LID DRIVEN CAVITY PROBLEM

Computational Project Report - II

Submitted in partial fulfilment of the requirements for the course of

Applied Computational Methods in Mechanical Sciences (ME426)

in

Mechanical Engineering

By

Sthavishtha B.R. (13ME125)

Under the supervision of

Dr. Ranjith M.

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL

MANGALORE, INDIA - 575025

Problem Statement

To compute incompressible flows in a **Lid Driven Cavity** using *Euler Explicit Method* on a uniform Cartesian Staggered Grid System

Tools Used

- The numerical code was implemented in *C programming language* using the open source IDE, *Code::Blocks*
- *Tecplot* software was used for plotting the contours
- *Microsoft Excel* was used for plotting the centreline velocity profiles

Computational resources available in the *Computational Fluid Dynamics Laboratory* were utilized for successfully completing this project.

Code for Lid-Driven Cavity Problem

```
//Staggered Finite-Volume approach to solve Lid Driven Cavity problem
// using Euler Explicit Method
#include<stdio.h>
#include<math.h>

//velocity boundary conditions
void vel_bcs(float uo[131][131],float vo[131][131],float po[131][131],int m,int n)
{
    int i,j;
    for(j=1;j<=n-1;j++)
    {
       uo[0][j]=0.0; // left boundary conditions
       uo[m-1][j]=0.0; //right boundary conditions
    }
    for(j=0;j<=n-1;j++)
    {
       vo[0][j]=-vo[1][j]; // left boundary conditions
       vo[m][j]=-vo[m-1][j]; //right boundary conditions
    }
    for(i=1;i<=m-1;i++)
    {
       vo[i][0]=0.0; // top boundary conditions
       vo[i][n-1]=0.0; //bottom boundary conditions
    }
    for(i=0;i<=m-1;i++)
       uo[i][0]=-uo[i][1]; //bottom boundary conditions
    for(i=1;i<=m-2;i++)
           uo[i][n]=2.0-uo[i][n-1]; // top boundary conditions
    for(i=1;i<=m-1;i++)
    {
        po[i][0]=po[i][1];
        po[i][n]=po[i][n-1];
    }
    for(j=1;j<=n-1;j++)
    {
        po[0][j]=po[1][j];
        po[m][j]=po[m][j];
    }
}
```

```
//f calculations at that point i,j according to u-cell
float f_calc(float uo[131][131],float vo[131][131],float re,int i,int j,int m,int n,float delta_x)
{
    float ue,uw,un,vn,vs,us,dudx,dudy,var_find;
    ue=(uo[i][j]+uo[i+1][j])/2;
    uw=(uo[i][j]+uo[i-1][j])/2;
    un=(uo[i][j]+uo[i][j+1])/2;
    us=(uo[i][j]+uo[i][j-1])/2;
    vn=(vo[i][j]+vo[i+1][j])/2;
    vs=(vo[i][j-1]+vo[i+1][j-1])/2;
    dudy=(uo[i][j+1]-2*uo[i][j]+uo[i][j-1])/(delta_x*delta_x);
    dudx=(uo[i+1][j]-2*uo[i][j]+uo[i-1][j])/(delta_x*delta_x);
    var_find=(dudy/re)+(dudx/re)-((ue*ue-uw*uw)/delta_x)-((un*vn-us*vs)/delta_x);
    return var_find;
}

//g calculations at that point i,j according to v-cell
float g_calc(float uo[131][131],float vo[131][131],float re,int i,int j,int m,int n,float delta_x)
{
    float ue,uw,ve,vw,vs,vn,dvdx,dvdy,var_find;
    ue=(uo[i][j]+uo[i][j+1])/2;
    uw=(uo[i-1][j]+uo[i-1][j+1])/2;
    ve=(vo[i][j]+vo[i+1][j])/2;
    vw=(vo[i][j]+vo[i-1][j])/2;
    vn=(vo[i][j]+vo[i][j+1])/2;
    vs=(vo[i][j]+vo[i][j-1])/2;
    dvdx=(vo[i+1][j]-2*vo[i][j]+vo[i-1][j])/(delta_x*delta_x);
    dvdy=(vo[i][j+1]-2*vo[i][j]+vo[i][j-1])/(delta_x*delta_x);
    var_find=(dvdy/re)+(dvdx/re)-((vn*vn-vs*vs)/delta_x)-((ue*ve-uw*vw)/delta_x);
    return var_find;
}

//Pressure calculation using SOR method
float sor(float f[131][131],float g[131][131],float po[131][131],float pn[131][131],float r,float delta_x,float omega,int m,int n)
{
    int i,j;
    float h,error=0.0,error_sum=0.0,ap,as,an,ae,aw;
    for(i=1;i<=m-1;i++)
        for(j=1;j<=n-1;j++)
        {
            if(i==1 && j==1)
            {
                aw=0.0;
                f[0][j]=0.0;
                ae=1.0;
                as=0.0;
                an=r*r;
                g[i][0]=0.0;
            }
            else if(i==m-1 && j==1)
            {
                ae=0.0;
                f[m-1][j]=0.0;
                an=r*r;
                as=0.0;
                aw=1.0;
```

```
                g[i][0]=0.0;
            }
            else if(j==n-1 && i==1)
            {
                as=r*r;
                g[i][n-1]=0.0;
                ae=1.0;
                an=0.0;
                aw=0.0;
                f[0][j]=0.0;
            }
            else if(j==n-1 && i==m-1)
            {
                an=0.0;
                g[i][n-1]=0.0;
                ae=0.0;
                as=r*r;
                aw=1.0;
                f[m-1][j]=0.0;
            }
            else if(i==1)
            {
                ae=1.0;
                an=r*r;
                as=r*r;
                aw=0.0;
                f[0][j]=0.0;
            }
