Numerical simulations using solvers developed for computational fluid dynamics

A project report submitted in partial fulfilment of the requirements of the course ME670 (Advanced computational fluid dynamics)

By

Name: Prashil Bhaskarrao Thool

Roll No: 224103105

Department of Mechanical Engineering Indian Institute of Technology Guwahati Guwahati-781039

March 2023

Contents

Chapter 1

Compact scheme

Question-1:

Consider a computational domain of width and height of 10 × 10. Take the left bottom corner as the origin (0, 0). At every nodal point of the domain the values of dependent variable ϕ is given according to

$$\boldsymbol{\phi_{ij= sin\, x_i\, cos\, y_j}}$$

Note: You should write a code such that it should give results for any order of accuracy (within the orders specified in the notes material. See table 22.1 on the course material) out of 4th, 6th and 8th orders. The code should ask for user input for the required order of accuracy. Then it should give result of the order of accuracy of the user input. Choose α and β values in table 22.1 of course material such that at least one of them should be non-zero

You can use the following expressions for the explicit fourth order expressions for boundary nodal points and the nodal points adjacent to the boundaries. Expression for 4th order forward differencing is:

$$\frac{\partial \phi}{\partial x} = \frac{1}{12\triangle x}(-25\phi_i + 48\phi_{i+1} - 36\phi_{i+2} + 16\phi_{i+3} - 3\phi_{i+4}) + O(\triangle x^4)$$

Expression for 4th backward differencing is:

$$\frac{\partial \phi}{\partial x} = \frac{1}{12\triangle x}(25\phi_i - 48\phi_{i-1} + 36\phi_{i-2} - 16\phi_{i-3} + 3\phi_{i-4}) + O(\triangle x^4)$$

Note that we have an analytical solution for the given dependent variable as:

$$\frac{\partial \phi}{\partial x} = cosx\, cosy$$

The programme output should contain at least the following:

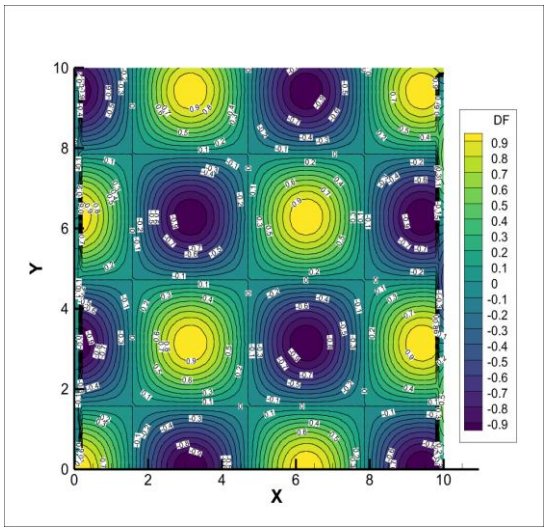
1. Plot contours of ∂f/∂x. Compare your numerical results obtained from your code with those of analytical expression given by Eq. (1) by showing contours in side by side figures.

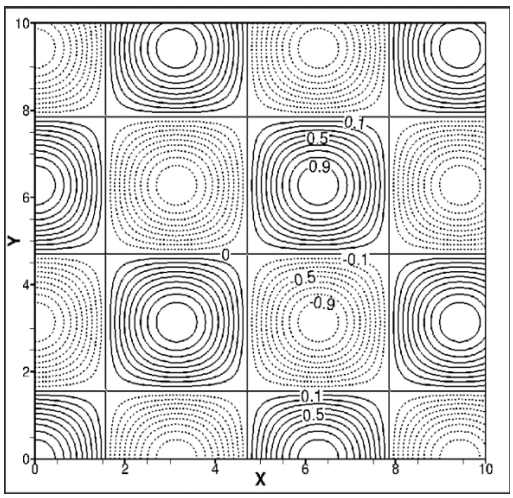
2. ∂f/∂x variation with y at mid-vertical line (x = 5) of the domain and compare with those of analytical solution. Plot your numerical data from compact scheme code and the data from analytical solution given by Eq. (1) in the same figure.

3. ∂f/∂x variation with x at mid-horizontal line (y = 5) of the domain and compare with those of analytical solution. Plot your numerical data from compact scheme code and the data from analytical solution given by Eq. (1) in the same figure.

4. Find out the value of ∂f/∂x at locations of point-A (x1, y1) = (3, 5) and point-B (x2, y2) = (6, 3), and compare with that of analytical value. Make a table of comparison. Table should contain 3 columns. First column should contain the values obtained from compact scheme, second column should contain analytical values and the last column should contain % difference between compact scheme value and analytical value. The table should contain two rows for point-A and point-B.

