

Solving Elliptical Differential Equations using C

Paht Juangphanich

I. Introduction

The purpose of this paper is to illustrate the procedure for solving an elliptical differential Equation using C. Half of the paper will be the numerical approach and the other half will be the details of the C program. It is essential to understand how to solve these types of equations. Methods used in solving elliptical differential equation may also be used in solving other types of differential equations.

II. Theory

Lets start with the Partial Differential Equation form of a basic elliptical differential equation.

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + Du_x + Eu_y + F = 0 \quad (1)$$

Examples of elliptical differential equations consist of the Laplace equation, Poisson equation, and the Helmholtz equation.

$$\nabla^2 u = 0 \text{ Laplace's Equation} \quad (2)$$

$$\nabla^2 u = g(x, y) \text{ Poisson} \quad (3)$$

$$\nabla^2 u + f(x, y)u = g(x, y) \text{ Helmholtz} \quad (4)$$

(5)

Lets start with a basic example of Poisson's equation and discretize using central differencing technique.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = g(x, y) \quad (6)$$

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} = g_{i,j} \quad (7)$$

(8)

Assuming that $dx = dy$. Adding these two equations together results in:

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} = 0 \quad (9)$$

Applying SOR(successive over relaxation) method:

$$u_{i,j} = u_{i,j} + w * r_{i,j}$$
$$r_{i,j} = \frac{(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j})}{4}$$

A. What if dx is not equal to dy

If dx is not equal to dy the procedure hasn't changed much. Only thing that will be different is the coefficients. Equation 7 can be reduced to:

$$u_{i,j} - \theta_x(u_{i+1,j} + u_{i-1,j}) - \theta_y(u_{i,j+1} + u_{i,j-1}) = -\delta^2 f(i, j) \quad (10)$$

$$\theta_x = \frac{(\Delta y)^2}{2((\Delta x)^2 + (\Delta y)^2)} \quad (11)$$

$$\theta_y = \frac{(\Delta x)^2}{2((\Delta x)^2 + (\Delta y)^2)} \quad (12)$$

$$\delta^2 = \frac{(\Delta y)^2(\Delta x)^2}{2((\Delta x)^2 + (\Delta y)^2)} \quad (13)$$

The SOR of equation 10 would be:

$$u_{i,j} = u_{i,j} + w * r_{i,j} \quad (14)$$

$$r_{i,j} = -u_{i,j} + \theta_x(u_{i+1,j} + u_{i-1,j}) + \theta_y(u_{i,j+1} + u_{i,j-1}) \quad (15)$$

$$(16)$$

w is the over-relaxation factor $1 < w < 2$. The optimal value of w can be calculated by the following equation however any value between the bounds could be used. It is important to note that the matrix that we need to form to solve the elliptical differential equation has to be a square matrix, otherwise there would be no solution. This is true with all partial differential equations. The number of equations has to equal the number of unknowns.

$$w = \frac{4}{2 + \sqrt{4 - \left(\cos\left(\frac{\pi}{n-1}\right) + \cos\left(\frac{\pi}{m-1}\right) \right)}} \quad (17)$$

III. Numerical Approach

The code below shows how to solve a basic elliptical differential equation in C. Note: dx does not have to equal to dy in this case.

The program below is used to solve for:

$$u(x, 0) = 20u(x, 4) = 1800 \quad 0 < x < 4$$

$$u(0, y) = 80u(4, y) = 00 \quad 0 < y < 4$$

Listing 1. Descriptive Caption Text

```

1  /*
   *   elliptic.c, compile with gcc -fopenmp elliptic.c
3  *
   *
5  *   Created by Paht Juangphanich on 2/5/11.
   *   Copyright 2011 Paht Juangphanich. All rights reserved.
7  *
   */
9  #include <stdio.h>
   #include <stdlib.h>
11 #include <math.h>
   #include <omp.h>
13
   #define pi 3.14159265
15
   void printresult(double **u, int n);
17 int main(){
   int n,i,j; // n = number of elements in x direction
19   // m = number of elements in y direction
   double ue,uw,un,us,a,b,ave;
21   double **u,w,relx;
   double err=1,tol=1E-6;
23   int max = 5000,c=0;
   printf("Please input how many elements in x and y:"); scanf("%d",&n);
25   printf("x1=0,x2="); scanf("%lf",&a);
   printf("y1=0,y2="); scanf("%lf",&b);
27   ue=0;
   uw=80;
29   us=20;
   un=180;
31   u=(double**) malloc(n*sizeof(double));
   #pragma omp parallel for default(none) shared(u,n) private(i)

```



```

33     for (i=0; i<n; ++i) {
34         u[i]=(double*)malloc(n*sizeof(double));
35     }
36     // Set boundaries
37     ave = (a*(ue+us)+b*(un+uw))/(2*a+2*b);
38     printf("ave = %lf\n", ave);
39     // Set interior nodes
40     #pragma omp parallel default(none) shared(u, ave, n) private(j, i)
41     for (i=0; i<n; ++i) {
42         #pragma omp for schedule(static, 10)
43         for (j=0; j<n; ++j) {
44             u[i][j] = ave;
45         }
46     }
47     // Set corner nodes
48     u[0][0] = (uw+us)/2;
49     u[0][n-1]=(un+uw)/2;
50     u[n-1][n-1]=(un+ue)/2;
51     u[n-1][0]=(ue+us)/2;
52     // Set wall nodes
53     #pragma omp parallel for default(none) shared(n, uw, un, us, ue, u) private(i) \
54         schedule(static, 10)
55     for (i=1; i<n-1; ++i) {
56         u[0][i] = uw;
57         u[i][0] = us;
58         u[n-1][i] = un;
59         u[i][n-1] = ue;
60     }
61     // SOR Parameter
62     w=4/(2+sqrt(4-pow(cos(pi/n-1)*2,2)));
63     printf("\nSOR proceed\n");
64     #pragma omp parallel default(none) shared(n, w, u, tol, max, err, c) private(i, j, relx)
65     {
66         while ((err>tol)&&(c<max)) {
67             #pragma omp master
68             err = 0;
69             for (i=1; i<n-1; ++i){
70                 #pragma omp for schedule(static, 10)
71                 for (j=1; j<n-1; ++j) {
72                     relx = w*(u[i][j+1]+u[i][j-1]+u[i-1][j]+u[i+1][j]-4*u[i][j])
73                     u[i][j]=u[i][j]+relx;
74                     if (err<abs(relx)){
75                         #pragma omp critical
76                         {
77                             err = abs(relx);
78                         }
79                     }
80                 }
81             }
82             #pragma omp flush(u)
83             #pragma omp atomic
84             ++c;
85         }
86     }
87     printf("err = %6.6lf\n", err);
88     printf("iter = %d\n", c);
89     // now u can be solved using LU or something

```



```

    printresult(u, n);
91     return 0;
    }
93
void printresult(double **u, int n){
95     FILE *fp;
    fp=fopen("results.txt", "w");
97     int i, j;
    for (i=0; i<n; ++i) {
99         for (j=0; j<n; ++j) {
                fprintf(fp, "%4.4lf ", u[i][j]);
101            }
            fprintf(fp, "\n");
103        }
    }
}

```