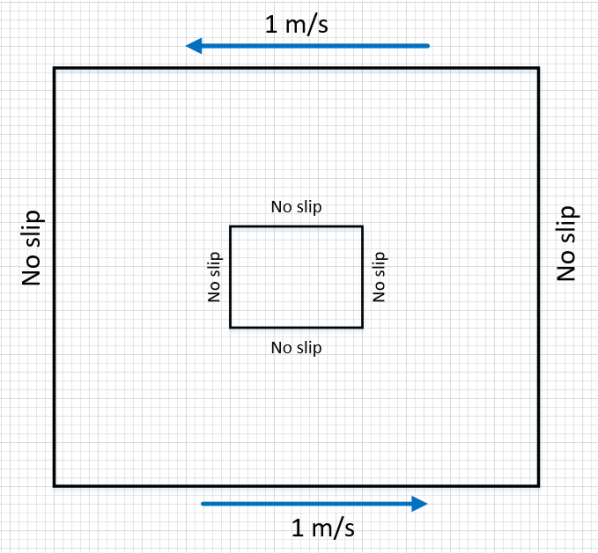
VAMSI KRISHNA M

LID-DRIVEN CAVITY USING SIMPLE ALGORITHM

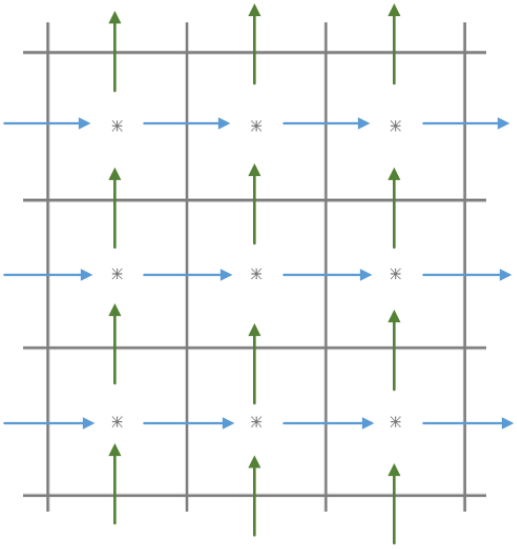
Problem statement:

Given a 2-D domain, square in shape (1 x 1), with the top and bottom lids moving at a velocity of -1 m/s and 1 m/s respectively. The domain contains a small solid piece in the centre, spanning two-sevenths of the length of the domain.

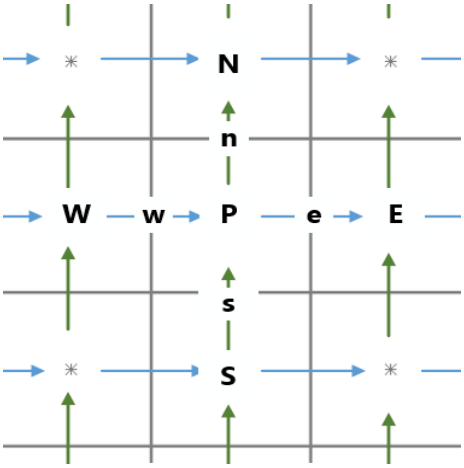
Boundary conditions:



Staggered Grid



A staggered grid is used alongside the collocated grid. The collocated grid points are the '*' points in the above attached figure. The velocities are calculated in the faces of the staggered grid and then copied into the collocated points by suitable mathematical operations. The collocated values are just the average of the staggered values.



Velocities are calculated at points denoted by lower case alphabets (Staggered grid), while pressure is calculated at the points denoted by upper case alphabets (Collocated grid),

The SIMPLE Algorithm:

1. Guess the pressure field p*.
2. Solve the momentum equations, to obtain u*, v*, w*.

$$a_e u_e^* = \Sigma a_{nb} u_{nb}^* + A_e(P_P^* - P_E^*)$$

$$a_n v_n^* = \Sigma a_{nb} v_{nb}^* + A_n(P_P^* - P_N^*)$$

3. Solve the p' equation.

$$a_P P_P' = \Sigma a_{NB} P_{NB}' + b$$

4. Calculate p from by adding p' to p*.

$$P = \alpha_p P' + P^*$$

5. Calculate u, v, w from their starred values using the velocity correction formulas.

$$u_e = u_e^* + \alpha_u d_e(P_P' - P_E')$$

$$v_n = v_n^* + \alpha_v d_n(P_P' - P_N')$$

6. Treat the corrected pressure p as a new guessed pressure p*, return to step 2, and repeat the whole procedure until a converged solution is obtained.