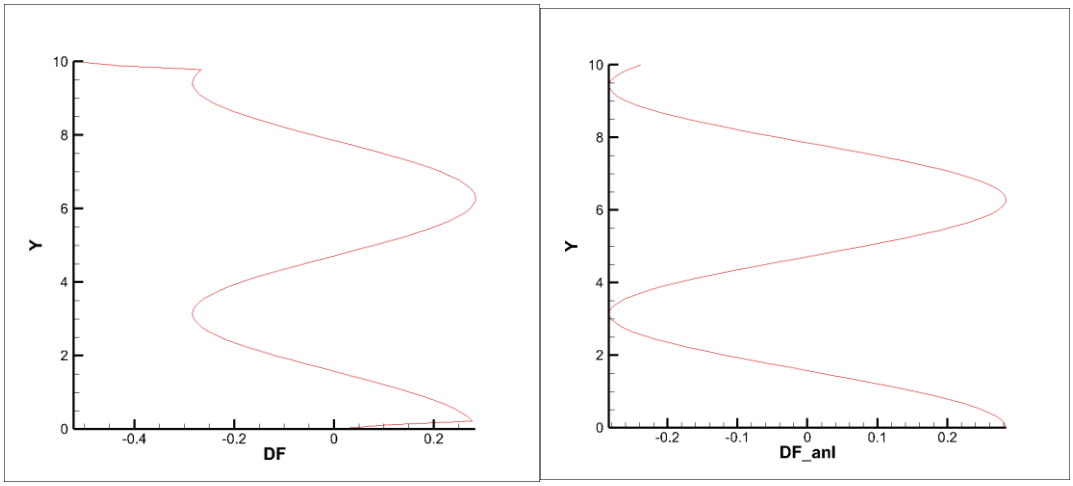
1. Ans





fig(a): Numerical results fig(b): Analytical results

2. Comparison of ∂f/∂x variation along mid-vertical line (line x = 0.5)



3. Fig: Comparison of ∂f/∂x variation along mid-vertical line (line x = 0.5)

4. Comparison of ∂f/∂x variation along mid-horizontal line (line y = 0.5)

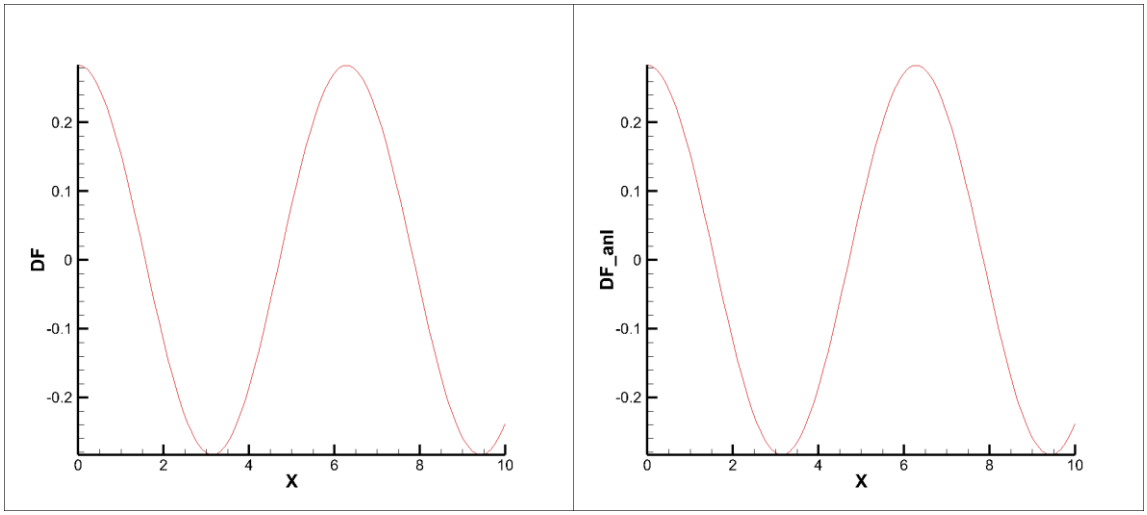


Fig: Comparison of ∂f/∂x variation along mid-horizontal line (line y = 0.5)

5. COMPARISON TABLE

	Numerical solution	Analytical solution	% error
Point A	-0.280822011	-0.280823435	0.000142464
Point B	-0.950556557	-0.950561379	0.000482226

1.1 The code of compact scheme

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
# define n 91
int main(void)
{
   double phi[n][n],df_anl[n][n],df[n][n],tem[n][n];
   double dx , dy,x,y,alpha, beta,a,b,c,cons,error;
   int i, j,order,iteration=1;
   dx = 10.0/(n-1);
   dy = 10.0/(n-1);
   cons = (1.0/(12.0*dx));



   FILE *file2;
   file2 = fopen("phi.plt","w");
   fprintf(file2,"VARIABLES= \"X\",\"Y\",\"PHI\"\n");
   fprintf(file2,"ZONE F=POINT\n ");
   fprintf(file2,"I=%d, j=%d\n",n,n);
    // value of phi at every point
   for ( i = 0; i <n; i++)
```

```
{
   for ( j = 0; j < n; j++)
   {
      x= dx*i;
      y=dx*j;
      phi[i][j]= sin(x)*cos(y);
      fprintf(file2,"%f\t%f\t%f\n",j*dx,i*dx,phi[i][j]);
      //printf("%f\t",phi[i][j]);
   }

}
FILE*file1;
file1 = fopen("analytical.plt","w");
 fprintf(file1,"VARIABLES= \"X\",\"Y\",\"DF_ANA\"\n");
fprintf(file1,"ZONE F=POINT\n ");
fprintf(file1,"I=%d, J=%d\n",n,n);
// analytical solution of problem
for ( i = 0; i < n; i++)
{
   for (  j= 0; j <n; j++)
   {
      x = dx*i;
      y = dx*j;
      df_anl [i][j] = cos(x)*cos(y);
```

```
        //printf("%f\t",df_anl[i][j]);
        fprintf(file1,"%f\t%f\t%f\n",j*dx,i*dx,df_anl[i][j]);
        df[i][j]=0.0;


    }

}
printf("choose the order of acurracy\n");
scanf("%d",& order);
// if and else  conditions for the order of the accuracy
 if (order == 4)
 {
    alpha = 0.0;
    beta=0.0;
    a = (2.0/3.0)*(alpha+2);
    b= (1.0/3.0)*(4*alpha-1);
    c = 0.0;
    printf(" 4th order \n");

