# ME 543 ASSIGNMENT-02
# Flow in 2D Lid-driven Square cavity

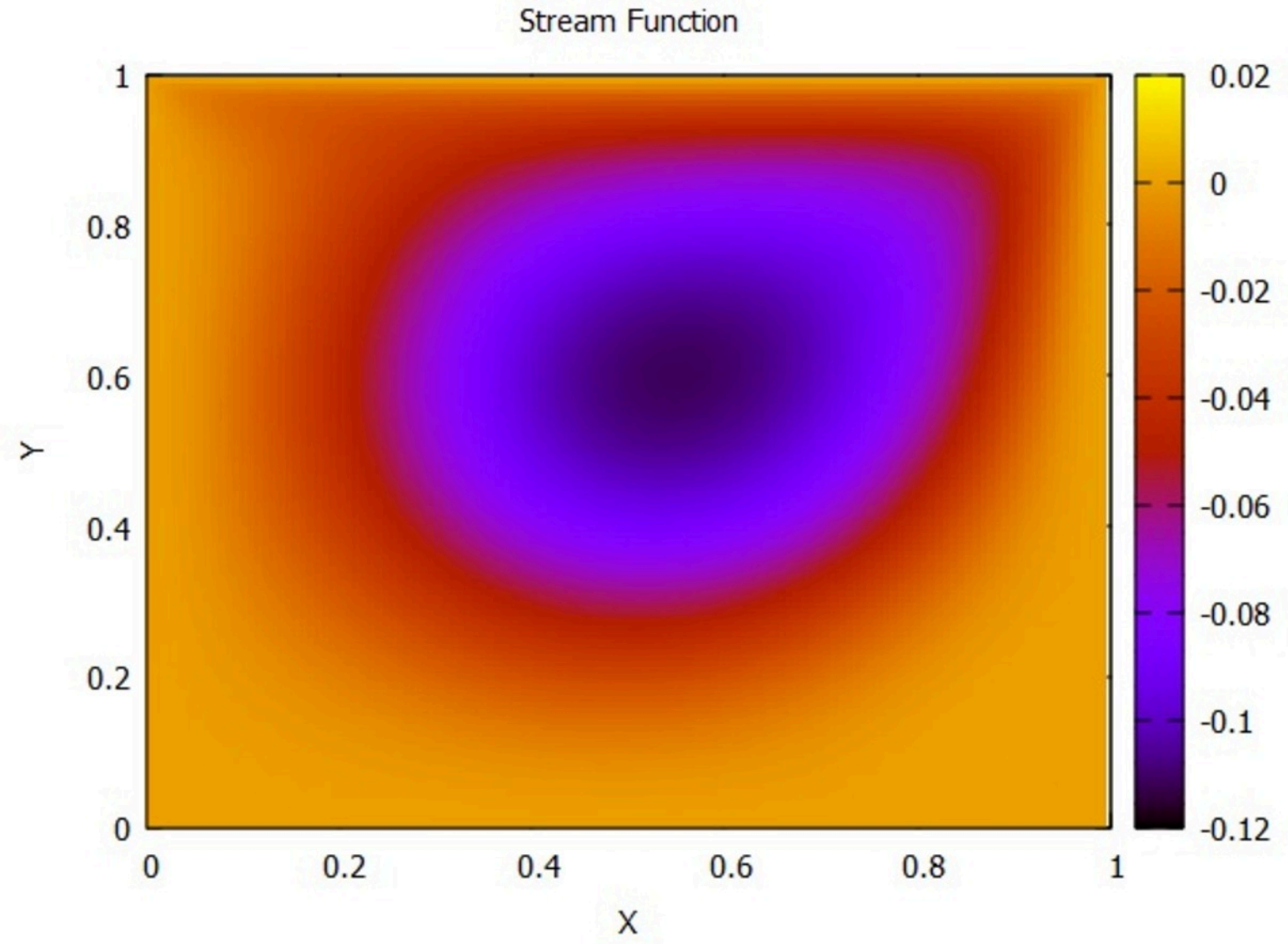
- Course instructor-
- Anoop k. Dass

