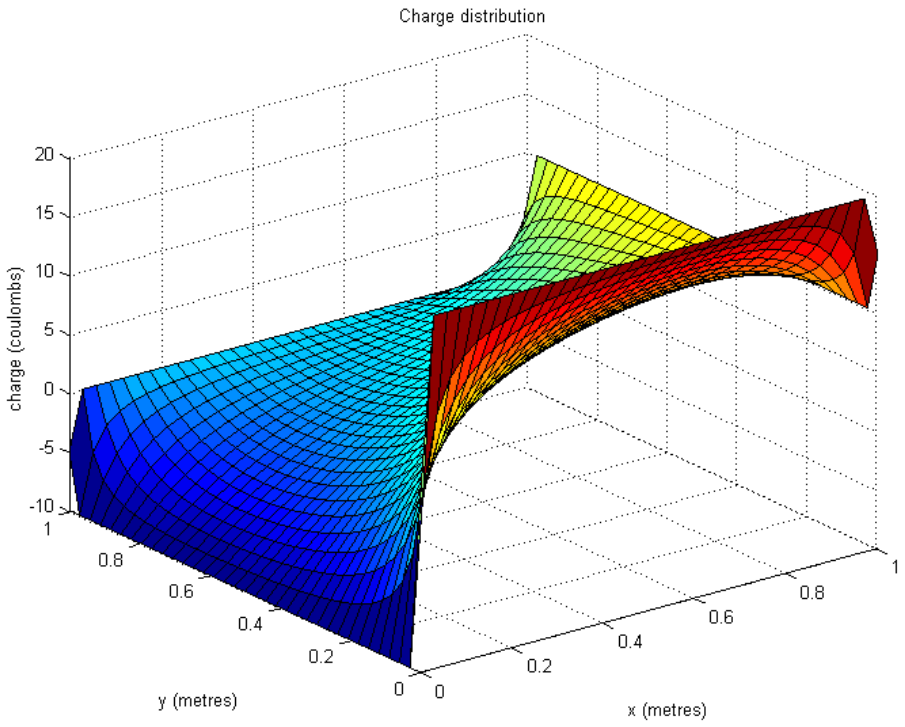
Evaluating Charge Distribution Using Approximate Methods



Allan Bruce

Department of Engineering

UNIVERSITY OF ABERDEEN

Contents

List of figures

1. The Problem

The problem given is to calculate the charge at several distinct points equally spaced within a given system. The system is arranged as shown below in figure 1.

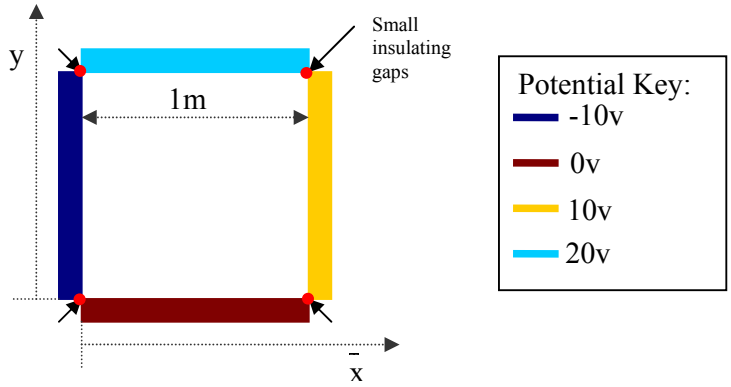


Figure 1: Charge Problem

There are four charged edges of a square with 1m gap between each opposite side. At each corner there is a small insulating gap with a relative permitivity of 1. The dielectric within the square also has a permitivity of 1. Each side has a charge as labelled in figure 1. These edges satisfy the boundary conditions. It is desirable to find the charge distribution within the system using approximate methods and a programming language of our choice.

2. Approximate Methods

The method used to solve the problem was successive over relaxation. This method structures the problem into a grid which is easily programmable using an array. The method is a derivation of the Finite Difference Method as detailed below.