 }
 else if (order==6)
 {
  alpha = 1.0/3.0;
```

```
 beta = 0;
 a = (1.0/6.0)*(alpha+9);
 b= (1.0/15.0)*(32*alpha-9);
 c = (1.0/10.0)*(-3*alpha+1);
 printf(" 6th order \n");
}
else if (order == 8)
{
 alpha = 3.0/8.0;
 beta = (1.0/20.0)*(-3+8*alpha);
 a = (1.0/6.0)*(12-7*alpha);
 b= (1.0/150.0)*(568*alpha-183);
 c = (1.0/50.0)*(9*alpha-4);
 printf(" 8th order \n");
}
else if (order == 10)
{
 alpha = 1.0/2.0;
 beta = (1.0/20.0)*(-3+8*alpha);
 a = (1.0/6.0)*(12-7*alpha);
 b= (1.0/150.0)*(568*alpha-183);
 c = (1.0/50.0)*(9*alpha-4);
 printf(" 10th order \n");
```

```
    }



   for ( i = 0; i <n ; i++)
    {
       // bottom boundary
       df[i][0] = cons*(-25*phi[i][0]+48*phi[i][1]-
36*phi[i][2]+16*phi[i][3]-3*phi[i][4]);
       //adjecent points
       df[i][1] = cons*(-25*phi[i][1]+48*phi[i][2]-
36*phi[i][3]+16*phi[i][4]-3*phi[i][5]);


    }




  //  printf("\n");
    for ( i = 0; i <n ; i++)
    {
       // top boundary
       df[i][n-1] = cons*(25*phi[i][n-1]-48*phi[i][n-2]+36*phi[i][n-
3]-16*phi[i][n-4]+3*phi[i][n-5]);
       // adjecent points
```

```
      df[i][n-2] = cons*(25*phi[i][n-2]-48*phi[i][n-3]+36*phi[i][n-
4]-16*phi[i][n-5]+3*phi[i][n-6]);




    }



    for ( j = 0; j < n; j++)

    {

      //left boundary

      df[0][j] = cons*(-25*phi[0][j]+48*phi[1][j]-
36*phi[2][j]+16*phi[3][j]-3*phi[4][j]);

      //adjecent points

      df[1][j] = cons*(-25*phi[1][j]+48*phi[2][j]-
36*phi[3][j]+16*phi[4][j]-3*phi[5][j]);



    }



    for ( j =0; j <n; j++)

    {

        // right boundary

        df[n-1][j] = cons*(25*phi[n-1][j]-48*phi[n-2][j]+36*phi[n-
3][j]-16*phi[n-4][j]+3*phi[n-5][j]);

        // adjecent points
```

```
        df[n-2][j] = cons*(25*phi[n-2][j]-48*phi[n-3][j]+36*phi[n-4][j]-16*phi[n-5][j]+3*phi[n-6][j]);


    }

   // printing .m file for visualization

   FILE*file6;
   file6 = fopen("df_bound.m","w");
   fprintf(file6,"df_bound=[");
    // printing in plt file
   for ( i = 0; i <n; i++)
   {
     for ( j = 0; j <n; j++)
     {

      //printf("%f\t",df[i][j]);
      fprintf(file6,"%f\t",df[i][j]);



     }
     //printf("\n");
     fprintf(file6,"\n");
   }
```

```
FILE*file3;
file3 = fopen("error.txt","w");
error =1.0;
while (error>1e-6 )
{
   // assining the df values to tem
   for ( i = 0; i <n; i++)
   {
      for ( j = 0; j < n; j++)
      {
         tem[i][j]= df[i][j];
      }

   }
   // point gausss seidel
   for ( i = 2; i < n-2; i++)
   {
      for ( j = 2; j < n-2; j++)
      {
```

```
            df[i][j] =  a*0.5*(phi[i+1][j]-phi[i-1][j])/dx +
b*0.25*(phi[i+2][j]-phi[i-2][j])/dx + c*(phi[i+1][j]-phi[i-
1][j])/(6.0*dx)
                    -beta*df[i-2][j]-alpha*df[i-1][j]-alpha*df[i+1][j]-
beta*df[i+2][j];


      }


   }
   // error
   error=0.0;
   for ( i = 2; i < n-2; i++)
   {
      for ( j = 2; j < n-2; j++)
      {
         error = error+pow((df[i][j]-tem[i][j]),2);
      }


   }
   error = sqrt(error/(n*n));
   printf("iteration %d\t",iteration);
   printf("error %.9f\n",error);
   fprintf(file3,"%d\t%.9f\n",iteration,error);
   iteration=iteration+1;
```

```
}
FILE*file4;
file4 = fopen("df.plt","w");

fprintf(file4,"VARIABLES = \"X\",\"Y\",\"DF\"\n");
fprintf(file4,"ZONE F=POINT\n");
fprintf(file4,"I=%d, J=%d\n",n,n);

 // printing in plt file
for ( i = 0; i <n; i++)
{
  for ( j = 0; j <n; j++)
  {

   printf("%f\t",df[i][j]);
   fprintf(file4,"%f\t%f\t%f\n",j*dx,i*dx,df[i][j]);


  }

}
// line graph variation y at x= 5
 FILE*file7;
```

```
file7 = fopen("df_variation_y.plt","w");
fprintf(file7,"VARIABLES = \"DF\",\"Y\"\n");
fprintf(file7, "ZONE F =POINT\n");
fprintf(file7,"I=%d\n",n);
for ( j = 0; j < n; j++)
{
  double ypos;
  ypos = j*dx;
  fprintf(file7,"%f\t%f\n",df[(n-1)/2][j],ypos);
}
// line graph variation x at y= 5
FILE*file8;
file8 = fopen("df_variation_x.plt","w");
fprintf(file8,"VARIABLES = \"X\",\"DF\"\n");
fprintf(file8, "ZONE F =POINT\n");
fprintf(file8,"J=%d\n",n);
for ( i = 0; i < n; i++)
{
   double xpos;
  xpos = i*dx;
  fprintf(file8,"%f\t%f\n",xpos,df[i][(n-1)/2]);
}
// line graph analytical variation y at x= 5
 FILE*file5;
```

