KANSAS STATE UNIVERSITY

ME-820-A, PROJECT -II

Simulation of Lid Driven Cavity

A. Gairola

Contents

Acknowledgement

I offer my sincerest gratitude to Prof. Mingjun Wei for his help, invaluable guidance and impeccable technical advices during the course of last semester. Without his help and guidance it would have been difficult for me to achieve anything in this project. At last I would like to express my sincere gratitude to the authors of 'Applied Numerical Linear Algebra' and 'Numerical Simulation in Fluid Dynamics: A Practical Introduction' for writing excellent yet easy to understand monographs.

1. Introduction

The scope of the current project is to solve a classical benchmark problem i.e, lid driven cavity via the marker and cell method (**MAC**) method, which uses a simple finite difference scheme with an explicit first-order time discretization. It may be applied to the computation of flows in fixed domains as well as to the simulation of free boundary value problems. [Griebel et al., 1998]

The current problem in hand is widely employed to evaluate numerical methods and to validate codes for solving the Navier-Stokes equations. The classical problem ask to solve the mentioned Navier-Stokes equation on a square domain with 'Dirichlet boundary' conditions on all the sides, with three stationary sides and one moving side.

2. The Navier Stokes Equation

The objective of the current project is to solve the equations[2.1, 2.2, 2.3] on the domain in the region $0 \leq x \leq 1$ and $0 \leq y \leq 1$ via the marker and cell method (1.). A pictorial view of the computational domain is shown in the figure [1a]. It is to be noted that if the mentioned equations are solved for all the variables located on the nodes of the figure [1a] then the problem of checkerboarding may arise owing to the discretization of the continuity equation [McDonough, 2007]. To circumvent this marker and cell method solves the equations for the three variables i.e. u,v and p on three separate domains shifted from each other by half a grid spacing to the bottom, to the left, and to the lower left, respectively.

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{2.1}$$

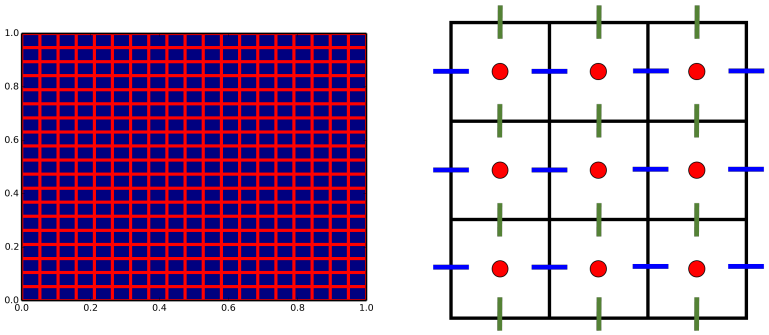
$$\frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y} + \frac{\partial p}{\partial x} = \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{2.2}$$

$$\frac{\partial v}{\partial t} + \frac{\partial v^2}{\partial y} + \frac{\partial uv}{\partial x} + \frac{\partial p}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{2.3}$$

Owing to which not all extremal grid points come to lie on the domain boundary. Hence the vertical boundaries, for instance, carry no v-values, just as the horizontal boundaries carry no u-values. For this reason, *an extra boundary strip of grid cells is introduced so that the boundary conditions may be applied by averaging the nearest grid points on either side*. [Griebel et al., 1998]

2.1 Discretization

Considering the staggered grid in figure [2] the continuity equation is discretized at the center of each cell. The second derivatives which forms the diffusive terms are replaced by



(a) A representative domain (b) Staggered grid

Figure 1: Domain

their discrete counterparts in the following equations

$$(\frac{\partial u}{\partial x})_{(i,j)} = \frac{u_{(i,j)} - u_{(i-1,j)}}{\delta x} \tag{2.4}$$

$$(\frac{\partial v}{\partial x})_{(i,j)} = \frac{v_{(i,j)} - v_{(i-1,j)}}{\delta y} \tag{2.5}$$

$$(\frac{\partial^2 u}{\partial x^2})_{(i,j)} = \frac{u_{(i-1,j)} - 2u_{(i,j)} + u_{i+1,j}}{\delta x^2} \tag{2.6}$$

$$(\frac{\partial^2 u}{\partial y^2})_{(i,j)} = \frac{u_{(i,j-1)} - 2u_{(i,j)} + u_{(i,j+1)}}{\delta y^2} \tag{2.7}$$

$$(\frac{\partial^2 v}{\partial x^2})_{(i,j)} = \frac{v_{(i-1,j)} - 2v_{(i,j)} + v_{i+1,j}}{\delta x^2} \tag{2.8}$$

$$(\frac{\partial^2 v}{\partial y^2})_{(i,j)} = \frac{v_{(i,j-1)} - 2v_{(i,j)} + v_{(i,j+1)}}{\delta y^2} \tag{2.9}$$

