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

Conjugate Gradient Method

Question-1:

Code for conjugate gradient method:

Consider 2D conduction problem governed by equation.

$$\frac{\partial^2 T}{\partial X^2} + \frac{\partial^2 T}{\partial Y^2} = -8\mu^2 \sin(2\mu x) \sin(2\mu y)$$

Take a square domain of width and height of 1 unit. The boundary conditions for all the boundaries are T = 0. Discretise the equation. Then solve the obtained system of linear equations by writing code for conjugate gradient method. This problem has an analytical solution which is given by series solution given by

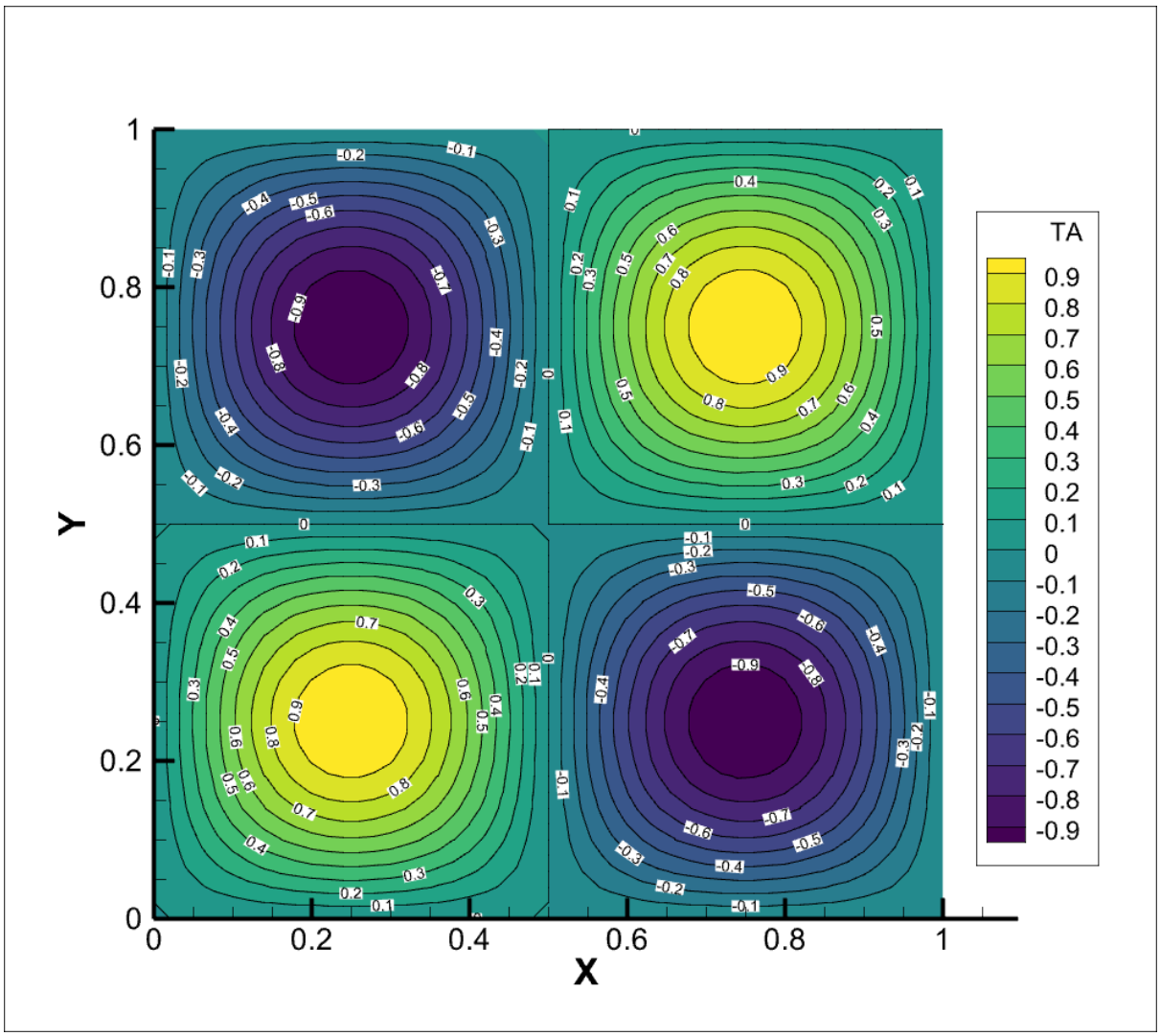
$$T(x, y) = \sin(2\mu x) \sin(2\mu y)$$

(a) Compare the temperature contours obtained by the code and with those obtained by using the above analytical expression (show the temperature contours figures side by side).

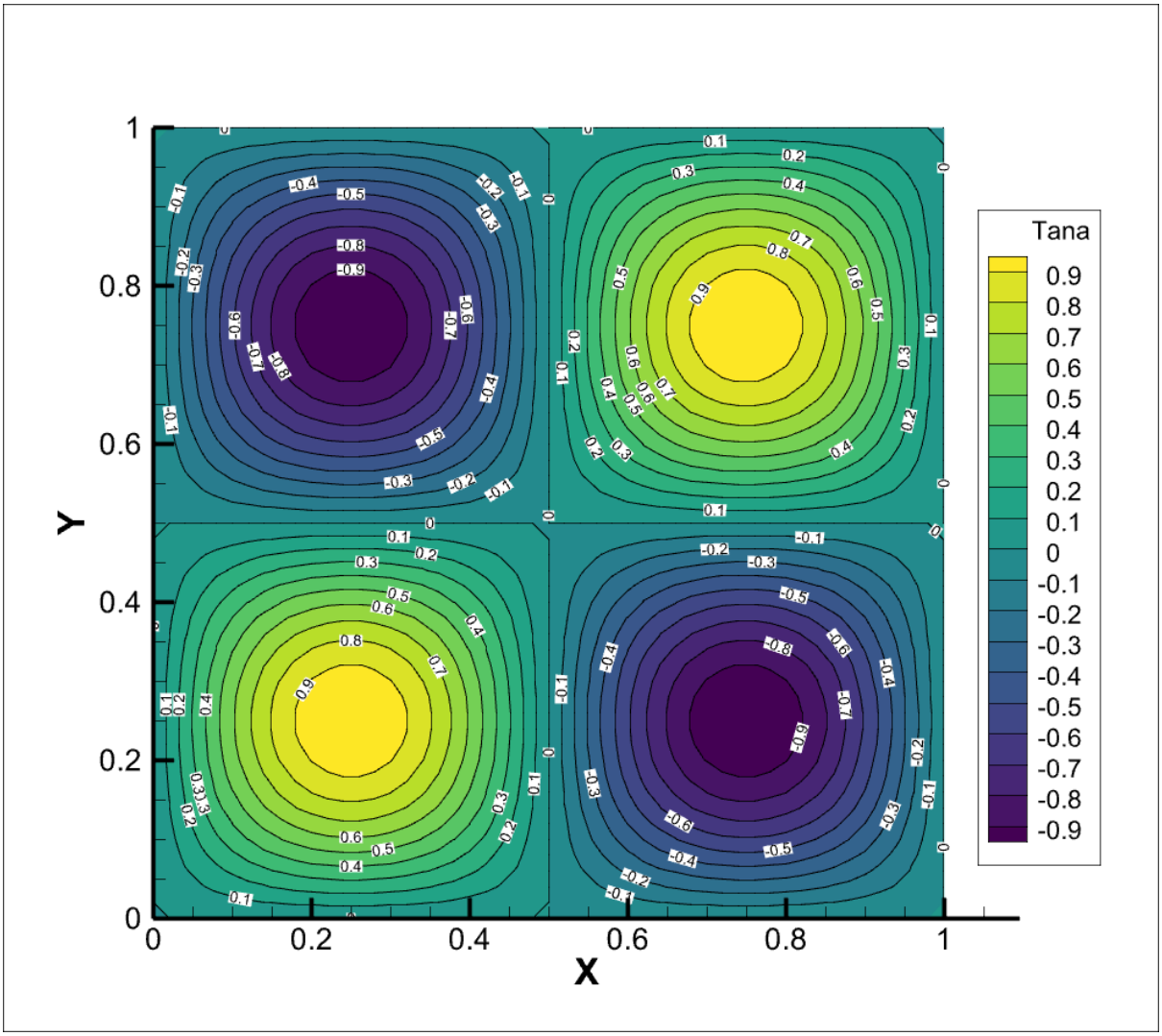
(b) Compare the temperature variation with y along mid-vertical plane x = 0.5 from the code and the above analytical expression. Plot both temperature profiles in the same figure.

