ASSIGNMENT 1

ME 543 Computational Fluid Mechanics

By ----

MASTER OF TECHNOLOGY

COMPUTATIONAL MECHANICS

INDIAN INSTITUTE OF TECHNOLOGY, GUWAHATI

DATE OF SUBMISSION : 5-OCT-2020

COURSE INSTRUCTOR: PROF. ANOOP K DASS

1.GAUSS SEIDEL METHOD

```
#include <stdio.h>
#include <math.h>
#include<stdlib.h>

#define Nx 21
#define Ny 41
#define errmx 0.01

int main()
{
	int i, j, k = 0;
	/* i,j Array Index ; Nx,Ny No. of grids ; k Iteration counter*/
	double bet, x[Nx], y[Ny], es = 0, err = 1, dx = 0.05, dy = 0.05,
u[Nx][Ny], ukpl[Nx][Ny];
	/* bet is Beta parameter ; es dummy variable for error (err) ;
errmx Maximum error value ; dx,dy Grid spacing ; u Temperature array */
	bet = dx / dy;

	/* Temperature array declaration*/
	for (i = 0; i < Nx; i++) /* BC and IC defining*/
	{
		for (j = 0; j < Ny; j++)
		{
			if (j == 0)
			{
				u[i][j] = 100;
			}
			else
			{
				u[i][j] = 0;
			}
		}
	}
	/* ukpl - Dummy variable to be used in difference scheme*/
	for (j = 0; j < Ny; j++)
	{
		for (i = 0; i < Nx; i++)
		{
			ukpl[i][j] = u[i][j];
		}
	}

	printf("\n i \t j \t T(Gauss Seidel) \n");

	/* Initiation of iteration with specific to given condition on
error*/
	while (err > errmx)
	{
		k = k + 1;
		/*printf("\n K = %d \n", k);*/
		for (i = 1; i < (Nx - 1); i++)
		{
			for (j = 1; j < (Ny - 1); j++)
			{
```

```
                    ukpl[i][j] = (ukpl[i + 1][j] + ukpl[i - 1][j] +
(bet * bet * (ukpl[i][j + 1] + ukpl[i][j - 1]))) / (2 * (1 + (bet *
bet)));
                    /* Gauss Seidel Scheme*/
               }
          }
          for (i = 0; i < Nx; i++)
          {
               for (j = 0; j < Ny; j++)
               {
                    es = es + fabs((ukpl[i][j] - u[i][j]));  /* Error
Evaluation */
               }
          }
          for (i = 0; i < Nx; i++)
          {
               for (j = 0; j < Ny; j++)
               {
                    u[i][j] = ukpl[i][j];  /* updating u */
               }
          }

          err = es;
          es = 0;
     }
     for (i = 0; i < Nx; i++)
     {
          for (j = 0; j < Ny; j++)
          {
               printf("%d \t %d \t %lf\n", i, j, u[i][j]);
               /* Converged temperature values at each grid */
          }
     }
     printf("Error is %lf \n", err);
     printf("Total number of iteration = %d \n", k);

     /* Grid generation to save the output */
     x[0]=0.0;
     y[0]=0.0;

     for(j = 0; j < Ny; j++)
          {
               y[j+1]=y[j]+dy;
          }

     for(i = 0; i < Nx; i++)
          {
               x[i+1]=x[i]+dx;
          }
     FILE *g,*p;
     g=fopen("Gauss_Seidal.dat","w");
     fprintf(g,"Variable =\"X\",\"Y\",\"Temprature\"\n",Ny,Nx);
     fprintf(g,"\tI=%d\tJ=%d\n",Nx,Ny);
     for(i = 0; i < Nx; i++)
     {
          for(j = 0; j < Ny; j++)
          {
               fprintf(g,"%lf\t%lf\t%lf\n",x[i],y[j],u[i][j]);
```

```
                /* saving of T values into file along with grids */
            }
        }
        fclose(g);
        p=fopen("Gauss_Seidal_nodes.dat","w");
        for(j = 0; j < Ny; j++)
        {
            fprintf(p,"%d\t%d\t%lf\n",11,j+1,u[10][j]);
            /* At i=11 and for all j T values */
        }
        fclose(p);
}
```