```
 file5 = fopen("df_analytical_variation_y.plt","w");
 fprintf(file5,"VARIABLES = \"DF_anl\",\"Y\"\n");
 fprintf(file5, "ZONE F =POINT\n");
 fprintf(file5,"I=%d\n",n);

for ( j = 0; j < n; j++)
{
   double ypos;
   ypos = j*dx;
   fprintf(file5,"%f\t%f\n",df_anl[(n-1)/2][j],ypos);
}
// line graph  anlytical variation x at y= 5
 FILE*file9;
 file9 = fopen("df_analytical_variation_x.plt","w");
 fprintf(file9,"VARIABLES = \"X\",\"DF_anl\"\n");
 fprintf(file9, "ZONE F =POINT\n");
 fprintf(file9,"J=%d\n",n);
 for ( i = 0; i < n; i++)
 {
   double xpos;
   xpos = i*dx;
   fprintf(file9,"%f\t%f\n",xpos,df_anl[i][(n-1)/2]);
 }
```

```
    fclose(file9);
    // point A and B for numerical and anlytical and errror
calculations
    file9 = fopen("POINT_A_AND_POINT_B.txt","w");
    fprintf(file9,"numA\tnumB\tanlA\tanlB\tdiffA\tdiffB\n");

fprintf(file9,"%.9f\t%.9f\t%.9f\t%.9f\t%.9f\t%.9f",df[27][45],df[
54][27],df_anl[27][45],df_anl[54][27],(df[27][45]-
df_anl[27][45])*100,(df[54][27]-df_anl[54][27])*100);