The terms shown in the following equation 2.10 need special treatment as, for example, to discretize $\frac{\partial(uv)}{\partial x}$ at the midpoint of the right edge of cell (i , j) [2], we need suitable values of the product uv lying in the two vertical directions. The solution implemented here is to use averages of u and v taken at the locations marked with an 'x' in Figure 2, which gives the discrete term.

$$(\frac{\partial(uv)}{\partial y})_{(i,j)}, (\frac{\partial(u^2)}{\partial x})_{(i,j)} \tag{2.10}$$

$$(\frac{\partial(uv)}{\partial y})_{i,j} = (1/\delta y)\left((\frac{v_{i,j} + v_{i+1,j}}{2}\frac{u_{i,j} + u_{i,j+1}}{2}) - (\frac{v_{i,j-1} + v_{i+1,j-1}}{2}\frac{u_{i,j-1} + u_{i,j}}{2})\right) \tag{2.11}$$

$$(\frac{\partial(u^2)}{\partial x})_{i,j} = (1/\delta x)\left((\frac{(u_{i,j} + u_{i+1,j})^2}{4} - \frac{(u_{i-1,j} + u_{i,j})^2}{4})\right) \tag{2.12}$$

Similarly for the two remaining terms.

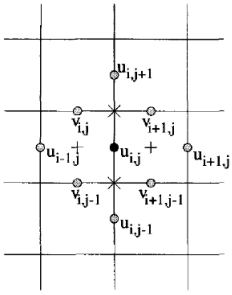


Figure 2: Staggered grid [Griebel et al., 1998]

3. Algorithm

The following algorithm was used to write the computer program:

```
Assign initial values to u,v, p
While t<tend and error>tolerance
    set boundary values for u and v
    compute F and G
    set it=0
    while it <itmax and errorp>tolerancep
        set boundary values for p
        perform SOR cycle and compute p
        compute the error
        it=it+1
   compute unew and vnew
   t=t+dt
   n=n+1
```

[Griebel et al., 1998].

3.1 Python Program

Corresponding to the algorithm mentioned above the following program was written in python by implementing all the necessary boundary conditions in the way mentioned in section 1 of this report:

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import linalg as LA
plt.rcParams['axes.linewidth'] = 2
g = 21
u = np.zeros((g+1, g))
un=np.zeros((g,g))
vn=np.zeros((g,g))
```

```
v = np.zeros((g, g+1))
p = np.zeros((g+1, g+1))
dx = float(1.0/(g-1))
dy = float(1.0/(g-1))
print dx
Re = 1.0
#Time step selection
fac=(1.0/(0.25*Re))
fac2=(0.25*Re*dx**2)
if fac>fac2:
    dt=fac2
elif fac<=fac2:
    dt=fac
print dt

#dt = 0.01

p[:,:]=1.0
w = 1.5
C=np.copy(p)
print type(C)
print np.shape(C)

F=np.zeros((g+1,g))
G=np.zeros((g,g+1))
error=1.0

t=0
Tf=1000.0
err=1
while (t<=Tf) and (err>10**-5):

    #print t
    #Boundary Values
    u[:,0]=0
    u[:,-1]=0
    v[:,0]=-v[:,1]
    v[:,-1]=-v[:,-2]
    v[0,:]=0
    v[-1,:]=0
    u[-1,:]=2-u[-2,:]
    u[0,:]=-u[1,:]
    speed=np.sqrt(un**2+vn**2)









    #print u

    for j in range(1,g):
        for i in range(1,g-1):
            F[j,i]=u[j,i]+((dt/(Re*dx**2))*(u[j,i+1]-2*u[j,i]+u[j,i-1]))\
```