2.TIME MARCHING METHOD

```
#include <stdio.h>
#include <math.h>
#include<stdlib.h>

#define Nx 21
#define Ny 41
#define errmx 0.01

int main()
{
      int i, j, k = 0;
      /* i,j Array Index ; Nx,Ny No. of grids ; k Iteration counter*/
      double bet, x[Nx], y[Ny], es = 0, err = 1, dx = 0.05, dy = 0.05,
u[Nx][Ny], ukpl[Nx][Ny];
      /* bet is Beta parameter ; es dummy variable for error (err) ;
errmx Maximum error value ; dx,dy Grid spacing ; u Temperature array */
      bet = dx / dy;

      /* Temperature array declaration*/
      for (i = 0; i < Nx; i++) /* BC and IC defining*/
      {
            for (j = 0; j < Ny; j++)
            {
                  if (j == 0)
                  {
                        u[i][j] = 100;
                  }
                  else
                  {
                        u[i][j] = 0;
                  }
            }
      }
      /* ukpl - Dummy variable to be used in difference scheme*/
      for (j = 0; j < Ny; j++)
      {
            for (i = 0; i < Nx; i++)
            {
                  ukpl[i][j] = u[i][j];
            }
      }

      printf("\n i \t j \t T(Time Marching) \n");

      /* Initiation of iteration with specific to given condition on
error*/
      while (err > errmx)
      {
            k = k + 1;
            /*printf("\n K = %d \n", k);*/
            for (i = 1; i < (Nx - 1); i++)
            {
                  for (j = 1; j < (Ny - 1); j++)
                  {
                        ukpl[i][j] = (u[i + 1][j] + u[i - 1][j] +  u[i][j
+ 1] + u[i][j - 1] ) / (4);
```

```
                    /* Time Marching method*/
              }
       }
       for (i = 0; i < Nx; i++)
       {
              for (j = 0; j < Ny; j++)
              {
                     es = es + fabs((ukpl[i][j] - u[i][j]));
                     /* Error Evaluation */
              }
       }
       for (i = 0; i < Nx; i++)
       {
              for (j = 0; j < Ny; j++)
              {
                     u[i][j] = ukpl[i][j];  /* updating u */
              }
       }

       err = es;
       es = 0;
}
for (i = 0; i < Nx; i++)
{
       for (j = 0; j < Ny; j++)
       {
              printf("%d \t %d \t %lf\n", i, j, u[i][j]);
              /* Converged temperature values at each grid */
       }
}
printf("Error is %lf \n", err);
printf("Total number of iteration = %d \n", k);

/* Grid generation to save the output */
x[0]=0.0;
y[0]=0.0;

for(j = 0; j < Ny; j++)
       {
              y[j+1]=y[j]+dy;
       }

for(i = 0; i < Nx; i++)
       {
              x[i+1]=x[i]+dx;
       }
FILE *g,*p;
g=fopen("Time_marching.dat","w");
fprintf(g,"Variable =\"X\",\"Y\",\"Temprature\"\n",Ny,Nx);
fprintf(g,"\tI=%d\tJ=%d\n",Nx,Ny);
for(i = 0; i < Nx; i++)
{
       for(j = 0; j < Ny; j++)
       {
              fprintf(g,"%lf\t%lf\t%lf\n",x[i],y[j],u[i][j]);
              /* saving of T values into file along with grids */
       }
}
```

```
        fclose(g);
        p=fopen("Time_marching_nodes.dat","w");
        for(j = 0; j < Ny; j++)
        {
                fprintf(p,"%d\t%d\t%lf\n",11,j+1,u[10][j]);
                /* At i=11 and for all j T values */
        }
        fclose(p);
}
```

