PNS - Assignment 3

Patrick Sinnott 17326757

December 2020

1 Question 1

I have attached a C++ code that successfully uses the SOR method to approximate the laplace equation with the given Dirichlet Boundary Conditions. The given task was to evaluate the formula: $\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0$ over the Dirichlet boundary conditions shown in the following diagram:

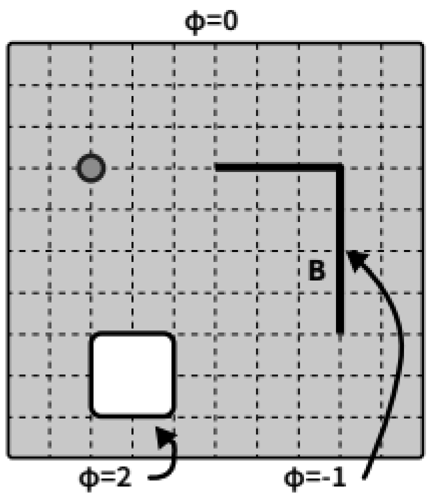
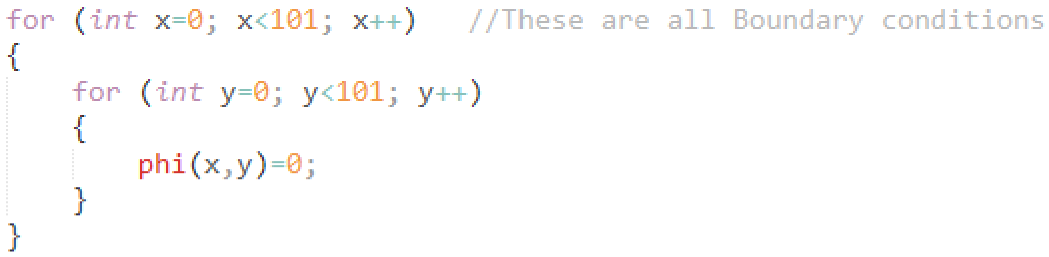


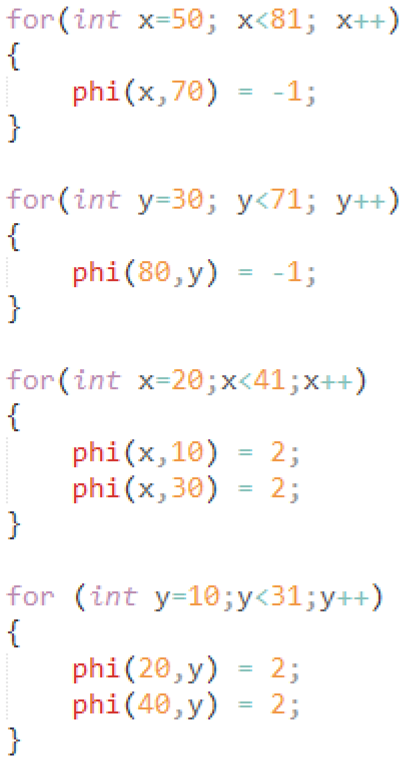
Figure 1: Region Ω, with boundary data

The first step was to establish the field class which would be used to describe the 2-D array which will describe our area of interest. After the field class has been defined we can define an object Field and declare its size. I decided to define a 100x100 square as this would give our answer sufficient accuracy.

When we have our object defined it is then time to establish the given boundary conditions. This can be done with a few simple for-loops which assign the boundary values to the necessary points.

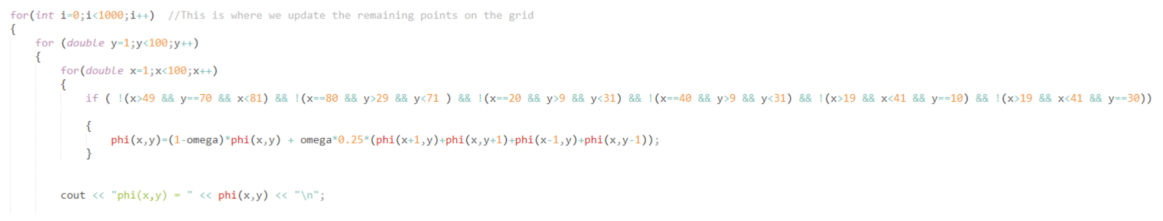


```
for (int x=0; x<101; x++)   //These are all Boundary conditions
{
    for (int y=0; y<101; y++)
    {
        phi(x,y)=0;
    }
}
```



```
for(int x=50; x<81; x++)
{
    phi(x,70) = -1;
}

for(int y=30; y<71; y++)
{
    phi(80,y) = -1;
}

for(int x=20;x<41;x++)
{
    phi(x,10) = 2;
    phi(x,30) = 2;
}

for (int y=10;y<31;y++)
{
    phi(20,y) = 2;
    phi(40,y) = 2;
}
```

