HW3, Math 228A

Date due 11/09/2010 UC Davis, California Fall 2010

Nasser M. Abbasi

Fall 2010 Compiled on January 4, 2020 at 11:12pm [public]

Contents

0.1 residual error animation

Animation of the residual error R as it changes during iterative solution. Tolerance used for these is h^2, stopping criteria used is relative error < tolerance Only SOR was done.

0.1.1 SOR with $h = 2^{-7}$

These animations are large in size. (Click on any to see in actual size, will open in new window)

0.1.2 SOR with $h = 2^{-5}$

These animations are smaller in size. (Click on any to see in actual size, will open in new window)

0.2 Animation for density plot animations of solvers for problem 1

Animation plots below show solver as it updates each grid point by point in its main loop

The above shows each of the solvers (Jacobi, Gauss-Seidel, SOR) in the process of updating the grid during one iteration of the main loop.

Notice the how GS and SOR solvers update the solution immediately (left to right, down to top numbering is used), while Jacobi solver updates the solution only at the end each iterative step.

This one below was done for $h = 2^{-3}$. Clicking on it shows animation.

0.3 Animations of iterative solution

Animations of iterative solution (Click on image to see animation), Stopping criteria used is relative residual method, tolerance is h^2

HW3, Math 228A
Date due 11/09/2010

UC Davis, California Fall 2010

Nasser M. AbbasiFall 2010 Compiled on January 4, 2020 at 11:12pm [public]

Contents

0.4 Problem 1

Problem statement

1. Use Jacobi, Gauss-Seidel, and SOR (with optimal ω) to solve

$$\Delta u = -\exp\left(-(x-0.25)^2 - (y-0.6)^2\right)$$

on the unit square $(0,1)\times(0,1)$ with homogeneous Dirichlet boundary conditions. Find the solution for mesh spacings of $h = 2^{-5}$, 2^{-6}, and 2^{-7}. What tolerance did you use? What stopping criteria did you use? What value of ω did you use? Report the number of iterations it took to reach convergence for each method for each mesh.

Figure 1: Problem 1

Answer

Using the method of splitting, the iteration matrix T is found, for each method, for solving $Au = f$.

Let $A = M - N$, then $Au = f$ becomes

$$\begin{aligned}(M-N)u &= f\\ Mu &= Nu + f\\ u^{[k+1]} &= M^{-1}Nu^{[k]} + M^{-1}f \end{aligned} \tag{1}$$

0.4.1 Finding iterative matrix for the different solvers

The Jacobi method

For the Jacobi method $M = D$ and $N = L+U$, where D is the diagonal of A, L is the negative of the strictly lower triangle matrix of A and U is negative of the strictly upper triangle matrix of A. Hence(1) becomes

$$\begin{aligned}u^{[k+1]} &= D^{-1}(L+U)u^{[k]} + D^{-1}f\\ u^{[k+1]} &= Tu^{[k]} + C\end{aligned}$$

Where T is called the Jacobi iteration matrix. Since $A = D - L - U$, hence $L + U = D - A$, therefore the iteration matrix T can be written as

$$\begin{aligned}T &= D^{-1}(D-A)\\ &= I - D^{-1}A\end{aligned}$$

or

$$\begin{aligned}T &= (I - D^{-1}A)\\ C &= D^{-1}f\end{aligned}$$

The Gauss-Seidel method

For Gauss-Seidel, $M = D - L$ and $N = U$, hence (1) becomes

$$u^{[k+1]} = (D-L)^{-1}\,U\,u^{[k]} + (D-L)^{-1}\,f$$

or

$$u^{[k+1]} = Tu^{[k]} + C$$

where

$$\begin{aligned}T &= (D-L)^{-1}\,U\\ C &= (D-L)^{-1}\,f\end{aligned}$$

The SOR method

For SOR, $M = \frac{1}{\omega}(D - \omega L)$ and $N = \frac{1}{\omega}((1-\omega)D + \omega U)$. Hence (1) becomes

$$\begin{aligned} u^{[k+1]} &= \left[\frac{1}{\omega}(D - \omega L)\right]^{-1}\left[\frac{1}{\omega}((1-\omega)D + \omega U)\right]u^{[k]} + \left[\frac{1}{\omega}(D - \omega L)\right]^{-1} f \\ &= (D - \omega L)^{-1}((1-\omega)D + \omega U)\,u^{[k]} + \omega(D - \omega L)^{-1} f \\ &= Tu^{[k]} + C \end{aligned}$$

where

$$\begin{aligned} T &= (D - \omega L)^{-1}((1-\omega)D + \omega U) \\ C &= \omega(D - \omega L)^{-1} \end{aligned}$$

Summary of iterative matrices used

This table summarizes the expression for the iterative matrix T and for the matrix C in the equation $u^{[k+1]} = Tu^{[k]} + C$ for the different methods used.

method	T	C
Jacobi	$(I - D^{-1}A)$	$D^{-1}f$
GS	$(D - L)^{-1}U$	$(D - L)^{-1}f$
SOR	$(D - \omega L)^{-1}((1-\omega)D + \omega U)$	$\omega(D - \omega L)^{-1}$

The discretized algebraic equation resulting from approximating $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y)$ with Dirichlet boundary conditions is based on the use of the standard 5 point Laplacian

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{1}{h^2}\left(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j}\right)$$

which has local truncation error $O(h^2)$.

The notation used above is based on the following grid formating

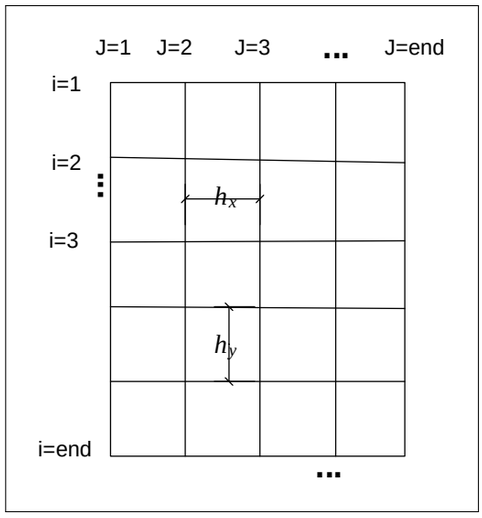

Figure 2: Grid notation

The derivation of the above formula is as follows: Consider a grid where the mesh spacing in the x-direction is h_x and in the y-direction is h_y. Then $\frac{\partial^2 u}{\partial x^2}$ is approximated, at position (i, j) by $\frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{h_x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ is approximated, at position (i, j), by $\frac{U_{i,j-1} - 2U_{i,j} + U_{i,j+1}}{h_y^2}$ therefore $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \approx \frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{h_x^2} + \frac{U_{i,j-1} - 2U_{i,j} + U_{i,j+1}}{h_y^2}$, and since $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f_{i,j}$ at that position, this results in

$$\frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{h_x^2} + \frac{U_{i,j-1} - 2U_{i,j} + U_{i,j+1}}{h_y^2} = f_{i,j}$$

Solving for $U_{i,j}$ from the above gives

$$\begin{aligned} U_{i,j} &= \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{i-1,j}h_y^2+U_{i+1,j}h_y^2+U_{i,j-1}h_x^2+U_{i,j+1}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j} \\ &= \frac{1}{2\left(h_x^2+h_y^2\right)}\left(\left(U_{i-1,j}+U_{i+1,j}\right)h_y^2+\left(U_{i,j-1}+U_{i,j+1}\right)h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j} \end{aligned}$$

The following table shows the formula for updating $U_{i,j}$ using each of the 3 methods for the 2D case, under the assumption of uniform mesh spacing, i.e. when $h_x = h_y$ which simplifies the above formula as given below

method	Formula for updating $U_{i,j}$, uniform mesh
Jacobi	$U_{i,j}^{[k+1]} = \frac{1}{4}\left(U_{i-1,j}^{[k]}+U_{i+1,j}^{[k]}+U_{i,j-1}^{[k]}+U_{i,j+1}^{[k]}-h^2f_{i,j}\right)$
GS	$U_{i,j}^{[k+1]} = \frac{1}{4}\left(U_{i-1,j}^{[k+1]}+U_{i+1,j}^{[k]}+U_{i,j-1}^{[k+1]}+U_{i,j+1}^{[k]}-h^2f_{i,j}\right)$
SOR	$U_{i,j}^{[k+1]} = \frac{\omega}{4}\left(U_{i-1,j}^{[k+1]}+U_{i+1,j}^{[k]}+U_{i,j-1}^{[k+1]}+U_{i,j+1}^{[k]}-h^2f_{i,j}\right)+(1-\omega)U_{i,j}^{[k]}$

note that for SOR, the general formula, using nonuniform mesh can be derived as follows

$$\begin{aligned} U_{(gs)i,j}^{[k+1]} &= \frac{1}{4}\left(U_{i-1,j}^{[k+1]}+U_{i+1,j}^{[k]}+U_{i,j-1}^{[k+1]}+U_{i,j+1}^{[k]}-h^2f_{i,j}\right) \\ U_{(sor)i,j}^{[k+1]} &= U_{i,j}^{[k]}+\omega\left(U_{(gs)i,j}^{[k+1]}-U_{i,j}^{[k]}\right) \end{aligned}$$

The second formula above can be simplified by substituing the first equation into it to yield

$$\begin{aligned} U_{(sor)i,j}^{[k+1]} &= U_{i,j}^{[k]}+\omega\left(\frac{1}{4}\left(U_{i-1,j}^{[k+1]}+U_{i+1,j}^{[k]}+U_{i,j-1}^{[k+1]}+U_{i,j+1}^{[k]}-h^2f_{i,j}\right)-U_{i,j}^{[k]}\right) \\ &= \frac{\omega}{4}\left(U_{i-1,j}^{[k+1]}+U_{i+1,j}^{[k]}+U_{i,j-1}^{[k+1]}+U_{i,j+1}^{[k]}-h^2f_{i,j}\right)+(1-\omega)U_{i,j}^{[k]} \end{aligned}$$

Which is what shown in the table above. Using the same procedure, but for the general case, results in

$$\begin{aligned} U_{(gs)i,j}^{[k+1]} &= \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{i-1,j}^{[k+1]}h_y^2+U_{i+1,j}^{[k]}h_y^2+U_{i,j-1}^{[k+1]}h_x^2+U_{i,j+1}^{[k]}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j} \\ U_{(sor)i,j}^{[k+1]} &= U_{i,j}^{[k]}+\omega\left(U_{(gs)i,j}^{[k+1]}-U_{i,j}^{[k]}\right) \end{aligned}$$

Hence, again, the second equation above becomes

$$\begin{aligned} U_{(sor)i,j}^{[k+1]} &= U_{i,j}^{[k]}+\omega\left(\frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{i-1,j}^{[k+1]}h_y^2+U_{i+1,j}^{[k]}h_y^2+U_{i,j-1}^{[k+1]}h_x^2+U_{i,j+1}^{[k]}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}-U_{i,j}^{[k]}\right) \\ &= \omega\left[\frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{i-1,j}^{[k+1]}h_y^2+U_{i+1,j}^{[k]}h_y^2+U_{i,j-1}^{[k+1]}h_x^2+U_{i,j+1}^{[k]}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}\right]+(1-\omega)U_{i,j}^{[k]} \end{aligned}$$

