     &                (u(xmin,ymin)-u(xmin,ymax-1)))/xmax

     zeta(xmax,ymax)=0.5*((v(xmin,ymax)-v(xmax-1,ymax))-
     &                (u(xmax,ymin)-u(xmax,ymax-1)))/xmax

     endif

     enddo
     enddo

     do x=xmin,xmax
     do y=ymin,ymax

     p(x,y)=cs2*rho(x,y)

     enddo
     enddo

     do y=ymin,ymax

     ut(c,y)=umax*(1.0-((y-lc)/l)**2.0)
     vt(c,y)=0.0

     enddo
```

```
      do x=xmin,xmax
      do y=ymin,ymax

      vel(x,y)=sqrt(u(x,y)**2.0+v(x,y)**2.0)

      enddo
      enddo

      do y=ymin,ymax

      err(y)=sqrt((ut(c,y)-u(c,y))**2.0+(vt(c,y)-
v(c,y))**2.0)/umax*100

      enddo

      end subroutine calculate
```

APPENDIX C

ISOTHERMAL FLOW OF SQUARE CYLINDER IN CROSS FLOW

```
!Created by:
!Abang Ma'aruf Bin Abang Bussri
!Faculty of Mechanical Engineering
!Universiti Malaysia Pahang
!November 2009
!abang.maaruf@gmail.com
!
!main program
program isothermal

parameter(xmin=1,ymin=1,xmax=100,ymax=50,xdim=xmax+1,ydim=
ymax+1,tsmax=10000)

double precision
f(0:8,0:xdim,0:ydim),f0(0:8,0:xdim,0:ydim)
double precision fpoi(0:8,0:xdim,0:ydim)
double precision
rho(0:xdim,0:ydim),u(0:xdim,0:ydim),v(0:xdim,0:ydim),p(0:x
dim,0:ydim)
double precision cx(0:8),cy(0:8),w(0:8)
double precision zeta(0:xdim,0:ydim)
double precision tau,omega
double precision rho0,u0,v0
double precision cs2,nu,re,umax,grad

logical wall(xmin:xmax,ymin:ymax)
logical cyl(xmin:xmax,ymin:ymax)

call
initial(xmin,ymin,xmax,ymax,xdim,ydim,f,f0,rho,u,v,cx,cy,w
,rho0,u0,v0,omega,cs2,nu,re,umax,grad,wall,cyl)

write(*,*) 'Reynolds Number = ' ,re

write(*,*) 'time step size for display?'
read(*,*) deltat

!iteration start
do ts=0,tsmax

call
collide(xmin,ymin,xmax,ymax,xdim,ydim,f,f0,omega,wall,cyl)
call stream(xmin,ymin,xmax,ymax,xdim,ydim,f,cx,cy)
call
boundary(xmin,ymin,xmax,ymax,xdim,ydim,f,cx,cy,wall,cyl)
call force(xmin,ymin,xmax,ymax,xdim,ydim,f,fpoi,u,v,cx,cy)
```

```
call
compute(xmin,ymin,xmax,ymax,xdim,ydim,f,fpoi,rho,u,v,p,cx,
cy,w,zeta,rho0,cs2,grad,wall,cyl)

!time step elapsed for display in console
if(mod(ts,deltat).eq.0)then

write(*,*) ts

endif

!recompute equilibrium distribution function
call
equilibrium(xmin,ymin,xmax,ymax,xdim,ydim,f0,rho,u,v,cx,cy
,w)

enddo

!create data file for velocity profile
open(unit=10,file='velocity.dat',status='replace',action='
write')

do x=xmin,xmax
do y=ymin,ymax

write(10,*) x,y,u(x,y),v(x,y)

enddo
enddo

close(10)

!create data file for vorticity
open(unit=30,file='vorticity.dat',status='replace',action=
'write')

do x=xmin,xmax
do y=ymin,ymax

write(30,*) x,y,zeta(x,y)

enddo
enddo

close(30)

!create data file for local pressure
open(unit=40,file='pressure.dat',status='replace',action='
write')

do x=xmin,xmax
do y=ymin,ymax

write(40,*) x,y,p(x,y)

enddo
```

```
enddo

close(40)

!creat data file channel and cylinder coordinate for
plotting
open(unit=50,file='obstacle.dat',status='replace',action='
write')

do x=xmin,xmax
do y=ymin,ymax

if(wall(x,y).or.cyl(x,y))then

write(50,*) x,y

endif

enddo
enddo

close(50)

endprogram isothermal

!initialization
subroutine
initial(xmin,ymin,xmax,ymax,xdim,ydim,f,f0,rho,u,v,cx,cy,w
,rho0,u0,v0,omega,cs2,nu,re,umax,grad,wall,cyl)

double precision
f(0:8,0:xdim,0:ydim),f0(0:8,0:xdim,0:ydim)
double precision
rho(0:xdim,0:ydim),u(0:xdim,0:ydim),v(0:xdim,0:ydim)
double precision cx(0:8),cy(0:8),w(0:8)
!double precision theta(0:8),pi
double precision rho0,u0,v0,omega
double precision a,b,xc,yc,x1,x2,y1,y2,l,umax
double precision cs2,nu,re,grad

logical wall(xmin:xmax,ymin:ymax)
logical cyl(xmin:xmax,ymin:ymax)