```
                +(((dt/(Re*dy**2)))*(u[j+1,i]-2*u[j,i]+u[j-1,i]))\
                -(dt/dx)*(((u[j,i]+u[j,i+1])/2)**2-((u[j,i-1]+u[j,i])/2)**2)\
                -(dt/dy)*(((v[j,i]+v[j,i+1])/2)*((u[j,i]+u[j+1,i])/2)-(((v[j-1,i]+v
[j,i+1])/2)*((u[j-1,i]+u[j,i])/2)))
    #print u[-2,:]
    F[-1,:]=2-F[-2,:]
    F[0,:]=-F[1,:]
    F[:,0]=u[:,0]
    F[:,-1]=u[:,-1]
    #print F

    for j in range(1,g-1):
        for i in range(1,g):
            G[j,i]=v[j,i]+((dt/(Re*dx**2))*(v[j,i+1]-2*v[j,i]+v[j,i-1]))\
                +(((dt/(Re*dy**2)))*(v[j+1,i]-2*v[j,i]+v[j-1,i]))\
                -(dt/dy)*(((v[j,i]+v[j+1,i])/2)**2-((v[j-1,i]+v[j,i])/2)**2)\
                -(dt/dx)*((((u[j,i]+u[j+1,i])/2)*((v[j,i]+v[j,i+1])/2))-(((u[j,i
-1]+u[j+1,i-1])/2)*((v[j,i-1]+v[j,i])/2)))
    G[0,:]=v[0,:]
    G[-1,:]=v[-1,0]
    G[:,0]=-G[:,1]
    G[:,-1]=-G[:,-2]
    #C=[]
    temp=1


    p2 = np.copy(p)
    #print type(F)
    #print type(G)
    error = 1
    while error > 10**-5:
        p[:,0] = p[:,1]
        p[:,-1] = p[:,-2]
        p[0,:] = p[1,:]
        p[-1,:] = p[-2,:]
        p1 = np.copy(p)
        for i in range(1,g):
            for j in range(1,g):
                C[j,i]=((F[j,i]-F[j,i-1])/(dx*dt))+((G[j,i]-G[j-1,i])/(dy*dt))
                p2[j,i] = (1-w)*p[j,i]+w*(((dy**2)*(p[j,i-1]+p[j,i+1])+(dx**2)*(p[j
-1,i]+p[j+1,i])-(dx**2)*(dy**2)*C[j,i])/(2*(dx**2+dy**2)))

                p[j,i] = p2[j,i]
                #print const
        error = LA.norm(p1-p)
    for j in range(1,g):
        for i in range(1,g-1):
            u[j,i]=F[j,i]-(dt/dx)*(p[j,i+1]-p[j,i])

    for j in range(1,g-1):
        for i in range(1,g):
            v[j,i]=G[j,i]-(dt/dx)*(p[j+1,i]-p[j,i])
    t=t+dt
```