Hence, for non-uniforma mesh the above update table becomes

method	Formula for updating $U_{i,j}$, non-uniform mesh
Jacobi	$U_{i,j}^{[k+1]} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(\left(U_{i-1,j}+U_{i+1,j}\right)h_y^2+\left(U_{i,j-1}+U_{i,j+1}\right)h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}$
GS	$U_{i,j}^{[k+1]} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{i-1,j}^{[k+1]}h_y^2+U_{i+1,j}^{[k]}h_y^2+U_{i,j-1}^{[k+1]}h_x^2+U_{i,j+1}^{[k]}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}$
SOR	$U_{i,j}^{[k+1]} = \omega\left[\frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{i-1,j}^{[k+1]}h_y^2+U_{i+1,j}^{[k]}h_y^2+U_{i,j-1}^{[k+1]}h_x^2+U_{i,j+1}^{[k]}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}\right]+(1-\omega)U_{i,j}^{[k]}$

The residual formula is $R^{[k]} = f - Au^{[k]}$, which can be written using the above notations as

$$R_{i,j} = f_{i,j}-\left(\frac{U_{i-1,j}-2U_{i,j}+U_{i+1,j}}{h_x^2}+\frac{U_{i,j-1}-2U_{i,j}+U_{i,j+1}}{h_y^2}\right)$$

and for a uniform mesh the above becomes

$$R_{i,j} = f_{i,j}-\frac{1}{h^2}\left(U_{i-1,j}+U_{i+1,j}+U_{i,j-1}+U_{i,j+1}-4U_{i,j}\right)$$

0.4.2 Tolerance used

The tolerance ϵ used is based on the global error in the numerical solution. For this problem $\|e\| = Ch^2$ where C is a constant C. Two different values for the constant were tried in the implementation: 0.1 and 1.

0.4.3 stopping criteria

The stopping criteria used was based on the use of the relative residual. Given

$$R^{[k]} = f - Au^{[k]}$$

The iterative process was stopped when the following condition was satisfied

$$\frac{\left\|R\left[^{k}\right]\right\|}{\|f\|} < \epsilon$$

Other possible stopping criterion are (but were not used) are

1. Absolute error: convergence achieved when $\left\|u^{[k+1]} - u^{[k]}\right\| < \epsilon$
2. Relative error: convergence achieved when $\frac{\|u^{[k+1]} - u^{[k]}\|}{\|u^{[k]}\|} < \epsilon$

The above two criterion are not used as they do not perform as well for Jacobi and Gauss-Seidel.

0.4.4 ω used for SOR

The ω_{opt} used is based on the relation to the spectral radius of the *Jacobi* iteration matrix. This relation was derived in class

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho_{Jacobian}^2}}$$

where for the $2D$ Poisson problem $\rho_{Jacobian}$ is given as $\cos(\pi h)$ where h is the grid spacing. Using this in the above and approximating for small h results in

$$\omega_{opt} \approx 2(1 - \pi h)$$

For the given h values in the problem, the following values were used for ω_{opt}

$h_x = h_y = h$	ω_{opt}
2^{-3}	1.214 6
2^{-4}	1.607 3
2^{-5}	1.803 7
2^{-6}	1.901 8
2^{-7}	1.950 9

Notice that the above approximation formula is valid for small h and will not result in good convergence for SOR if used for large h.

Update 12/29/2010: After more analysis, the following formula is found to produce better results

$$t = cos(\pi\ h_x) + cos(\pi h_y)$$
$$poly(x) = x^2 t^2 - 16x + 16$$

Solve the above polynomial for x and then take the smaller root. This will be ω_{opt}, Using this results in

$h_x = h_y = h$	ω_{opt}
2^{-3}	1.4464
2^{-4}	1.6763
2^{-5}	1.821
2^{-6}	1.9064
2^{-7}	1.9509

0.4.5 Algorithm details

In this section, some of the details of the implementation are described for reference. The Matrix A is not used directly in the iterative method, but its structure is briefly described.

In the discussion below, updating the grid is described for the Jacobi method, showing how the new values of the dependent variable u are calculated.

Numbering system and grid updating

The numbering system used for the grid is the one described in class. The indices for the unknown $u_{i,j}$ are numbered row wise, left to right, bottom to top. This follows the standard Cartesian coordinates system. The reason for this, is to allow the use of the standard formula for the 5 point Laplacian. The following diagram illustrate the 5 point Laplacian borrowed from the text book, figure 5, page 61 "*Finite Difference Methods for Ordinary and Partial Differential Equations*" by Randall J. LeVeque

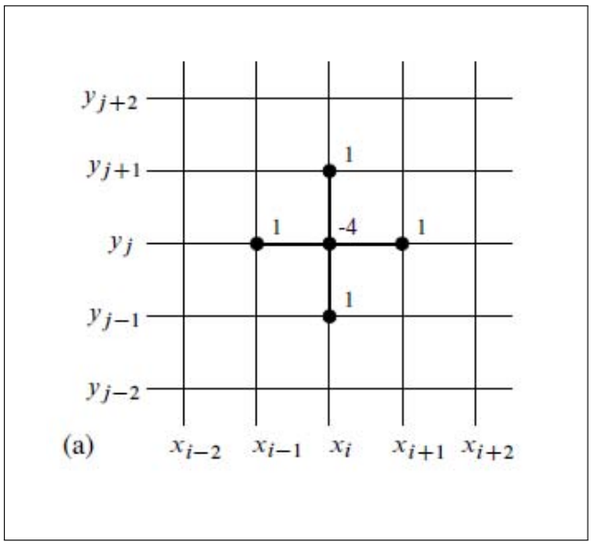

Figure 3: 5-point stencile for Laplacian at x(i,j)

Lower case n is used to indicate the number of unknowns along one dimension, and upper case N is used to indicate the total number of unknowns.

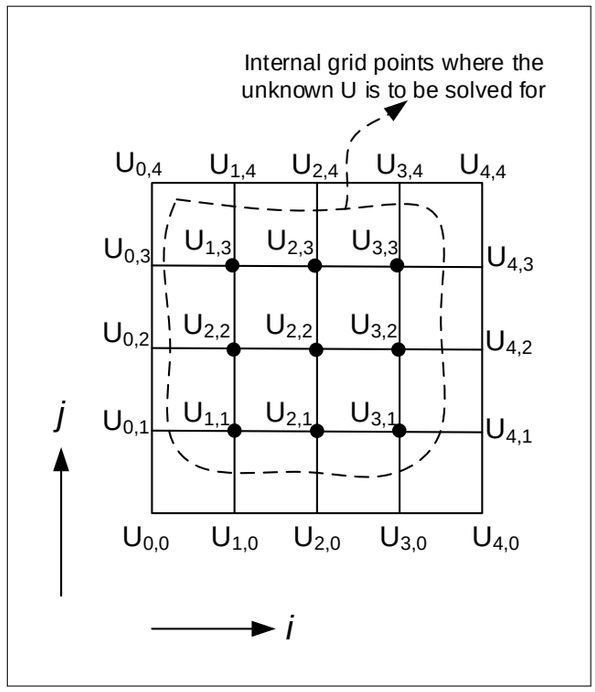

Figure 4: stencil move

In the diagram above, $n = 3$ is the number of unknowns on each one row or one column, and since there are 3 internal rows, there will be 9 unknowns, all are located on internal grid points. There are a total of 25 grid points, 16 of which are on the boundaries and 9 internal.

To update the grid, the 5 point Laplacian is centered at each internal grid point and then moved left to right, bottom to top, each time an updated value of the unknown is generated and stored in an auxiliary grid.

In the Jacobian method, the new value at the location $x_{i,j}$ is not used in the calculation to update its neighbors when the stencil is moved, but

Only when the whole grid has been swept by the Laplacian will the grid be updated by copying the content of the auxiliary grid into it.

In the Gauss-Seidel and SOR, this not the case. Once a grid point have been updated, its new value is used immediately in the update of its neighbor as the Laplacian sweeps the grid. No auxiliary grid is required. In other words, the updates happens 'in place'.

Continuing this example, the following diagram shows how each grid point is updated (In a parallel computation, these operations can all be done at once for the Jacobi solver, but not for the Gauss-Seidel or the SOR solver, unless different numbering scheme is used such as black-red numbering).

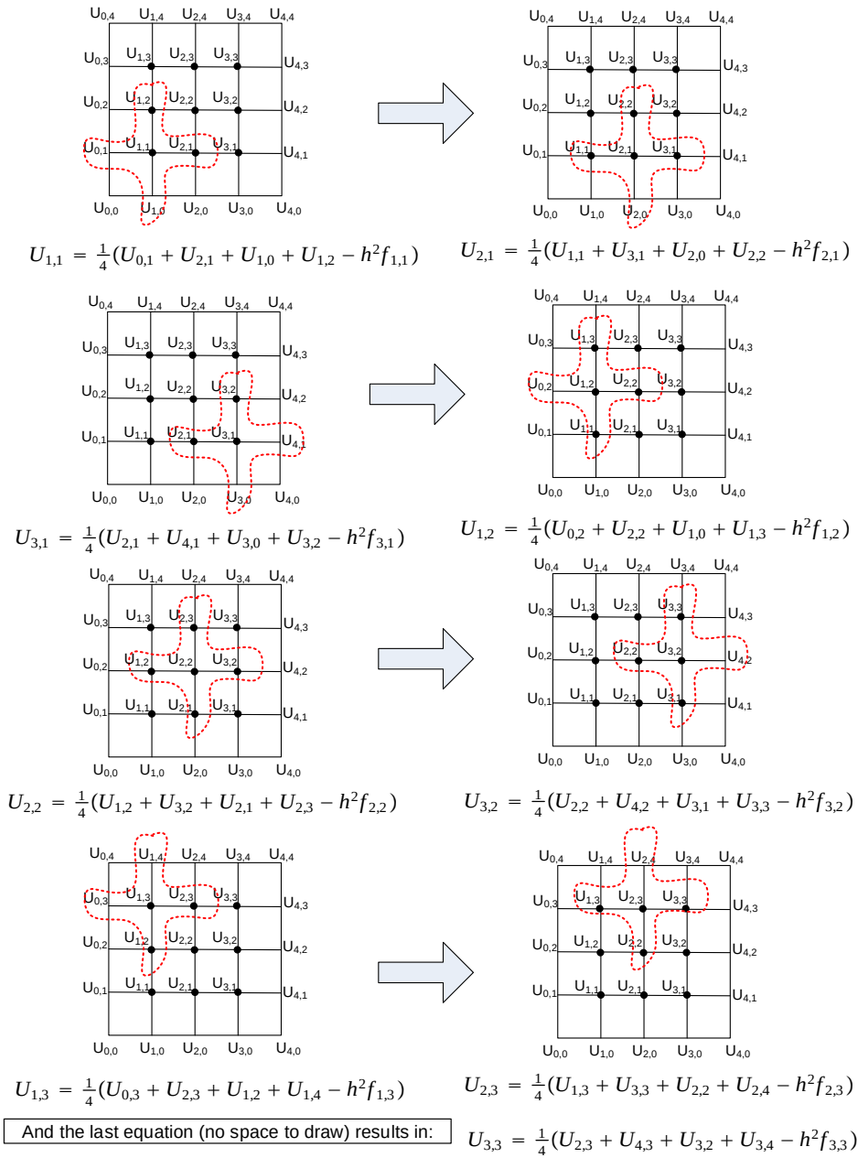

Figure 5: stencil move 2

Now that the grid has been updated once, the process is repeated again. This process is continued until convergence is achieved.