!cylinder profiles
a=20.0
b=a
xc=30.0
yc=(ymax-ymin)/2.0+1.0
x1=xc-a/2.0
x2=x1+a
y1=yc-b/2.0
y2=y1+b

!single time relaxation
tau=0.55
omega=1.0/tau
```

```
cs2=1.0/3.0 !square speed of sound (lattice constant)
nu=cs2*(tau-1.0/2.0) !kinematic viscosity

!controlled dimensionless parameter
re=100.0 !reynolds number

!initial condition
rho0=1.0
u0=0.0
v0=0.0

rho(xmin:xmax,ymin:ymax)=rho0
u(xmin:xmax,ymin:ymax)=u0
v(xmin:xmax,ymin:ymax)=v0

l=(ymax-ymin)/2.0
umax=re*nu/(2.0*l)
grad=2.0*rho0*nu*umax/l**2.0 !pressure gradian, g=-dp/dx


!pi=4.0*atan(1.0)

!cx(0)= 0.0
!cy(0)=0.0

!do i=1,4

!theta(i)=(i-1.0)*pi/2.0
!cx(i)=cos(theta(i))
!cy(i)=sin(theta(i))

!enddo

!do i=5,8

!theta(i)=(i-5.0)*pi/2.0+pi/4.0
!cx(i)=sqrt(2.0)*cos(theta(i))
!cy(i)=sqrt(2.0)*sin(theta(i))

!enddo

!macroscopic velocities
cx(0)= 0.0;cy(0)= 0.0
cx(1)= 1.0;cy(1)= 0.0
cx(2)= 0.0;cy(2)= 1.0
cx(3)=-1.0;cy(3)= 0.0
cx(4)= 0.0;cy(4)=-1.0
cx(5)= 1.0;cy(5)= 1.0
cx(6)=-1.0;cy(6)= 1.0
cx(7)=-1.0;cy(7)=-1.0
cx(8)= 1.0;cy(8)=-1.0

!weight coefficient
w(0)=4.0/9.0
w(1:4)=1.0/9.0
```

```
w(5:8)=1.0/36.0

!define solid node boundaries
wall(xmin:xmax,ymin:ymax)=.false.
cyl(xmin:xmax,ymin:ymax)=.false.

wall(xmin:xmax,ymin)=.true.
wall(xmin:xmax,ymax)=.true.
cyl(x1:x2,y1:y2)=.true.

call
equilibrium(xmin,ymin,xmax,ymax,xdim,ydim,f0,rho,u,v,cx,cy
,w)

!initial density distribution function
f(0:8,xmin:xmax,ymin:ymax)=f0(0:8,xmin:xmax,ymin:ymax)

endsubroutine initial

!equilibrium distribution function
subroutine
equilibrium(xmin,ymin,xmax,ymax,xdim,ydim,f0,rho,u,v,cx,cy
,w)

double precision f0(0:8,0:xdim,0:ydim)
double precision
rho(0:xdim,0:ydim),u(0:xdim,0:ydim),v(0:xdim,0:ydim)
double precision cx(0:8),cy(0:8),w(0:8)

do x=xmin,xmax
do y=ymin,ymax
do i=0,8

f0(i,x,y)=w(i)*rho(x,y)*(1.0+3.0*(cx(i)*u(x,y)+cy(i)*v(x,y
))+(9.0/2.0)*(cx(i)*u(x,y)+cy(i)*v(x,y))**2.0-
(3.0/2.0)*(u(x,y)**2.0+v(x,y)**2.0))

enddo
enddo
enddo

endsubroutine equilibrium

!collision step
subroutine
collide(xmin,ymin,xmax,ymax,xdim,ydim,f,f0,omega,wall,cyl)

double precision
f(0:8,0:xdim,0:ydim),f0(0:8,0:xdim,0:ydim),ftemp(0:8,0:xdi
m,0:ydim)
double precision omegav,omegac

logical wall(xmin:xmax,ymin:ymax)
logical cyl(xmin:xmax,ymin:ymax)

ftemp(0:8,xmin:xmax,ymin:ymax)=f(0:8,xmin:xmax,ymin:ymax)
```

```
do x=xmin,xmax
do y=ymin,ymax
do i=0,8

!only fluid node are considered
if(.not.(wall(x,y).or.cyl(x,y)))then

f(i,x,y)=(1.0-omega)*ftemp(i,x,y)+omega*f0(i,x,y)

endif

enddo
enddo
enddo

endsubroutine collide

!streaming step
subroutine stream(xmin,ymin,xmax,ymax,xdim,ydim,f,cx,cy)

double precision
f(0:8,0:xdim,0:ydim),ftemp(0:8,0:xdim,0:ydim)
double precision cx(0:8),cy(0:8)

ftemp(0:8,xmin:xmax,ymin:ymax)=f(0:8,xmin:xmax,ymin:ymax)

do x=xmin,xmax
do y=ymin,ymax
do i=0,8

f(i,x+cx(i),y+cy(i))=ftemp(i,x,y)

enddo
enddo
enddo

!periodic boundary condition
do y=ymin,ymax

!west
f(1,xmin,y+cy(1))=ftemp(1,xmax,y)
f(5,xmin,y+cy(5))=ftemp(5,xmax,y)
f(8,xmin,y+cy(8))=ftemp(8,xmax,y)