3.PSOR

```
#include <stdio.h>
#include <math.h>
#include<stdlib.h>

#define Nx 21
#define Ny 41
#define errmx 0.01
#define M_PI 3.14159265358979323846

int main()
{
	int i, j, k = 0;
	/* i,j Array Index ; Nx,Ny No. of grids ; k Iteration counter*/
	double omega, alpha, q_1, q_2, bet, x[Nx], y[Ny], es = 0, err = 1,
dx = 0.05, dy = 0.05, u[Nx][Ny], ukpl[Nx][Ny];
	/* bet is Beta parameter ; es dummy variable for error (err) ;
errmx Maximum error value ; dx,dy Grid spacing ; u Temperature array */
	bet = dx / dy;
	q_1 = (M_PI)/(Nx - 1);
	q_2 = (M_PI)/(Ny - 1);
	alpha = ((cos(q_1))+(bet*bet*cos(q_2)))/(1+(bet*bet));
	alpha = pow(alpha,2);
	omega = (2-(2*sqrt(1-alpha)))/(alpha);

	/* Temperature array declaration*/
	for (i = 0; i < Nx; i++) /* BC and IC defining*/
	{
		for (j = 0; j < Ny; j++)
		{
			if (j == 0)
			{
				u[i][j] = 100;
			}
			else
			{
				u[i][j] = 0;
			}
		}
	}
	/* ukpl - Dummy variable to be used in difference scheme*/
	for (j = 0; j < Ny; j++)
	{
		for (i = 0; i < Nx; i++)
		{
			ukpl[i][j] = u[i][j];
		}
	}

	printf("\n i \t j \t T(PSOR) \n");

	/* Initiation of iteration with specific to given condition on
error*/
	while (err > errmx)
	{
		k = k + 1;
		/*printf("\n K = %d \n", k);*/
```

```
            for (i = 1; i < (Nx - 1); i++)
            {
                  for (j = 1; j < (Ny - 1); j++)
                  {
                        ukpl[i][j] = ((1 - omega)*ukpl[i][j]) +
omega*((ukpl[i + 1][j] + ukpl[i - 1][j] + (bet * bet * (ukpl[i][j + 1] +
ukpl[i][j - 1]))) / (2 * (1 + (bet * bet))));
                        /* PSOR*/
                  }
            }
            for (i = 0; i < Nx; i++)
            {
                  for (j = 0; j < Ny; j++)
                  {
                        es = es + fabs((ukpl[i][j] - u[i][j]));
                        /* Error Evaluation */
                  }
            }
            for (i = 0; i < Nx; i++)
            {
                  for (j = 0; j < Ny; j++)
                  {
                        u[i][j] = ukpl[i][j];  /* updating u */
                  }
            }

            err = es;
            es = 0;
      }
      for (i = 0; i < Nx; i++)
      {
            for (j = 0; j < Ny; j++)
            {
                  printf("%d \t %d \t %lf\n", i, j, u[i][j]);
                  /* Converged temperature values at each grid */
            }
      }
      printf("Error is %lf \n", err);
      printf("Total number of iteration = %d \n", k);

      /* Grid generation to save the output */
      x[0]=0.0;
      y[0]=0.0;

      for(j = 0; j < Ny; j++)
            {
                  y[j+1]=y[j]+dy;
            }

      for(i = 0; i < Nx; i++)
            {
                  x[i+1]=x[i]+dx;
            }
      FILE *g,*p;
      g=fopen("PSOR.dat","w");
      fprintf(g,"Variables=\"X\",\"Y\",\"Temprature\"\n",Ny,Nx);
      fprintf(g,"\tI=%d\tJ=%d\n",Nx,Ny);
      for(i = 0; i < Nx; i++)
```

```
        {
                for(j = 0; j < Ny; j++)
                {
                        fprintf(g,"%lf\t%lf\t%lf\n",x[i],y[j],u[i][j]);
                        /* saving of T values into file along with grids */
                }
        }
        fclose(g);
        p=fopen("PSOR_nodes.dat","w");
        for(j = 0; j < Ny; j++)
        {
                fprintf(p,"%d\t%d\t%lf\n",11,j+1,u[10][j]);
                /* At i=11 and for all j T values */
        }
        fclose(p);
}
```