0.4.6 Structure of $Au = f$

The structure of $Au = f$ system matrix is now described. As an example, for number of unknowns = 9 the following characteristics of the matrix A can be seen

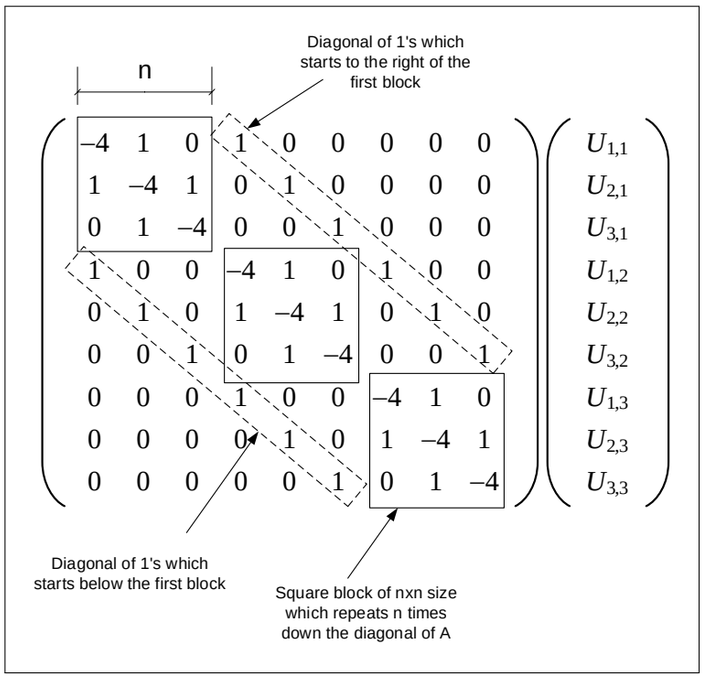

Figure 6: A structure

Structure of the A matrix for elliptic 2D PDE with Dirchillet boundary conditions for non-uniform mesh

This section was added at a later time here for completion, and is not required for the HW. Below the A matrix form is derived for the case for non-uniform mesh (this means h_x is not neccessarly the same as h_y) and also, the number of grid points in the x-direction is not the same as the number of grid points in the y-direction.

To ease the derivation, we will use a 5×5 grid as an example, hence this will generate 9 equations as there are 9 unknowns. From this derivation we will be able to see the form of the A matrix.

In addition, in this derivation, we will use i to represent row number, and j to represent column number, as this more closely matches the convention.

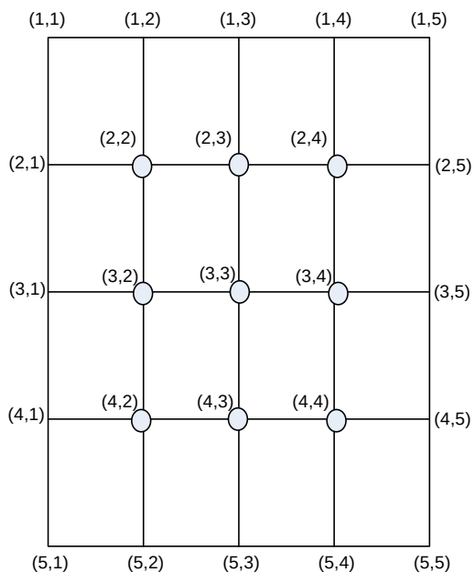

Figure 7: new grid

The unknowns are shown above in the circles. From earlier, we found that the discretization

for the elliptic PDE at a node is

$$U_{i,j} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{i,j-1}h_y^2 + U_{i,j+1}h_y^2 + U_{i-1,j}h_x^2 + U_{i+1,j}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}$$

We will now write the 9 equations for each node, starting with $(2,2)$ to node $(4,4)$

$$U_{2,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,1}h_y^2 + U_{2,3}h_y^2 + U_{1,2}h_x^2 + U_{3,2}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,2}$$

$$U_{2,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,2}h_y^2 + U_{2,4}h_y^2 + U_{1,3}h_x^2 + U_{3,3}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,3}$$

$$U_{2,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,3}h_y^2 + U_{2,5}h_y^2 + U_{1,4}h_x^2 + U_{3,4}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,4}$$

$$U_{3,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,1}h_y^2 + U_{3,3}h_y^2 + U_{2,2}h_x^2 + U_{4,2}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,2}$$

$$U_{3,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,2}h_y^2 + U_{3,4}h_y^2 + U_{2,3}h_x^2 + U_{4,3}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,3}$$

$$U_{3,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,3}h_y^2 + U_{3,5}h_y^2 + U_{2,4}h_x^2 + U_{4,4}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,4}$$

$$U_{4,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,1}h_y^2 + U_{4,3}h_y^2 + U_{3,2}h_x^2 + U_{5,2}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,2}$$

$$U_{4,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,2}h_y^2 + U_{4,4}h_y^2 + U_{3,3}h_x^2 + U_{5,3}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,3}$$

$$U_{4,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,3}h_y^2 + U_{4,5}h_y^2 + U_{3,4}h_x^2 + U_{5,4}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,4}$$

Now, moving the knowns to the right, results in

$$\begin{aligned}
2\left(h_x^2+h_y^2\right)U_{2,2} - U_{2,3}h_y^2 - U_{3,2}h_x^2 &= -h_x^2h_y^2f_{2,2} + U_{2,1}h_y^2 + U_{1,2}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,3} - U_{2,2}h_y^2 - U_{2,4}h_y^2 - U_{3,3}h_x^2 &= -h_x^2h_y^2f_{2,3} + U_{1,3}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,4} - U_{2,3}h_y^2 - U_{3,4}h_x^2 &= -h_x^2h_y^2f_{2,4} + U_{2,5}h_y^2 + U_{1,4}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{3,2} - U_{3,3}h_y^2 - U_{2,2}h_x^2 - U_{4,2}h_x^2 &= -h_x^2h_y^2f_{3,2} + U_{3,1}h_y^2\\
2\left(h_x^2+h_y^2\right)U_{3,3} - U_{3,2}h_y^2 - U_{3,4}h_y^2 - U_{2,3}h_x^2 - U_{4,3}h_x^2 &= -h_x^2h_y^2f_{3,3}\\
2\left(h_x^2+h_y^2\right)U_{3,4} - U_{3,3}h_y^2 - U_{2,4}h_x^2 - U_{4,4}h_x^2 &= -h_x^2h_y^2f_{3,4} + U_{3,5}h_y^2\\
2\left(h_x^2+h_y^2\right)U_{4,2} - U_{4,3}h_y^2 - U_{3,2}h_x^2 &= -h_x^2h_y^2f_{4,2} + U_{4,1}h_y^2 + U_{5,2}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,3} - U_{4,2}h_y^2 - U_{4,4}h_y^2 - U_{3,3}h_x^2 &= -h_x^2h_y^2f_{4,3} + U_{5,3}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,4} - U_{4,3}h_y^2 - U_{3,4}h_x^2 &= -h_x^2h_y^2f_{4,4} + U_{4,5}h_y^2 + U_{5,4}h_x^2
\end{aligned}$$

Now, we can write $Ax = b$ as, letting $\beta = 2\left(h_x^2+h_y^2\right)$

$$\begin{pmatrix}
\beta & -h_y^2 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0\\
-h_y^2 & \beta & -h_y^2 & 0 & -h_x^2 & 0 & 0 & 0 & 0\\
0 & -h_y^2 & \beta & 0 & 0 & -h_x^2 & 0 & 0 & 0\\
-h_x^2 & 0 & 0 & \beta & -h_y^2 & 0 & -h_x^2 & 0 & 0\\
0 & -h_x^2 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & -h_x^2 & 0\\
0 & 0 & -h_x^2 & 0 & -h_y^2 & \beta & 0 & 0 & -h_x^2\\
0 & 0 & 0 & -h_x^2 & 0 & 0 & \beta & -h_y^2 & 0\\
0 & 0 & 0 & 0 & -h_y^2 & 0 & -h_y^2 & \beta & -h_y^2\\
0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & -h_y^2 & \beta
\end{pmatrix}
\begin{pmatrix}
U_{22}\\U_{23}\\U_{24}\\U_{32}\\U_{33}\\U_{34}\\U_{42}\\U_{43}\\U_{44}
\end{pmatrix} =
\begin{pmatrix}
-h_x^2h_y^2f_{2,2} + U_{2,1}h_y^2 + U_{1,2}h_x^2\\
-h_x^2h_y^2f_{2,3} + U_{1,3}h_x^2\\
-h_x^2h_y^2f_{2,4} + U_{2,5}h_y^2 + U_{1,4}h_x^2\\
-h_x^2h_y^2f_{3,2} + U_{3,1}h_y^2\\
-h_x^2h_y^2f_{3,3}\\
-h_x^2h_y^2f_{3,4} + U_{3,5}h_y^2\\
-h_x^2h_y^2f_{4,2} + U_{4,1}h_y^2 + U_{5,2}h_x^2\\
-h_x^2h_y^2f_{4,3} + U_{5,3}h_x^2\\
-h_x^2h_y^2f_{4,4} + U_{4,5}h_y^2 + U_{5,4}h_x^2
\end{pmatrix}$$

Now we will do another case $n_x \neq n_y$

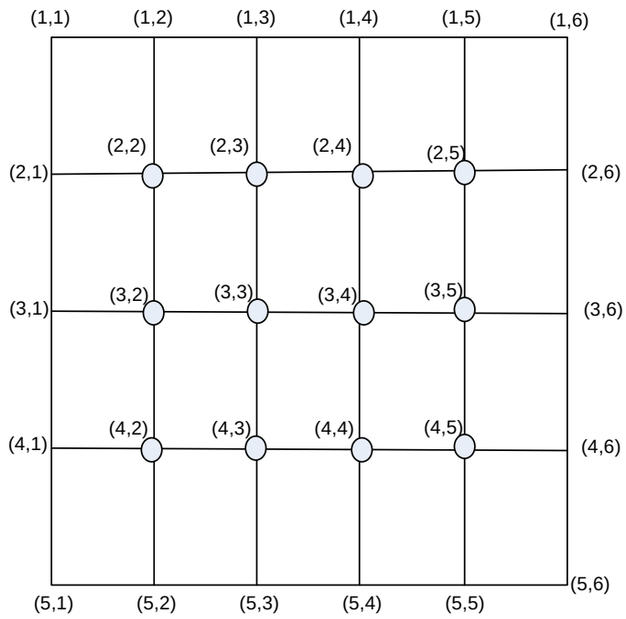

Figure 8: new grid 2

The unknowns are shown above in the circles. From earlier, we found that the discretization for the elliptic PDE at a node is

$$U_{i,j} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{i,j-1}h_y^2 + U_{i,j+1}h_y^2 + U_{i-1,j}h_x^2 + U_{i+1,j}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{i,j}$$