2.1 Finite Difference Method

Consider the 2D heat flow problem as shown in figure 2.

Figure 2: 2D Heat Flow Problem

The temperature can be evaluated using the Laplace partial differential equation

$$\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial x^2} = 0$$

If we now consider a first order Taylor series expansion of T in the positive direction of x to form the forward difference

$$T(x+\Delta x, y) = T(x,y) + \frac{\partial T(x,y)}{\partial x}\Delta x$$
$$\therefore \frac{\partial T(x,y)}{\partial x} = \frac{T(x+\Delta x, y) - T(x,y)}{\Delta x}$$

Similarly expand in the negative direction of x to form the backward difference

$$\frac{\partial T(x,y)}{\partial x} = \frac{T(x,y) - T(x-\Delta x, y)}{\Delta x}$$

Now consider a second order Taylor series expansion of T in both directions of x

$$T(x+\Delta x, y) = T(x,y) + \frac{\partial T(x,y)}{\partial x}\Delta x + \frac{\partial^2 T(x,y)}{\partial x^2}\frac{\Delta x^2}{2} \quad \text{.....(1)}$$

and

$$T(x-\Delta x, y) = T(x,y) + \frac{\partial T(x,y)}{\partial x}(-\Delta x) + \frac{\partial^2 T(x,y)}{\partial x^2}\frac{(-\Delta x)^2}{2} \quad \text{.....(2)}$$

Now, subtracting these two equation gives the first order difference equation

$$\boxed{\frac{\partial T(x,y)}{\partial x} = \frac{T(x+\Delta x, y) - T(x-\Delta x, y)}{2\Delta x}}$$

or more generally, the one dimensional first order difference equation

$$f'(x_0) \approx \frac{f(x_0+\Delta x) - f(x_0-\Delta x)}{2\Delta x}$$

Addition of equations (1) and (2) yields the second order difference equation. The one dimensional second order difference equation is

$$\boxed{f''(x_0) \approx \frac{f(x_0+\Delta x) - 2f(x_0)f + (x_0-\Delta x)}{(\Delta x)^2}}$$

This can now be extended into two dimensions as shown

$$f''(x_0) \approx \frac{[f(x+\Delta x, y) - 2f(x,y)f + (x-\Delta x, y)] + [f(x, y+\Delta y) - 2f(x,y)f + (x, y-\Delta y)]}{(\Delta x)^2}$$

If we now adapt this to be able to be applied to an array, then $f(x+\Delta x, y)$ becomes $f(x+1, y)$ with $(\Delta x = \Delta y = h)$ defined. Since this is an approximation, the process should be repeated until convergence is reached. This now becomes an iterative step with the appropriate equations set up. This approximation will be labelled the residual error and evaluated as

$$R(i,j) = \phi(i+1,j) + \phi(i-1,j) + \phi(i,j+1) + \phi(i,j-1) - 4\phi(i,j) - h^2 g(i,j)$$

Where g(i,j) is the charge distribution in a dielectric and defined by

$$g(x,y) = \frac{-ax(y-1)}{\varepsilon_0}$$

with a=1nCm^{-5}, ε_0 is the permitivity of free space (8.8542x10-12 Fm^{-1}), x and y are the distances in the respective directions. This residual error can now be used to update the charge distribution matrix with a better approximation using

$$\phi_{k+1}(i,j) = \phi_k(i,j) + \frac{\omega}{4} R_k(i,j)$$

where ω is the relaxation factor. For ω=1 successive relaxation would be used. For successive over relaxation, the relaxation factor must be $1<\omega<2$. This relaxation factor depicts how quickly the system will converge. Optimal values of ω need to be studied but the optimal value is believed to be the lower root of the equation

$$t^2\omega^2 - 16\omega + 16 = 0 \quad \text{.....(3)}$$

$$\text{where } t = \cos\left(\frac{\pi}{N_x}\right) + \cos\left(\frac{\pi}{N_y}\right)$$

N_x and N_y are the number of intervals along the x and y axes respectively.
For this method to work, suitable initial interior conditions need to be set.