The coefficients in steps 3 and 6 are calculated by different schemes such as upwind, central difference, hybrid, and power-law.

Convergence is achieved when the source term, b, in step 3 is below the tolerance value.

The SIMPLE algorithm was coded in C++ programming language.

```
#include<bits/stdc++.h>
#define TOL 1e-6
using namespace std;
int counter;

//To find the max value
double max(double a, double b)
{
	if(a>b)
	return a;
	return b;
}

// Used in calculation of coefficients
double A(double f, double d)
{
	switch (counter)
	{

		case 1: return 1 - 0.1*fabs(f/d);
		case 2: return 1;
		case 3: return max(0,1 - 0.1*fabs(f/d));
		case 4: return max(0, pow(1-0.1*fabs(f/d),5));

	}
}

int main()
{
	int n = 71 , iter =0;
	double mu = 0.01, rho =1;
	double dy = 1.0/(n-1), dx = 1.0/(n-1);
	double vfactor = 1, ufactor = 1, pfactor=1;

	/*  counter =
		central difference , 1
		upwind, 2
		hybrid, 3
		power law, 4
	*/
	counter = 1;

	//conductances , convective fluxes, coefficients
	double D = mu*dy/dx;
	double fe,fw,fn,fs,ae,aw,an,as,ap;

	// Staggered variables
	double ustar[n+1][n], u[n+1][n], de[n+1][n];
	double vstar[n][n+1], v[n][n+1], dn[n][n+1];
	double pstar[n+1][n+1], pnew[n+1][n+1], p_c[n+1][n+1];
	double b[n+1][n+1];

	// Collocated variables
	double ufinal[n][n],vfinal[n][n],pfinal[n][n];

	// To store the number of iterations each method takes
```

```
fstream ifile;
ifile.open("iter_values.txt",ios::out);

label:
double error =1;

// Initialization of u,v,p with the Boundary conditions

for(int i=0;i<n+1;i++)
for(int j=0;j<n;j++)
{
	ustar[i][j]=0;
	u[i][j]=0;
	if(i==0)                                    //bottom wall
		u[i][j] = 2;
	if(i==n)
		u[i][j] = -2;              //top wall


}
for(int i=0;i<n;i++)
for(int j=0;j<n+1;j++)
{
	vstar[i][j]=0;
	v[i][j]=0;

}
for(int i=0;i<n;i++)
for(int j=0;j<n;j++)
{
	ufinal[i][j]=0;
	vfinal[i][j]=0;
	if(i==0)                                    // bottom wall
		ufinal[i][j]=1;
	if(i==n-1)                          //top wall
		ufinal[i][j] = -1;

}
for(int i=0;i<n+1;i++)
for(int j=0;j<n+1;j++)
{
	pstar[i][j] = 0;
	pnew[i][j]=0;
	p_c[i][j]=0;
	b[i][j]=0;
}


// Iterative loop

while(error > TOL)
{
	// ustar calculation

	for(int i = 1;i<n;i++)
	for(int j=1; j<n-1;j++)
	{
		fe = (u[i][j+1] + u[i][j])/2*dy* rho;
		fw = (u[i][j] + u[i][j-1])/2*dy* rho;
```

```
                fn = (v[i][j+1] + v[i][j])/2*dx* rho;
                fs = (v[i-1][j+1] + v[i-1][j])/2*dx * rho;

                ae = D*A(fe,D) + max(-fe,0);
                aw = D*A(fw,D) + max(fw,0);
                an = D*A(fn,D) + max(-fn,0);
                as = D*A(fs,D) + max(fs,0);

                ap = ae+aw+an+as;
                de[i][j] = dy/ap;
                ustar[i][j] = (ae*u[i][j+1] + aw*u[i][j-1] + an*u[i+1][j] +
as*u[i-1][j])/ap + de[i][j]*(pstar[i][j] - pstar[i][j+1]);

        }

        // Boundary conditions of ustar
        for(int i =0;i<n+1;i++)

        {
                ustar[i][0] = 0;                //left wall
                ustar[i][n-1] = 0;              //right wall
        }
        for(int j = 0; j<n;j++)
        {
                ustar[n][j] = -2-ustar[n-1][j];  //top wall
                ustar[0][j] = 2 - ustar[1][j]; //bottom wall
        }
        for(int i = 27;i<=44;i++)
        for(int j = 25;j<=45;j++)
        ustar[i][j]=0;

        for(int j=25;j<=45;j++)
        {
                ustar[25][j] = - ustar[26][j];
                ustar[46][j] = - ustar[45][j];
        }

        // vstar calculation
        for(int i = 1;i<n-1;i++)
        for(int j = 1; j<n;j++)
        {
                fe = (u[i+1][j] + u[i][j])/2*dy* rho;
                fw = (u[i+1][j-1] + u[i][j-1])/2*dy* rho;
                fn = (v[i+1][j] + v[i][j])/2*dx* rho;
                fs = (v[i][j] + v[i-1][j])/2*dx * rho;

                ae = D*A(fe,D) + max(-fe,0);
                aw = D*A(fw,D) + max(fw,0);
                an = D*A(fn,D) + max(-fn,0);
                as = D*A(fs,D) + max(fs,0);