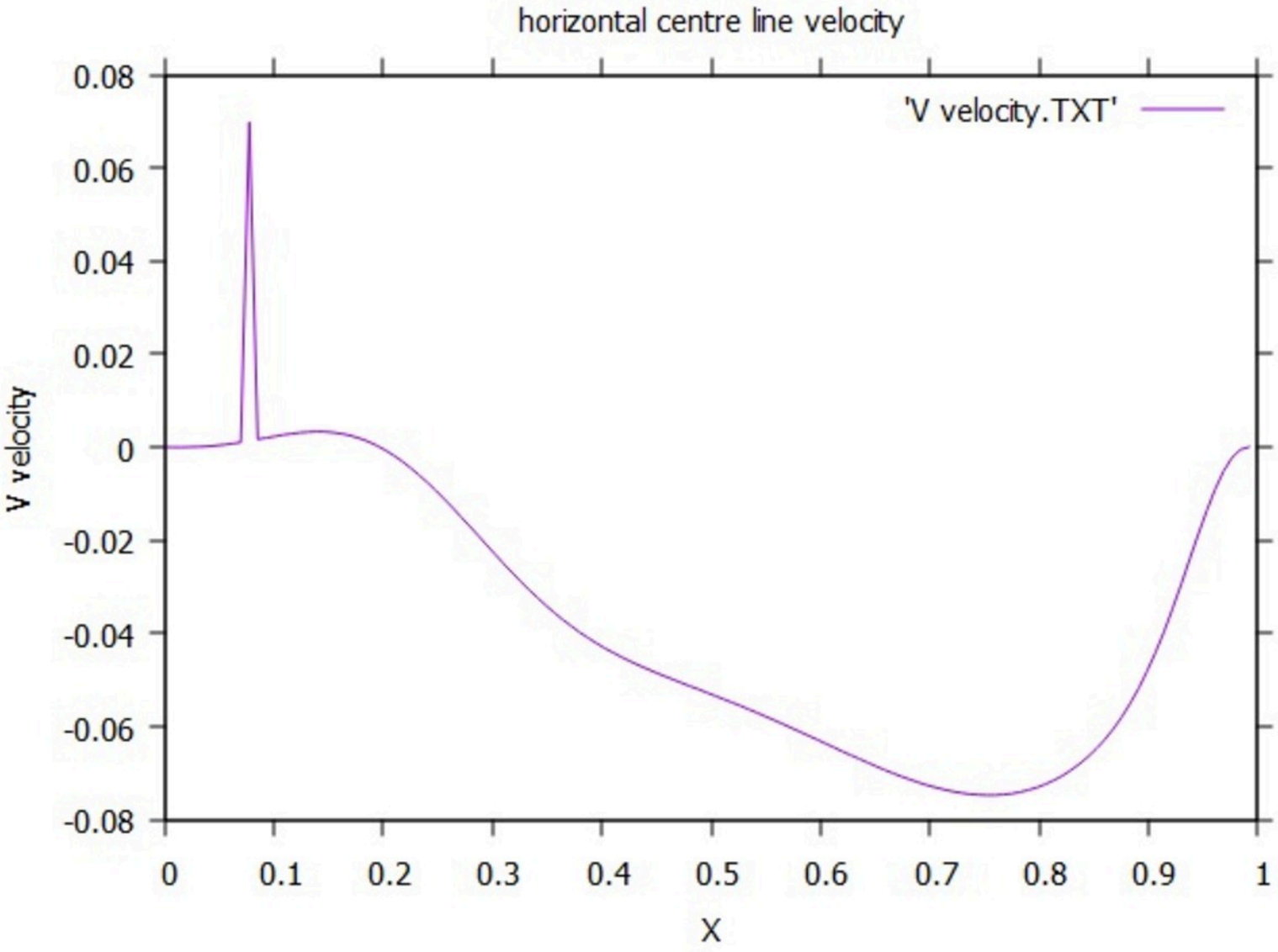

- Submitted by-
- Babalesh kumar
- Roll no 214103007
- Mtech
- Aerodynamics and propulsion