3. MATLAB

It was decided for this program to be written in MATLAB. MATLAB provides powerful manipulation and ease of use for working with mathematical problems. It was originally intended for rapid matrix manipulation, for example finding the inverse, or eigenvalues. MATLAB has been enhanced to include many different fields of mathematics and since has become a very powerful tool for engineers

worldwide. MATLAB also allows relatively simple implementation of a graphical user interface, or GUI, which results in programs written being easier to use.
The syntax of MATLAB is similar to that of the ANSI c programming language but does not require specific libraries to be included nor such a rigorous use of end-of-command semi-colons. The language will not be discussed in detail here but many texts are available for understanding and using MATLAB.

4. The Program

As this is a fairly trivial program it is not necessary to follow a complete software engineering process, however it was useful to construct a basic plan. Figure 3 shows a basic block diagram of the programs execution.

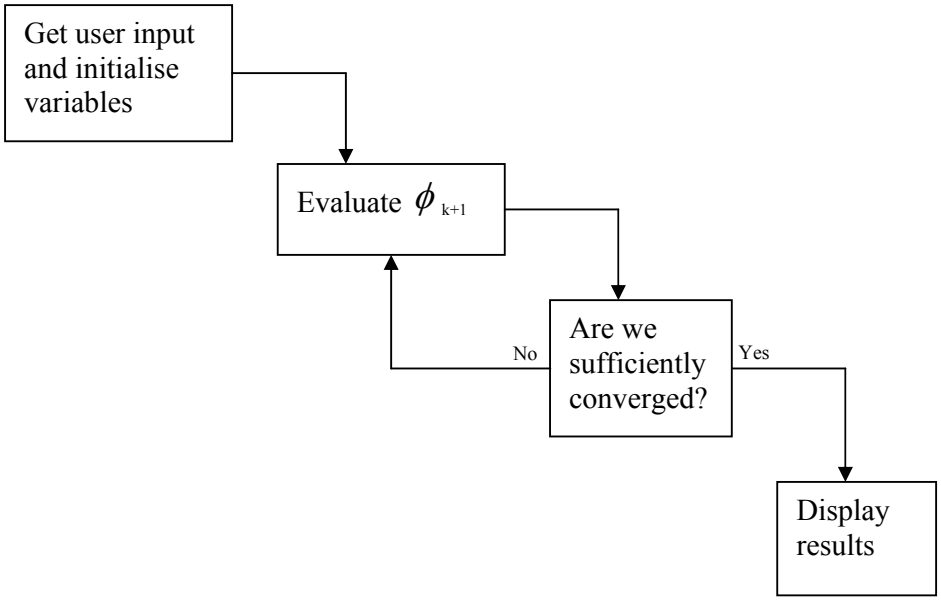


Figure 3: Program Block Diagram

From this block diagram, some pseudo-code was easily obtained:

```
get user input
initialise variables
while not converged
        evaluate next iteration
        if error < threshold then converged
do
output results
```

User input was required to allow the program to find a solution for a specified number of points within the dielectric. From this, it was possible to initialise variables, N_x, N_y and h. a and ε_0 are constants so could also be initialised.
A double loop was set up to cover the range of i and j across the whole array as shown

```
for i=1:intervals
                for j=1:intervals
                        {
                        …
                        }
                end
        end
```

