CO25: Laplace's Equation

A. Kulanthaivelu
University College

1 Abstract

The aim of this experiment was to solve Laplace's equation numerically over a square domain for Dirichlet boundary conditions. The solution was arrived at using the successive over-relaxation method with the results plotted and compared against the analytic solution. It was found that the numerical and analytic solution matched each other very closely. The over-relaxation parameter was then varied and the effect on the rate of convergence and on the form of convergence was observed. It was verified that the most suitable choice for the parameter was around 1.35 for our case, very close to the analytically determined optimum value of $\frac{4}{3}$ and that when this parameter is above 2 the iteration does not converge.

2 Introduction

In this experiment we aim to solve Laplace's equation

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = 0 \tag{1}$$

Laplace's equation is an elliptic partial differential equation having second order derivatives of all variables with the same sign on same side of the equation.

This equation can be solved analytically using the separation of variables method, let

$$\psi(x, y) = X(x)Y(y) \tag{2}$$

Substituting this into (1) and dividing by XY we get

$$\frac{1}{X}\frac{d^2X}{dx^2} + \frac{1}{Y}\frac{d^2Y}{dy^2} = 0 \tag{3}$$

Each of these terms must be constant for the equality to hold for all of *X, Y* since each term is a function of different variables so we get two separate equations

$$\frac{d^2X}{dx^2} + k^2X = 0 \text{ and } \frac{d^2Y}{dy^2} - k^2Y = 0 \tag{4}$$

where k is an arbitrary constant.
Thus

$$X = A\cos kx + B\sin kx \text{ and } Y = C\cosh ky + D\sinh ky \tag{5}$$

where A, B, C and D are arbitrary constants.

For the boundary conditions $\psi(0, y) = \psi(x, 0) = 0$ for $0 \leq x \leq 1$, $0 \leq y \leq 1$, and $\psi(x, 1) = \sin(x)\sinh(1)$ and $\psi(1, y) = \sin(1)\sinh(y)$ we have $A = 0$, $B = 1$, $C = 0$, $D = 1$ and $k = 1$, so

$$\psi(x, y) = \sin(x)\sinh(y) \tag{6}$$

This is a particularly simple solution corresponding to the given boundary conditions. In general solutions to Laplace's equation will be more complicated and harder to solve analytically and thus it is better to try to solve the equation numerically. In this experiment we aim to do this and solve Laplace's equation numerically for the specified boundary conditions using the successive over-relaxation method and we compare the solution with the analytic solution to see how well our computer model works.

3 Solution

To compute the solution to Laplace's equation we use a procedure known as successive over-relaxation which is outlined below.

An approximation to Laplace's equation is

$$\frac{(\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j})}{(\delta x)^2} + \frac{(\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1})}{(\delta y)^2} = 0 \tag{7}$$

Where each term of the form $\psi_{i,j}$ corresponds to a grid point and $\delta x, \delta y$ are the distances between the grid points in the x and y directions.

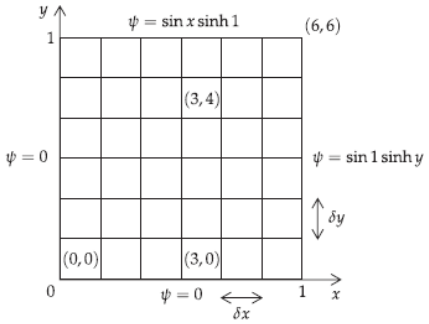

Figure 1: Plot of the grid layout for a 7 x 7 grid, with δx and δy the distance between neighbouring points in the x and y directions

Applying this approximation for $\nabla^2\psi$ for all the interior points in the domain we get a system of linear equations for the $\psi_{i,j}$

$$\psi_{i,j} = \frac{\frac{(\psi_{i+1,j} + \psi_{i-1,j})}{(\delta x)^2} + \frac{(\psi_{i,j+1} + \psi_{i,j-1})}{(\delta y)^2}}{2(\frac{1}{(\delta x)^2} + \frac{1}{(\delta y)^2})} \tag{8}$$

Systems of second order linear equations can be solved using direct or iterative numerical methods. The use of finite differencing, iterative methods and the origin of several of the equations used here are discussed in appendix B and C. In this case we will be using the iterative successive over-relaxation method solve the equations.

In the process of successive over-relaxation we use the equation

$$\psi_{i,j}^{m+1} = \alpha\bar{\psi}_{i,j} + (1 - \alpha)\psi_{i,j}^{m} \tag{9}$$

Where $\bar{\psi}_{i,j}$ is calculated from (8). The process is iterative and m is the iteration number.

For a uniform grid where $\delta x = \delta y = \delta$ this reduces the equation to

$$\begin{aligned} \psi_{i,j}^{m+1} &= \psi_{i,j}^{m} + \frac{\alpha}{4} R_{i,j} \\ R_{i,j}^{m} &= \psi_{i,j+1} + \psi_{i,j-1} + \psi_{i-1,j} + \psi_{i+1,j} - 4\psi_{i,j} \end{aligned} \tag{10}$$

We use the boundary conditions $\psi(0,y) = \psi(x,0) = 0$ for $0 \le x \le 1$, $0 \le y \le 1$, $\psi(x,1) = \sin(x)\sinh(1)$ and $\psi(1,y) = \sin(1)\sinh(y)$ on the other boundaries correspond to the simple analytic solution described previously.

To implement this procedure we sweep over the domain as shown in the following figure in a systematic manner.

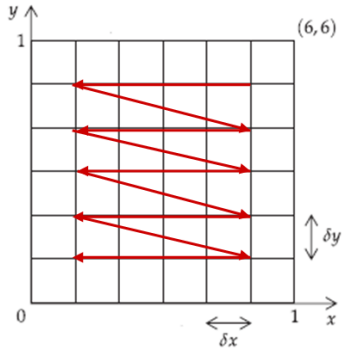

Figure 2: Diagram showing how we sweep across the grid domain

$R_{i,j}$ is the residual which we'd like to be zero which, of course, won't happen in general. However $R_{i,j}$ will get smaller for any (i,j) for a convergent iteration algorithm as the process continues. Therefore for a suitably convergent iteration process we can decide when to stop the algorithm, which can be for reaching a maximum number of iterations or when a predetermined condition is satisfied.