}
```

Chapter 2

Marker And Cell (MAC) algorithm

Question-2: Write a code for MAC algorithm to solve steady state flow field in a lid driven square cavity. Take height and width of the cavity as 1 unit. Obtain results for Reynolds number (Re) = 400. Take the equations in dimensionless form as given in Eqs. (8-56) to Eq. (8-58) on page number 55 given in the course material. There are two methods for solving: (a) compressibility method and (b) pressure Poisson equation method. Your code should be written based on pressure Poisson equation method.

The boundary conditions for left, bottom and right sides are

$$u = 0;\ v = 0;\ \frac{\partial p}{\partial \hat{n}} = 0$$

Where nˆ is a unit vector perpendicular to the wall. The boundary conditions for top side (for lid) are

$$u = 1;\ v = 0;\ \frac{\partial p}{\partial y} = 0$$

Here, u, v, p are dimensionless variables.

At the beginning of the iterations or time instant you need to give initial conditions or initial guessed values. Take the initial values as

$$u = 0;\ v = 0;\ p = 0$$

at all nodal points of the computational domain.

2.1 Physical system

The physical system is shown in figure 2.1

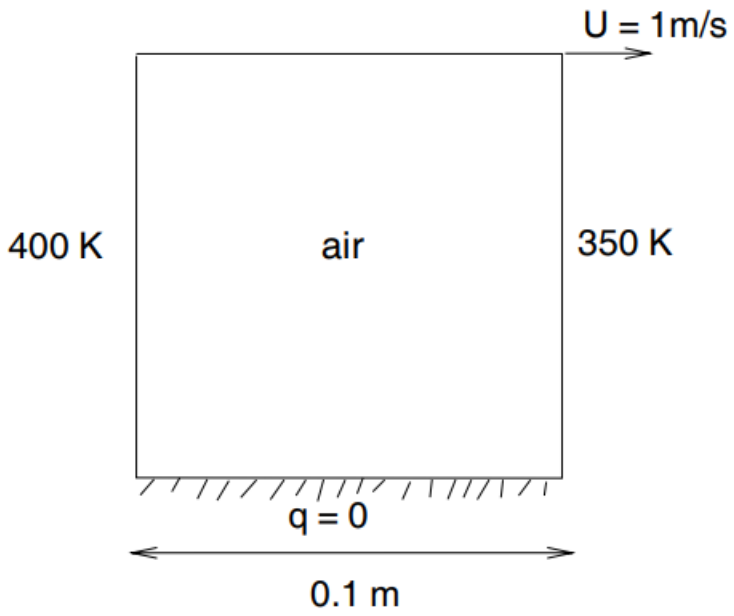
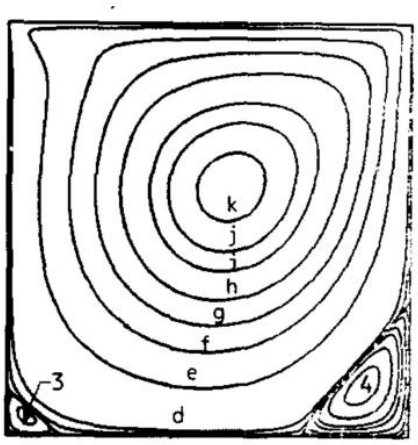
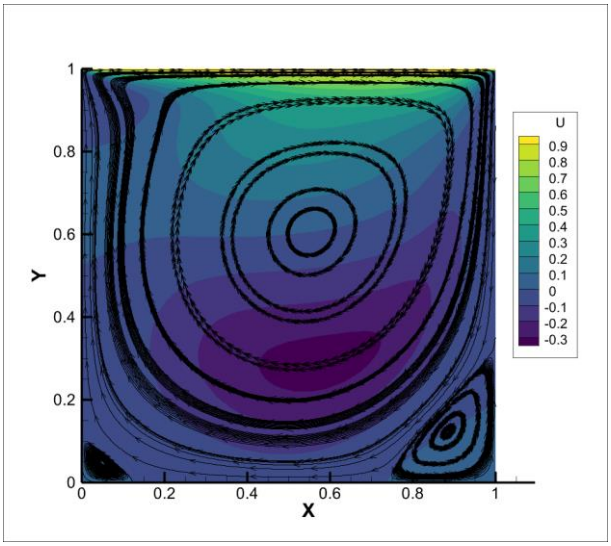


Fig: Schematic of lid driven cavity



(b) Ghia et al. [1]



Fig：Ghia et al.

Fig: Streamline MAC

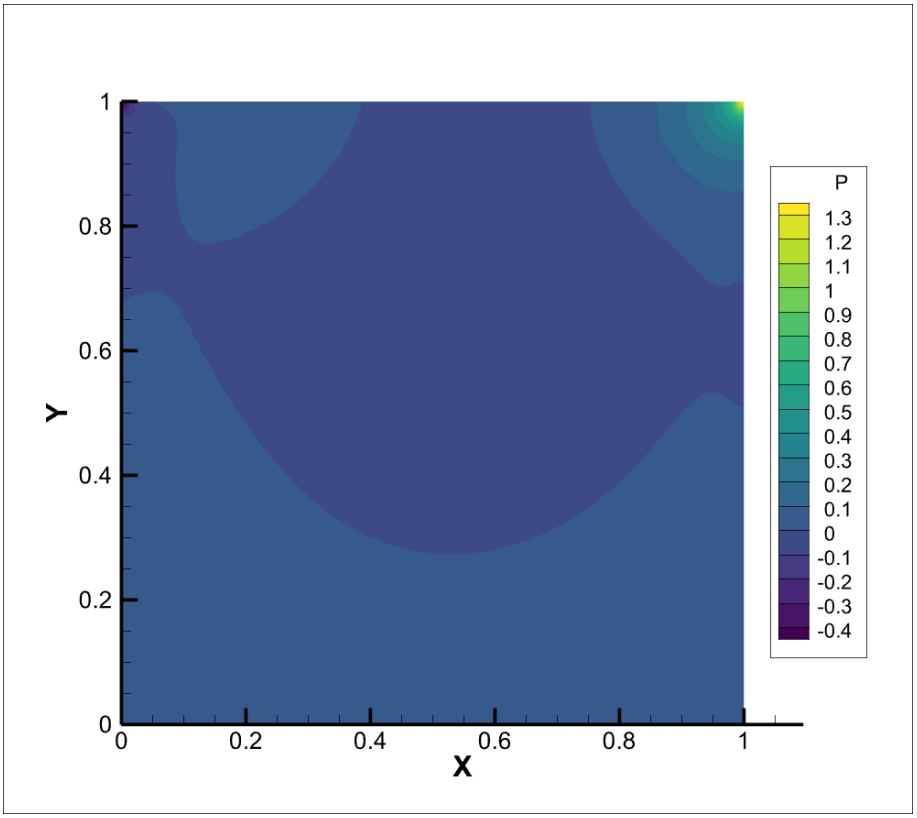


Fig: Contours of p for Re = 400

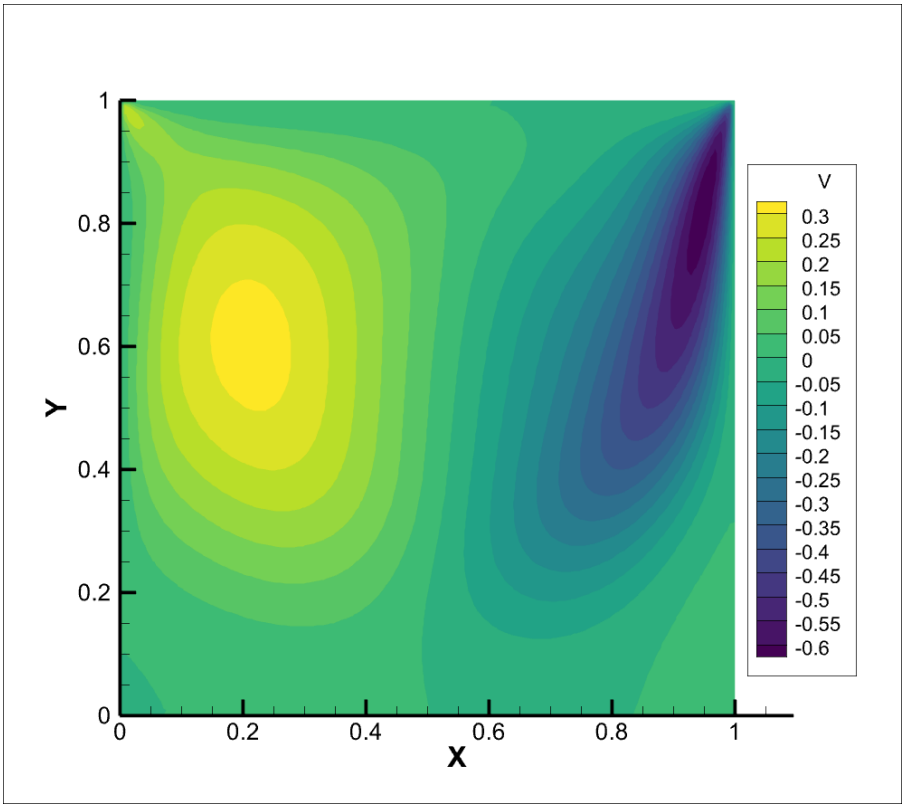


Fig: Contours of v for Re = 400

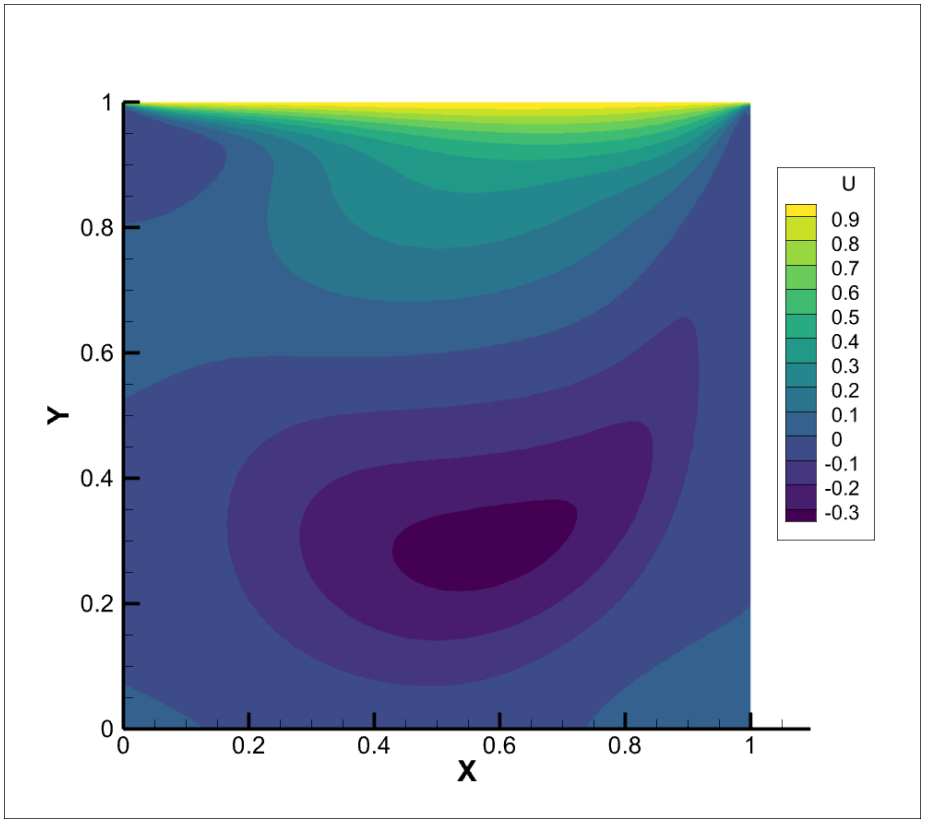


Fig: Contours of u for Re = 400

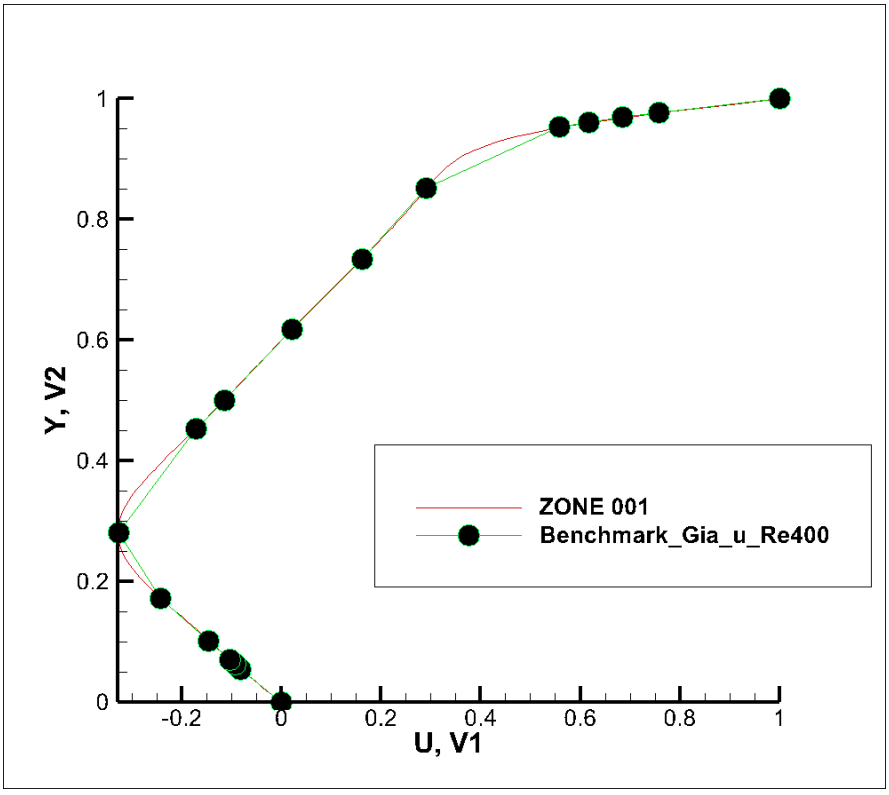


Fig: Comparison of u velocity profile along mid-vertical line (line x = 0.5) for Re = 400

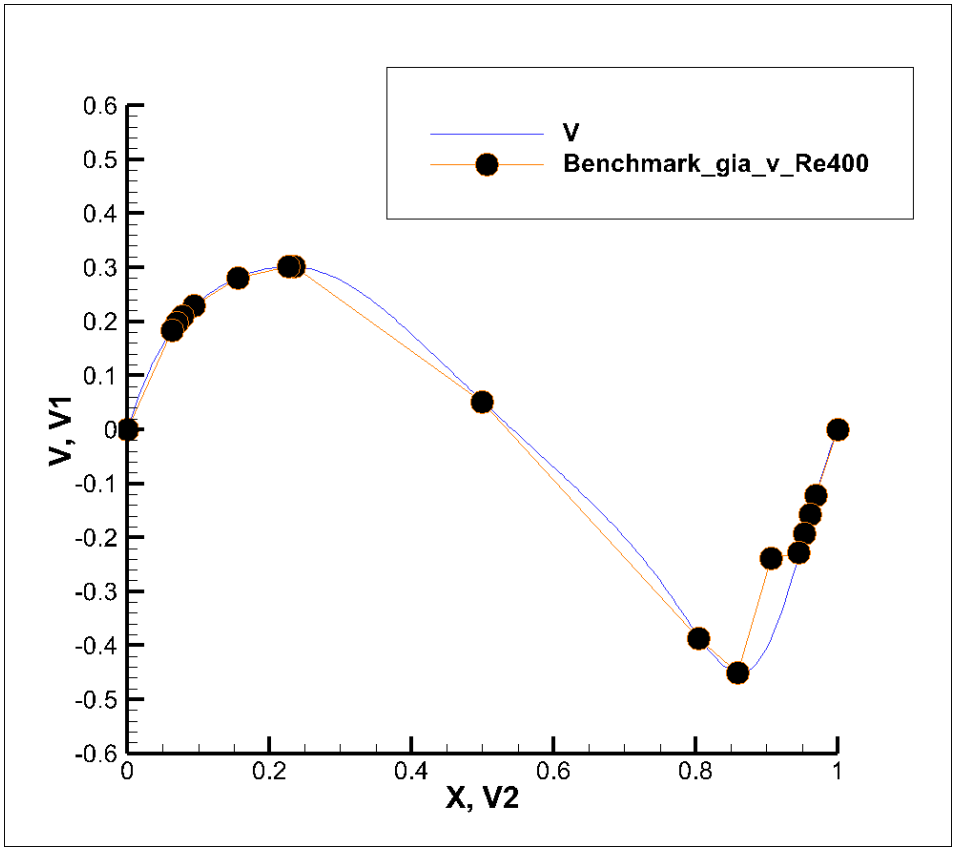


Fig: Comparison of v velocity profile along mid-horizontal line (line y = 0.5) for Re = 400

2.3 The code of MAC algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define N 128

int main (void)
{
        double u[N][N+1], un[N][N+1], uc[N][N],RHSU[N][N+1];
        double v[N+1][N], vn[N+1][N], vc[N][N],RHSV[N+1][N] ;