!east
f(3,xmax,y+cy(3))=ftemp(3,xmin,y)
f(6,xmax,y+cy(6))=ftemp(6,xmin,y)
f(7,xmax,y+cy(7))=ftemp(7,xmin,y)

enddo

do x=xmin,xmax

!south
f(2,x+cx(2),ymin)=ftemp(2,x,ymax)
```

```
f(5,x+cx(5),ymin)=ftemp(5,x,ymax)
f(6,x+cx(6),ymin)=ftemp(6,x,ymax)

!north
f(4,x+cx(4),ymax)=ftemp(4,x,ymin)
f(7,x+cx(7),ymax)=ftemp(7,x,ymin)
f(8,x+cx(8),ymax)=ftemp(8,x,ymin)

enddo

f(5,xmin,ymin)=ftemp(5,xmax,ymax)!south-west
f(8,xmin,ymax)=ftemp(8,xmax,ymin)!north-west
f(6,xmax,ymin)=ftemp(6,xmin,ymax)!south-east
f(7,xmax,ymax)=ftemp(7,xmin,ymin)!north-east

endsubroutine stream

!bounceback boundary condition
subroutine
boundary(xmin,ymin,xmax,ymax,xdim,ydim,f,cx,cy,wall,cyl)

double precision
f(0:8,0:xdim,0:ydim),ftemp(0:8,0:xdim,0:ydim)
double precision cx(0:8),cy(0:8)

logical wall(xmin:xmax,ymin:ymax)
logical cyl(xmin:xmax,ymin:ymax)

ftemp(0:8,xmin:xmax,ymin:ymax)=f(0:8,xmin:xmax,ymin:ymax)

do x=xmin,xmax
do y=ymin,ymax
do i=0,8

if(wall(x,y))then

f(0,x,y)=ftemp(0,x,y)
f(1,x,y)=ftemp(3,x,y)
f(2,x,y)=ftemp(4,x,y)
f(3,x,y)=ftemp(1,x,y)
f(4,x,y)=ftemp(2,x,y)
f(5,x,y)=ftemp(7,x,y)
f(6,x,y)=ftemp(8,x,y)
f(7,x,y)=ftemp(5,x,y)
f(8,x,y)=ftemp(6,x,y)

endif

enddo
enddo
enddo

endsubroutine boundary

!forcing terms
```

```
subroutine
force(xmin,ymin,xmax,ymax,xdim,ydim,f,fpoi,u,v,cx,cy)

double precision
f(0:8,0:xdim,0:ydim),ftemp(0:8,0:xdim,0:ydim)
double precision fpoi(0:8,0:xdim,0:ydim)
double precision u(0:xdim,0:ydim),v(0:xdim,0:ydim)
double precision cx(0:8),cy(0:8)

ftemp(0:8,xmin:xmax,ymin:ymax)=f(0:8,xmin:xmax,ymin:ymax)

do x=xmin,xmax
do y=ymin,ymax
do i=0,8

f(i,x,y)=ftemp(i,x,y)+fpoi(i,x,y)

enddo
enddo
enddo

endsubroutine force

!compute value of density,velocities,pressure and forces
subroutine
compute(xmin,ymin,xmax,ymax,xdim,ydim,f,fpoi,rho,u,v,p,cx,
cy,w,zeta,rho0,cs2,grad,wall,cyl)

double precision f(0:8,0:xdim,0:ydim)
double precision fpoi(0:8,0:xdim,0:ydim)
double precision
rho(0:xdim,0:ydim),u(0:xdim,0:ydim),v(0:xdim,0:ydim),p(0:x
dim,0:ydim)
double precision
zeta(0:xdim,0:ydim),pstat(0:xdim,0:ydim),pdyn(0:xdim,0:ydi
m)
double precision utemp(0:xdim,0:ydim),vtemp(0:xdim,0:ydim)
double precision rhou(0:xdim,0:ydim),rhov(0:xdim,0:ydim)
double precision cx(0:8),cy(0:8),w(0:8)
double precision rho0
double precision grad

logical wall(xmin:xmax,ymin:ymax)
logical cyl(xmin:xmax,ymin:ymax)

utemp(xmin:xmax,ymin:ymax)=u(xmin:xmax,ymin:ymax)
vtemp(xmin:xmax,ymin:ymax)=v(xmin:xmax,ymin:ymax)

do x=xmin,xmax
do y=ymin,ymax

rho(x,y)=f(0,x,y)
rhou(x,y)=f(0,x,y)*cx(0)
rhov(x,y)=f(0,x,y)*cy(0)

do i=1,8
```

```
rho(x,y)=rho(x,y)+f(i,x,y)
rhou(x,y)=rhou(x,y)+f(i,x,y)*cx(i)
rhov(x,y)=rhov(x,y)+f(i,x,y)*cy(i)

enddo
enddo
enddo

do x=xmin,xmax
do y=ymin,ymax

!no slip u=0,v=0
if(wall(x,y))then

rho(x,y)=rho0
u(x,y)=0.0
v(x,y)=0.0

elseif(cyl(x,y))then

rho(x,y)=rho0
u(x,y)=0.0
v(x,y)=0.0

else

!rho cannot equal to zero
if(rho(x,y).ne.0.)then

u(x,y)=rhou(x,y)/rho(x,y)
v(x,y)=rhov(x,y)/rho(x,y)