The number α is the over-relaxation parameter, since it has a value greater than 1, and the optimum value for this parameter, α_{opt}, is given by

$$\alpha_{opt} = \frac{2}{1 + \sqrt{1 - \left\{ \frac{\cos\left(\frac{\pi}{p}\right) + \cos\left(\frac{\pi}{q}\right)}{2} \right\}^2}} \tag{11}$$

Which in our case where $p = q$ reduces to

$$\alpha_{opt} = \frac{2}{1 + \sin(\frac{\pi}{p})} \tag{12}$$

In our process we use the new values of $\psi_{i,j}$ to calculate the residual once they have been calculated themselves to make the programming easier.
We iterate from the boundaries where ψ is non-zero down to the boundaries where it is zero and at each iteration three values of ψ, one in the upper half, one in the middle and one in the lower half of the domain, were printed on a table to show convergence.

We initially take the domain to be a 7 x 7 grid and initially take $\alpha = 1.35$. The solution generated is then checked for accuracy by comparing it with the analytic solution. We then run the program for $\alpha = 1.10, 1.25, 1.45$ and 2.10 and check the rate of convergence.
The maximum iteration number was taken to be 30 with the program stopping if convergence was reached sooner.

4 Results

The numerical results were taken and a contour and surface plot were created. The analytic solution was also plotted to allow for comparison between the analytic and numerical solution and the two solutions have been placed side by side for this reason. A further surface plot has been included with the numerical solution plotted together with the analytic solution to show how closely they match.
It can be seen that the numerical contour plot has kinks in the contour lines, this is due to the fact that we only applied the over-relaxation method to discrete points in a grid surface so only points at the intersection of grid lines have been calculated. These kinks can be minimised by using a finer grid.
Following this, tables of the values of $\psi_{3,1}$, $\psi_{3,3}$ and $\psi_{3,5}$, i.e values from the bottom, middle and top of the domain, respectively have been displayed for different over-relaxation parameters. The data has been plotted on graphs of the ψ values against the iteration number to show whether the convergence was monotonic or oscillatory. Two separate graphs, one with the oscillatory convergence over-relaxation parameter and one without this parameter have been included since the oscillations with the parameter are so large in amplitude they mask the convergence lines of the other parameters.