We will now write the 12 equations for each node, starting with $(2,2)$ to node $(4,5)$

$$U_{2,2} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{2,1}h_y^2 + U_{2,3}h_y^2 + U_{1,2}h_x^2 + U_{3,2}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{2,2}$$

$$U_{2,3} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{2,2}h_y^2 + U_{2,4}h_y^2 + U_{1,3}h_x^2 + U_{3,3}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{2,3}$$

$$U_{2,4} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{2,3}h_y^2 + U_{2,5}h_y^2 + U_{1,4}h_x^2 + U_{3,4}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{2,4}$$

$$U_{2,5} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{2,4}h_y^2 + U_{2,6}h_y^2 + U_{1,5}h_x^2 + U_{3,5}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{2,5}$$

$$U_{3,2} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{3,1}h_y^2 + U_{3,3}h_y^2 + U_{2,2}h_x^2 + U_{4,2}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{3,2}$$

$$U_{3,3} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{3,2}h_y^2 + U_{3,4}h_y^2 + U_{2,3}h_x^2 + U_{4,3}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{3,3}$$

$$U_{3,4} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{3,3}h_y^2 + U_{3,5}h_y^2 + U_{2,4}h_x^2 + U_{4,4}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{3,4}$$

$$U_{3,5} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{3,4}h_y^2 + U_{3,6}h_y^2 + U_{2,5}h_x^2 + U_{4,5}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{3,4}$$

$$U_{4,2} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{4,1}h_y^2 + U_{4,3}h_y^2 + U_{3,2}h_x^2 + U_{5,2}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{4,2}$$

$$U_{4,3} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{4,2}h_y^2 + U_{4,4}h_y^2 + U_{3,3}h_x^2 + U_{5,3}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{4,3}$$

$$U_{4,4} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{4,3}h_y^2 + U_{4,5}h_y^2 + U_{3,4}h_x^2 + U_{5,4}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{4,4}$$

$$U_{4,5} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{4,4}h_y^2 + U_{4,6}h_y^2 + U_{3,5}h_x^2 + U_{5,5}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{4,5}$$

Now, moving the knowns to the right, results in

$$\begin{aligned}
2\left(h_x^2+h_y^2\right)U_{2,2}-U_{2,3}h_y^2-U_{3,2}h_x^2 &= -h_x^2h_y^2f_{2,2}+U_{2,1}h_y^2+U_{1,2}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,3}-U_{2,2}h_y^2-U_{2,4}h_y^2-U_{3,3}h_x^2 &= -h_x^2h_y^2f_{2,3}+U_{1,3}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,4}-U_{2,3}h_y^2-U_{2,5}h_y^2-U_{3,4}h_x^2 &= -h_x^2h_y^2f_{2,4}+U_{1,4}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,5}-U_{2,4}h_y^2+U_{3,5}h_x^2 &= -h_x^2h_y^2f_{2,5}+U_{2,6}h_y^2+U_{1,5}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{3,2}-U_{3,3}h_y^2-U_{2,2}h_x^2-U_{4,2}h_x^2 &= -h_x^2h_y^2f_{3,2}+U_{3,1}h_y^2\\
2\left(h_x^2+h_y^2\right)U_{3,3}-U_{3,2}h_y^2-U_{3,4}h_y^2-U_{2,3}h_x^2-U_{4,3}h_x^2 &= -h_x^2h_y^2f_{3,3}\\
2\left(h_x^2+h_y^2\right)U_{3,4}-U_{3,3}h_y^2-U_{3,5}h_y^2-U_{2,4}h_x^2-U_{4,4}h_x^2 &= -h_x^2h_y^2f_{3,4}\\
2\left(h_x^2+h_y^2\right)U_{3,5}-U_{3,4}h_y^2+U_{2,5}h_x^2+U_{4,5}h_x^2 &= -h_x^2h_y^2f_{3,4}+U_{3,6}h_y^2\\
2\left(h_x^2+h_y^2\right)U_{4,2}-U_{4,3}h_y^2-U_{3,2}h_x^2 &= -h_x^2h_y^2f_{4,2}+U_{4,1}h_y^2+U_{5,2}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,3}-U_{4,2}h_y^2-U_{4,4}h_y^2-U_{3,3}h_x^2 &= -h_x^2h_y^2f_{4,3}+U_{5,3}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,4}-U_{4,5}h_y^2-U_{4,3}h_y^2-U_{3,4}h_x^2 &= -h_x^2h_y^2f_{4,4}+U_{5,4}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,5}-U_{4,4}h_y^2-U_{3,5}h_x^2 &= -h_x^2h_y^2f_{4,5}+U_{4,6}h_y^2+U_{5,5}h_x^2
\end{aligned}$$

Now, we can write $Ax=b$ as, letting $\beta=2\left(h_x^2+h_y^2\right)$

$$\begin{pmatrix}
\beta & -h_y^2 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
-h_y^2 & \beta & -h_y^2 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & -h_y^2 & \beta & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0\\
-h_x^2 & 0 & 0 & 0 & \beta & -h_y^2 & 0 & 0 & -h_x^2 & 0 & 0 & 0\\
0 & -h_x^2 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & -h_x^2 & 0 & 0\\
0 & 0 & -h_x^2 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & -h_x^2 & 0\\
0 & 0 & 0 & -h_x^2 & 0 & 0 & -h_y^2 & \beta & 0 & 0 & 0 & -h_x^2\\
0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & \beta & -h_y^2 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & -h_y^2 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & -h_y^2 & \beta & -h_y^2\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & -h_y^2 & \beta
\end{pmatrix}
\begin{pmatrix} U_{22}\\ U_{23}\\ U_{24}\\ U_{25}\\ U_{32}\\ U_{33}\\ U_{34}\\ U_{35}\\ U_{42}\\ U_{43}\\ U_{44}\\ U_{45}\end{pmatrix}
=
\begin{pmatrix}
-h_x^2h_y^2f_{2,2}+U_{2,1}h_y^2+U_{1,2}h_x^2\\
-h_x^2h_y^2f_{2,3}+U_{1,3}h_x^2\\
-h_x^2h_y^2f_{2,4}+U_{2,5}h_y^2+U_{1,4}h_x^2\\
-h_x^2h_y^2f_{2,5}+U_{2,6}h_y^2+U_{1,5}h_x^2\\
-h_x^2h_y^2f_{3,2}+U_{3,1}h_y^2\\
-h_x^2h_y^2f_{3,3}\\
-h_x^2h_y^2f_{3,4}+U_{3,5}h_y^2\\
-h_x^2h_y^2f_{3,4}+U_{3,6}h_y^2\\
-h_x^2h_y^2f_{4,2}+U_{4,1}h_y^2+U_{5,2}h_x^2\\
-h_x^2h_y^2f_{4,3}+U_{5,3}h_x^2\\
-h_x^2h_y^2f_{4,4}+U_{4,5}h_y^2+U_{5,4}h_x^2\\
-h_x^2h_y^2f_{4,5}+U_{4,6}h_y^2+U_{5,5}h_x^2
\end{pmatrix}$$

To create the above matrix, as sparse matrix, call the following Matlab code

```
A=lap2d(4,3,hx,hy);
%Where in the above, 4 is nx, which is the number of nodes in the x-
    direction.
%These are internal nodes. ny is number of nodes in the y-direction.
%hx is the spacing between nodes in the x-direction.
%hy is spacing between nodes in the y-direction

function L2 = lap2d(nx,ny,hx,hy)
Lx=lap1dy(nx,hx,hy);  %makes the MAIN block, only one block
Ly=lap1dx(ny,hx,hy);  %makes the off block, only one block

Ix=speye(nx);
Iy=speye(ny);

Lm=kron(Iy,Lx); %does the central diagonal
Lo=kron(Ly,Ix); %does the off diagonal
L2=Lm+Lo;

%------------------------
function L=lap1dy(n,hx,hy)
e=ones(n,1);
T1=2*(hx^2+hy^2);
B=[-e*hy^2 (1/2)*T1*e -e*hy^2];
L=spdiags(B,[-1 0 1],n,n);

%-------------------------------------
%
```

```
%
function L=lap1dx(n,hx,hy)
e=ones(n,1);
T1=2*(hx^2+hy^2);
B=[-e*hx^2 (1/2)*T1*e -e*hx^2];
L=spdiags(B,[-1 0 1],n,n);
```

.

A matrix format for sparse storage for elliptic 2D with non-homegenous Nuemman boundary conditions

This section was added at later time, and not required for this HW.

To be able to forumlate the A matrix as sparse matrix, we need to find what the form will be in the case when one or more of the boundary conditions has Nuemman coditions on it. We will start as above, and start with assuming Nuemman conditions now exist on the left edge and find out what the form of the A matrix will turn out to be.

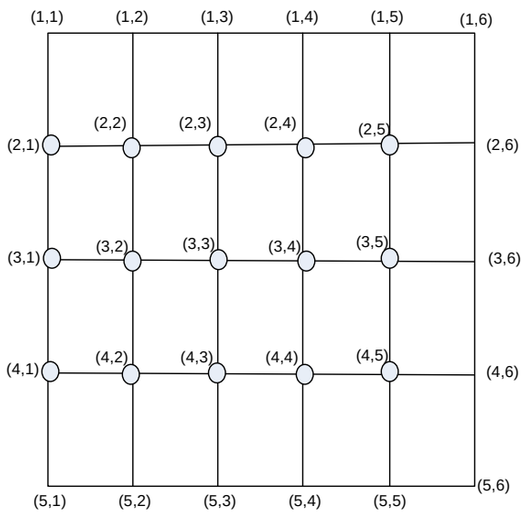

Figure 9: new grid 2 nuemman on left

The unknowns are shown above in the circles. From earlier, we found that the discretization for the elliptic PDE at an internal node is (note that in the above, i is the row number, and j is column number)

$$U_{i,j} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{i,j-1}h_y^2 + U_{i,j+1}h_y^2 + U_{i-1,j}h_x^2 + U_{i+1,j}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{i,j}$$

And for Nuemman, non-homegenouse boundary conditions, the left edge unknowns are given by

$$U_{i,1} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(2U_{i,2}\ h_y^2 + \left(U_{i-1,1} + U_{i+1,1}\right)h_x^2 - h_y^2 h_x g_{i,1}\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{i,1}$$