                ap = ae+aw+an+as;
                dn[i][j] = dx/ap;
                vstar[i][j] = (ae*v[i][j+1] + aw*v[i][j-1] + an*v[i+1][j] +
as*v[i-1][j])/ap + dn[i][j]*(pstar[i][j] - pstar[i+1][j]);

        }

        //Boundary conditons for vstar
```

```
            for(int j = 0; j<n+1;j++)
            {
                  vstar[n][j] = 0;       //top wall
                  vstar[0][j] = 0;       //bottom wall
            }
            for(int i =0;i<n;i++)

            {
                  vstar[i][0] = -vstar[i][1];    //left wall
                  vstar[i][n] = -vstar[i][n-1];  //right wall
            }
            for(int i = 25;i<=45;i++)
            for(int j = 27;j<=44;j++)
            vstar[i][j]=0;

            for(int i=25;i<=45;i++)
            {
                  vstar[i][25] = - vstar[i][26];
                  vstar[i][46] = - vstar[i][45];
            }

            //Intialization of pressure correction
            for(int i=0;i<n+1;i++)
            for(int j=0;j<n+1;j++)
            p_c[i][j]=0;

            // Pressure correction calculation
            for(int i=1;i<n;i++)
            for(int j=1;j<n;j++)
            {
                  ae = rho*de[i][j]*dy;
                  aw = rho*de[i][j-1]*dy;
                  an = rho*dn[i][j]*dy;
                  as = rho*dn[i-1][j]*dy;

                  b[i][j] = rho*(ustar[i][j-1] - ustar[i][j])*dy  +
rho*(vstar[i-1][j] - vstar[i][j])*dx;

                  p_c[i][j] = (ae*p_c[i][j+1] + aw*p_c[i][j-1] + an*p_c[i+1][j]
+ as*p_c[i-1][j] + b[i][j])/ap;
            }


            // Correcting the pressure
            for(int i=1;i<n;i++)
            for(int j=1;j<n;j++)
            {
                  pnew[i][j] = pfactor*p_c[i][j] + pstar[i][j];
            }

            //Boundary conditions for pressure
            for(int i = 0;i<n+1;i++)
            {
                  pnew[i][0] = pnew[i][1];
                  pnew[i][n] = pnew[i][n-1];

            }
```

```
		for(int j = 0;j<n+1;j++)
		{
			pnew[0][j] = pnew[1][j];
			pnew[n][j] = pnew[n-1][j];
		}

		// Updation of u
		for(int i = 1;i<n;i++)
		for(int j=1; j<n-1;j++)
		{
			u[i][j] = ustar[i][j] + ufactor*de[i][j]*(p_c[i][j] -
p_c[i][j+1]);
		}

		// Boundary conditions of u
		for(int i =0;i<n+1;i++)
		{
			u[i][0] = 0;
			u[i][n-1] = 0;
		}

		for(int j = 0; j<n;j++)
		{
			u[n][j] = -2 -u[n-1][j];
			u[0][j] = 2 - u[1][j];
		}

		for(int i = 27;i<=44;i++)
		for(int j = 25;j<=45;j++)
		u[i][j]=0;

		for(int j=25;j<=45;j++)
		{
			u[25][j] = - u[26][j];
			u[46][j] = - u[45][j];
		}

		// Updation of v
		for(int i = 1;i<n-1;i++)
		for(int j=1; j<n;j++)
		{
			v[i][j] = vstar[i][j] + vfactor*dn[i][j]*(p_c[i][j] -
p_c[i+1][j]);
		}

		//Boundary conditions of v
		for(int j = 0; j<n+1;j++)
		{
			v[n][j] = 0;
			v[0][j] = 0;
		}
		for(int i =0;i<n;i++)
		{
			v[i][0] = -v[i][1];
			v[i][n] = -v[i][n-1];
		}
		for(int i = 25;i<=45;i++)
		for(int j = 27;j<=44;j++)
		v[i][j]=0;
```

```
	for(int i=25;i<=45;i++)
	{
		v[i][25] = - v[i][26];
		v[i][46] = - v[i][45];
	}

	// Copying pnew values into pstar

	for(int i=0;i<n+1;i++)
	for(int j=0;j<n+1;j++)
	pstar[i][j] = pnew[i][j];

	// Calculation of error
	error = 0;

	for(int i=1;i<n;i++)
	for(int j=1;j<n;j++)
	error+= fabs(b[i][j]);


	cout<<"iteration = "<<++iter<<endl;

}

// Calculations of collocated variables
for(int i= 0;i<n;i++)
for(int j=0;j<n;j++)
{
	ufinal[i][j] = (u[i][j] + u[i+1][j])/2;
	vfinal[i][j] = (v[i][j] + v[i][j+1])/2;

}

// Storing the output in files
fstream ufile,vfile;
switch (counter)
{
	case 1: ufile.open("u_vel_central.txt",ios::out);
			vfile.open("v_vel_central.txt",ios::out);

			break;
	case 2: ufile.open("u_vel_upwind.txt",ios::out);
			vfile.open("v_vel_upwind.txt",ios::out);

			break;
	case 3: ufile.open("u_vel_hybrid.txt",ios::out);
			vfile.open("v_vel_hybrid.txt",ios::out);

			break;
	case 4: ufile.open("u_vel_powerlaw.txt",ios::out);
			vfile.open("v_vel_powerlaw.txt",ios::out);

			break;

}
```