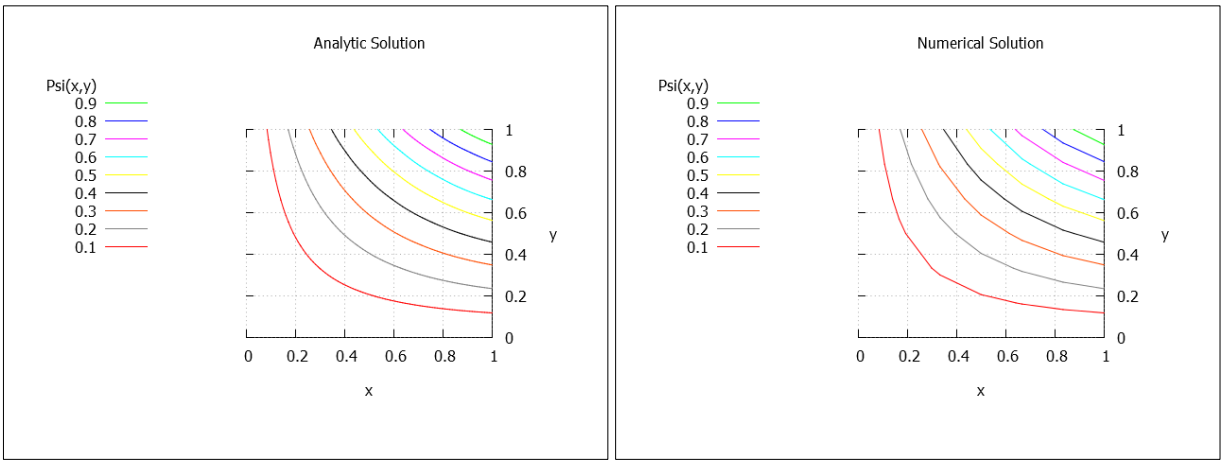

Figure 3: Contour plots of the analytic and numerical solution

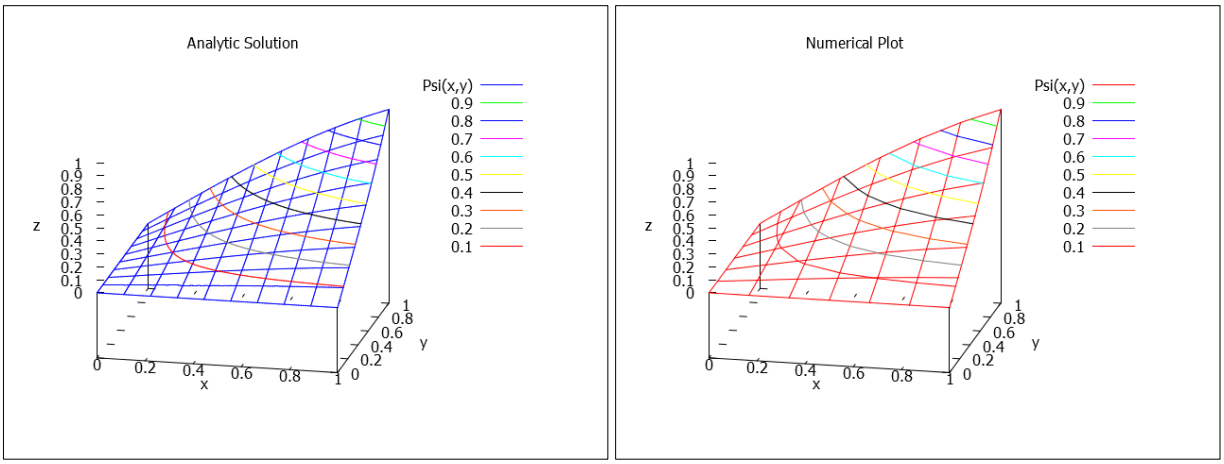

Figure 4: Surface plot of the analytic and numerical solution

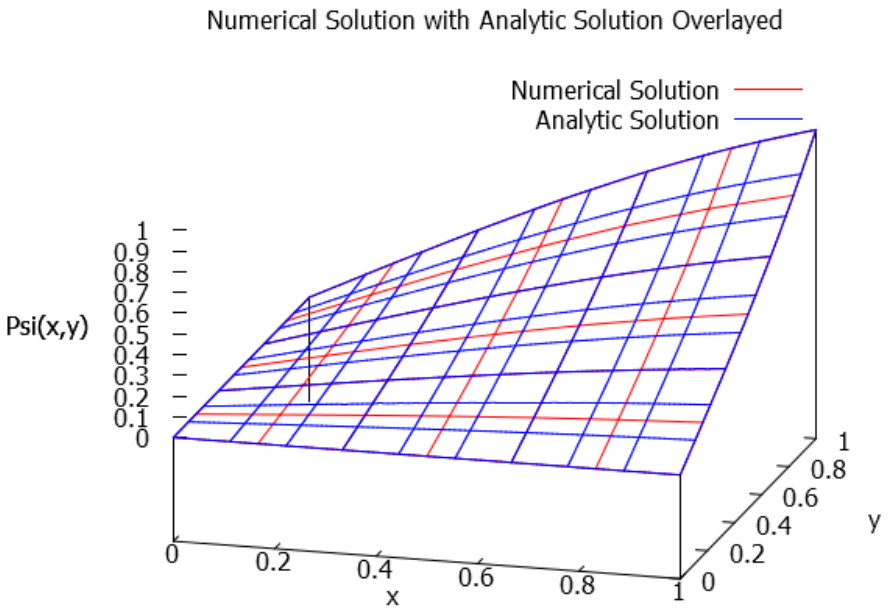

Figure 5: Analytic solution surface plot with the numerical solution surface plot together

	$\psi_{3,1}$ *values for different values of* α				
Iteration Number (m)	$\alpha = 1.35$	$\alpha = 1.10$	$\alpha = 1.25$	$\alpha = 1.45$	$\alpha = 2.10$
1	0.352019	0.314472	0.336207	0.368947	0.509688
2	0.402670	0.364251	0.387136	0.418535	0.544769
3	0.425805	0.392645	0.412924	0.438213	0.517282
4	0.437664	0.410443	0.427529	0.446782	0.482725
5	0.443508	0.422122	0.436011	0.449444	0.413236
6	0.446255	0.430002	0.441007	0.449598	0.342248
7	0.447234	0.435396	0.443871	0.448485	0.275574
8	0.447380	0.439111	0.445451	0.447323	0.359939
9	0.447447	0.441678	0.446336	0.447360	1.054654
10	0.447459	0.443455	0.446831	0.447386	0.444048
11	0.447465	0.444685	0.447108	0.447451	0.260993
12	0.447464	0.445538	0.447264	0.447452	-0.078393
13	0.447463	0.446128	0.447351	0.447463	0.647661
14	0.447462	0.446538	0.447400	0.447465	0.566599
15	0.447462	0.446822	0.447427	0.447464	0.623218
16	-	0.447018	0.447443	0.447463	0.755547
17	-	0.447154	0.447451	0.447462	0.444760
18	-	0.447249	0.447456	0.447462	-0.065631
19	-	0.447314	0.447459	-	-0.426198
20	-	0.447360	0.447460	-	0.801807
21	-	0.447391	0.447461	-	2.249705
22	-	0.447413	0.447462	-	-0.167974
23	-	0.447428	0.447462	-	0.347933
24	-	0.447439	0.447462	-	-1.028298
25	-	0.447446	-	-	1.072978
26	-	0.447451	-	-	0.119061
27	-	0.447454	-	-	1.216003
28	-	0.447457	-	-	2.161240
29	-	0.447458	-	-	0.333073
30	-	0.447460	-	-	-2.331911

Figure 6: Table showing $\psi_{3,1}$ values for different values of α

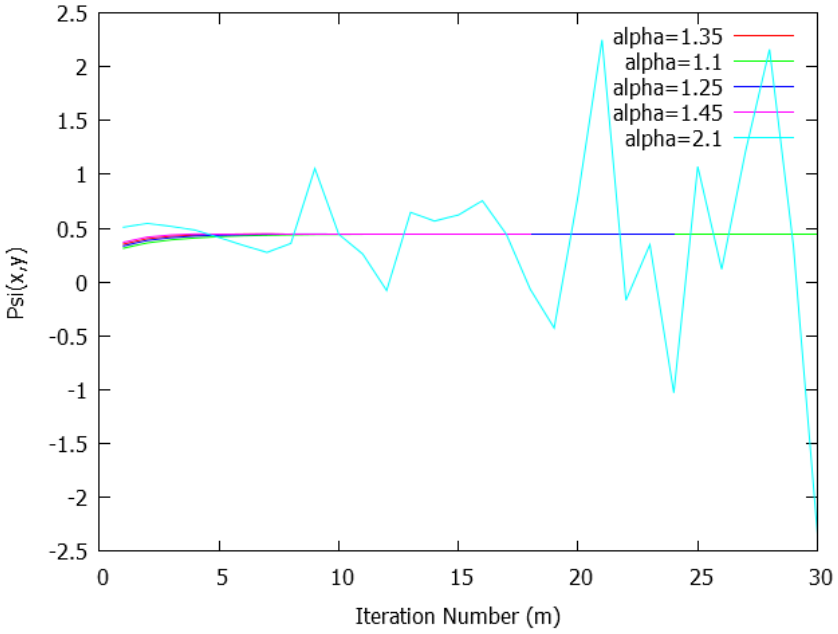

Figure 7: Plot showing the convergence of $\psi_{3,1}$ for all different values of α

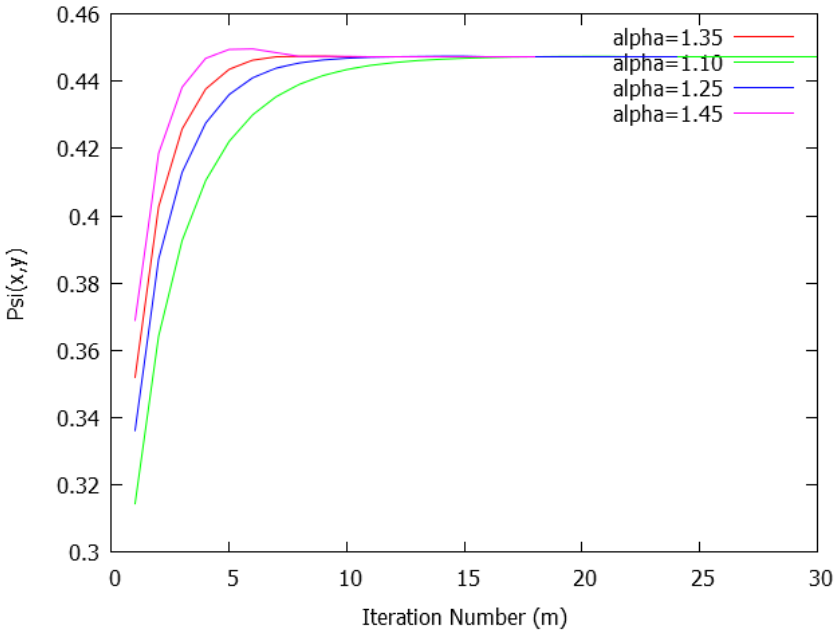

Figure 8: Plot showing the convergence of $\psi_{3,1}$ for all different values of α excluding $\alpha = 2.10$