Iterations were carried out using the equations derived above. To verify convergence, a variable was setup to obtain the sum of all values of the Residual Error array, R(i,j). If the average error was detected to be below 10^{-5} then the system was thought to have sufficiently converged. Once converged, the charge matrix, $\phi_{(i,j)}$ and a graphical output in the form of a 3D plot was displayed. As mentioned earlier, it is necessary to choose suitable initial interior conditions, these were chosen as 5. The insulating gaps needed an initial value as well, these were chosen as the midway value between each charged side.
The whole program is shown below.

```
fprintf('\n\n\n\n\n\n\n\n\n\n\n\n\n\n');
points=input('Please enter number of points in the 1m gap '); % user input - amount of points
iterations=input('Please enter maximum number of iterations '); % user input - max iterations
Nx=points+1; Ny=Nx; h=1/Nx; %Set h and Nx
thi=zeros(points+2,points+2);R=thi; % Setup matrices with all zeros
thi(1,1)=5; thi(1,2:Nx)=20; thi(1,Nx+1)=15; thi(2:Ny,1)=-10; thi(Ny+1,1)=-5; thi(2:Ny,Nx+1)=10;
thi(Ny+1,Nx+1)=5; thi(Ny+1,2:Nx)=0; thi(2:Ny,2:Nx)=5;% Setup initial values of thi
thinew=thi; % Have thinew the same as thi initialised
a=10^-9; E=8.8542*10^-12; % Set constants
t=cos(pi/Nx) + cos(pi/Ny); % Evaluate t for matrix
temp=[t^2 -16 16]; ws=roots(temp); %Find the roots of equation to find values for w
fprintf('\nStats:\n')
fprintf('distance between points %f m\n',h)

if ws(1)>ws(2)
   w=ws(2);
else
   w=ws(1);
end

tic; % Measure time taken
for loop=1:iterations
   error=0; %Set error to 0 at beginning of each iteration
   for i=2:Nx
      for j=2:Ny
         x=(i-1)*h; %Find distance in x
         y=1-(j-1)*(h); %Find distance in y
         g(i,j)=-a*x*(y-1)/E; % Evaluate charge equation
         R(i,j) = thi(i+1,j) + thi(i-1,j) + thi(i,j+1) + thi(i,j-1) - 4*thi(i,j) - (h^2)*g(i,j); % Find Residual
         thinew(i,j)=thi(i,j) + (w/4)*R(i,j); % Ierative step to update thi with respect to residual
         error=error+abs( thinew(i,j) - thi(i,j) ); %Evaluate an error for stopping if converged
         thi=thinew;
      end
   end

   if error<(10^-5)/points^2 % If small enough error then converged
      break % Therefore break the iterations
   end

end
```

```
%Display results
timetaken=toc;
fprintf('After %d iterations and %f seconds, the final result is:',loop,timetaken)
thi(2:Nx,2:Ny)

%Plot data in 3D graph
x=linspace(0,1,Nx+1);
y=x;
[X,Y]=meshgrid(x,y);
surf(X,Y,thi);
xlabel('x (metres)')
ylabel('y (metres)')
zlabel('charge (coulombs)')
title('Charge distribution')
```

This program also asks the user for a maximum number of iterations as an upper limit. This will safeguard against an infinite loop if the algorithm does not converge. The program also measures the time taken to calculate the charge distribution and display it.

5. Problems

Initially, the program did not converge. This was found to be due to several bugs. The calculation of g(i,j) was wrong as 'a' was used instead of '-a', ε_0 was also declared with the wrong value. With these corrections, the program then converged. In the original problem, a sample set of actual charge values was supplied for a 3x3 grid system. The programs output was compared to this but found to have a relatively large error. One reason for this error was the way the program was originally executed. The program was evaluating the charges of areas rather than points which were affecting the values of N_x, N_y and h. As a result ω was not optimal using the supplied equation and extra iterations were required for convergence. With this problem fixed, the program resulted in a lower error and ω was now believed to be optimal. The error could still be improved so further examination of the program was required. It was found that the reason for further error was in the declaration of y. The Φ matrix was setup from +20v on the first row down to 0v on the bottom row, and y was first defined to follow this direction. This led for g(i,j) to be miscalculated. To fix the problem, it was required for y to be replaced with (1-y). The reason for this was due to the 1m gap in the dielectric.