(c) Compare the temperature variation with y along mid-horizontal plane y = 0.5 from the code and the above analytical expression. Plot both temperature profiles in the same figure.

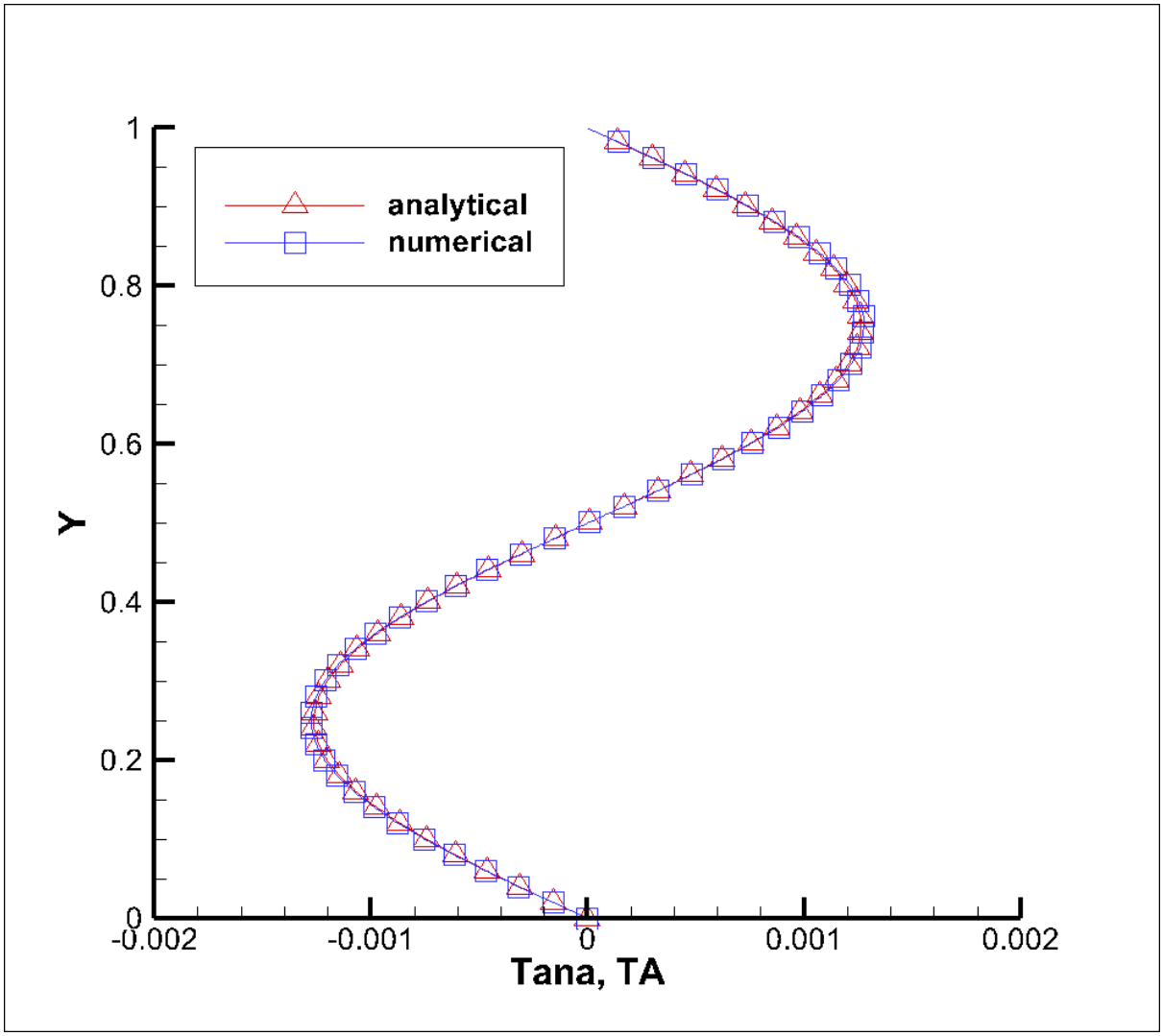
1. Ans



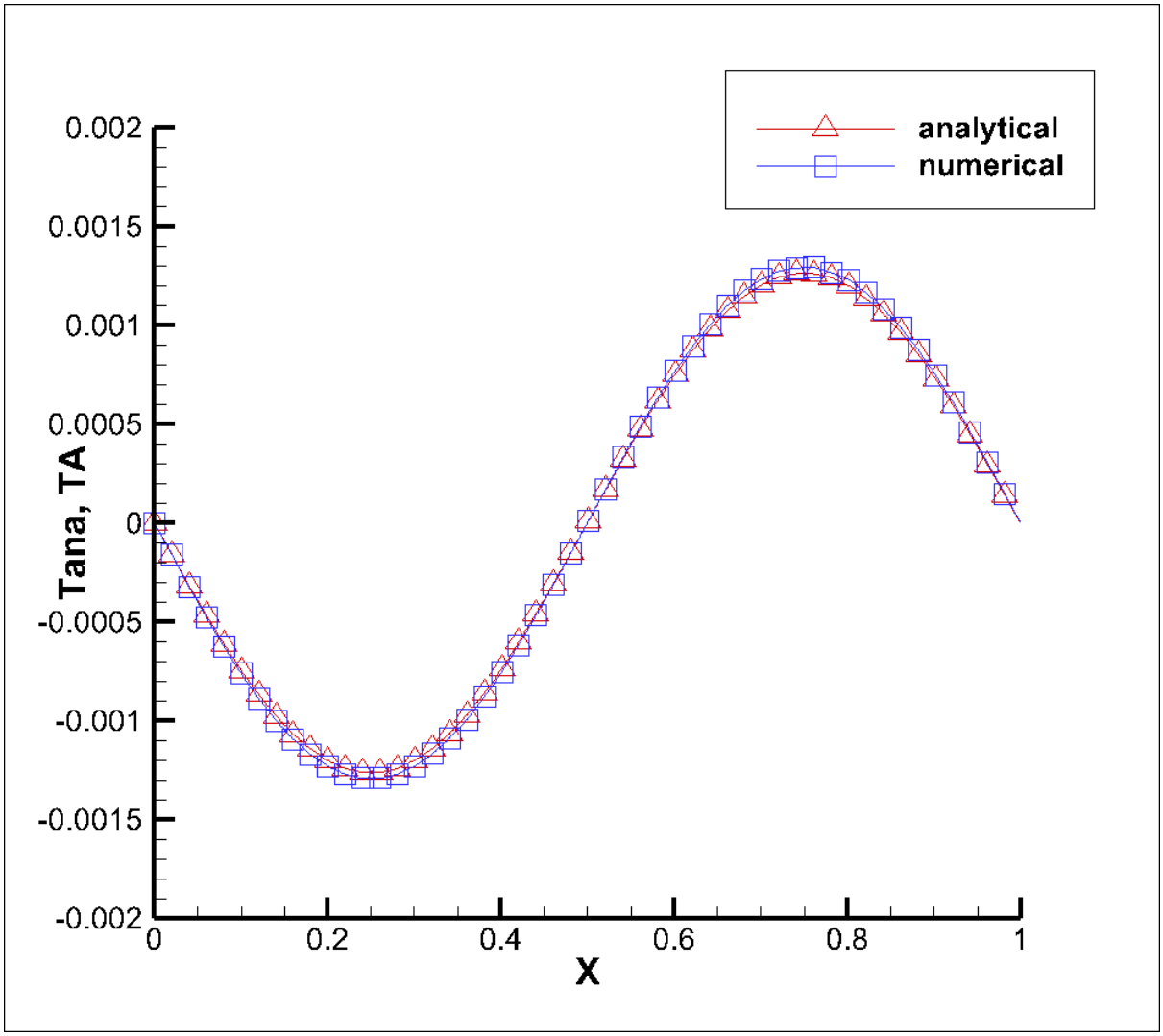
fig(a): Numerical



fig(b): Analytical results



2. Comparison of T variation along mid-vertical line (line x = 0.5)



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

1.1 The code of Conjugate Gradient method

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

# define ni 51
# define nj 51
// defining the function for 1 d locations
int L(int i, int j)
{
   int l;
   l = (i-1) *nj + (j-1);
   return l;
}
// main methode
int main(void)
{
   double  T[ni*nj], rold[ni*nj], r[ni*nj] ,
b[ni*nj],p[ni*nj],Ap[ni*nj],TA[ni][nj] ;
   double ae,aw,an,as,ap,dx,dy,gamma,pi=(22.0/7);
   double alpha,beta,error,rtr,ptAp,rtorto;
   double
lp[ni*nj],ue[ni*nj],lw[ni*nj],un[ni*nj],ls[ni*nj],Tem[ni*nj],Tana[