        double p[N+1][N+1], pn[N+1][N+1], pc[N][N];
        double m[N+1][N+1];
        int i, j, step;
        double dx, dy, dt, delta, error, Re,beta;
        step =1;
        dx = 1.0/(N-1);
        dy = 1.0/(N-1);
        dt = 0.001;
        error = 1.0;
        Re = 400.0;
   beta = dx/dy;

        // Initializing u
                for (i=0; i<=(N-1); i++)
                {
```

```
                for (j=0; j<=(N); j++)
                {
                        u[i][j] = 0.0;
                        u[i][N] = 1.0;
                        u[i][N-1] = 1.0;
    //printf("%f\t",v[i][j]);
                }
        }

// Initializing v
        for (i=0; i<=(N); i++)
        {
                for (j=0; j<=(N-1); j++)
                {
                        v[i][j] = 0.0;
    //printf("%f\t",v[i][j]);
                }
        }

// Initializing p
        for (i=0; i<=(N); i++)
        {
                for (j=0; j<=(N); j++)
                {
                        p[i][j] = 1.0;
                }
```

```
        }


    while (error > 1e-6)
    {
        // Solve RHSU
        for (i=1; i<=(N-2); i++)
        {
            for (j=1; j<=(N-1); j++)
            {
              RHSU[i][j] = u[i][j] - (dt/dx)*0.25*( pow((u[i+1][j]+u[i][j]),2)-
pow( (u[i-1][j]+u[i][j]),2))-
        (dt/dy)*0.25*((u[i][j+1]+u[i][j])*(v[i+1][j]+v[i][j])-(u[i][j-
1]+u[i][j])*(v[i][j-1]+v[i+1][j-1]))
        + (dt/(Re*dx*dx))*(u[i-1][j]-2*u[i][j]+u[i+1][j])+ (dt/(Re*dy*dy))*(u[i][j-
1]-2*u[i][j]+u[i][j+1]);
                                                                    //
printf("%f",RHSU[i][j]);
            }
        }




        // Solves RHSV
        for (i=1; i<=(N-1); i++)
        {
            for (j=1; j<=(N-2); j++)
```

```
                {
                        RHSV[i][j] = v[i][j] -
(dt/dx)*0.25*((u[i][j+1]+u[i][j])*(v[i+1][j]+v[i][j])-(u[i-1][j]+u[i-1][j+1])*(v[i-
1][j]+v[i][j]))
                    -(dt/dy)*0.25*( pow((v[i][j]+v[i][j+1]),2)-pow((v[i][j]+v[i][j-
1]),2))
                    +(dt/(Re*dx*dx))*(v[i-1][j]-
2*v[i][j]+v[i+1][j])+(dt/(Re*dy*dy))*(v[i][j-1]-2*v[i][j]+v[i][j+1]);

                }
            }