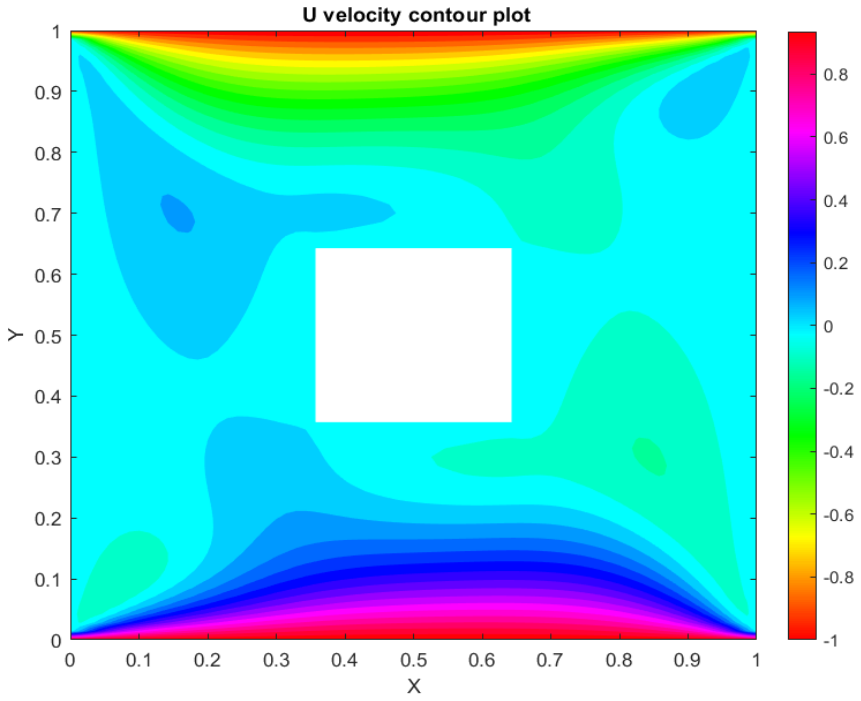
```
	for(int i =0;i<n;i++)
	{
		for(int j=0;j<n;j++)
		{
			ufile<<ufinal[i][j]<<"\t";
			vfile<<vfinal[i][j]<<"\t";

		}
		ufile<<endl;
		vfile<<endl;

	}

	ifile<<"Iteration in method "<<counter<<" = "<<iter<<endl;
	cout<<"Iteration in method "<<counter<<" = "<<iter<<endl;

	// Run the loop for different methods
	if(counter<=3)
	{	counter++;
		iter =0;
		goto label;
	}

return 0;
}
```