else

u(x,y)=0.0
v(x,y)=0.0

endif

endif

enddo
enddo

!pressure driven poiseuille flow
do x=xmin,xmax
do y=ymin,ymax
do i=0,8

fpoi(i,x,y)=3.0*w(i)*grad*rho(x,y)*cx(i)

enddo
enddo
enddo
```

```
!vorticity function
do x=xmin,xmax
do y=ymin,ymax

zeta(x,y)=(v(x,y)-v(x-1.0,y))-(u(x,y)-u(x,y-1.0))

enddo
enddo

!pressure
do x=xmin,xmax
do y=ymin,ymax

pstat(x,y)=cs2*rho(x,y)-grad*(x-xmax)!static pressure
pdyn(x,y)=(1.0/2.0)*rho(x,y)*(u(x,y)**2.0+v(x,y)**2.0)!dyn
amic pressure
p(x,y)=pstat(x,y)+pdyn(x,y)!stagnation pressure

enddo
enddo

endsubroutine compute
```

APPENDIX D

THERMAL FLOW OF HEATED SQUARE CYLINDER IN CROSS FLOW

```
!Created by:
!Abang Ma'aruf Bin Abang Bussri
!Faculty of Mechanical Engineering
!Universiti Malaysia Pahang
!November 2009
!abang.maaruf@gmail.com
!
!main program
program thermal

parameter(xmin=1,ymin=1,xmax=100,ymax=50,xdim=xmax+1,ydim=
ymax+1,tsmax=8000)

double precision
f(0:8,0:xdim,0:ydim),f0(0:8,0:xdim,0:ydim)
double precision
g(0:8,0:xdim,0:ydim),g0(0:8,0:xdim,0:ydim)
double precision
fpoi(0:8,0:xdim,0:ydim),fbuoy(0:8,0:xdim,0:ydim),h(0:8,0:x
dim,0:ydim)
double precision
rho(0:xdim,0:ydim),u(0:xdim,0:ydim),v(0:xdim,0:ydim),t(0:x
dim,0:ydim),p(0:xdim,0:ydim)
double precision cx(0:8),cy(0:8),w(0:8)
double precision zeta(0:xdim,0:ydim)
double precision tauv,tauc,omegav,omegac
double precision rho0,u0,v0,tc,th
double precision cs2,nu,chi,re,umax,ra,pr,grad,grav,beta

logical wall(xmin:xmax,ymin:ymax)
logical cyl(xmin:xmax,ymin:ymax)

call
initial(xmin,ymin,xmax,ymax,xdim,ydim,f,f0,g,g0,rho,u,v,t,
cx,cy,w,rho0,u0,v0,tc,th,omegav,omegac,cs2,nu,chi,re,umax,
ra,pr,grad,grav,beta,wall,cyl)

write(*,*) 'Reynolds Number = ' ,re
write(*,*) 'Rayleigh Number = ' ,ra

write(*,*) 'time step size for display?'
read(*,*) deltat

!iteration start
do ts=0,tsmax
```

```
call
collide(xmin,ymin,xmax,ymax,xdim,ydim,f,g,f0,g0,omegav,ome
gac,wall,cyl)
call stream(xmin,ymin,xmax,ymax,xdim,ydim,f,g,cx,cy)
call
boundary(xmin,ymin,xmax,ymax,xdim,ydim,f,g,cx,cy,wall,cyl)
call
force(xmin,ymin,xmax,ymax,xdim,ydim,f,g,fpoi,fbuoy,h,u,v,c
x,cy)
call
compute(xmin,ymin,xmax,ymax,xdim,ydim,f,g,fpoi,fbuoy,h,rho
,u,v,t,p,cx,cy,w,zeta,rho0,tc,th,cs2,grad,grav,beta,wall,c
yl)

!time step elapsed for display in console
if(mod(ts,deltat).eq.0)then

write(*,*) ts

endif

!recompute equilibrium distribution function
call
equilibrium(xmin,ymin,xmax,ymax,xdim,ydim,f0,g0,rho,u,v,t,
cx,cy,w)

enddo

!create data file for velocity profile
open(unit=10,file='velocity.dat',status='replace',action='
write')

do x=xmin,xmax
do y=ymin,ymax

write(10,*) x,y,u(x,y),v(x,y)

enddo
enddo

close(10)

!create data file for temperature
open(unit=20,file='temperature.dat',status='replace',actio
n='write')

do x=xmin,xmax
do y=ymin,ymax

write(20,*) x,y,t(x,y)

enddo
enddo

close(20)
```

```
!create data file for vorticity
open(unit=30,file='vorticity.dat',status='replace',action=
'write')

do x=xmin,xmax
do y=ymin,ymax

write(30,*) x,y,zeta(x,y)

enddo
enddo

close(30)

!create data file for local pressure
open(unit=40,file='pressure.dat',status='replace',action='
write')

do x=xmin,xmax
do y=ymin,ymax

write(40,*) x,y,p(x,y)

enddo
enddo

close(40)