6. Observations and Results

Values of ω were tested to find which gave the optimal result or least iterations to achieve convergence. Figure 4 shows a graph of ω plotted against number of

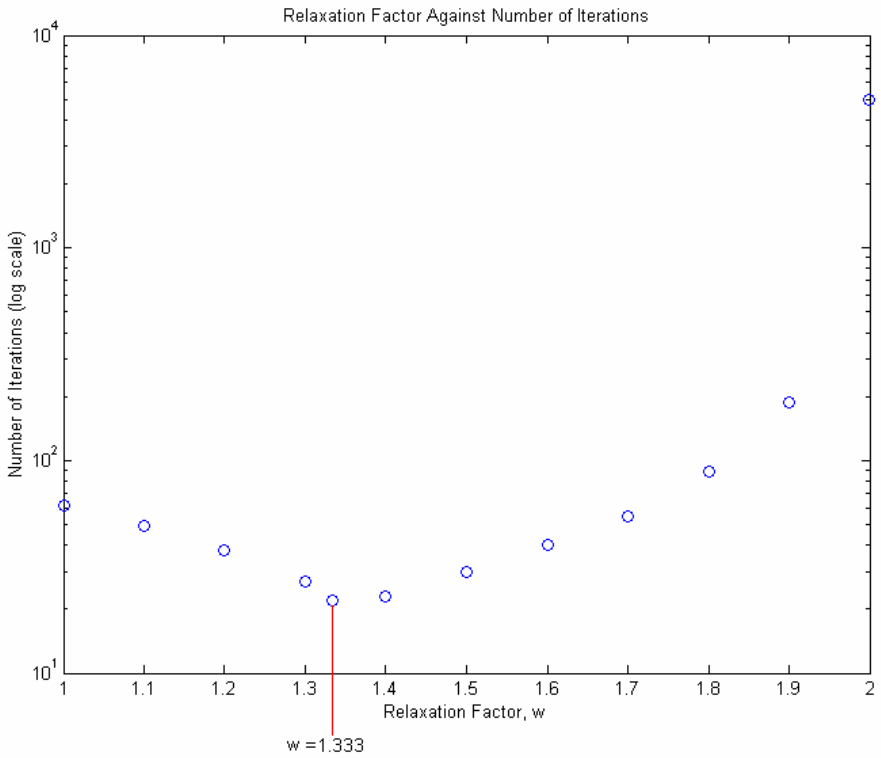


Figure 4: Determining the Optimal Value of ω.

iterations for a 5x5 grid of charges within the dielectric. For ω=2, There was almost no convergence therefore ω was taken as 1.999 for this result. Even at this value, there were still a lot of iterations so to display this, a logarithmic scale was used to display the number of iterations. It can be seen that the optimal value is around 1.333, which is the value obtained by using equation (3).

The value of h was varied but this just gave erroneous results. This is due to the fact that h is the distance between points and requires to be set to a unique value for each grid.

Figures 5 & 6 show the charge distribution of a 33x33 grid (h=0.029412m). The time taken for this to reach a solution was approximately 13.5 seconds.