```
112
113
114
115     for j in range(g):
116         for i in range(g):
117             un[j,i]=(u[j,i]+u[j+1,i])/2
118     for j in range(g):
119         for i in range(g):
120             vn[j,i]=((v[j,i]+v[j,i+1]))/2
121
122     err=LA.norm(np.sqrt(un**2+vn**2)-speed)
123     print "Error",err,"Time", t
124
125
126
127
128 print np.shape(un)
129 print np.size(vn)
130
131 X1=np.linspace(0,1,np.size(un[:,0]))
132 Y1=np.linspace(0,1,np.size(un[:,0]))
133 print np.shape(X1)
134 Y,X=np.meshgrid(Y1,X1)
135 speed=np.sqrt(un*un+vn*vn)
136 print speed
137 plt.streamplot(X1,Y1,un,vn,density=1,  color=un, linewidth=2, cmap=plt.cm.autumn)
138 plt.xlim(0,1)
139 plt.ylim(0,1)
140 plt.savefig('15.png',format='png',dpi=300)
141 plt.show()
142 plt.pcolor(un)
143 plt.xlim(0,g)
144 plt.ylim(0,g)
145 plt.savefig('16.png',format='png',dpi=300)
146 plt.colorbar()
147 plt.show()
148 plt.pcolor(vn)
149 plt.xlim(0,g)
150 plt.ylim(0,g)
151 plt.savefig('17.png',format='png',dpi=300)
152 plt.show()
153 np.savetxt('U5.txt',un)
154 np.savetxt('V5.txt',vn)
```

Listing 1: Python list for MAC method

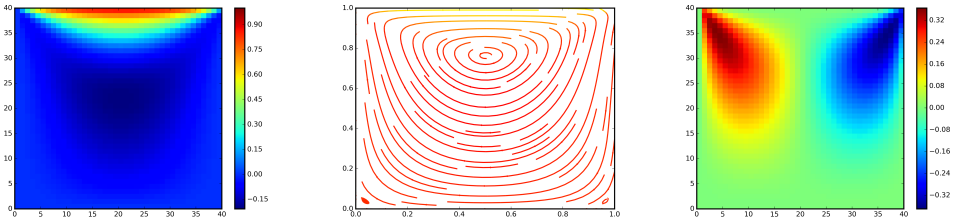
4. Results

The capability of the program was tested to simulate the driven cavity at different Reynold's numbers. Following results were obtained for the Reynold's number 1, 100, 400 and 1000. Subsequently these results were compared with the standard benchmarks

of Ghia [Ghia et al., 1982] and Kim & Moin. [Kim and Moin, 1985]

4.1 Re=1

Following results are obtained for a domain of 40×40 for Reynold's number 1.



(a) U velocity (b) Streamplot (c) V velocity

Figure 3: Results for Re=1

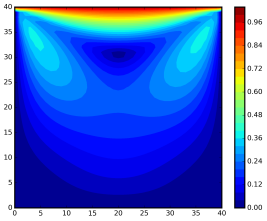
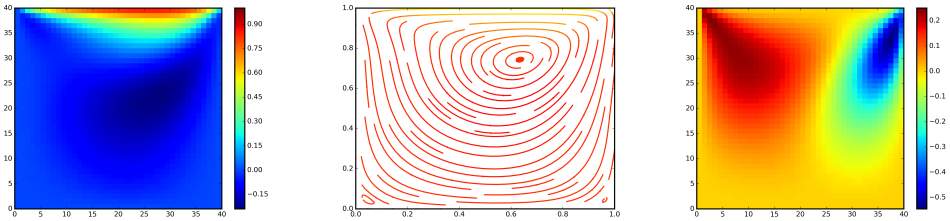


Figure 4: Norm of the velocity

4.2 Re=100

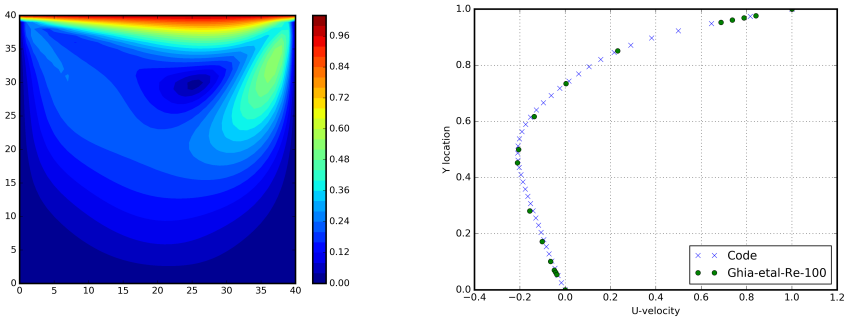
For the Reynold's number 100 on a domain of 40×40 following results were obtained. The



(a) U velocity (b) Streamplot (c) V velocity

Figure 5: Results for Re=100

results were further benchmarked against the results of Ghia [Ghia et al., 1982] in the following figure.



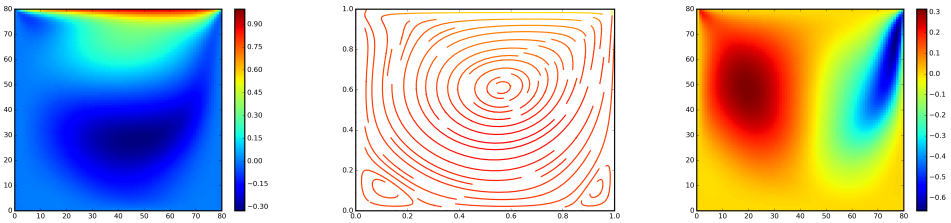
(a) Norm of the velocity

(b) Comparison of U-velocity

Figure 6: Results for Re=100

4.3 Re=400

Following results were obtained for the Reynold's number 400 on a domain of 80×80 which were later benchmarked against the results of Ghia [Ghia et al., 1982] and Kim & Moin [Kim and Moin, 1985]

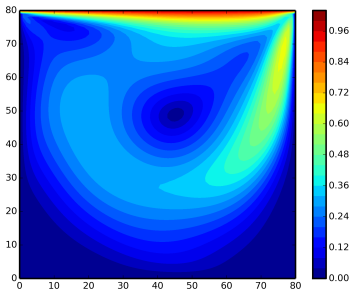


(a) U velocity

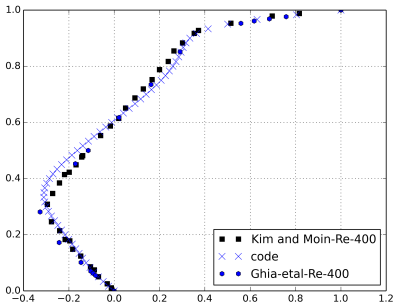
(b) Streamplot

(c) V velocity

Figure 7: Results for Re=400