	$\psi_{3,3}$ *values for different values of* α				
Iteration Number (m)	$\alpha = 1.35$	$\alpha = 1.10$	$\alpha = 1.25$	$\alpha = 1.45$	$\alpha = 2.10$
1	0.147077	0.100233	0.125561	0.173025	0.505413
2	0.202275	0.145697	0.178124	0.228360	0.461089
3	0.230313	0.177631	0.209130	0.251295	0.391197
4	0.242906	0.199801	0.226990	0.256045	0.123349
5	0.247783	0.215182	0.237120	0.253889	-0.150230
6	0.249209	0.225841	0.242727	0.250863	-0.033367
7	0.249757	0.233229	0.245886	0.250103	0.682462
8	0.249884	0.238348	0.247656	0.249714	0.523637
9	0.249911	0.241897	0.248647	0.249759	0.663138
10	0.249914	0.244356	0.249202	0.249853	-0.048112
11	0.249913	0.246061	0.249513	0.249900	-0.524197
12	0.249911	0.247242	0.249688	0.249912	-0.073616
13	0.249910	0.248061	0.249785	0.249916	0.812679
14	0.249910	0.248628	0.249840	0.249912	0.757577

15	0.249910	0.249022	0.249871	0.249911	1.197147
16	-	0.249294	0.249888	0.249910	-0.551323
17	-	0.249483	0.249898	0.249910	-0.951715
18	-	0.249614	0.249903	0.249910	-0.225042
19	-	0.249705	0.249906	-	0.840941
20	-	0.249768	0.249908	-	1.514333
21	-	0.249811	0.249909	-	1.915990
22	-	0.249842	0.249909	-	-1.507910
23	-	0.249863	0.249909	-	-1.301141
24	-	0.249877	0.249910	-	-0.971275
25	-	0.249887	-	-	1.004673
26	-	0.249894	-	-	3.218828
27	-	0.249899	-	-	2.546947
28	-	0.249902	-	-	-2.695828
29	-	0.249905	-	-	-1.760342
30	-	0.249906	-	-	-3.117623

Figure 9: Table showing $\psi_{3,3}$ values for different values of α

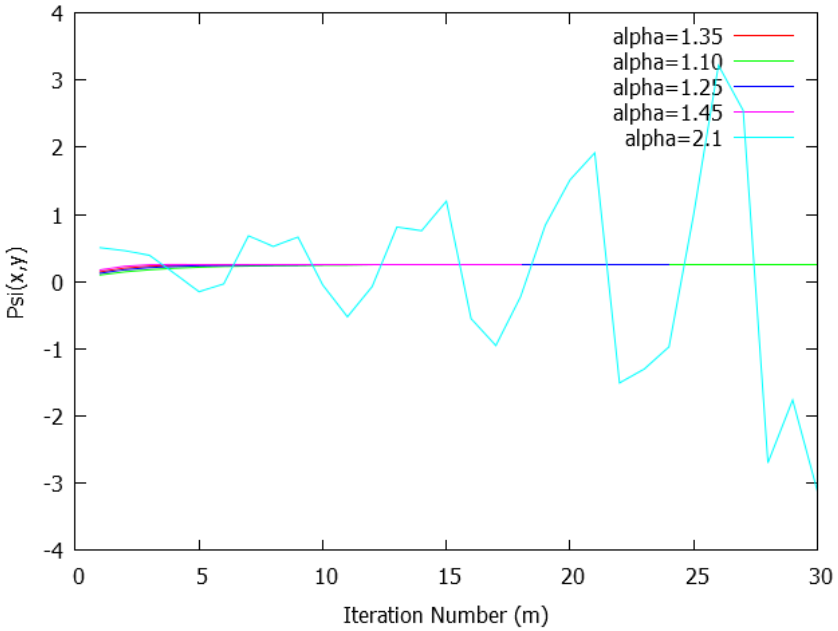

Figure 10: Plot showing the convergence of $\psi_{3,3}$ for all different values of α

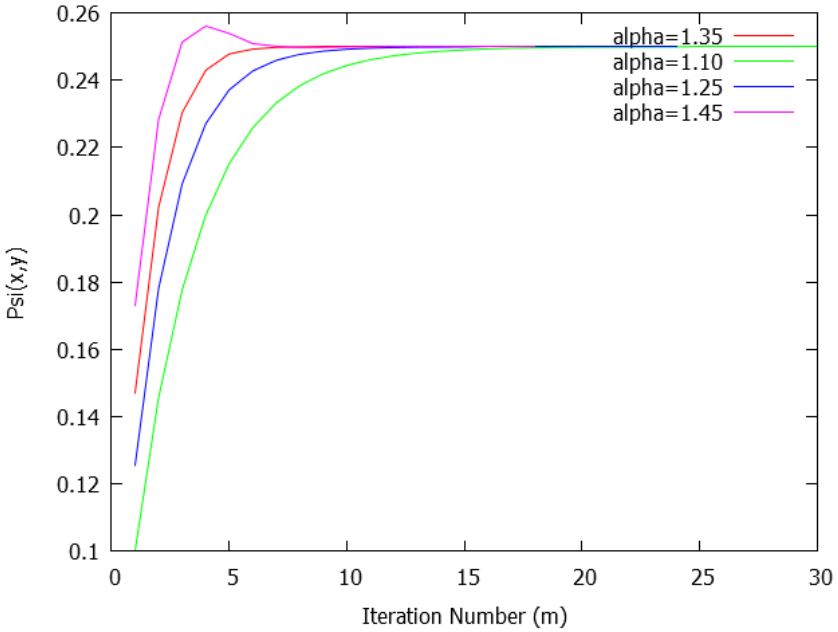

Figure 7: Plot showing the convergence of $\psi_{3,3}$ for all different values of α excluding $\alpha = 2.10$