            else if(j==1)
            {
                aw=1.0;
                ae=1.0;
                as=0.0;
                an=r*r;
                g[i][0]=0.0;
            }
            else if(j==n-1)
            {
                an=0.0;
                g[i][n-1]=0.0;
                ae=1.0;
                as=r*r;
                aw=1.0;
            }
            else if(i==m-1)
            {
                ae=0.0;
                f[m-1][j]=0.0;
                an=r*r;
                as=r*r;
                aw=1.0;
            }
            else
            {
                ae=1.0;
                an=r*r;
                as=r*r;
                aw=1.0;
            }
            ap=-(an+as+aw+ae);
            h=((f[i][j]-f[i-1][j])+(g[i][j]-g[i][j-1]))*delta_x;
```

```
                pn[i][j]=(omega*(h-aw*pn[i-1][j]-ae*pn[i+1][j]-as*pn[i][j-1]-an*pn[i][j+1])/ap)+(1-omega)*po[i][j];
                error=fabs(pn[i][j]-po[i][j]);
                error_sum=error+error_sum;
                po[i][j]=pn[i][j];
            }
            error=error_sum/((m-1)*(n-1));  //overall error - similar to mean error
            return error;
}

//computation of u and v velocities and returning the maximum error
float vel_calc(float un[131][131],float vn[131][131],float uo[131][131],float vo[131][131],float f[131][131],float g[131][131],float po[131][131],float delta_x,float delta_t,int m,int n)
{
    int i,j;
    float err_uo,err_un,err_vo,err_vn;
    for(i=1;i<=m-2;i++)
        for(j=1;j<=n-1;j++)
        {
         un[i][j]=uo[i][j]+f[i][j]*delta_t-(((po[i+1][j]-po[i][j])*delta_t)/delta_x);
         err_uo=fabs(un[i][j]-uo[i][j]);
         if(i==1 && j==1)
            err_un=err_uo;
         if(err_uo>err_un)
            err_un=err_uo;
        uo[i][j]=un[i][j];
        }
    for(i=1;i<=m-1;i++)
        for(j=1;j<=n-2;j++)
        {
         vn[i][j]=vo[i][j]+g[i][j]*delta_t-(((po[i][j+1]-po[i][j])*delta_t)/delta_x);
         err_vo=fabs(vn[i][j]-vo[i][j]);
         if(i==1 && j==1)
            err_vn=err_vo;
         if(err_vo>err_vn)
            err_vn=err_vo;
        vo[i][j]=vn[i][j];
        }
    if(err_un>=err_vn)
        return err_un;
    else
        return err_vn;
}

//to equate the new and old pressures
void pressure_new_old(float po[131][131],float pn[131][131],int m,int n)
{
    int i,j;
    for(i=1;i<=m-1;i++)
        for(j=1;j<=n-1;j++)
        po[i][j]=pn[i][j];
}

//calculates the u and v velocities at the specified nodes
void compute_uv(float uo[131][131],float vo[131][131],float po[131][131],int m,int n,float delta_x)
```

```
{
    int i,j;
    FILE* fid1,*fid2;
    float up[131][131]={0.0},vp[131][131]={0.0};
    float delta_xn;
    fid1=fopen("streamline.plt","w");
    fid2=fopen("pressure.plt","w");
    delta_xn=1/(float)(m-2);
    for(i=1;i<=m-1;i++)
        for(j=1;j<=n-1;j++)
        {
            up[i][j]=(uo[i][j]+uo[i-1][j])/2.0;
            vp[i][j]=(vo[i][j]+vo[i][j-1])/2.0;
        }
    for(i=1;i<=m-1;i++)
        for(j=1;j<=n-1;j++)
        {
            fprintf(fid1,"%f\t %f\t %f\t %f\n",(i-1)*delta_xn,(j-1)*delta_xn,up[i][j],vp[i][j]);
            fprintf(fid2,"%f\t %f\t %f\n",(i-1)*delta_xn,(j-1)*delta_xn,po[i][j]);
        }
    fclose(fid1);
    fclose(fid2);
}

void main()
{
    //initialization and declaration of variables
    float delta_x,r=1.0,uo[131][131]={0.0},un[131][131]={0.0},vo[131][131]={0.0},vn[131][131]={0.0},po[131][131]={0.0},pn[131][131]={0.0},re=100.0;
    int m=129,n=129,iter=0,i,j,iter1=0;
    float ae=1.0,an=1.0,aw=1.0,as=1.0,f[131][131]={0.0},g[131][131]={0.0},error_p=1.0,omega=1.5,error_vel=1.0,t,delta_t=1e-3;
    delta_x=1/(float)(m-1); //computing parameters
    //looping
    while(error_vel>1e-8)
    {
        iter1=0;
        vel_bcs(uo,vo,po,m,n); //velocity boundary conditions
        for(i=1;i<=m-2;i++)
            for(j=1;j<=n-1;j++)
                 f[i][j]=f_calc(uo,vo,re,i,j,m,n,delta_x); //equating f to all grid points
        for(i=1;i<=m-1;i++)
            for(j=1;j<=n-2;j++)
                g[i][j]=g_calc(uo,vo,re,i,j,m,n,delta_x); //equating g to all grid points
        // convergence of pressure equations
        while(error_p>1e-8)
        {
            iter1++;
            error_p=sor(f,g,po,pn,r,delta_x,omega,m,n);
        }
        error_p=1.0;
        //computing u and v velocities
        error_vel=vel_calc(un,vn,uo,vo,f,g,po,delta_x,delta_t,m,n);
        //pressure_new_old(po,pn,m,n);
        iter++;
```

```
            t=iter*delta_t;
            printf("%d\t %f\t %d\n",iter,error_vel,iter1);
        }
        compute_uv(uo,vo,po,m,n,delta_x);
    }
```