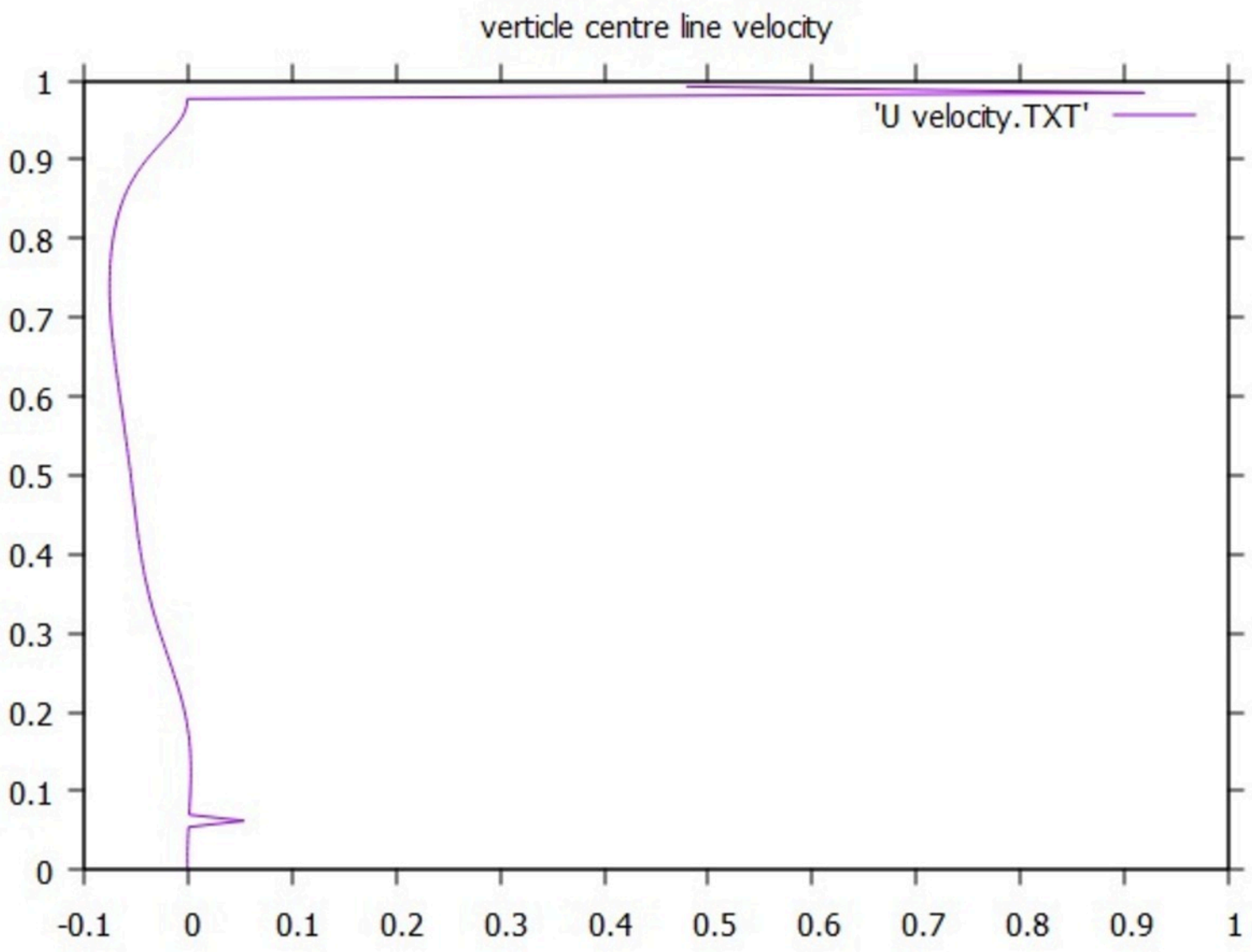
```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define Ngx 129
#define Ngy 129
#define errormax 0.000001
int main()
{
    int i, j, n = 0;            /* i,j Array Index ; n Iteration count*/
    double Re = 400.0, beta, u_velocity = 1.0, x[Ngx], y[Ngy]; /*Ngx,Ngy No. of
grid*/
    double edummy = (double)0.0, error = (double)1.0;
    double delx = (double)1 / (double)Ngx, dely = delx;
    /*delx, dely Grid spacing*/

    double shi[Ngx][Ngy];
    double shi_dm[Ngx][Ngy], omega[Ngx][Ngy], omega_dm[Ngx][Ngy];
    printf("Reynolds Number=%lf\tUvelocitylid=%lf\tError=%lf\tdelx=%lf\tdely=%lf\
n", Re, u_velocity, error, delx, dely);
    double edummyNr = 0, edummyDr = 0;
    double uvelocity[1][j], vvelocity[1][j];
    /*  edummy dummy variable for error */
    beta = 1;
    /* array declaration*/
    for (i = 0; i < Ngx; i++)
    {
        for (j = 0; j < Ngy; j++)
        {
            shi[i][j] = 0, shi_dm[i][j] = 0;
            if (i == 0 || i == 128)
            {
                omega[i][j] = 0, omega_dm[i][j] = 0;
            }
            else
            {
                if (j == 128)
                {
                    omega[i][j] = -2 * 1 / dely, omega_dm[i][j] = -2 * 1 / dely;
                }
                else
                {
                    omega[i][j] = 0, omega_dm[i][j] = 0;
                }
            }
        }
    } /* Initiation of iteration */
    while (error > errormax)
    {
        n = n + 1;
        for (i = 1; i < (Ngx - 1); i++)
        {
            for (j = 1; j < (Ngy - 1); j++)
            {
                shi_dm[i][j] = (shi_dm[i + 1][j] + shi_dm[i - 1][j] + (beta * beta
* (shi_dm[i][j + 1] + shi_dm[i][j - 1])) + (delx * delx * omega[i][j])) / (2 * (1 +
(beta * beta)));
                /* Gauss Seidel Scheme for Stream function*/
            }
```

```
        }
        /*  finding vorticity at boundaries*/
        for (j = 1; j < (Ngy - 1); j++)
        {
            omega[0][j] = (-2 * shi_dm[1][j]) / (delx * delx), omega_dm[0][j] =
omega[0][j];
            omega[128][j] = (-2 * shi_dm[127][j]) / (delx * delx), omega_dm[128][j]
= omega[128][j];
        }
        for (i = 1; i < (Ngx - 1); i++)
        {
            omega[i][0] = (-2 * shi_dm[i][1]) / (dely * dely), omega_dm[i][0] =
omega[i][0];
            omega[i][128] = ((-2 * shi_dm[i][127]) - (2 * dely * u_velocity)) /
(dely * dely), omega_dm[i][128] = omega[i][128];
        }
        for (i = 1; i < (Ngx - 1); i++)
        {
            for (j = 1; j < (Ngy - 1); j++)
            {
                omega_dm[i][j] = (omega[i + 1][j] + omega_dm[i - 1][j] + (beta *
beta * (omega[i][j + 1] + omega_dm[i][j - 1])) - (0.25 * beta * Re * (omega[i + 1]
[j] - omega_dm[i - 1][j]) * (shi[i][j + 1] - shi[i][j - 1])) + (0.25 * beta * Re *