!creat data file channel and cylinder coordinate for
plotting
open(unit=50,file='obstacle.dat',status='replace',action='
write')

do x=xmin,xmax
do y=ymin,ymax

if(wall(x,y).or.cyl(x,y))then

write(50,*) x,y

endif

enddo
enddo

close(50)

endprogram thermal

!initialization
subroutine
initial(xmin,ymin,xmax,ymax,xdim,ydim,f,f0,g,g0,rho,u,v,t,
cx,cy,w,rho0,u0,v0,tc,th,omegav,omegac,cs2,nu,chi,re,umax,
ra,pr,grad,grav,beta,wall,cyl)
```

```
double precision
f(0:8,0:xdim,0:ydim),f0(0:8,0:xdim,0:ydim)
double precision
g(0:8,0:xdim,0:ydim),g0(0:8,0:xdim,0:ydim)
double precision
rho(0:xdim,0:ydim),u(0:xdim,0:ydim),v(0:xdim,0:ydim),t(0:x
dim,0:ydim)
double precision cx(0:8),cy(0:8),w(0:8)
!double precision theta(0:8),pi
double precision rho0,u0,v0,tc,th,omegav,omegac
double precision a,b,xc,yc,x1,x2,y1,y2,l,umax
double precision cs2,nu,chi,re,ra,pr,grad,grav,beta

logical wall(xmin:xmax,ymin:ymax)
logical cyl(xmin:xmax,ymin:ymax)

!cylinder profiles
a=6.0
b=a
xc=30.0
yc=(ymax-ymin)/2.0+1.0
x1=xc-a/2.0
x2=x1+a
y1=yc-b/2.0
y2=y1+b

!single time relaxation
tauv=0.55
tauc=0.7
omegav=1.0/tauv
omegac=1.0/tauc

cs2=1.0/3.0 !square speed of sound (lattice constant)
nu=cs2*(tauv-1.0/2.0) !dynamic viscosity
chi=cs2*(tauc-1.0/2.0) !thermal diffusity

!controlled dimensionless parameter
re=1000.0 !reynolds number
ra=50000.0 !rayleigh number
pr=nu/chi !prandtl number

!initial condition
rho0=1.0
u0=0.0
v0=0.0
tc=0.0
th=1.0

rho(xmin:xmax,ymin:ymax)=rho0
u(xmin:xmax,ymin:ymax)=u0
v(xmin:xmax,ymin:ymax)=v0
t(xmin:xmax,ymin:ymax)=tc
t(x1:x2,y1:y2)=th

l=(ymax-ymin)/2.0
umax=re*nu/(2.0*l)
```

```
grad=2.0*rho0*nu*umax/l**2.0 !pressure gradian, g=-dp/dx

grav=0.001102
beta=ra*nu*chi/((ymax-ymin)**3.0*grav*(th-tc))

!pi=4.0*atan(1.0)

!cx(0)= 0.0
!cy(0)=0.0

!do i=1,4

!theta(i)=(i-1.0)*pi/2.0
!cx(i)=cos(theta(i))
!cy(i)=sin(theta(i))

!enddo

!do i=5,8

!theta(i)=(i-5.0)*pi/2.0+pi/4.0
!cx(i)=sqrt(2.0)*cos(theta(i))
!cy(i)=sqrt(2.0)*sin(theta(i))

!enddo

!macroscopic velocities
cx(0)= 0.0;cy(0)= 0.0
cx(1)= 1.0;cy(1)= 0.0
cx(2)= 0.0;cy(2)= 1.0
cx(3)=-1.0;cy(3)= 0.0
cx(4)= 0.0;cy(4)=-1.0
cx(5)= 1.0;cy(5)= 1.0
cx(6)=-1.0;cy(6)= 1.0
cx(7)=-1.0;cy(7)=-1.0
cx(8)= 1.0;cy(8)=-1.0

!weight coefficient
w(0)=4.0/9.0
w(1:4)=1.0/9.0
w(5:8)=1.0/36.0

!define solid node boundaries
wall(xmin:xmax,ymin:ymax)=.false.
cyl(xmin:xmax,ymin:ymax)=.false.

wall(xmin:xmax,ymin)=.true.
wall(xmin:xmax,ymax)=.true.
cyl(x1:x2,y1:y2)=.true.

call
equilibrium(xmin,ymin,xmax,ymax,xdim,ydim,f0,g0,rho,u,v,t,
cx,cy,w)

!initial density distribution function
f(0:8,xmin:xmax,ymin:ymax)=f0(0:8,xmin:xmax,ymin:ymax)
```

```
g(0:8,xmin:xmax,ymin:ymax)=g0(0:8,xmin:xmax,ymin:ymax)

endsubroutine initial

!equilibrium distribution function
subroutine
equilibrium(xmin,ymin,xmax,ymax,xdim,ydim,f0,g0,rho,u,v,t,
cx,cy,w)

double precision f0(0:8,0:xdim,0:ydim)
double precision g0(0:8,0:xdim,0:ydim)
double precision
rho(0:xdim,0:ydim),u(0:xdim,0:ydim),v(0:xdim,0:ydim),t(0:x
dim,0:ydim)
double precision cx(0:8),cy(0:8),w(0:8)