ni][nj];
```

```
int i, j, iter=0,l,k;

// calculating the coefficients
dx = 1.0/(ni-1);
dy = 1.0/(nj-1);
gamma = dx/dy;
ae = 1.0;
aw = 1.0;
an = gamma*gamma;
as = gamma * gamma;
ap = -2*(1+(gamma*gamma));
//printf("%f\n",an);
// crating a matrix and initializing matrix
for ( i = 1; i <=ni; i++)
{
   for ( j = 1; j <=nj; j++)
   {
      l = L(i,j);
      lw[l] = aw;
      ls[l] = as;
      lp[l] = ap;
      un[l] = an;
      ue[l] = ae;
      T[l] = 0.0;
```

```
b[l] =0.0;
// boundary conditions
b[l] =  -8 *(pi*pi)*(sin(2*pi*(i-1)*dx)*sin(2*pi*(j-
1)*dy))*(dx*dx) ;

if (i==1)
{
   lp[l] = 1.0;
   lw[l] = 0.0;
   ls[l] = 0.0;
   ue[l] = 0.0;
   un[l] = 0.0;
   b[l] =0.0;

}
if (j== 1)
{
   lp[l] = 1.0;
   lw[l] = 0.0;
   ls[l] = 0.0;
   ue[l] = 0.0;
   un[l] = 0.0;
   b[l] =0.0;
}
```

```
        if (i==ni)
        {
           lp[l] = 1.0;
           lw[l] = 0.0;
           ls[l] = 0.0;
           ue[l] = 0.0;
           un[l] = 0.0;
           //b[l] =0.0;