	$\psi_{3,5}$ *values for different values of* α				
Iteration Number (m)	$\alpha = 1.35$	$\alpha = 1.10$	$\alpha = 1.25$	$\alpha = 1.45$	$\alpha = 2.10$
1	0.054750	0.314472	0.043025	0.070443	0.375843
2	0.070280	0.030930	0.059239	0.082213	0.144130
3	0.077733	0.045007	0.069002	0.085371	-0.026336
4	0.079090	0.055543	0.073923	0.081129	-0.269634
5	0.079938	0.063040	0.076723	0.080839	0.200076
6	0.080218	0.068306	0.078295	0.080422	0.197413
7	0.080298	0.071978	0.079180	0.080256	0.187813
8	0.080311	0.074530	0.079676	0.080231	0.194958
9	0.080309	0.076302	0.079954	0.080264	0.052339
10	0.080308	0.077531	0.080109	0.080302	-0.140040
11	0.080308	0.078383	0.080196	0.080314	-0.337861
12	0.080308	0.078974	0.080245	0.080309	0.075727
13	0.080308	0.079383	0.080273	0.080309	1.196408
14	0.080308	0.079667	0.080288	0.080308	-0.141097
15	0.080307	0.079863	0.080297	0.080308	-0.126723
16	-	0.080000	0.080301	0.080307	-0.773739
17	-	0.080094	0.080304	0.080307	0.424620
18	-	0.080160	0.080306	0.080307	0.114508
19	-	0.080205	0.080306	-	0.411540
20	-	0.080236	0.080307	-	0.835799
21	-	0.080258	0.080307	-	0.101373
22	-	0.080273	0.080307	-	-1.153495
23	-	0.080284	0.080307	-	-1.438693
24	-	0.080291	0.080307	-	1.269467
25	-	0.080296	-	-	2.669040
26	-	0.080300	-	-	-1.021406
27	-	0.080302	-	-	0.218529
28	-	0.080304	-	-	-2.427730
29	-	0.080305	-	-	1.122001
30	-	0.080306	-	-	-1.240453

Figure 12: Table showing $\psi_{3,5}$ values for different values of α

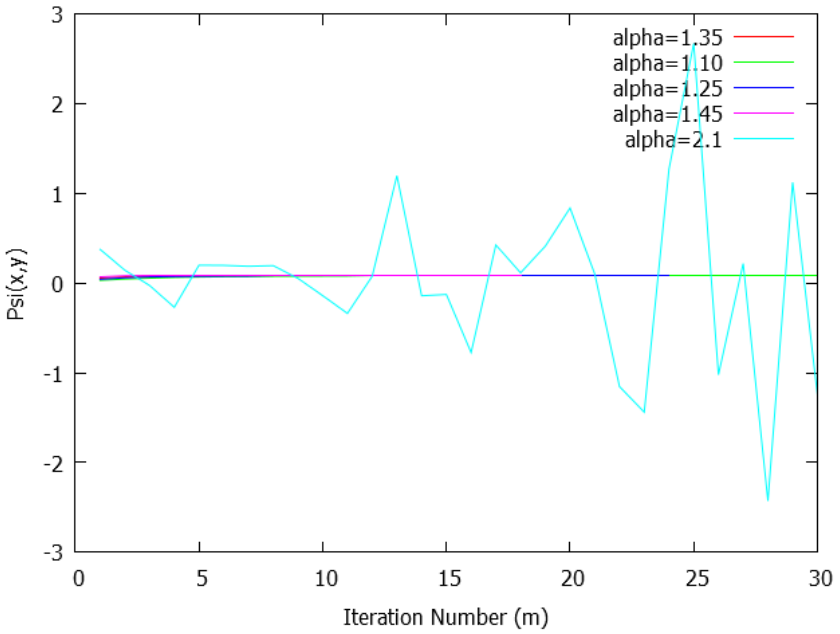

Figure 13: Plot showing the convergence of $\psi_{3,5}$ for all different values of α

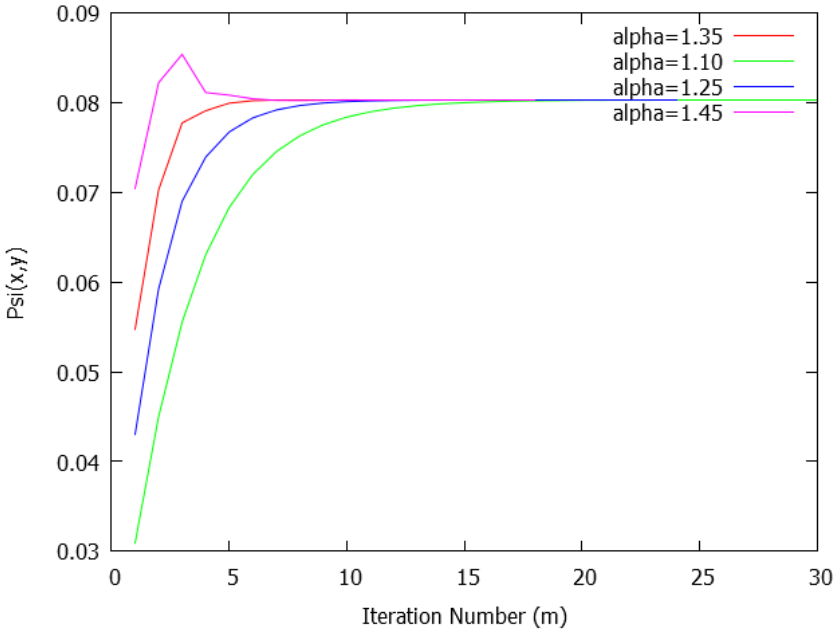

Figure 14: Plot showing the convergence of $\psi_{3,5}$ for all different values of α excluding $\alpha = 2.10$

5 Conclusion

Using the computer model we calculated the solution to Laplace's equation corresponding to the boundary values which generation the solution $\psi(x, y) = \sin(x)\sinh(y)$. We found, using a contour diagram and a surface plot, that the numerical solution corresponded closely with the analytic solution as the contour and surface plots of the two appeared to be very similar.
We found that varying the over relaxation parameter resulted in different levels of convergence and having a parameter greater than two resulted in vast oscillations of the individual $\psi_{i,j}$ values throughout the iteration process, not converging to any single value. Furthermore we recognise that out of the values 1.10, 1.25, 1.35 and 1.45, the value that results in the quickest convergence during iteration was 1.35, which corresponds with the analytically determined optimum value of 4/3.