We will now write the 15 equations for each node, starting with $(2,1)$ to node $(4,5)$

$$U_{2,1} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(2U_{2,2}\ h_y^2 + \left(U_{1,1}+U_{3,1}\right)h_x^2 - h_y^2 h_x g_{2,1}\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{2,1}$$

$$U_{2,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,1}h_y^2 + U_{2,3}h_y^2 + U_{1,2}h_x^2 + U_{3,2}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{2,2}$$

$$U_{2,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,2}h_y^2 + U_{2,4}h_y^2 + U_{1,3}h_x^2 + U_{3,3}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{2,3}$$

$$U_{2,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,3}h_y^2 + U_{2,5}h_y^2 + U_{1,4}h_x^2 + U_{3,4}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{2,4}$$

$$U_{2,5} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,4}h_y^2 + U_{2,6}h_y^2 + U_{1,5}h_x^2 + U_{3,5}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{2,5}$$

$$U_{3,1} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(2U_{3,2}\ h_y^2 + \left(U_{2,1}+U_{4,1}\right)h_x^2 - h_y^2 h_x g_{3,1}\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{3,1}$$

$$U_{3,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,1}h_y^2 + U_{3,3}h_y^2 + U_{2,2}h_x^2 + U_{4,2}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{3,2}$$

$$U_{3,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,2}h_y^2 + U_{3,4}h_y^2 + U_{2,3}h_x^2 + U_{4,3}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{3,3}$$

$$U_{3,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,3}h_y^2 + U_{3,5}h_y^2 + U_{2,4}h_x^2 + U_{4,4}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{3,4}$$

$$U_{3,5} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,4}h_y^2 + U_{3,6}h_y^2 + U_{2,5}h_x^2 + U_{4,5}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{3,4}$$

$$U_{4,1} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(2U_{4,2}\ h_y^2 + \left(U_{3,1}+U_{5,1}\right)h_x^2 - h_y^2 h_x g_{4,1}\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{4,1}$$

$$U_{4,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,1}h_y^2 + U_{4,3}h_y^2 + U_{3,2}h_x^2 + U_{5,2}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{4,2}$$

$$U_{4,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,2}h_y^2 + U_{4,4}h_y^2 + U_{3,3}h_x^2 + U_{5,3}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{4,3}$$

$$U_{4,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,3}h_y^2 + U_{4,5}h_y^2 + U_{3,4}h_x^2 + U_{5,4}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{4,4}$$

$$U_{4,5} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,4}h_y^2 + U_{4,6}h_y^2 + U_{3,5}h_x^2 + U_{5,5}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)} f_{4,5}$$

Now, moving the knowns to the right, results in

$$\begin{aligned}
2\left(h_x^2+h_y^2\right)U_{2,1} - 2U_{2,2}\ h_y^2 - U_{3,1}h_x^2 &= U_{1,1}h_x^2 - h_y^2 h_x g_{2,1} - h_x^2 h_y^2 f_{2,1}\\
2\left(h_x^2+h_y^2\right)U_{2,2} - U_{2,3}h_y^2 - U_{3,2}h_x^2 &= -h_x^2 h_y^2 f_{2,2} + U_{2,1}h_y^2 + U_{1,2}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,3} - U_{2,2}h_y^2 - U_{2,4}h_y^2 - U_{3,3}h_x^2 &= -h_x^2 h_y^2 f_{2,3} + U_{1,3}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,4} - U_{2,3}h_y^2 - U_{2,5}h_y^2 - U_{3,4}h_x^2 &= -h_x^2 h_y^2 f_{2,4} + U_{1,4}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,5} - U_{2,4}h_y^2 + U_{3,5}h_x^2 &= -h_x^2 h_y^2 f_{2,5} + U_{2,6}h_y^2 + U_{1,5}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{3,1} - 2U_{3,2}\ h_y^2 - U_{4,1}h_x^2 - U_{2,1}h_x^2 &= -h_y^2 h_x g_{3,1} - h_x^2 h_y^2 f_{3,1}\\
2\left(h_x^2+h_y^2\right)U_{3,2} - U_{3,3}h_y^2 - U_{2,2}h_x^2 - U_{4,2}h_x^2 &= -h_x^2 h_y^2 f_{3,2} + U_{3,1}h_y^2\\
2\left(h_x^2+h_y^2\right)U_{3,3} - U_{3,2}h_y^2 - U_{3,4}h_y^2 - U_{2,3}h_x^2 - U_{4,3}h_x^2 &= -h_x^2 h_y^2 f_{3,3}\\
2\left(h_x^2+h_y^2\right)U_{3,4} - U_{3,3}h_y^2 - U_{3,5}h_y^2 - U_{2,4}h_x^2 - U_{4,4}h_x^2 &= -h_x^2 h_y^2 f_{3,4}\\
2\left(h_x^2+h_y^2\right)U_{3,5} - U_{3,4}h_y^2 + U_{2,5}h_x^2 + U_{4,5}h_x^2 &= -h_x^2 h_y^2 f_{3,4} + U_{3,6}h_y^2\\
2\left(h_x^2+h_y^2\right)U_{4,1} - 2U_{4,2}\ h_y^2 - U_{3,1}h_x^2 &= U_{5,1}h_x^2 - h_y^2 h_x g_{4,1} - h_x^2 h_y^2 f_{4,1}\\
2\left(h_x^2+h_y^2\right)U_{4,2} - U_{4,3}h_y^2 - U_{3,2}h_x^2 &= -h_x^2 h_y^2 f_{4,2} + U_{4,1}h_y^2 + U_{5,2}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,3} - U_{4,2}h_y^2 - U_{4,4}h_y^2 - U_{3,3}h_x^2 &= -h_x^2 h_y^2 f_{4,3} + U_{5,3}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,4} - U_{4,5}h_y^2 - U_{4,3}h_y^2 - U_{3,4}h_x^2 &= -h_x^2 h_y^2 f_{4,4} + U_{5,4}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,5} - U_{4,4}h_y^2 - U_{3,5}h_x^2 &= -h_x^2 h_y^2 f_{4,5} + U_{4,6}h_y^2 + U_{5,5}h_x^2
\end{aligned}$$

Now, we can write $Ax = b$ as, letting $\beta = 2\left(h_x^2+h_y^2\right)$

$$\begin{pmatrix}
\beta & -2h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & -h_y^2 & \beta & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0\\
-h_x^2 & 0 & 0 & 0 & 0 & \beta & -2h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0\\
0 & -h_x^2 & 0 & 0 & 0 & 0 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0\\
0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0\\
0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2\\
0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & \beta & -2h_y^2 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & \beta & -h_y^2 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_y^2 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2
\end{pmatrix}
\begin{pmatrix}
U_{21}\\ U_{22}\\ U_{23}\\ U_{24}\\ U_{25}\\ U_{31}\\ U_{32}\\ U_{33}\\ U_{34}\\ U_{35}\\ U_{41}\\ U_{42}\\ U_{43}\\ U_{44}\\ U_{45}
\end{pmatrix}
=
\begin{pmatrix}
U_{1,1}h_x^2 - h_y^2 h_x g_{2,1} - h_x^2 h_y^2 f_{2,1}\\
-h_x^2 h_y^2 f_{2,2} + U_{2,1}h_y^2 + U_{1,2}h_x^2\\
-h_x^2 h_y^2 f_{2,3} + U_{1,3}h_x^2\\
-h_x^2 h_y^2 f_{2,4} + U_{2,5}h_y^2 + U_{1,4}h_x^2\\
-h_x^2 h_y^2 f_{2,5} + U_{2,6}h_y^2 + U_{1,5}h_x^2\\
-h_y^2 h_x g_{3,1} - h_x^2 h_y^2 f_{3,1}\\
-h_x^2 h_y^2 f_{3,2} + U_{3,1}h_y^2\\
-h_x^2 h_y^2 f_{3,3}\\
-h_x^2 h_y^2 f_{3,4} + U_{3,5}h_y^2\\
-h_x^2 h_y^2 f_{3,4} + U_{3,6}h_y^2\\
U_{5,1}h_x^2 - h_y^2 h_x g_{4,1} - h_x^2 h_y^2 f_{4,1}\\
-h_x^2 h_y^2 f_{4,2} + U_{4,1}h_y^2 + U_{5,2}h_x^2\\
-h_x^2 h_y^2 f_{4,3} + U_{5,3}h_x^2\\
-h_x^2 h_y^2 f_{4,4} + U_{4,5}h_y^2 + U_{5,4}h_x^2\\
-h_x^2 h_y^2 f_{4,5} + U_{4,6}h_y^2 + U_{5,5}h_x^2
\end{pmatrix}$$

The above is the matrix for the case of non-homogeneous Neumann boundary conditions on the left edge.

We will now do the same edge, but with homogeneous boundary conditions to see the difference. Recall, that when the edge is insulated, then

$$U_{i,1} = \frac{h_x h_y}{\left(h_x^2+h_y^2\right)}\left(\frac{h_x}{2h_y}\left(U_{i-1,1}+U_{i+1,1}\right) - \frac{h_y}{h_x}U_{i,2}\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,1}$$

Using the above, we write the 15 equations starting with (2,1) to node (4,5)

$$U_{2,1} = \frac{h_x h_y}{\left(h_x^2 + h_y^2\right)}\left(\frac{h_x}{2h_y}\left(U_{1,1} + U_{3,1}\right) - \frac{h_y}{h_x}U_{2,2}\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{2,1}$$

$$U_{2,2} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{2,1}h_y^2 + U_{2,3}h_y^2 + U_{1,2}h_x^2 + U_{3,2}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{2,2}$$

$$U_{2,3} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{2,2}h_y^2 + U_{2,4}h_y^2 + U_{1,3}h_x^2 + U_{3,3}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{2,3}$$

$$U_{2,4} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{2,3}h_y^2 + U_{2,5}h_y^2 + U_{1,4}h_x^2 + U_{3,4}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{2,4}$$

$$U_{2,5} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{2,4}h_y^2 + U_{2,6}h_y^2 + U_{1,5}h_x^2 + U_{3,5}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{2,5}$$

$$U_{3,1} = \frac{h_x h_y}{\left(h_x^2 + h_y^2\right)}\left(\frac{h_x}{2h_y}\left(U_{2,1} + U_{4,1}\right) - \frac{h_y}{h_x}U_{3,2}\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{3,1}$$

$$U_{3,2} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{3,1}h_y^2 + U_{3,3}h_y^2 + U_{2,2}h_x^2 + U_{4,2}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{3,2}$$

$$U_{3,3} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{3,2}h_y^2 + U_{3,4}h_y^2 + U_{2,3}h_x^2 + U_{4,3}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{3,3}$$

$$U_{3,4} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{3,3}h_y^2 + U_{3,5}h_y^2 + U_{2,4}h_x^2 + U_{4,4}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{3,4}$$

$$U_{3,5} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{3,4}h_y^2 + U_{3,6}h_y^2 + U_{2,5}h_x^2 + U_{4,5}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{3,4}$$

$$U_{4,1} = \frac{h_x h_y}{\left(h_x^2 + h_y^2\right)}\left(\frac{h_x}{2h_y}\left(U_{3,1} + U_{5,1}\right) - \frac{h_y}{h_x}U_{4,2}\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{4,1}$$

$$U_{4,2} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{4,1}h_y^2 + U_{4,3}h_y^2 + U_{3,2}h_x^2 + U_{5,2}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{4,2}$$

$$U_{4,3} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{4,2}h_y^2 + U_{4,4}h_y^2 + U_{3,3}h_x^2 + U_{5,3}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{4,3}$$

$$U_{4,4} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{4,3}h_y^2 + U_{4,5}h_y^2 + U_{3,4}h_x^2 + U_{5,4}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{4,4}$$

$$U_{4,5} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{4,4}h_y^2 + U_{4,6}h_y^2 + U_{3,5}h_x^2 + U_{5,5}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)}f_{4,5}$$