OUTPUT:

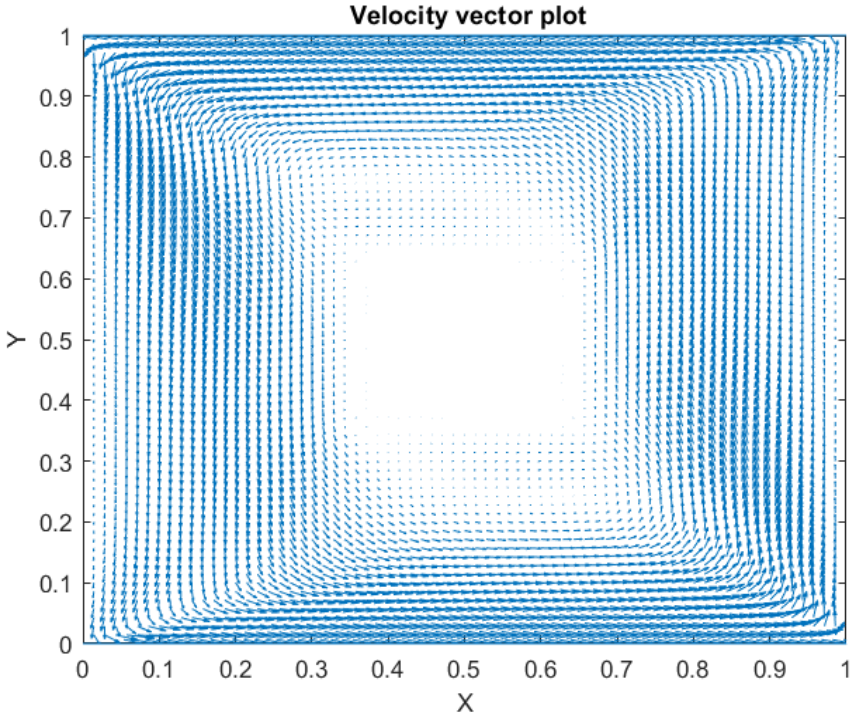
SCHEME	NO. OF ITERATIONS TO CONVERGE
CENTRAL DIFFERENCE SCHEME	3478
UPWIND SCHEME	3555
HYBRID SCHEME	3478
POWER-LAW SCHEME	3313

The number of iterations can be reduced by properly choosing the reduction factors, i.e., the vfactor, ufactor and pfactor variables in the code.

RESULTS:



The U-velocity contour plot has perfect symmetry as expected. The bpundary conditions can be be seen clearly at the top and bottom lids.



Since the top and bottom lids are moving in opposite directions a giant vortex is created throughout the domain. That can be seen in the velocity vector plot above.