            // Solves pressure
            for (i=1; i<=(N-1); i++)
            {
                for (j=1; j<=(N-1); j++)
                {
                        pn[i][j] = (1.0/(2.0*(1+beta)))*(pn[i+1][j]+pn[i-
1][j]+pow(beta,2)*(pn[i][j+1]+pn[i][j-1])
                    -(pow(dx,2)/dt)*((RHSU[i][j]-RHSU[i-1][j])/dx +(RHSV[i][j]-
RHSV[i][j-1])/dy));

                }
```

```
        }


        // Boundary conditions
        for (i=1; i<=(N-1); i++)
        {
                pn[i][0] = pn[i][1];
                pn[i][N] = pn[i][N-1];
        }

        for (j=0; j<=(N); j++)
        {
                pn[0][j] = pn[1][j];
                pn[N][j] = pn[N-1][j];
        }

// Solves u-momentum
for (i=1; i<=(N-2); i++)
        {
                for (j=1; j<=(N-1); j++)
                {
                 un[i][j] =  - (dt/dx)*(pn[i+1][j]-pn[i][j])+RHSU[i][j];
                }
        }
// Boundary conditions
        for (j=1; j<=(N-1); j++)
```

```
        {
                un[0][j] = 0.0;
                un[N-1][j] = 0.0;
        }

        for (i=0; i<=(N-1); i++)
        {
                un[i][0] = -un[i][1];
                un[i][N] = 2 - un[i][N-1];
        }
// Solves v-momentum
        for (i=1; i<=(N-1); i++)
        {
                for (j=1; j<=(N-2); j++)
                {
                        vn[i][j] = - (dt/dy)*(pn[i][j+1]-pn[i][j])+RHSV[i][j];

                }
        }

// Boundary conditions
        for (j=1; j<=(N-2); j++)
        {
                vn[0][j] = -vn[1][j];
                vn[N][j] = -vn[N-1][j];
        }
```

```
		for (i=0; i<=(N); i++)
		{
			vn[i][0] = 0.0;
			vn[i][N-1] = 0.0;
		}

		// Displaying error
		error = 0.0;

		for (i=1; i<=(N-1); i++)
		{
			for (j=1; j<=(N-1); j++)
			{
				m[i][j] = (  ( un[i][j]-un[i-1][j] )/dx + ( vn[i][j]-vn[i][j-1] )/dy  );
				error = error + fabs(m[i][j]);
			}
		}

	//printf("Error is %9.8lf for the step %d\n", error, step);

		if (step%1000 ==1)
		{
		printf("Error is %.9lf for the step %d\n", error, step);
		}
```

```
// Iterating u
for (i=0; i<=(N-1); i++)
{
        for (j=0; j<=(N); j++)
        {
                u[i][j] = un[i][j];