The method with which we solved Laplace's equation has several advantages:

- The successive over-relaxation algorithm is an acceleration of the Gauss-Siedel method for a suitable over-relaxation parameter which in turn is faster than the Jacobi method so it is the fastest of the three iterative methods in terms of convergence. Furthermore it could be considered the most efficient of the three methods, using less computer storage than the other methods by using few iterations but operating on a similar if not shorter time span.
- The process has the considerable advantage of being able to be applied to solve equations over irregular domains and can be used to tackle almost any elliptical partial differential equation such as Poisson's equation, the biharmonic equation or the steady stokes equations.
- In principle the successive over-relaxation procedure can be used for solving hyperbolic partial differential equations such as the wave equation or parabolic partial differential equations such as the diffusion equation which have no mixed derivatives. In this case of the diffusion equation other methods are best used in conjunction to solve the problem such as the Crank-Nicholson method. Furthemore stability of the solution needs to be taken into account as not all finite difference methods lead to unconditionally stable solutions, but one's that diverge as the procedure continues, such as with the certain methods for solving the heat equation as the system evolves in time.
- The method is $\mathcal{O}(\delta^2)$ accurate so halving the spacing between points can increase your accuracy by a factor of 4. However, decreasing the spacing will increase the number of mesh

points and so increase the number of equations to solve which can significantly increase computation time.

On the other hand the algorithm comes with a few disadvantages:

- The rate of convergence is critically dependant on α. The value of α that is chosen depends on the problem and the optimum value is not easy to calculate analytically. In fact, it depends on the analytic solution to the problem you are solving numerically. This weakness in the S.O.R. method could overcome by mapping the problem to one in which the analytic solution is known. Otherwise α has to be found empirically.
- If $0 < \alpha < 1$, successive over relaxation method converges but the convergence rate is slower (deceleration) than the Gauss-Seidel method described in appendix C.
- Asymptotic rate of convergence in successive over relaxation is not attained until $\mathcal{O}(N)$ iterations for an N x N grid. The error usually grows by a factor of 20 before convergence sets in. However a trivial modification can solve this. Based on observation $\alpha = \alpha_{opt}$ is not necessarily a good initial choice. Chebyshev acceleration odd even ordering changes α at each half sweep. The norm of the error decreases with each iteration and we get an asymptotic rate of convergence identical to what we would have had if we didn't use Chebyshev acceleration.
- While the method converges under general conditions, it typically makes slower progress than competing methods.

Nevertheless, in terms of our problem, the successive over-relaxation method was an efficient and effective algorithm for solving Laplace's equation with Dirichlet boundary conditions, producing highly accurate results within the domain we chose which closely followed the true values and these results were produced in a short amount of time given the correct over-relaxation parameter. Even whilst the over-relaxation parameter is not at the optimum value it still produces quite accurate results in a relatively small interval of time given the over-relaxation parameter lies between 1 and 2.

References

[1] *CO25 Practical Script,* Oxford Physics, 2009

[2] A. O'Hare, *Numerical Methods for Physicists,* Oxford Physics, 2005

[3] G.D. Smith, *Numerical Solution of Partial Differential Equations: Finite Difference Methods*, 3rd edition, Oxford University Press, 1985

[4] R. S. Varga, *Matrix Iterative Analysis*, 2nd edition, Springer, 2000

Appendix A Source Code

The source code used to compute the solution to Laplace's equation using the previously described method of successive over-relaxation:

```
/*************************************************************************
*  CO25 Laplace's Equation
*  A program designed to find the numerical solution to Laplace's equation
*  using successive over-relaxation on prespecified boundary conditions.
*************************************************************************/

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

/*Function used to determine whether the residuals have converged*/
double r(double R[5][5])
{ int i, j;
  int B=0;

/*If any R[j][i] is less than 0.000001 then the value of B returned is greater
  than 0*/
  for(j=0; j<5; j+=1)
  {for(i=0; i<5; i+=1)
    if(fabs(R[j][i])<0.000001)
     {B=B+0;}
    else
     {B=B+1;}}

  return B;
}

int main()
{
   /*The variables and arrays used are declared here and the psi-array is
     initialised using the parameters specified corresponding to the solution
     psi(x,y)=sin(x)sinh(y) on the boundaries of the domain and the values
     are arranged in such a way so that we can iterated from the non-zero
     boundaries down to the zero ones so that we are accessing elements
     sequentially along the rows and columns*/
   int i, j, k;
   double x, y;
   double psi[7][7] = {{sin(1.0)*sinh(1.0), sin(5.0/6.0)*sinh(1.0),
                        sin(4.0/6.0)*sinh(1.0), sin(3.0/6.0)*sinh(1.0),
                        sin(2.0/6.0)*sinh(1.0), sin(1.0/6.0)*sinh(1.0), 0},
                       {sin(1.0)*sinh(5.0/6.0), 0, 0, 0, 0, 0, 0},
                       {sin(1.0)*sinh(4.0/6.0), 0, 0, 0, 0, 0, 0},
                       {sin(1.0)*sinh(3.0/6.0), 0, 0, 0, 0, 0, 0},
                       {sin(1.0)*sinh(2.0/6.0), 0, 0, 0, 0, 0, 0},
                       {sin(1.0)*sinh(1.0/6.0), 0, 0, 0, 0, 0, 0},
                       {0, 0, 0, 0, 0, 0, 0} };
   double R[5][5];

     FILE *fout;

   if ((fout=fopen("R.txt","w"))==NULL)
    { printf("Cannot open %s\n", "R.txt");
    exit(EXIT_FAILURE);}

/*The intial values of psi are set using the finite difference approximation
  of Laplace's equation*/
   for(j=1; j<6; j+=1)
   { for(i=1; i<6; i+=1)
```

```
   { psi[j][i]=(1.0/4.0)*(psi[j][i+1] + psi[j][i-1] + psi[j+1][i] + psi[j-1][i]);}
   }

/*The successive over relaxation process is implemented here and the procedure
  is iterated 30 times or until the residuals converge*/
   for(k=0; k<=30; k+=1)
   {
    { for(j=1; j<6; j+=1)
    { for(i=1; i<6; i+=1)
    { R[j-1][i-1]= psi[j+1][i] + psi[j-1][i] + psi[j][i-1] + psi[j][i+1] - 4*psi[j][i];
      psi[j][i]= psi[j][i] + (1.35/4.0)*R[j-1][i-1];}
   }}

   fprintf(fout,"%f\t %f\t %f\n", psi[1][3], psi[3][3], psi[5][3]);

   if(r(R)==0)
   {break;}
     }

   if ((fout=fopen("L_E.txt","w"))==NULL)
    { printf("Cannot open %s\n", "L_E.txt");
   exit(EXIT_FAILURE);}

/*A table of the psi[j][i] values is printed alongside the corresponding x and
  y values*/
   for(j=0; j<=6; j+=1)
   { for(i=0; i<=6; i+=1)
   { x=1.0-(double)i/(6.0);
     y=1.0-(double)j/(6.0);
     fprintf(fout,"%f\t %f\t %f\n", x, y, psi[j][i]);}
}

   fclose(fout);

   exit(0);
}
```