4.PSOR WITH DIFFERENT OMEGA

```
#include <stdio.h>
#include <math.h>
#include<stdlib.h>

#define Nx 21
#define Ny 41
#define errmx 0.01
#define M_PI 3.14159265358979323846

int main()
{
	int i, j, k = 0;
	/* i,j Array Index ; Nx,Ny No. of grids ; k Iteration counter*/
	double omega, w_i = 0.8, w_f = 2.0, bet, x[Nx], y[Ny], es = 0, err
= 1, dx = 0.05, dy = 0.05, u[Nx][Ny], ukpl[Nx][Ny];
	/* bet is Beta parameter ; es dummy variable for error (err) ;
errmx Maximum error value ; dx,dy Grid spacing ; u Temperature array */
	bet = dx / dy;

	FILE *g;
	g=fopen("PSOR.dat","w");
	fprintf(g,"\t Omega \t Iterations \n");

	for (w_i = 0.8; w_i <= 2.0; w_i=w_i+0.1)
	{
		/* Temperature array declaration*/
		for (i = 0; i < Nx; i++) /* BC and IC defining*/
		{
			for (j = 0; j < Ny; j++)
			{
				if (j == 0)
				{
					u[i][j] = 100;
				}
				else
				{
					u[i][j] = 0;
				}
			}
		}
		/* ukpl - Dummy variable to be used in difference scheme*/
		for (j = 0; j < Ny; j++)
		{
			for (i = 0; i < Nx; i++)
			{
				ukpl[i][j] = u[i][j];
			}
		}

		err = 1;
		k = 0;

		/* Initiation of iteration with specific to given condition
on error*/
		while (err > errmx)
		{
```

```
                    k = k + 1;
                    /*printf("\n K = %d \n", k);*/
                    for (i = 1; i < (Nx - 1); i++)
                    {
                          for (j = 1; j < (Ny - 1); j++)
                          {
                                ukpl[i][j] = ((1 - w_i)*ukpl[i][j]) +
w_i*((ukpl[i + 1][j] + ukpl[i - 1][j] + (bet * bet * (ukpl[i][j + 1] +
ukpl[i][j - 1]))) / (2 * (1 + (bet * bet))));
                                /* PSOR*/
                          }
                    }
                    for (i = 0; i < Nx; i++)
                    {
                          for (j = 0; j < Ny; j++)
                          {
                                es = es + fabs((ukpl[i][j] - u[i][j]));
                                /* Error Evaluation */
                          }
                    }
                    for (i = 0; i < Nx; i++)
                    {
                          for (j = 0; j < Ny; j++)
                          {
                                u[i][j] = ukpl[i][j];  /* updating u */
                          }
                    }

                    err = es;
                    es = 0;
              }
              printf("Total number of iteration = %d for %lf \n",k,w_i);
              fprintf(g,"\t %lf \t %d \n",w_i,k);

        }
        fclose(g);
}
```