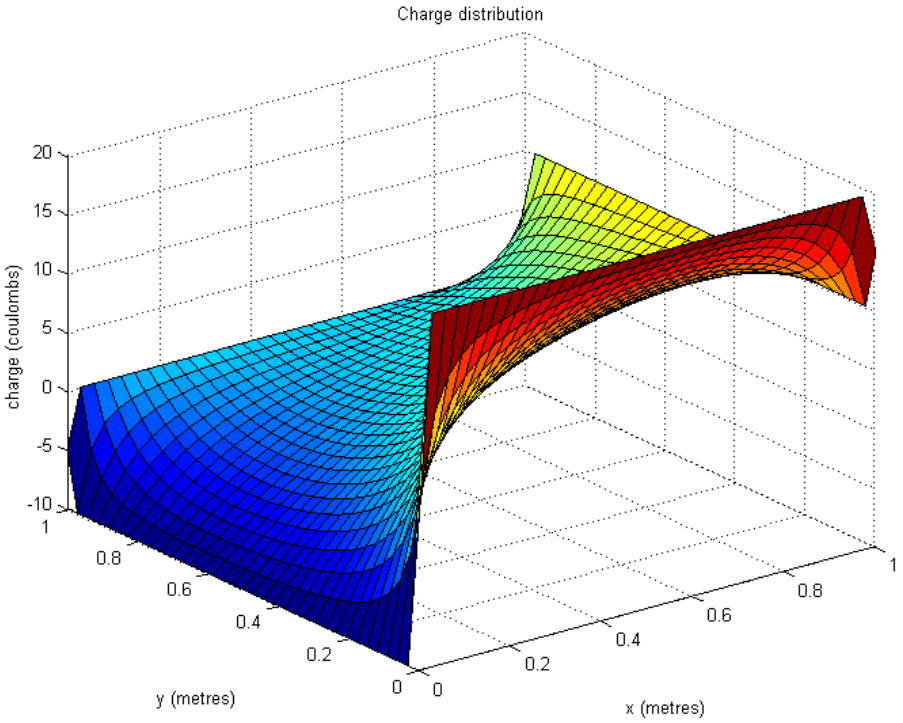


Figure 5: Charge Distribution of 33x33 grid (isometric view)

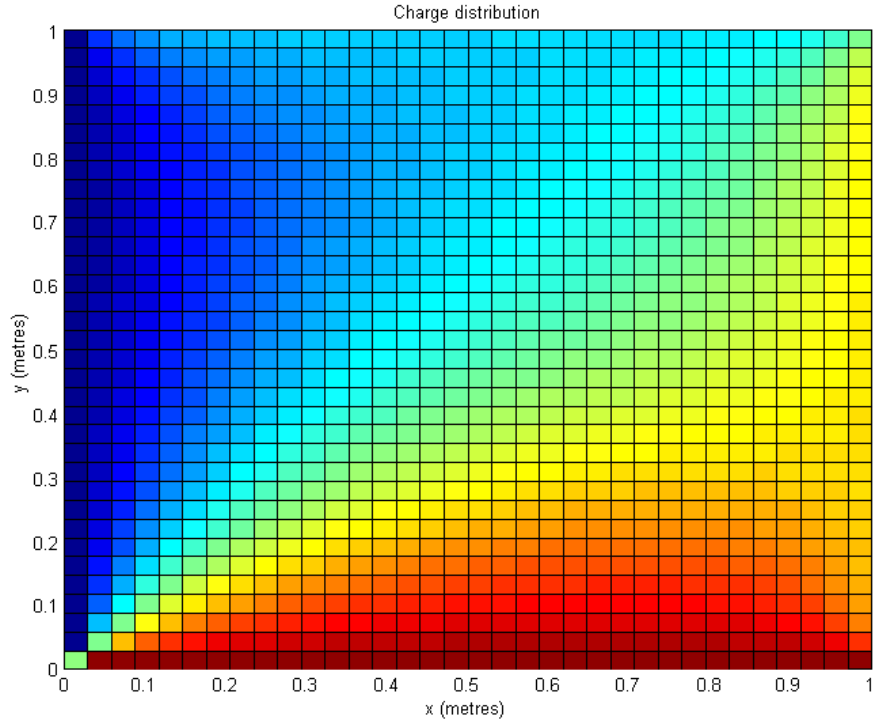


Figure 6: Charge Distribution of 33x33 grid (plan view)

7. Conclusion

The example values for the 3x3 grid of charges given were derived from Fourier analysis and believed to be exact to four significant figures. The program used approximate methods in iterative steps to obtain a solution. It was found that the error of these approximations was within 0.25% of the range of potential. This is accurate enough for most calculations. Accuracy may be improved by extending the finite difference method from a third order Taylor series expansion but this would require further computational power and further algebraic manipulation.

The relaxation factor, ω, was varied and tested to evaluate the optimal value compared with the value calculated from the given equation (3). This was found to be very accurate. It was not possible to obtain an error for this as the test was to evaluate how many iterations the program took to converge. For small grids, the program took less than 100 iterations to complete. The number of iterations can only vary by a whole number therefore the results would have an error greater than 1%. To test the optimal value of ω would require a more accurate measurement than this. Unfortunately, systems requiring over 100 iterations took a very long time to complete. Because of this, it was not possible to check how accurate the optimal value of ω was but this could be the basis of further study with extra time allowed.