(a) Norm of the velocity

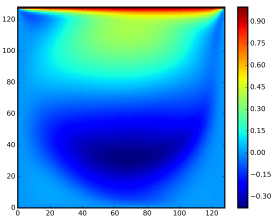


(b) Comparison of U-velocity

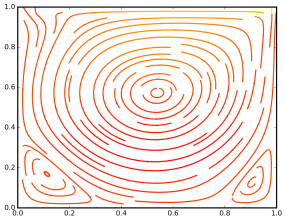
Figure 8: Results for Re=400

4.4 Re=1000

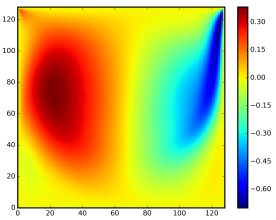
Results were also obtained for Reynold's number of 1000 on a 128×128 domain.



(a) U velocity

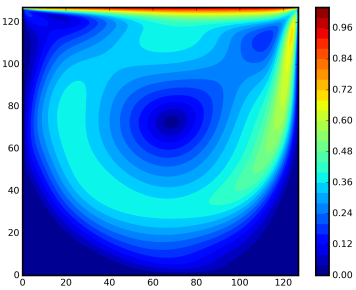


(b) Streamplot

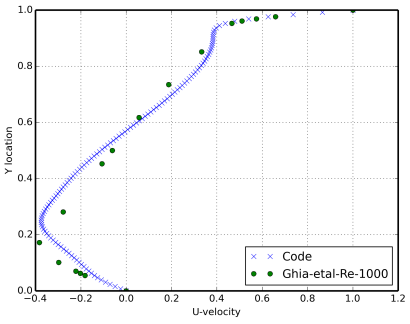


(c) V velocity

Figure 9: Results for Re=1000



(a) Norm of the velocity



(b) Comparison of U-velocity

Figure 10: Results for Re=1000

5. Conclusion

After comparing the obtained solutions with the benchmark results it is clear that there is a certain degree of discrepancy between the two with the increasing Reynold's number, this can be attributed to the error tolerance used in the pressure poisson step (10^{-5}), a rather aggressive tolerance may correct this issue, however owing to the slow nature of the solver for higher number of grids and aggressive tolerance levels it was not possible during the course of this project to implement the same, an alternative and faster implementation for the pressure poisson solver (SLOR or ADI) may speed up the process in the future, GPU computation may further accelerate the process.

Nevertheless at Re=100 and Re=400 solver was able to match the benchmark results quite well which implies that the implementation of the marker and cell method was done correctly.

Thus during the course of this project a standard method to solve the incompressible Navier-Stokes equation was implemented and validated against the standard benchmark results of Ghia and Kim & Moin, some discrepancies were found at higher Reynold's number between the calculated and the benchmark solutions which can be corrected by implementing an aggressive error tolerance in the pressure poisson step and possibly by refining the grids further, however to do that a faster implementation is needed to solve the pressure poisson equation.

References

[Ghia et al., 1982] Ghia, U., Ghia, K. N., and Shin, C. (1982). High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of computational physics*, 48(3):387–411.

[Griebel et al., 1998] Griebel, M., Dornseifer, T., and Neunhoeffer, T. (1998). *Numerical simulation in fluid dynamics: a practical introduction*. SIAM.

[Kim and Moin, 1985] Kim, J. and Moin, P. (1985). Application of a fractional-step method to incompressible navier-stokes equations. *Journal of computational physics*, 59(2):308–323.

[McDonough, 2007] McDonough, J. (2007). Mathematics, algorithms and implementations.