Appendix B Finite Difference Approximations

When a function U and its derivatives are single-valued, finite and continuous functions then we can write by Taylor's theorem

$$\begin{aligned} U(x_0+\delta x) &= U(x_0) + \delta x U'(x_0) + \frac{\delta x^2}{2}U''(x_0) + \frac{\delta x^3}{6}U'''(x_0) + \cdots \\ U(x_0-\delta x) &= U(x_0) - \delta x U'(x_0) + \frac{\delta x^2}{2}U''(x_0) - \frac{\delta x^3}{6}U'''(x_0) + \cdots \end{aligned} \tag{B.1}$$

We can add these to get

$$U(x_0+\delta x) + U(x_0-\delta x) = 2U(x_0) + \delta x^2 U''(x_0) + \mathcal{O}(\delta x^4) \tag{B.2}$$

Which we can rearrange to get

$$U''(x_0) = \left(\frac{d^2U}{dx^2}\right)_{x=x_0} \approx \frac{1}{\delta x^2}\{U(x_0+\delta x) - 2U(x_0) + U(x_0-\delta x)\} \tag{B.3}$$

with a leading error of order δx^2 on the right hand side.

We can extend this to functions of two variables to get

$$\psi_{xx} = (\frac{\partial^2 \psi}{\partial x^2})_{x=x_0} \approx \frac{1}{\delta x^2}\{\psi(x_0 + \delta x) - 2\psi(x_0) + \psi(x_0 - \delta x)\}$$
$$\psi_{yy} = (\frac{\partial^2 \psi}{\partial y^2})_{y=y_0} \approx \frac{1}{\delta y^2}\{\psi(y_0 + \delta y) - 2\psi(y_0) + \psi(y_0 - \delta y)\} \quad \text{(B.4)}$$

with errors of order δx^2 and δy^2 respectively.
Thus we can approximate the Laplace's equation to

$$\nabla^2 \psi \approx \frac{(\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j})}{(\delta x)^2} + \frac{(\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1})}{(\delta y)^2} = 0 \quad \text{(B.5)}$$

Where the subscripts refer to ψ evaluated at the position $x = i\delta x$ and $y = j\delta y$.
We can rearrange this to get the system of linear equations

$$\psi_{i,j} \approx \frac{\frac{(\psi_{i+1,j} + \psi_{i-1,j})}{(\delta x)^2} + \frac{(\psi_{i,j+1} + \psi_{i,j-1})}{(\delta y)^2}}{2(\frac{1}{(\delta x)^2} + \frac{1}{(\delta y)^2})} \quad \text{(B.6)}$$

Which, if $\delta x = \delta y = \delta$, reduces to

$$\psi_{i,j} \approx \frac{1}{4}\{\psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1}\} \quad \text{(B.7)}$$

On the Laplace operator in general, if $\delta x = \delta y = \delta$ we can discretise the Laplacian into a five point difference approximation

$$\nabla^2 \psi \simeq \frac{1}{\delta^2}\{\psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1} - 4\psi_{i,j}\} \quad \text{(B.8)}$$

which has a truncation error $\mathcal{O}(\delta^2)$.
We can compute more accurate finite-difference formulae with truncation errors of higher order powers of δ by using more terms in the Taylor expansion of the function. One example of this is the nine point difference approximation to the Laplacian.

$$\nabla^2 \psi \simeq \frac{1}{\delta^2}\left\{-\frac{1}{12}\psi_{i-2,j} + \frac{4}{3}\psi_{i-1,j} - \frac{15}{6}\psi_{i,j} + \frac{4}{3}\psi_{i+1,j} - \frac{1}{12}\psi_{i+2,j}\right\}$$
$$+ \frac{1}{\delta^2}\left\{-\frac{1}{12}\psi_{i,j-2} + \frac{4}{3}\psi_{i,j-1} - \frac{15}{6}\psi_{i,j} + \frac{4}{3}\psi_{i,j+1} - \frac{1}{12}\psi_{i,j+2}\right\} \quad \text{(B.9)}$$

Appendix C Iterative Methods

A finite difference method for approximating a boundary value problem leads to a system of algebraic simultaneous equations. For linear boundary-value problems these equations are always linear but their number is generally large and for this reason their solution is a major problem in itself. Methods of solution belong essentially to either the class of direct methods or the class of iterative methods.
Direct methods solve the system of equations in a known number of arithmetic operations and errors in the solution arise entirely from rounding errors introduced during computations. They are generally elimination methods such as systematic gauss elimination or triangular decomposition.

The matrices associated with difference equations approximating partial differential equations are band matrices, i.e. matrices with non-zero elements lying between two sub-diagonals parallel to the main diagonal. They are also usually 'sparse', i.e. the number of zero elements in the matrix is much greater than the number of non-zero elements. For this type of matrix standard Gaussian elimination is inefficient in the sense that it 'fills-in' the zero elements within the band with non-zero numbers that have to be stored in the computer and used at subsequent stages of the elimination process.

With iterative methods however no arithmetic is associated with the zero coefficients so considerably fewer numbers have to be stored in the computer. They can be used to solve systems of equations that are too large for the use of direct methods. Programming and data handling are also much simpler than for direct methods. Another advantage not possessed by direct methods, is their frequent extension to the solution of sets of non-linear equations.