(omega[i][j + 1] - omega_dm[i][j - 1]) * (shi[i + 1][j] - shi[i - 1][j]))) / (2 *
(1 + (beta * beta)));
                /* Gauss Seidel Scheme for Vorticity*/
            }
        }
        for (i = 0; i < Ngx; i++)
        {
            for (j = 0; j < Ngy; j++)
            {
                edummyNr = edummyNr + fabs((omega_dm[i][j] - omega[i][j])),
edummyDr = edummyDr + fabs(omega_dm[i][j]);
            }
        }
        edummy = edummy + (edummyNr / edummyDr); /* Error finding */
        for (i = 0; i < Ngx; i++)
        {
            for (j = 0; j < Ngy; j++)
            {
                shi[i][j] = shi_dm[i][j], omega[i][j] = omega_dm[i][j]; /* updating
stream function and vorticity */
            }
        }
        error = edummy, edummy = 0, edummyNr = 0, edummyDr = 0;
    }
    printf("\nError is %lf \n", error);
    printf("Total number of iteration = %d \n", n);
    /* Grid generation to save the output */
    x[0] = 0.0;
    y[0] = 0.0;
    for (j = 0; j < Ngy; j++)
    {
        y[j + 1] = y[j] + dely;
    }
    for (i = 0; i < Ngx; i++)
    {
        x[i + 1] = x[i] + delx;
```

```
    }
    FILE *g, *p, *vv, *uv;
    g = fopen("stream_fn.dat", "w");
    fprintf(g, "\ngx\t\t Y\t\t Stream_Function\n", Ngy, Ngx);
    for (i = 0; i < Ngx; i++)
    {
        for (j = 0; j < Ngy; j++)
        {
            fprintf(g, "%lf \t %lf \t %lf \n", x[i], y[j], shi[i][j]); /* save the
stream function values into file*/
        }
    }
    fclose(g);
    p = fopen("vorticity.dat", "w");
    fprintf(p, "\ngx\t\t Y\t\t Vorticity\n", Ngy, Ngx);
    for (i = 0; i < Ngx; i++)
    {
        for (j = 0; j < Ngy; j++)
        {
            fprintf(p, "%lf \t %lf \t %lf \n", x[i], y[j], omega[i][j]); /* save
the vorticity values into file */
        }
    }
    fclose(p);
    for (j = 0; j < Ngy; j++)
    {
        uvelocity[0][j] = (shi[64][j + 1] - shi[64][j - 1]) / (2 * dely),
vvelocity[0][j] = (shi[65][j] - shi[63][j]) / (2 * delx);
    }
    uv = fopen("U_velocity.dat", "w");
    fprintf(uv, "\ngx\t Y\t\tU-velocity\n", Ngy, Ngx);
    for (j = 0; j < Ngy; j++)
    {
        fprintf(uv, "L/2 \t %lf \t %lf \n", y[j], uvelocity[0][j]); /* save the u-
velocity values into file */
    }
    fclose(uv);
    vv = fopen("V_velocity.dat", "w");
    fprintf(vv, "X\t Y\t\tV-velocity\n", Ngy, Ngx);
    for (j = 0; j < Ngy; j++)
    {
        fprintf(vv, "L/2 \t %lf \t %lf \n", y[j], vvelocity[0][j]); /* save the v-
velocity values into file */
    }
    fclose(vv);
}
```


Stream Function
1
0.8
0.6
0.4
0.2
0
Y
0
0.2
0.4
0.6
0.8
1
X
0.02
0
-0.02
-0.04
-0.06
-0.08
-0.1
-0.12


horizontal centre line velocity
'V velocity.TXT'
V velocity
X
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


verticle centre line velocity
'U velocity.TXT'
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1