        }
        if (j==nj)
        {
           lp[l] = 1.0;
           lw[l] = 0.0;
           ls[l] = 0.0;
           ue[l] = 0.0;
           un[l] = 0.0;
           b[l] = 0.0;
        }

        //printf("%f\t",lw[l]);
    }
    //printf("\n");
}
for ( i =1; i <=ni; i++)
```

```
    {
        for ( j = 1; j <=nj; j++)
        {
            l = L(i,j);
            r[l] = b[l] - ( ue[l] * T[l+nj] + un[l] *T[l+1] + lp[l]*T[l] + ls[l]
*T[l-1] + lw[l] * T[l-nj] );
            p[l] = r[l];
            //printf("%f\t",p[l]);
        }
        //printf("\n");

    }
    do
    {
        /* code */


        // calculating alpha
        rtr =0;
        ptAp = 0;

        for ( i = 1; i <= ni; i++)
        {
            for ( j = 1; j <= nj; j++)
```

```
        {
            l = L(i,j);
            Ap[l] = (lp[l]*p[l]+lw[l]*p[l-nj]+ls[l]*p[l-
1]+ue[l]*p[l+nj]+un[l]*p[l+1]);
            rtr = rtr + r[l] *r[l];
            ptAp = ptAp + Ap[l] * p[l];