After we have assigned our boundary conditions it is simply a matter of creating a process which will update the remainder of the points with the given formula whilst leaving the boundary points unaltered. I achieved this using a somewhat sloppy if statement that updates the each of the points as an average of its 4 nearest neighbours provided the point in question is not one of the boundary points. I decided to set n=1,000 (Run for 1,000 steps) to provide sufficient accuracy to my solution.



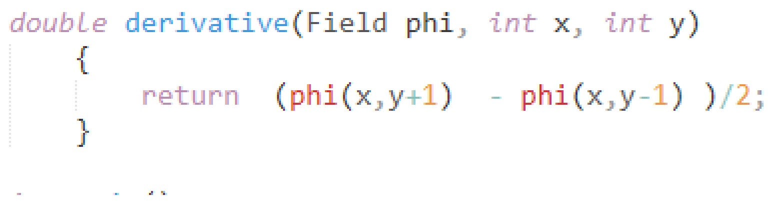
```
for(int i=0;i<1000;i++)  //This is where we update the remaining points on the grid
{
    for (double y=1;y<100;y++)
    {
        for(double x=1;x<100;x++)
        {
            if ( !(x>49 && y==70 && x<81) && !(x==80 && y>29 && y<71 ) && !(x==20 && y>9 && y<31) && !(x==40 && y>9 && y<31) && !(x>19 && x<41 && y==10) && !(x>19 && x<41 && y==30))

            {
                phi(x,y)=(1-omega)*phi(x,y) + omega*0.25*(phi(x+1,y)+phi(x,y+1)+phi(x-1,y)+phi(x,y-1));
            }


        cout << "phi(x,y) = " << phi(x,y) << "\n";
```

I am aware that this method may not be the most legible, however it works to a good degree of accuracy and runs rather efficiently.

I first used the The Gauss-Seidel iteration to calculate my grid points which is the same as the SOR method but it has omega = 1. I then experimented with a number of different values of omega and eventually found that omega = 1.75 gave a satisfactory solution.

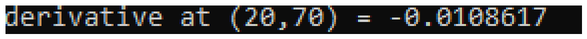
When I had the array filled with the correct values of $\phi(x, y)$ the next step was to calculate the value of $\frac{\partial\phi}{\partial y}\left(\frac{2}{10}, \frac{7}{10}\right)$

To do this I needed to define a function to calculate the y-derivative of $\phi(x, y)$ at a given point and this was achieved rather easily using the following function:



```
double derivative(Field phi, int x, int y)
    {
        return  (phi(x,y+1)  - phi(x,y-1) )/2;
    }
```

Using this I calculated the following value for the y-derivative at the given point:

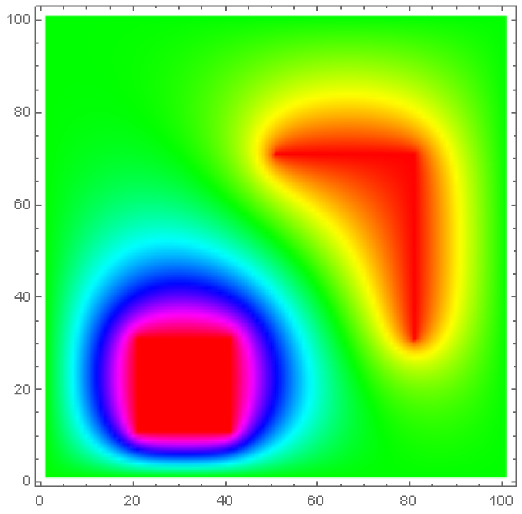


```
derivative at (20,70) = -0.0108617
```

As a further check to validate my solution I decided I should graph my resulting array in the form of a heat map. I wrote a piece of code that exported my array of values to a text file which I was able to plot using Mathematica resulting in the following graph:

```
In[•]:= ListDensityPlot[Import["C:\\Users\\Pa\\Desktop\\lab1\Heat.txt", "Data"], PlotRange → All, ColorFunction → Hue]
```

Out[•]=



This Heat-map matches the given diagram and validates my solution.