do x=xmin,xmax
do y=ymin,ymax
do i=0,8

f0(i,x,y)=w(i)*rho(x,y)*(1.0+3.0*(cx(i)*u(x,y)+cy(i)*v(x,y
))+(9.0/2.0)*(cx(i)*u(x,y)+cy(i)*v(x,y))**2.0-
(3.0/2.0)*(u(x,y)**2.0+v(x,y)**2.0))
g0(i,x,y)=w(i)*rho(x,y)*t(x,y)*((3.0/2.0)*((cx(i)**2.0+cy(
i)**2.0)-
(u(x,y)**2.0+v(x,y)**2.0))+3.0*((3.0/2.0)*(cx(i)**2.0+cy(i
)**2.0)-
1.0)*(cx(i)*u(x,y)+cy(i)*v(x,y))+(9.0/2.0)*(cx(i)*u(x,y)+c
y(i)*v(x,y))**2.0)

enddo
enddo
enddo

endsubroutine equilibrium

!collision step
subroutine
collide(xmin,ymin,xmax,ymax,xdim,ydim,f,g,f0,g0,omegav,ome
gac,wall,cyl)

double precision
f(0:8,0:xdim,0:ydim),f0(0:8,0:xdim,0:ydim),ftemp(0:8,0:xdi
m,0:ydim)
double precision
g(0:8,0:xdim,0:ydim),g0(0:8,0:xdim,0:ydim),gtemp(0:8,0:xdi
m,0:ydim)
double precision omegav,omegac

logical wall(xmin:xmax,ymin:ymax)
logical cyl(xmin:xmax,ymin:ymax)

ftemp(0:8,xmin:xmax,ymin:ymax)=f(0:8,xmin:xmax,ymin:ymax)
gtemp(0:8,xmin:xmax,ymin:ymax)=g(0:8,xmin:xmax,ymin:ymax)

do x=xmin,xmax
do y=ymin,ymax
do i=0,8

!only fluid node are considered
if(.not.(wall(x,y).or.cyl(x,y)))then

f(i,x,y)=(1.0-omegav)*ftemp(i,x,y)+omegav*f0(i,x,y)
g(i,x,y)=(1.0-omegac)*gtemp(i,x,y)+omegac*g0(i,x,y)

endif

enddo
enddo
enddo

endsubroutine collide

!streaming step
subroutine stream(xmin,ymin,xmax,ymax,xdim,ydim,f,g,cx,cy)

double precision
f(0:8,0:xdim,0:ydim),ftemp(0:8,0:xdim,0:ydim)
double precision
g(0:8,0:xdim,0:ydim),gtemp(0:8,0:xdim,0:ydim)
double precision cx(0:8),cy(0:8)

ftemp(0:8,xmin:xmax,ymin:ymax)=f(0:8,xmin:xmax,ymin:ymax)
gtemp(0:8,xmin:xmax,ymin:ymax)=g(0:8,xmin:xmax,ymin:ymax)

do x=xmin,xmax
do y=ymin,ymax
do i=0,8

f(i,x+cx(i),y+cy(i))=ftemp(i,x,y)
g(i,x+cx(i),y+cy(i))=gtemp(i,x,y)

enddo
enddo
enddo

!periodic boundary condition
do y=ymin,ymax

!west
f(1,xmin,y+cy(1))=ftemp(1,xmax,y)
f(5,xmin,y+cy(5))=ftemp(5,xmax,y)
f(8,xmin,y+cy(8))=ftemp(8,xmax,y)

!east
f(3,xmax,y+cy(3))=ftemp(3,xmin,y)
f(6,xmax,y+cy(6))=ftemp(6,xmin,y)
f(7,xmax,y+cy(7))=ftemp(7,xmin,y)

!west
g(1,xmin,y+cy(1))=gtemp(1,xmax,y)
g(5,xmin,y+cy(5))=gtemp(5,xmax,y)
```

```
g(8,xmin,y+cy(8))=gtemp(8,xmax,y)

!east
g(3,xmax,y+cy(3))=gtemp(3,xmin,y)
g(6,xmax,y+cy(6))=gtemp(6,xmin,y)
g(7,xmax,y+cy(7))=gtemp(7,xmin,y)


enddo

do x=xmin,xmax

!south
f(2,x+cx(2),ymin)=ftemp(2,x,ymax)
f(5,x+cx(5),ymin)=ftemp(5,x,ymax)
f(6,x+cx(6),ymin)=ftemp(6,x,ymax)

!north
f(4,x+cx(4),ymax)=ftemp(4,x,ymin)
f(7,x+cx(7),ymax)=ftemp(7,x,ymin)
f(8,x+cx(8),ymax)=ftemp(8,x,ymin)

!south
g(2,x+cx(2),ymin)=gtemp(2,x,ymax)
g(5,x+cx(5),ymin)=gtemp(5,x,ymax)
g(6,x+cx(6),ymin)=gtemp(6,x,ymax)

!north
g(4,x+cx(4),ymax)=gtemp(4,x,ymin)
g(7,x+cx(7),ymax)=gtemp(7,x,ymin)
g(8,x+cx(8),ymax)=gtemp(8,x,ymin)

enddo

f(5,xmin,ymin)=ftemp(5,xmax,ymax)!south-west
f(8,xmin,ymax)=ftemp(8,xmax,ymin)!north-west
f(6,xmax,ymin)=ftemp(6,xmin,ymax)!south-east
f(7,xmax,ymax)=ftemp(7,xmin,ymin)!north-east

g(5,xmin,ymin)=gtemp(5,xmax,ymax)!south-west
g(8,xmin,ymax)=gtemp(8,xmax,ymin)!north-west
g(6,xmax,ymin)=gtemp(6,xmin,ymax)!south-east
g(7,xmax,ymax)=gtemp(7,xmin,ymin)!north-east

endsubroutine stream

!bounceback boundary condition
subroutine
boundary(xmin,ymin,xmax,ymax,xdim,ydim,f,g,cx,cy,wall,cyl)

double precision
f(0:8,0:xdim,0:ydim),ftemp(0:8,0:xdim,0:ydim)
double precision
g(0:8,0:xdim,0:ydim),gtemp(0:8,0:xdim,0:ydim)
double precision cx(0:8),cy(0:8)
```