The efficient use of iterative methods is very dependent however upon the direct calculation or estimation of the value or values of some acceleration parameter, and upon the coefficient matrix being well-conditioned, otherwise convergence will be slow and the volume of arithmetic enormous but with optimum acceleration parameters the volume of arithmetic of iterative methods for large sets of equations may well be less than for direct methods.

Here we describe three such iterative methods:

Jacobi Method
Consider the system of linear equations

$$\sum_{j=1}^{m} a_{ij} x_j = b_i \tag{C.1}$$

We can write this is as

$$A\mathbf{x} = \mathbf{b} \tag{C.2}$$

where we assume A is a non-singular matrix.
We can represent A as the matrix sum

$$A = D - E - F \tag{C.3}$$

where D is a diagonal matrix and E and F are strictly upper and lower diagonal matrices respectively whose entries are the negatives of A below and above the main diagonal of A.

$$D\mathbf{x} = (E + F)\mathbf{x} + \mathbf{b} \tag{C.4}$$

We can write

$$x_i = \frac{1}{a_{ii}}(b_i - \sum_{j=1, j\neq i}^{m} a_{ij} x_j) \tag{C.5}$$

Denote the first approximation to x_i by $x_i^{(1)}$, the second by $x_i^{(2)}$ and so on. Assume that n of them have been calculated. Then the Jacobi iterative method expresses the $(n+1)^{th}$ iterative values exclusively in terms of the n^{th} iterative values such that

$$x_i^{(n+1)} = \frac{1}{a_{ii}}\left\{ b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(n)} - \sum_{j=i+1}^{m} a_{ij} x_j^{(n)} \right\} \tag{C.6}$$

In terms of matrices we can say

$$D\mathbf{x}^{(m+1)} = (E + F)\mathbf{x}^{(m)} + \mathbf{b} \tag{C.7}$$

Since D is non-singular we can write

$$\mathbf{x}^{(m+1)} = D^{-1}(E + F)\mathbf{x}^{(m)} + D^{-1}\mathbf{b} \tag{C.8}$$

Define the point Jacobi matrix associated with A to be

$$J = D^{-1}(E + F) \tag{C.9}$$

The Jacobi method converges as the number of iterations increases but it is the slowest of all converging iteration methods.

Gauss-Siedel Method

For the Gauss-Siedel method the $(n+1)^{th}$ iterative values are used as soon as they are available and the iteration equation is given by

$$x_i^{(n+1)} = \frac{1}{a_{ii}}\left\{ b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(n+1)} - \sum_{j=i+1}^{m} a_{ij} x_j^{(n)} \right\} \tag{C.10}$$

Or in matrix notation

$$\mathbf{x}^{(m+1)} = (D - E)^{-1} F\mathbf{x}^{(m)} + (D - E)^{-1}\mathbf{b} \tag{C.11}$$

Define the Gauss-Siedel matrix associated with A

$$C = (D - E)^{-1} F \tag{C.12}$$

The Gauus-Siedel method converges twice as fast as the Jacobi algorithm since when we go through grid points in the natural order we don't need to keep two copies of the solution, one for the old values at iteration k and another for the new values at iteration k+1; we can use an immediate replacement, therefore it uses less storage and is twice as fast.

The Successive Over-Relaxation (SOR) Method

The SOR method is an acceleration of the Gauss-Siedel method which uses an over correction relaxation parameter which lies between 1 and 2, anticipating future corrections.

$$x_i^{(n+1)} = \frac{\alpha}{a_{ii}}\left\{ b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(n+1)} - \sum_{j=i+1}^{m} a_{ij} x_j^{(n)} \right\} + (1 - \alpha) x_i^{(n)} \tag{C.13}$$

$$(D - \alpha E)\mathbf{x}^{(m+1)} = \{(1 - \alpha)D + \alpha E\}\mathbf{x}^{(m)} + \alpha\mathbf{b} \tag{C.14}$$

As $(D - \alpha E)$ is non singular for any choice of alpha then with $L = D^{-1}E$ and $U = D^{-1}F$ this takes the form

$$\mathbf{x}^{(m+1)}=(I - \alpha L)^{-1}\{(1 - \alpha)I + \alpha U\}\mathbf{x}^{(m)} + \alpha(I - \alpha L)^{-1}D^{-1}\mathbf{b} \tag{C.15}$$

and we can define a point successive over relaxation matrix

$$L_w = (I - \alpha L)^{-1}\{(1 - \alpha)I + \alpha U\} \tag{C.16}$$

Over relaxation can give faster convergence than the Gauss-Siedel method under certain mathematical restrictions generally satisfied by matrices arising from finite differencing. However, a problem of paramount importance associated with the SOR method is the determination of a suitable value for the acceleration parameter alpha. Ideally, we want the optimum value alpha opt of alpha which minimises the spectral radius of the SOR iteration matrix since it maximises the rate of convergence of the method. At the present time no formula exists for the determination of alpha opt for an arbitrary set of linear equations. However it can be calculated for many of the difference equation approximations for first and second order partial differential equations because their matrices are often of a special type called 2-cyclic matrices.

Let $A = [a_{ij}]$ be a complex nxn complex matrix with eigenvalues $\lambda_i, 1 \leq i \leq n$. Then

$$\rho(A) = \max_{1\leq i\leq n}|\lambda_i| \tag{C.17}$$

is the spectral radius of the matrix A. Geometrically if all the eigenvalues are plotted in the complex z-plane then $\rho(A)$ is the radius of the smallest disk, centred at the origin, which includes all eigenvalues of A.
It can be proved that when the coefficient matrix for the system of linear equations (C.2) is block tridiagonal with non-zero diagonal elements and all the eigenvalues of the Jacobi iteration matrix B associated with A are real and such that $0 < \rho(B) < 1$ then

$$\alpha_{opt} = \frac{2}{1 + \sqrt{\{1 - \rho^2(J)\}}} \tag{C.18}$$

and the SOR method applied to the equations $A\mathbf{x} = \mathbf{b}$ converges for all α in the range $0 < \alpha < 2$.

For a function which satisfies Laplace's equation on the points of the rectangular domain of sides $p\delta$ and $q\delta$ with Dirichlet boundary conditions it can be shown that the spectral radius of the Jacobi matrix is

$$\rho(J) = \frac{1}{2}((\cos\left(\frac{\pi}{p}\right) + \cos\left(\frac{\pi}{q}\right)) \tag{C.19}$$

which gives the optimum value of α used throughout our procedure.