Now, moving the knowns to the right, results in

$$\frac{\left(h_x^2+h_y^2\right)}{h_x h_y}U_{2,1}-\frac{h_x}{2h_y}U_{3,1}+\frac{h_y}{h_x}U_{2,2}=\frac{h_x}{2h_y}U_{1,1}-h_x h_y f_{2,1}$$
$$2\left(h_x^2+h_y^2\right)U_{2,2}-U_{2,3}h_y^2-U_{3,2}h_x^2=-h_x^2h_y^2f_{2,2}+U_{2,1}h_y^2+U_{1,2}h_x^2$$
$$2\left(h_x^2+h_y^2\right)U_{2,3}-U_{2,2}h_y^2-U_{2,4}h_y^2-U_{3,3}h_x^2=-h_x^2h_y^2f_{2,3}+U_{1,3}h_x^2$$
$$2\left(h_x^2+h_y^2\right)U_{2,4}-U_{2,3}h_y^2-U_{2,5}h_y^2-U_{3,4}h_x^2=-h_x^2h_y^2f_{2,4}+U_{1,4}h_x^2$$
$$2\left(h_x^2+h_y^2\right)U_{2,5}-U_{2,4}h_y^2+U_{3,5}h_x^2=-h_x^2h_y^2f_{2,5}+U_{2,6}h_y^2+U_{1,5}h_x^2$$
$$\frac{\left(h_x^2+h_y^2\right)}{h_x h_y}U_{3,1}-\frac{h_x}{2h_y}U_{2,1}-\frac{h_x}{2h_y}U_{4,1}+\frac{h_y}{h_x}U_{3,2}=-h_x h_y f_{3,1}$$
$$2\left(h_x^2+h_y^2\right)U_{3,2}-U_{3,3}h_y^2-U_{2,2}h_x^2-U_{4,2}h_x^2=-h_x^2h_y^2f_{3,2}+U_{3,1}h_y^2$$
$$2\left(h_x^2+h_y^2\right)U_{3,3}-U_{3,2}h_y^2-U_{3,4}h_y^2-U_{2,3}h_x^2-U_{4,3}h_x^2=-h_x^2h_y^2f_{3,3}$$
$$2\left(h_x^2+h_y^2\right)U_{3,4}-U_{3,3}h_y^2-U_{3,5}h_y^2-U_{2,4}h_x^2-U_{4,4}h_x^2=-h_x^2h_y^2f_{3,4}$$
$$2\left(h_x^2+h_y^2\right)U_{3,5}-U_{3,4}h_y^2+U_{2,5}h_x^2+U_{4,5}h_x^2=-h_x^2h_y^2f_{3,4}+U_{3,6}h_y^2$$
$$\frac{\left(h_x^2+h_y^2\right)}{h_x h_y}U_{4,1}-\frac{h_x}{2h_y}U_{3,1}+\frac{h_y}{h_x}U_{4,2}=\frac{h_x}{2h_y}U_{5,1}-h_x h_y f_{4,1}$$
$$2\left(h_x^2+h_y^2\right)U_{4,2}-U_{4,3}h_y^2-U_{3,2}h_x^2=-h_x^2h_y^2f_{4,2}+U_{4,1}h_y^2+U_{5,2}h_x^2$$
$$2\left(h_x^2+h_y^2\right)U_{4,3}-U_{4,2}h_y^2-U_{4,4}h_y^2-U_{3,3}h_x^2=-h_x^2h_y^2f_{4,3}+U_{5,3}h_x^2$$
$$2\left(h_x^2+h_y^2\right)U_{4,4}-U_{4,5}h_y^2-U_{4,3}h_y^2-U_{3,4}h_x^2=-h_x^2h_y^2f_{4,4}+U_{5,4}h_x^2$$
$$2\left(h_x^2+h_y^2\right)U_{4,5}-U_{4,4}h_y^2-U_{3,5}h_x^2=-h_x^2h_y^2f_{4,5}+U_{4,6}h_y^2+U_{5,5}h_x^2$$

Now, we can write $Ax=b$ as, letting $\beta=2\left(h_x^2+h_y^2\right)$ and $\gamma=\frac{\left(h_x^2+h_y^2\right)}{h_x h_y}$

$$\begin{pmatrix}
\gamma & \frac{h_y}{h_x} & 0 & 0 & 0 & -\frac{h_x}{2h_y} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & -h_y^2 & \beta & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0\\
-\frac{h_x}{2h_y} & 0 & 0 & 0 & 0 & \gamma & \frac{h_y}{h_x} & 0 & 0 & 0 & -\frac{h_x}{2h_y} & 0 & 0 & 0 & 0\\
0 & -h_x^2 & 0 & 0 & 0 & 0 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0\\
0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0\\
0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0\\
0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & 0 & 0 & 0 & 0 & -h_x^2\\
0 & 0 & 0 & 0 & 0 & -\frac{h_x}{2h_y} & 0 & 0 & 0 & 0 & \gamma & \frac{h_y}{h_x} & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & \beta & -h_y^2 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_y^2 & 0 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & -h_y^2\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta
\end{pmatrix}
\begin{pmatrix}
U_{21}\\U_{22}\\U_{23}\\U_{24}\\U_{25}\\U_{31}\\U_{32}\\U_{33}\\U_{34}\\U_{35}\\U_{41}\\U_{42}\\U_{43}\\U_{44}\\U_{45}
\end{pmatrix}=
\begin{pmatrix}
\frac{h_x}{2h_y}U_{1,1}-h_x h_y f_{2,1}\\
-h_x^2h_y^2f_{2,2}+U_{2,1}h_y^2+U_{1,2}h_x^2\\
-h_x^2h_y^2f_{2,3}+U_{1,3}h_x^2\\
-h_x^2h_y^2f_{2,4}+U_{2,5}h_y^2+U_{1,4}h_x^2\\
-h_x^2h_y^2f_{2,5}+U_{2,6}h_y^2+U_{1,5}h_x^2\\
-h_x h_y f_{3,1}\\
-h_x^2h_y^2f_{3,2}+U_{3,1}h_y^2\\
-h_x^2h_y^2f_{3,3}\\
-h_x^2h_y^2f_{3,4}+U_{3,5}h_y^2\\
-h_x^2h_y^2f_{3,4}+U_{3,6}h_y^2\\
\frac{h_x}{2h_y}U_{5,1}-h_x h_y f_{4,1}\\
-h_x^2h_y^2f_{4,2}+U_{4,1}h_y^2+U_{5,2}h_x^2\\
-h_x^2h_y^2f_{4,3}+U_{5,3}h_x^2\\
-h_x^2h_y^2f_{4,4}+U_{4,5}h_y^2+U_{5,4}h_x^2\\
-h_x^2h_y^2f_{4,5}+U_{4,6}h_y^2+U_{5,5}h_x^2
\end{pmatrix}$$

Storage requirements for the different solvers

The total number of grid points along the x or the y direction is given by $\frac{length}{h}+1$, where length is always 1, and hence n, the number of unknowns in one dimension is 2 less than the above number (since U is known at the boundaries).

It is important to note that the matrix A is not used explicitly to solve for the unknowns in the iterative schemes. Storage is needed only for the internal grid points, which is the number of the unknowns n^2. An auxiliary grid is used to hold the updated values in the Jacobian method. In addition, an auxiliary grid is required for $f_{i,j}$, the force function. Hence, in total $3n^2$ storage is needed.

Comparing this to the storage needed in the case of direct solver, where the storage for A alone is n^4. This shows the main advantage of the iterative methods compared to the direct method when A is a dense matrix. (Use of sparse matrix become necessary if direct solver is to be used for large n problems).

The following table summarizes the above discussion for the h values given in this problem. In this calculations, double precision (8 bytes) per grid point is assumed. For single precision half of this storage would be needed.

h	n	number of unknowns n^2	storage	size of A n^4	storage for dense A
2^{-5}	30	900	30 (k Bytes)	810,000	6.4 MBytes
2^{-6}	62	3844	0.1 (MB)	14,776,336	118 MBytes
2^{-7}	126	15 876	0.5 (MB)	252,047,376	2 GB

Loop algorithm for each method

This is a short description of the algorithm used by each method. In the following $u_{i,j}^{new}$ represents the new value (residing on the auxiliary grid) of the unknown, and $u_{i,j}^{current}$ is the current value. Initially $u^{current}$ is set to zero at each internal grid point. (any other initial value could also have been used).

Jacobi method algorithm

$k := 0$ $(*counter*)$
$\epsilon := Ch^2$ $(*tolerance*)$
$u^{current} := u^{new} := 0$ $(*initialize\ storage*)$
$f := f(x,y)$ $(*initialize\ f\ on\ grid\ points*)$
$CONVERGED := false$
$WHILE\ NOT\ CONVERGED$
  $LOOP\ i$
    $LOOP\ j$
      $u_{i,j}^{new} := \frac{1}{4}\left(u_{i-1,j}^{current} + u_{i+1,j}^{current} + u_{i,j-1}^{current} + u_{i,j+1}^{current} - h^2 f_{i,j}\right)$
      $R_{i,j} = f_{i,j} - \frac{1}{h^2}\left(u_{i-1,j}^{current} + u_{i+1,j}^{current} + u_{i,j-1}^{current} + u_{i,j+1}^{current} - 4u_{i,j}^{current}\right)$ $(*residual*)$
    $END\ LOOP\ j$
  $END\ LOOP\ i$
  $u^{current} := u^{new}$ $(*update*)$
  $k := k+1$
  $IF\ \left(\frac{\|R\|}{\|f\|} < \epsilon\right)\ THEN$ $(*Norms\ are\ grid\ norms.\ see\ code*)$
    $CONVERGED := true$
  $END\ IF$
$END\ WHILE$

Gauss-Seidel method

$k := 0$ (*counter*)
$\epsilon := Ch^2$ (*tolerance*)
$u^{current} := u^{new} := 0$ (*initialize storage*)
$f := f(x,y)$ (*initialize f on grid points*)
CONVERGED $:= false$
WHILE NOT CONVERGED
    LOOP i
        LOOP j
            $u_{ij}^{new} := \frac{1}{4}\left(u_{i-1,j}^{new} + u_{i+1,j}^{current} + u_{i,j-1}^{new} + u_{i,j+1}^{current} - h^2 f_{i,j}\right)$
            $R_{i,j} = f_{i,j} - \frac{1}{h^2}\left(u_{i-1,j}^{current} + u_{i+1,j}^{current} + u_{i,j-1}^{current} + u_{i,j+1}^{current} - 4u_{i,j}^{current}\right)$ (*residual*)
        END LOOP j
    END LOOP i
    $u^{current} := u^{new}$ (*update*)
    $k := k + 1$
    IF $\left(\frac{\|R\|}{\|f\|} < \epsilon\right)$ *THEN* (*Norms are grid norms. see code*)
        CONVERGED $:= true$
    END IF
END WHILE

SOR method

$k := 0$ (*counter*)
$\epsilon := Ch^2$ (*tolerance*)
$u^{current} := u^{new} := 0$ (*initialize storage*)
$f := f(x,y)$ (*initialize f on grid points*)
CONVERGED $:= false$
WHILE NOT CONVERGED
    LOOP i
        LOOP j
            $u_{ij}^{new} := \frac{\omega}{4}\left(u_{i-1,j}^{new} + u_{i+1,j}^{current} + u_{i,j-1}^{new} + u_{i,j+1}^{current} - h^2 f_{i,j}\right) + (1-\omega)\, u_{i,j}^{current}$
            $R_{i,j} = f_{i,j} - \frac{1}{h^2}\left(u_{i-1,j}^{current} + u_{i+1,j}^{current} + u_{i,j-1}^{current} + u_{i,j+1}^{current} - 4u_{i,j}^{current}\right)$ (*residual*)
        END LOOP j
    END LOOP i
    $u^{current} := u^{new}$ (*update*)
    $k := k + 1$
    IF $\left(\frac{\|R\|}{\|f\|} < \epsilon\right)$ *THEN* (*Norms are grid norms. see code*)
        CONVERGED $:= true$
    END IF
END WHILE

Notice that when $\omega = 1$, SOR becomes the same as *Gauss – Seidel*. When $\omega > 1$ the method is called overrelaxed and when $\omega < 1$ it is called underrelaxed.