        }
}

// Iterating v
for (i=0; i<=(N); i++)
{
        for (j=0; j<=(N-1); j++)
        {
                v[i][j] = vn[i][j];
        }
}

// Iterating p
for (i=0; i<=(N); i++)
{
        for (j=0; j<=(N); j++)
        {
                p[i][j] = pn[i][j];
        }
```

```
        }

        step = step + 1;

}

for (i=0; i<=(N-1); i++)
{
        for (j=0; j<=(N-1); j++)
        {
                uc[i][j] = 0.5*(u[i][j]+u[i][j+1]);
                vc[i][j] = 0.5*(v[i][j]+v[i+1][j]);
                pc[i][j] = 0.25*(p[i][j]+p[i+1][j]+p[i][j+1]+p[i+1][j+1]);
        }
}



// OUTPUT DATA
FILE *file2, *file3, *file1;
file2 = fopen("UVP.plt","w+t");
file3 = fopen("Central_U.plt","w+t");
file1 = fopen("Central_V.plt","w+t");

if ( file2 == NULL )
{
```

```
  printf("\nERROR when opening file\n");
  fclose( file2 );
        }

 else
        {
        fprintf( file2, "VARIABLES=\"X\",\"Y\",\"U\",\"V\",\"P\"\n");
        fprintf( file2, "ZONE  F=POINT\n");
        fprintf( file2, "I=%d, J=%d\n", N, N );

        for ( j = 0 ; j < (N) ; j++ )
        {
  for ( i = 0 ; i < (N) ; i++ )
  {
                double xpos, ypos;
                xpos = i*dx;
                ypos = j*dy;

                fprintf( file2, "%5.8lf\t%5.8lf\t%5.8lf\t%5.8lf\t%5.8lf\n", xpos,
ypos, uc[i][j], vc[i][j], pc[i][j] );
  }
        }
        }

        fclose( file2 );

        // CENTRAL --U
```

```
 fprintf(file3, "VARIABLES=\"U\",\"Y\"\n");
 fprintf(file3, "ZONE F=POINT\n");
 fprintf(file3, "I=%d\n", N );

 for ( j = 0 ; j < N ; j++ )
 {
        double ypos;
   ypos = (double) j*dy;

   fprintf( file3, "%5.8lf\t%5.8lf\n", (uc[N/2][j] + uc[(N/2)+1][j])/(2.), ypos );
 }
        // CENTRAL --V
 fprintf(file1, "VARIABLES=\"X\",\"V\"\n");
 fprintf(file1, "ZONE F=POINT\n");
 fprintf(file1, "J=%d\n", N);
 for ( i = 0 ; i < N ; i++ )
 {
        double xpos;
   xpos = (double) i*dx;
   fprintf( file1, "%5.8lf\t%5.8lf\n", xpos,(vc[(N/2)+1][i] + vc[(N/2)][i])/(2.) );
}
}
```

Bibliography

[1] U. Ghia, K. Ghia, C. Shin, High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method, Journal of Computational Physics 48 (1982) 387–411.