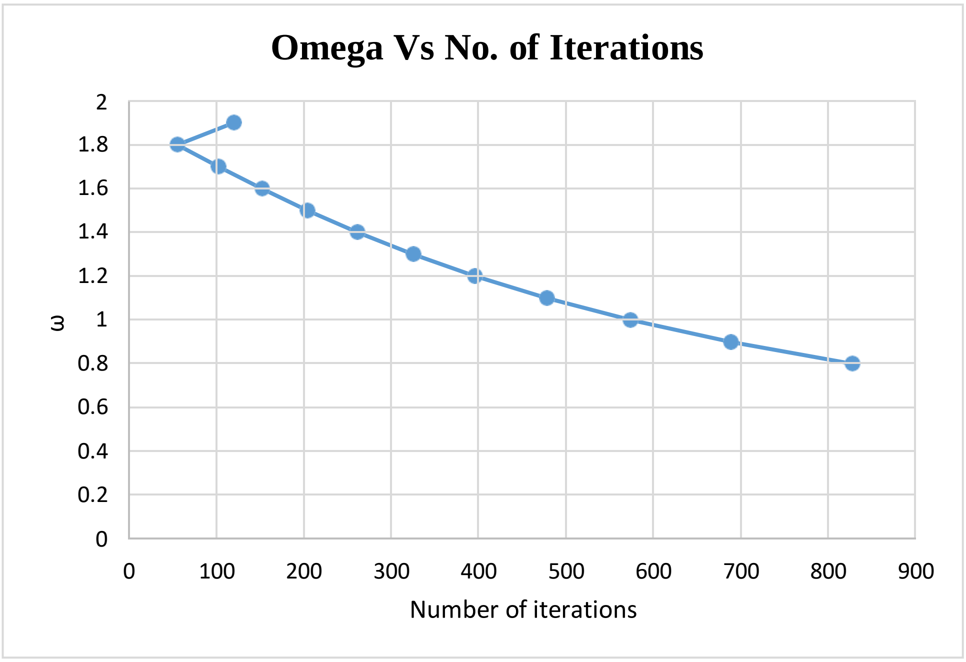
5.ANALYTICAL PART

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int main()
{
      FILE *g;
      g=fopen("Analytical.txt","w");
      float  pi=3.1416,X,Y,sum;
      float u[41][21];
      int n,i,j;
      for(i=0;i<41;i++)
      {
            for(j=0;j<21;j++)
            {
                  if(i==0)
                  u[i][j]=100;
                  else
                  u[i][j]=0;
            }
      }