0.4.7 Result of computation

The above 3 algorithms where implemented and ran for the 3 values of h given. The following tables summarizes the results obtained. The source code is in the appendix. This is a diagram illustrating the final converged solution from one of the runs.

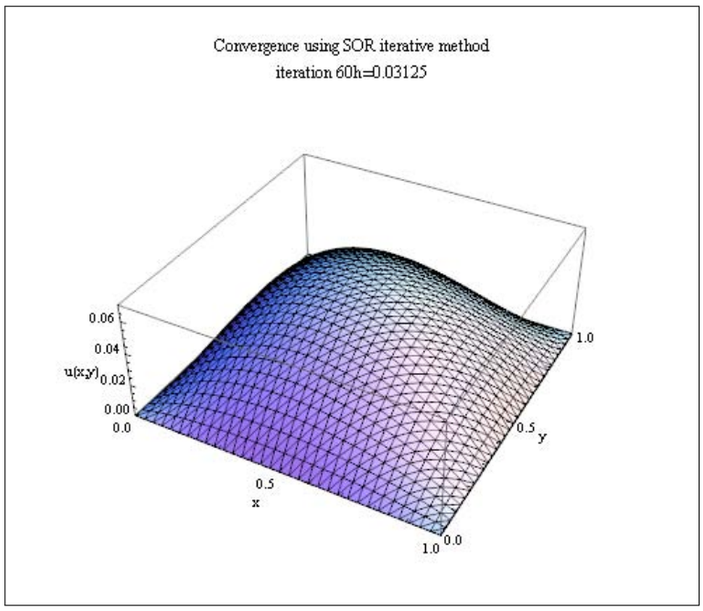

Figure 10: final convergedsolution

Number of steps to reach convergence

These table show the number of iterations to converge. The first was based on using $\varepsilon = 0.1h^2$ for tolerance and the second used $\varepsilon = h^2$

Number of iteration to reach convergence using tolerance $\varepsilon = 0.1h^2$

method	$h = 2^{-3}$ $n = 7$	$h = 2^{-4}$ $n = 15$	$h = 2^{-5}$ $n = 31$	$h = 2^{-6}$ $n = 63$	$h = 2^{-7}$ $n = 127$
Jacobi	82	400	1886	8689	note[1]
Gauss-Seidel	43	202	944	3446	19674
SOR	13	26	53	106	215

Number of iteration to reach convergence using tolerance $\varepsilon = h^2$

method	$h = 2^{-3}$ $n = 7$	$h = 2^{-4}$ $n = 15$	$h = 2^{-5}$ $n = 31$	$h = 2^{-6}$ $n = 63$	$h = 2^{-7}$ $n = 127$
Jacobi	52	281	1409	6779	31702
Gauss – Seidel	28	142	706	3391	15852
SOR	10	20	40	80	159

Convergence plots for each method

These error plots where generated only for tolerance $\varepsilon = 1 \times h^2$. They show how the log of the norm of the relative residual changes as the number of iterations changed until convergence is achieved.

In these plots, the *yaxis* is $\log\left(\frac{\|R^{[k]}\|}{\|f\|}\right)$ or $\log\left(\frac{\|f-Au^{[k]}\|}{\|f\|}\right)$, and the *xaxis* is the k value (the iteration number).

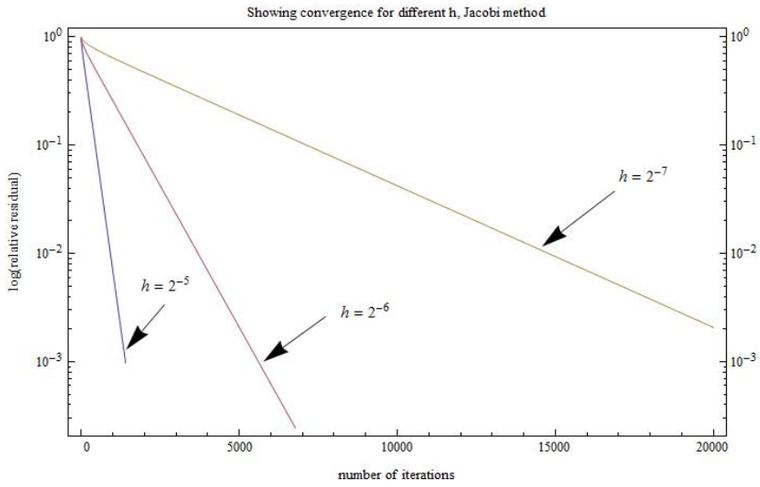

Figure 11: error plot Jacobi

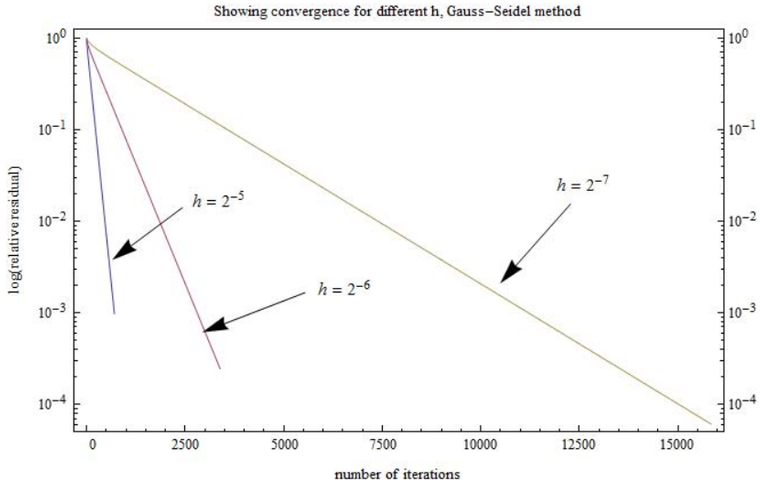

Figure 12: error plot GS

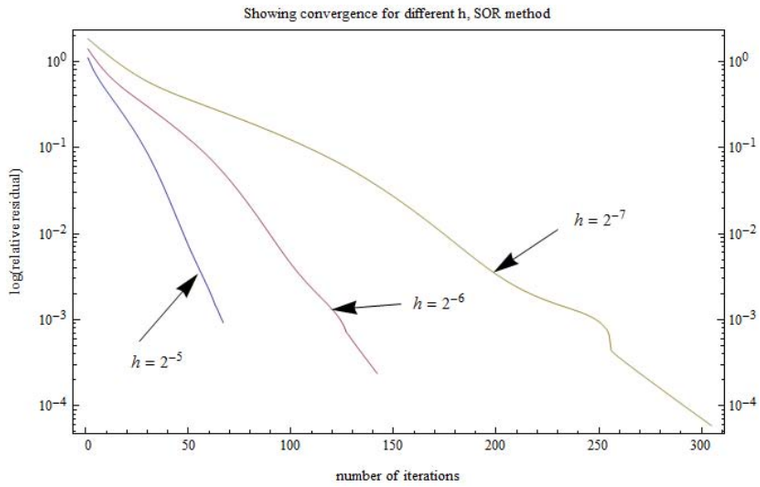

Figure 13: error plot SOR

Plots for comparing convergence of the 3 methods for $\epsilon = 1 \times h^2$ for different h values

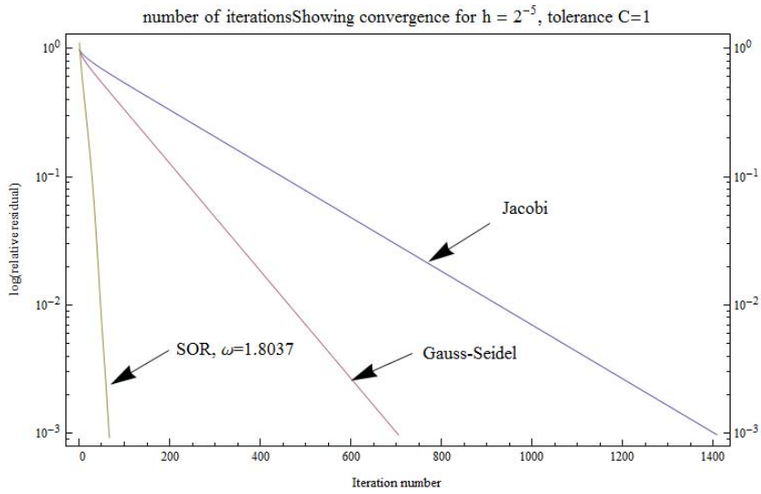

Figure 14: prob1 compare 3h 25

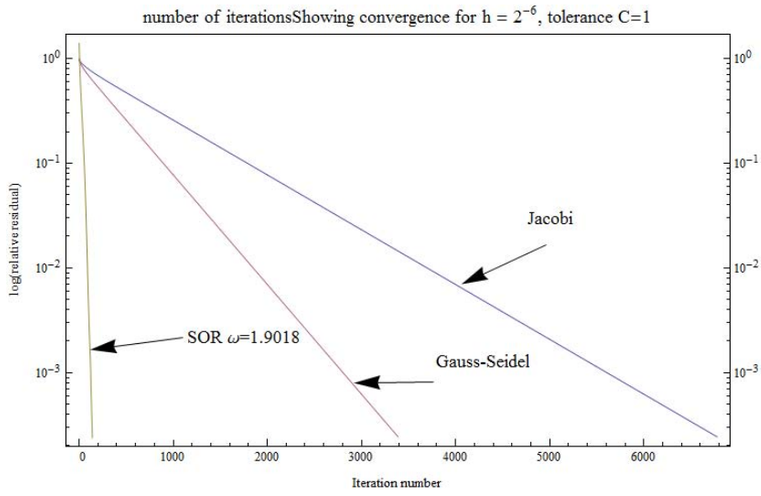

Figure 15: prob1 compare 3h 26

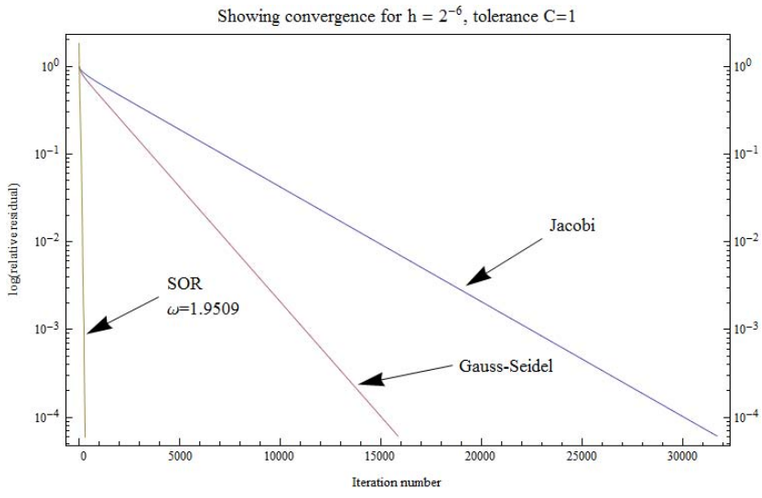

Figure 16: prob1 compare 3h 27

0.4.8 Conclusions and summary

1. SOR is the fastest iterative solver of the three solvers.
2. SOR method required calculation of an optimal ω to use. For this problem, this calculation was not difficult. In other problems it can be difficult to determine before hand the optimal ω. Some SOR methods use an adaptive ω where ω is readjusted as the solution progresses.