            Tem[l] = T[l];
            //printf("%f\t",p[l]);

        }

        //printf("\n");

    }
    alpha = rtr/ptAp;
    printf("alpha = %f\t",alpha);

    // updated value of T unknowns and assigning r to rold
    for ( i = 1; i <= ni; i++)
    {
        for ( j = 1; j <= nj; j++)
        {
            l = L(i,j);
```

```
        T[l] = T[l] + alpha * p[l];
        //printf("%f\t",r[l]);
        rold[l] = r[l];
    }
    //printf("\n");

}
// updated r
for ( i = 1; i <= ni; i++)
{
    for ( j = 1; j <= nj; j++)
    {
      l = L(i,j);
      r[l] = r[l] - alpha * Ap[l];
      //printf("%f\t",r[l]);
    }
    //printf("\n");
}
// calculating the beta
rtorto = 0.0;
rtr = 0.0;
for ( i = 1; i <= ni; i++)
{
    for ( j = 1; j <= nj; j++)
```

```
    {
        l = L(i,j);
        rtr = rtr + r[l] *r[l];
        rtorto =  rtorto + rold[l] * rold[l];
        //printf("%f\t",rtorto);
    }
    //printf("\n");

}
beta = rtr/rtorto ;
printf("beta = %f\t\t",beta);
// updating the direction
for ( i = 1; i <= ni; i++)
{
    for ( j = 1; j <= nj; j++)
    {
        l = L(i,j);
        p[l] = r[l] + beta * p[l];
        //printf("%f\t",p[l]);
    }
    //printf("\n");

}
// calculating error
```

```
    error =0.0;
    rtr =0.0;
    for ( i = 1; i <= ni; i++)
    {
       for ( j = 1; j <= ni; j++)
       {
          l = L(i,j);
          rtr = rtr + r[l]*r[l];
          rtorto = rtorto + rold[l] * rold[l];
          error = (rtr)  ;
          //printf("%f\t",rtr);
       }
       //printf("\n");

    }
    error = error/(ni*nj);
    iter = iter +1;
    printf("\titer = %d\t error = %.12f\n",iter,error);
} while (error>1e-10);
    for ( i = 1; i <= ni; i++)
    {
       for ( j = 1; j <= nj; j++)
       {
          l = L(i,j);
```

```
            TA[i-1][j-1] = T[l];
            printf("%f\t",TA[i-1][j-1]);
        }

    }
        // numerical solution
    FILE*file;
    file = fopen("T.plt","w");
    fprintf(file," VARIABLES = \"X\",\"Y\",\"TA\"\n");
    fprintf(file," ZONE F = POINT\n");
    fprintf(file,"I=%d, J=%d\n",ni,nj);
    for ( i = 1; i<= ni; i++)
    {
        for ( j = 1; j<= nj; j++)
        {
            l = L(i,j);
            TA[i-1][j-1] = T[l];
            //printf("%f\t",TA[i-1][j-1]);
            fprintf(file,"%f\t%f\t%f\n",(i-1)*dx,(j-1)*dy,TA[i-1][j-
1]);


        }
```

```
    }
  FILE*file1;
  file1 = fopen("Tanalytical.plt","w");
  fprintf(file1," VARIABLES = \"X\",\"Y\",\"Tana\"\n");
  fprintf(file1," ZONE F = POINT\n");
  fprintf(file1,"I=%d, J=%d\n",ni,nj);
    for ( i = 0; i< ni; i++)
    {
      for ( j = 0; j< nj; j++)
      {
        Tana[i][j] = sin(2*pi*i*dx)*sin(2*pi*j*dy);
        //printf("%f\t",TA[i-1][j-1]);
        fprintf(file1,"%f\t%f\t%f\n",(i)*dx,(j)*dy,Tana[i][j]);


      }

    }


}
```