      for(i=1;i<40;i++)
      {
            for(j=1;j<20;j++)
            {
                  X=i*0.05;
                  Y=j*0.05;
                  sum=0;
                  for(n=1;n<=110;n++)
                  {
                        sum=sum+((1-(pow(-1,n)))/(n*pi))*sinh((n*pi*(2-
X))/1)*sin(n*pi*Y/1)/sinh(n*pi*2/1);
                  }
                  u[i][j]=100*2*sum;
            }
      }
      printf("\n\n");
      for(i=0;i<41;i++)
      {
            for(j=0;j<21;j++)
            {
                  fprintf(g,"%0.6f\t",u[i][j]);
            }
            fprintf(g,"\n");
      }
}
```

TABLE 1

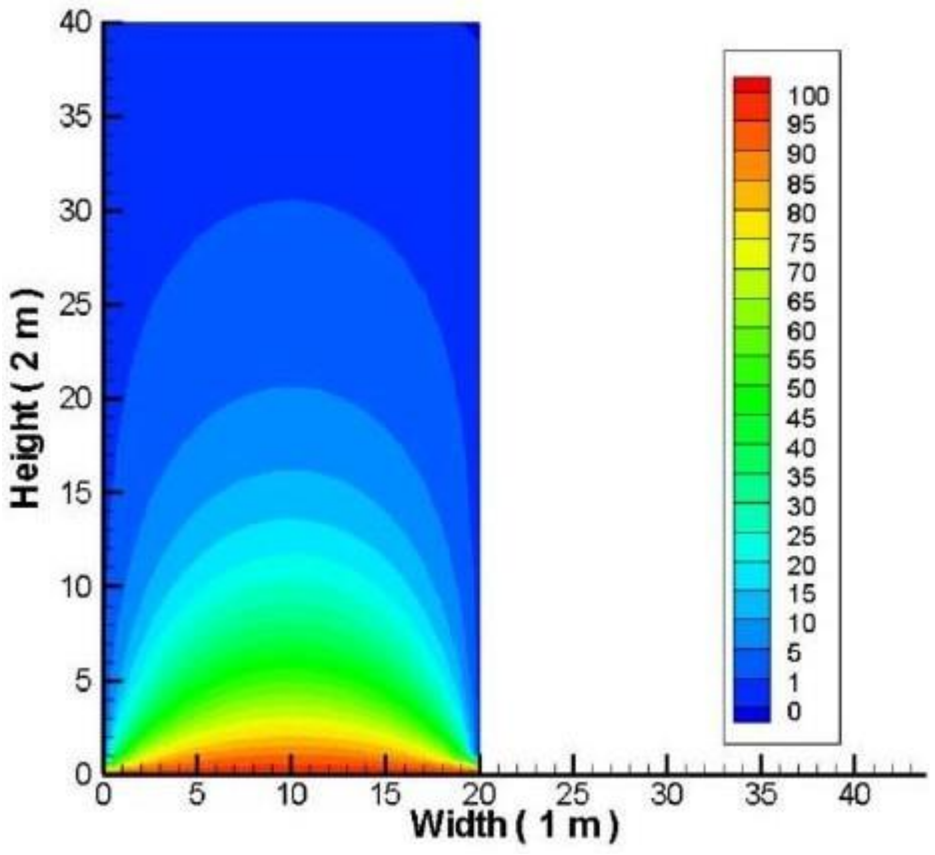
i	j	Temperature (^{o}C)			
		Gauss Seidel	**Time Marching**	**PSOR**	**Analytical**
11	1	100	100	100	100
11	2	90.00068	90.000545	90.000844	90.040497
11	3	80.250328	80.250056	80.250664	80.320564
11	4	70.966102	70.965692	70.96661	71.05175
11	5	62.311862	62.311315	62.312531	62.397831
11	6	54.390814	54.390129	54.391629	54.46579
11	7	47.249851	47.249032	47.250796	47.30756
11	8	40.89052	40.889569	40.891578	40.929092
11	9	35.281955	35.280876	35.28312	35.302517
11	10	30.372812	30.37161	30.374091	30.378119
11	11	26.100834	26.099515	26.102232	26.094181
11	12	22.399802	22.398373	22.401305	22.384403
11	13	19.204162	19.20263	19.205751	19.182808
11	14	16.45185	16.450224	16.453515	16.426819
11	15	14.085818	14.084107	14.087559	14.058867
11	16	12.054683	12.052897	12.05649	12.027115
11	17	10.312822	10.310971	10.314677	10.285572
11	18	8.820138	8.818234	8.822024	8.79385
11	19	7.54165	7.539703	7.543539	7.516745
11	20	6.446999	6.445022	6.448913	6.423728
11	21	5.509941	5.507946	5.511849	5.488438
11	22	4.707851	4.705851	4.709743	4.688162
11	23	4.021261	4.019268	4.023127	4.003369
11	24	3.433441	3.431468	3.435267	3.417292
11	25	2.93003	2.928089	2.931804	2.915539
11	26	2.498699	2.496804	2.500412	2.485768
11	27	2.128871	2.127033	2.130511	2.117392
11	28	1.811466	1.809697	1.813024	1.801327
11	29	1.538681	1.536994	1.540151	1.529775
11	30	1.30381	1.302216	1.305184	1.29603
11	31	1.101072	1.099581	1.102341	1.09432
11	32	0.925476	0.924099	0.926632	0.91966
11	33	0.772697	0.771445	0.773736	0.767735
11	34	0.638972	0.637853	0.63989	0.634789
11	35	0.521006	0.520028	0.521797	0.517537
11	36	0.415889	0.415061	0.416551	0.413081
11	37	0.321031	0.320358	0.321562	0.318837
11	38	0.234092	0.23358	0.23449	0.232476
11	39	0.152926	0.152582	0.153191	0.151863
11	40	0.075532	0.075358	0.075664	0.075004
11	41	0	0	0	0
Iterations		575	1076	51	

FIGURE 1



TEMPERATURE CONTOURS

Values in ^{0}C


Height (2 m)
Width (1 m)