```
logical wall(xmin:xmax,ymin:ymax)
logical cyl(xmin:xmax,ymin:ymax)

ftemp(0:8,xmin:xmax,ymin:ymax)=f(0:8,xmin:xmax,ymin:ymax)
gtemp(0:8,xmin:xmax,ymin:ymax)=g(0:8,xmin:xmax,ymin:ymax)

do x=xmin,xmax
do y=ymin,ymax
do i=0,8

if(wall(x,y))then

f(0,x,y)=ftemp(0,x,y)
f(1,x,y)=ftemp(3,x,y)
f(2,x,y)=ftemp(4,x,y)
f(3,x,y)=ftemp(1,x,y)
f(4,x,y)=ftemp(2,x,y)
f(5,x,y)=ftemp(7,x,y)
f(6,x,y)=ftemp(8,x,y)
f(7,x,y)=ftemp(5,x,y)
f(8,x,y)=ftemp(6,x,y)

g(0,x,y)=gtemp(0,x,y)
g(1,x,y)=gtemp(3,x,y)
g(2,x,y)=gtemp(4,x,y)
g(3,x,y)=gtemp(1,x,y)
g(4,x,y)=gtemp(2,x,y)
g(5,x,y)=gtemp(7,x,y)
g(6,x,y)=gtemp(8,x,y)
g(7,x,y)=gtemp(5,x,y)
g(8,x,y)=gtemp(6,x,y)

!thermal boundary condition at boundary nodes
elseif(cyl(x,y))then

f(0,x,y)=ftemp(0,x,y)
f(1,x,y)=ftemp(3,x,y)
f(2,x,y)=ftemp(4,x,y)
f(3,x,y)=ftemp(1,x,y)
f(4,x,y)=ftemp(2,x,y)
f(5,x,y)=ftemp(7,x,y)
f(6,x,y)=ftemp(8,x,y)
f(7,x,y)=ftemp(5,x,y)
f(8,x,y)=ftemp(6,x,y)

g(0,x,y)=-(gtemp(0,x,y)-
(cx(0)**2.0+cy(0)**2.0)*ftemp(0,x,y))+(cx(0)**2.0+cy(0)**2
.0)*f(0,x,y)
g(1,x,y)=-(gtemp(3,x,y)-
(cx(3)**2.0+cy(3)**2.0)*ftemp(3,x,y))+(cx(1)**2.0+cy(1)**2
.0)*f(1,x,y)
g(2,x,y)=-(gtemp(4,x,y)-
(cx(4)**2.0+cy(4)**2.0)*ftemp(4,x,y))+(cx(2)**2.0+cy(2)**2
.0)*f(2,x,y)
```

```
g(3,x,y)=-(gtemp(1,x,y)-
(cx(1)**2.0+cy(1)**2.0)*ftemp(1,x,y))+(cx(3)**2.0+cy(3)**2
.0)*f(3,x,y)
g(4,x,y)=-(gtemp(2,x,y)-
(cx(2)**2.0+cy(2)**2.0)*ftemp(2,x,y))+(cx(4)**2.0+cy(4)**2
.0)*f(4,x,y)
g(5,x,y)=-(gtemp(7,x,y)-
(cx(7)**2.0+cy(7)**2.0)*ftemp(7,x,y))+(cx(5)**2.0+cy(5)**2
.0)*f(5,x,y)
g(6,x,y)=-(gtemp(8,x,y)-
(cx(8)**2.0+cy(8)**2.0)*ftemp(8,x,y))+(cx(6)**2.0+cy(6)**2
.0)*f(6,x,y)
g(7,x,y)=-(gtemp(5,x,y)-
(cx(5)**2.0+cy(5)**2.0)*ftemp(5,x,y))+(cx(7)**2.0+cy(7)**2
.0)*f(7,x,y)
g(8,x,y)=-(gtemp(6,x,y)-
(cx(6)**2.0+cy(6)**2.0)*ftemp(6,x,y))+(cx(8)**2.0+cy(8)**2
.0)*f(8,x,y)

endif

enddo
enddo
enddo

endsubroutine boundary

!forcing terms
subroutine
force(xmin,ymin,xmax,ymax,xdim,ydim,f,g,fpoi,fbuoy,h,u,v,c
x,cy)

double precision
f(0:8,0:xdim,0:ydim),ftemp(0:8,0:xdim,0:ydim)
double precision
g(0:8,0:xdim,0:ydim),gtemp(0:8,0:xdim,0:ydim)
double precision
fpoi(0:8,0:xdim,0:ydim),fbuoy(0:8,0:xdim,0:ydim),h(0:8,0:x
dim,0:ydim)
double precision u(0:xdim,0:ydim),v(0:xdim,0:ydim)
double precision cx(0:8),cy(0:8)