Results

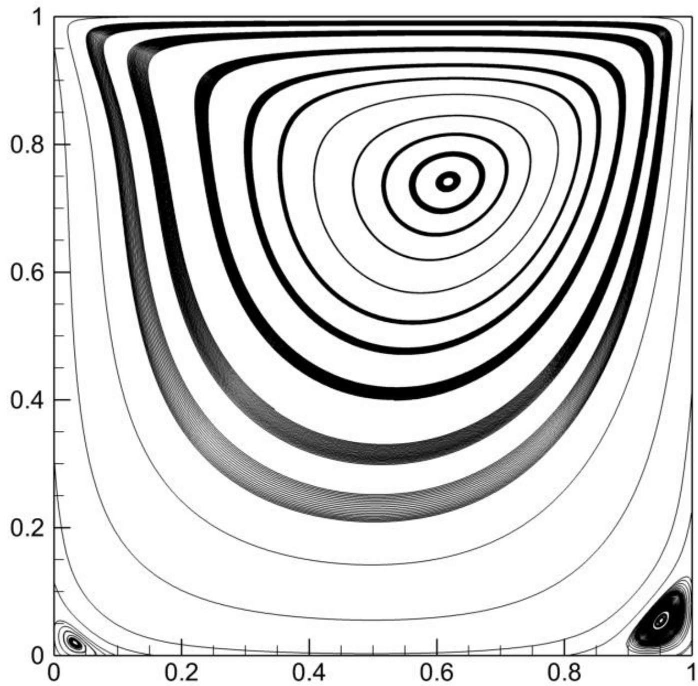
For 129 x 129 grid size, Parameters used were:

Over-Relaxation Parameter = 1.5

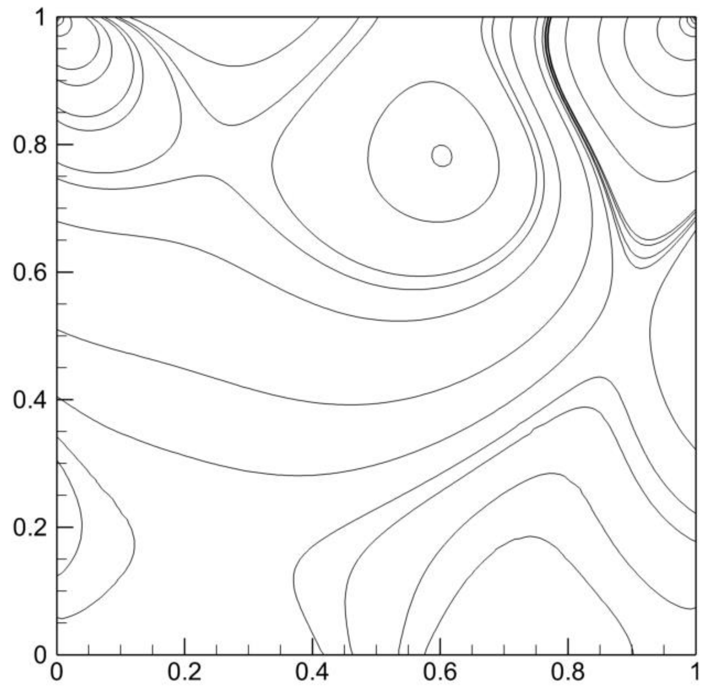
Convergence Error for Pressure = 10^{-8}