ftemp(0:8,xmin:xmax,ymin:ymax)=f(0:8,xmin:xmax,ymin:ymax)
gtemp(0:8,xmin:xmax,ymin:ymax)=g(0:8,xmin:xmax,ymin:ymax)

do x=xmin,xmax
do y=ymin,ymax
do i=0,8

f(i,x,y)=ftemp(i,x,y)+fpoi(i,x,y)+fbuoy(i,x,y)

g(i,x,y)=gtemp(i,x,y)+(1.0/2.0)*(cy(i)-
v(x,y))**2.0*(fpoi(i,x,y)+fbuoy(i,x,y))+f(i,x,y)*h(i,x,y)

enddo
enddo
```

```
enddo

endsubroutine force

!compute value of density,velocities,pressure and forces
subroutine
compute(xmin,ymin,xmax,ymax,xdim,ydim,f,g,fpoi,fbuoy,h,rho
,u,v,t,p,cx,cy,w,zeta,rho0,tc,th,cs2,grad,grav,beta,wall,c
yl)

double precision f(0:8,0:xdim,0:ydim)
double precision g(0:8,0:xdim,0:ydim)
double precision
fpoi(0:8,0:xdim,0:ydim),fbuoy(0:8,0:xdim,0:ydim),h(0:8,0:x
dim,0:ydim)
double precision
rho(0:xdim,0:ydim),u(0:xdim,0:ydim),v(0:xdim,0:ydim),t(0:x
dim,0:ydim),p(0:xdim,0:ydim)
double precision
zeta(0:xdim,0:ydim),pstat(0:xdim,0:ydim),pdyn(0:xdim,0:ydi
m)
double precision utemp(0:xdim,0:ydim),vtemp(0:xdim,0:ydim)
double precision
rhou(0:xdim,0:ydim),rhov(0:xdim,0:ydim),rhot(0:xdim,0:ydim
)
double precision cx(0:8),cy(0:8),w(0:8)
double precision rho0,tc,th
double precision grad,grav,beta

logical wall(xmin:xmax,ymin:ymax)
logical cyl(xmin:xmax,ymin:ymax)

utemp(xmin:xmax,ymin:ymax)=u(xmin:xmax,ymin:ymax)
vtemp(xmin:xmax,ymin:ymax)=v(xmin:xmax,ymin:ymax)

do x=xmin,xmax
do y=ymin,ymax

rho(x,y)=f(0,x,y)
rhou(x,y)=f(0,x,y)*cx(0)
rhov(x,y)=f(0,x,y)*cy(0)
rhot(x,y)=g(0,x,y)

do i=1,8

rho(x,y)=rho(x,y)+f(i,x,y)
rhou(x,y)=rhou(x,y)+f(i,x,y)*cx(i)
rhov(x,y)=rhov(x,y)+f(i,x,y)*cy(i)
rhot(x,y)=rhot(x,y)+g(i,x,y)

enddo
enddo
enddo

do x=xmin,xmax
do y=ymin,ymax
```

```
!no slip u=0,v=0
if(wall(x,y))then

rho(x,y)=rho0
u(x,y)=0.0
v(x,y)=0.0

elseif(cyl(x,y))then

rho(x,y)=rho0
u(x,y)=0.0
v(x,y)=0.0
t(x,y)=th

else

!rho cannot equal to zero
if(rho(x,y).ne.0.)then

u(x,y)=rhou(x,y)/rho(x,y)
v(x,y)=rhov(x,y)/rho(x,y)
t(x,y)=rhot(x,y)/rho(x,y)

else

u(x,y)=0.0
v(x,y)=0.0
t(x,y)=0.0

endif

endif

enddo
enddo

!adiabatic wall
t(xmin:xmax,ymin)=t(xmin:xmax,ymin+1)
t(xmin:xmax,ymax)=t(xmin:xmax,ymax-1)

!pressure driven poiseuille flow
do x=xmin,xmax
do y=ymin,ymax
do i=0,8

fpoi(i,x,y)=3.0*w(i)*grad*rho(x,y)*cx(i)

enddo
enddo
enddo

!buoyancy force
do x=xmin,xmax
do y=ymin,ymax
do i=0,8
```

```
fbuoy(i,x,y)=3.0*w(i)*grav*beta*t(x,y)*rho(x,y)*cy(i)

enddo
enddo
enddo

!effects of viscous heating
do x=xmin,xmax
do y=ymin,ymax
do i=0,8

h(i,x,y)=(cx(i)-utemp(x,y))*(u(x,y)-utemp(x,y))+(cy(i)-
vtemp(x,y))*(v(x,y)-vtemp(x,y))

enddo
enddo
enddo

!vorticity function
do x=xmin,xmax
do y=ymin,ymax

zeta(x,y)=(v(x,y)-v(x-1.0,y))-(u(x,y)-u(x,y-1.0))

enddo
enddo

!pressure
do x=xmin,xmax
do y=ymin,ymax

pstat(x,y)=cs2*rho(x,y)-grad*(x-xmax)!static pressure
pdyn(x,y)=(1.0/2.0)*rho(x,y)*(u(x,y)**2.0+v(x,y)**2.0)!dyn
amic pressure
p(x,y)=pstat(x,y)+pdyn(x,y)!stagnation pressure

enddo
enddo

endsubroutine compute
```