Convergence Error for Velocity = 10^{-8}

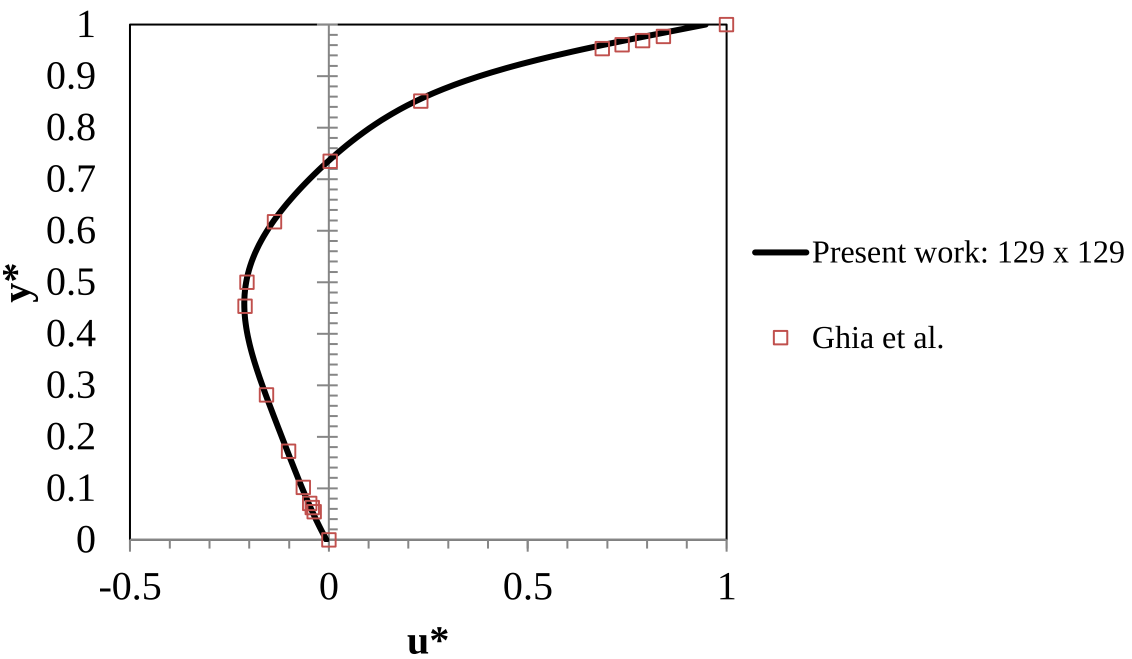
Streamline patterns at Re = 100



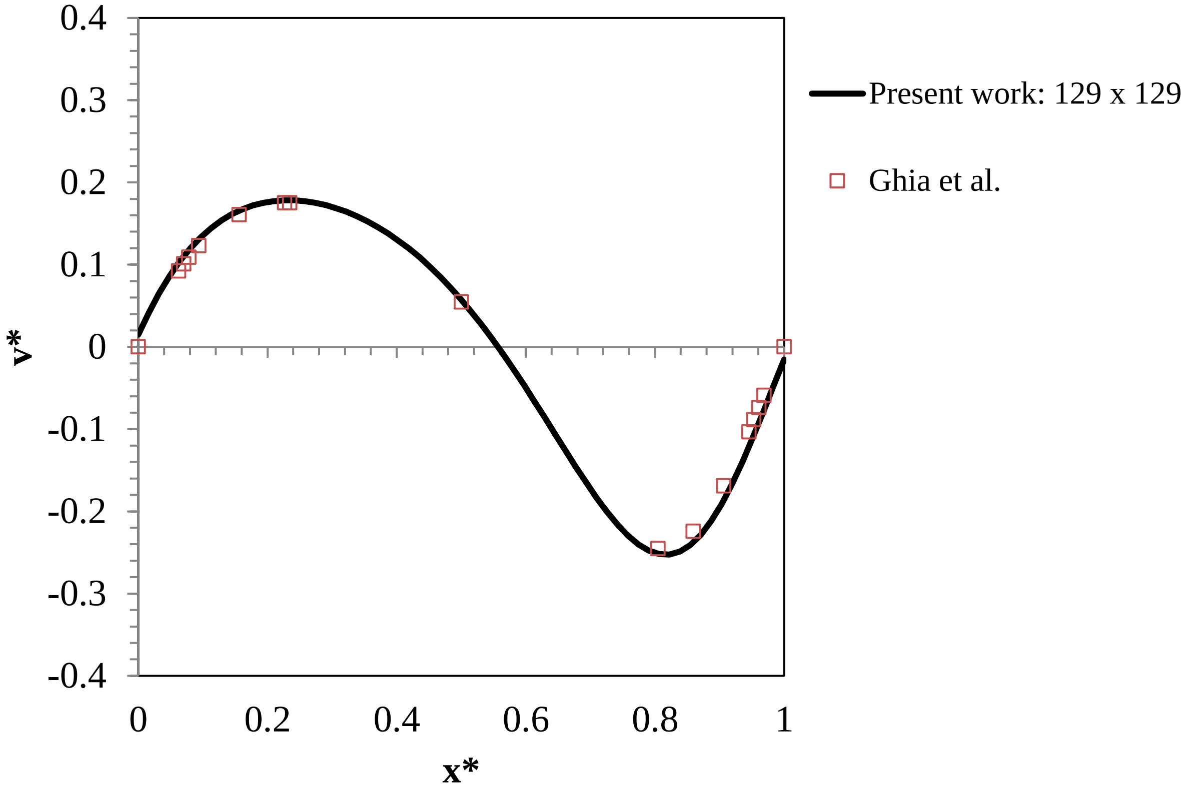
Pressure Contours at Re = 100



Validation of centreline u-velocity profile



Validation of centreline v-velocity profile



Conclusions

The streamline patterns and pressure contours developed by this method (pictures above) are consistent with the published results.