3. The use of relative residual to determine the condition of convergence required applying the matrix A without actually storing A.
4. Jacobi method required an additional auxiliary grid storage, hence its memory requirement was twice as much as Gauss-Seidel or SOR.
5. Jacobi method was the simplest to implement, but it was the slowest to converge.
6. Jacobi method is more suitable for use in parallel processing, where each grid point can be updated independent of the grid point next to it. This is not possible with Gauss-Seidel nor SOR due to the dependency of updates on its immediate grid points. However, if a red-black numbering is used, then it would be possible to implement these methods in parallel in 2 stages.
7. All methods are guaranteed to converge eventually, as the spectral radius of the iterative matrix for each method is less than one.

0.5 Problem 2

2. When solving parabolic equations numerically, one frequently needs to solve an equation of the form

$$u - \delta \Delta u = f,$$

where $\delta > 0$. The analysis and numerical methods we have discussed for the Poisson equation can be applied to the above equation. Suppose we are solving the above equation on the unit square with Dirichlet boundary conditions. Use the standard five point stencil for the discrete Laplacian.

(a) Analytically compute the eigenvalues of the Jacobi iteration matrix, and show that the Jacobi iteration converges.

(b) If $h = 10^{-2}$ and $\delta = 10^{-3}$, how many iterations of SOR would it take to reduce the error by a factor of 10^{-6}? How many iterations would it take for the Poisson equation? Use that the spectral radius of SOR is

$$\rho_{\text{sor}} = \omega_{\text{opt}} - 1,$$

where

$$\omega_{\text{opt}} = \frac{2}{1 + \sqrt{1 - \rho_J^2}},$$

and where ρ_J is the spectral radius of Jacobi.

Figure 17: Problem 2

0.5.1 Part(a)

Given

$$u - \delta \Delta u = f$$

And using standard 5 point Laplacian for the approximation of Δ, the above can be written as

$$u - \delta A u = f \tag{1}$$

Where A is the Jacobian matrix for 2D

$$\frac{1}{h^2} \begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & \ddots & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix}$$

Hence (1) becomes

$$(I - \delta A)\, u = f \tag{2}$$

Let

$$B = (I - \delta A) \tag{3}$$

Then (2) becomes

$$Bu = f \tag{3A}$$

To obtain the iterative matrix for the above system, the method of matrix splitting is used. Let $B = M - N$. Equation (3A) becomes

$$(M - N)\, u = f$$
$$Mu = Nu + f$$

M is selected so that it is invertible and such that M^{-1} is easy to compute, the iterative equation results

$$u^{[k+1]} = \left(M^{-1}N\right) u^{[k]} + M^{-1} f$$

Where iterative matrix T_j is

$$T_j = \left(M^{-1}N\right) \tag{3B}$$

For convergence it is required that the spectral radius of T_j be less than one. $\rho\left(T_j\right)$ is the largest eigenvalue of T_j in absolute terms. The largest eigenvalue of T_j is now found as follows.

For the Jacobi method let $M = D$, and $N = L + U$, where D is the diagonal of B, L is the negative of the strictly lower triangle matrix of B and U is negative of the strictly upper triangle matrix of B. (3B) becomes

$$T_j = D^{-1}(L + U)$$

But $B = D - L - U$, hence $L + U = D - B$ and the above becomes

$$T_j = D^{-1}(D - B)$$
$$= I - D^{-1}B \tag{4}$$

Now the spectral radius of T_j is determined. First D^{-1} is found. But first B needs to be determined. From (3)

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} - \frac{\delta}{h^2} \begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & \ddots & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix}$$

$$= \begin{pmatrix} 1 + \frac{4\delta}{h^2} & -\frac{\delta}{h^2} & 0 & -\frac{\delta}{h^2} & 0 & 0 & 0 \\ -\frac{\delta}{h^2} & 1 + \frac{4\delta}{h^2} & -\frac{\delta}{h^2} & 0 & -\frac{\delta}{h^2} & 0 & 0 \\ 0 & -\frac{\delta}{h^2} & 1 + \frac{4\delta}{h^2} & 0 & 0 & -\frac{\delta}{h^2} & 0 \\ -\frac{\delta}{h^2} & 0 & 0 & \ddots & 0 & 0 & -\frac{\delta}{h^2} \\ 0 & -\frac{\delta}{h^2} & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & -\frac{\delta}{h^2} & 0 & -\frac{\delta}{h^2} & 1 + \frac{4\delta}{h^2} & -\frac{\delta}{h^2} \\ 0 & 0 & 0 & -\frac{\delta}{h^2} & 0 & -\frac{\delta}{h^2} & 1 + \frac{4\delta}{h^2} \end{pmatrix}$$

Therefore, D the diagonal matrix of B is easily found

$$D = \begin{pmatrix} 1+\frac{4\delta}{h^2} & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1+\frac{4\delta}{h^2} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1+\frac{4\delta}{h^2} & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1+\frac{4\delta}{h^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1+\frac{4\delta}{h^2} \end{pmatrix}$$

Now that D is known, the eigenvalue μ_{kl} of the iteration matrix T_j shown in (4) can be written down as

$$\begin{aligned} T_j &= I - D^{-1}B \\ &\Rightarrow \\ \mu_{kl} &= 1 - \left(1+\frac{4\delta}{h^2}\right)^{-1} \lambda_{kl} \end{aligned} \tag{5}$$

where λ_{kl} is the eigenvalues of B. But from (3), $B = (I - \delta A)$, hence (5) becomes

$$\mu_{kl} = 1 - \left(1+\frac{4\delta}{h^2}\right)^{-1} (1 - \delta\nu_{kl}) \tag{6}$$

Where ν_{kl} is the eigenvalue of A, the standard Jacobi A matrix for 2D, with eigenvalues given in textbook (page 63, equation 3.15)

$$\nu_{kl} = \frac{2}{h^2}\left(\cos(k\pi h) + \cos(l\pi h) - 2\right)$$

Using this in (6) results in

$$\begin{aligned} \mu_{kl} &= 1 - \left(1+\frac{4\delta}{h^2}\right)^{-1}\left(1 - \frac{2\delta}{h^2}\cos(k\pi h) - \frac{2\delta}{h^2}\cos(l\pi h) + \frac{4\delta}{h^2}\right) \\ &= 1 - \left(\frac{h^2}{h^2+4\delta}\right)\left(1 - \frac{2\delta}{h^2}\cos(k\pi h) - \frac{2\delta}{h^2}\cos(l\pi h) + \frac{4\delta}{h^2}\right) \\ &= 1 - \left(\frac{h^2}{h^2+4\delta}\right) - \left(\frac{4\delta}{h^2+4\delta}\right) + \frac{2\delta}{h^2}\left(\frac{h^2}{h^2+4\delta}\right)\{\cos(k\pi h) + \cos(l\pi h)\} \\ &= \left(\frac{\overbrace{h^2+4\delta-h^2-4\delta}^{=0}}{h^2+4\delta}\right) + \left(\frac{2\delta}{h^2+4\delta}\right)\{\cos(k\pi h) + \cos(l\pi h)\} \\ &= \left(\frac{2\delta}{h^2+4\delta}\right)\{\cos(k\pi h) + \cos(l\pi h)\} \end{aligned}$$

The largest value of the above occurs when $\cos(k\pi h) + \cos(l\pi h)$ is maximum, which is 2. Therefore

$$\rho\left(T_j\right) = \left(\frac{4\delta}{h^2+4\delta}\right)$$

Which is less than one for any $\delta > 0$.

Hence it is now shown that Jacobi iteration converges for this system.

0.5.2 Part (b)

Reducing the error by factor 10^{-6} implies

$$\left\|e^k\right\| < 10^{-6}\left\|e^0\right\| \tag{1}$$

but by definition

$$\left\|e^k\right\| = \rho_{sor}^k \left\|e^0\right\|$$

Hence (1) becomes

$$\rho_{sor}^k < 10^{-6}$$

Where the solution for k at equality is found (rounded to the largest integer if needed). Hence the above becomes, after taking logarithms of both sides

$$\begin{aligned} k \log \rho_{sor} &= -6 \\ k &= \frac{-6}{\log \rho_{sor}} \end{aligned} \tag{2}$$

But

$$\rho_{sor} = \omega_{opt} - 1$$

Where

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho_j^2}}$$

And $\rho_j = \left(\frac{4\delta}{h^2+4\delta}\right)$ from part(a), hence the above becomes

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \left(\frac{4\delta}{h^2+4\delta}\right)^2}}$$

Hence

$$\rho_{sor} = \frac{2}{1 + \sqrt{1 - \left(\frac{4\delta}{h^2+4\delta}\right)^2}} - 1$$

Substituting the numerical values $h = 10^{-2}, \delta = 10^{-3}$ in the above results in

$$\begin{aligned} \rho_{sor} &= \frac{2}{1 + \sqrt{1 - \left(\frac{4\left(10^{-3}\right)}{\left(10^{-2}\right)^2+4\left(10^{-3}\right)}\right)^2}} - 1 \\ &= 0.64 \end{aligned}$$

Therefore, from (2)

$$\begin{aligned} k &= \frac{-6}{\log(0.64)} \\ &= 30.95 \end{aligned}$$

rounding up gives

$$k = 31$$

0.6 Problem 3

3. In this problem we compare the speed of SOR to a direct solve using Gaussian elimination. At the end of this assignment is MATLAB code to form the matrix for the 2D discrete Laplacian. The code for the 3D matrix is similar. Note that with 1 GB of memory, you can handle grids up to about 1000×1000 in 2D and $40 \times 40 \times 40$ in 3D with a direct solve. The range of grids you will explore depends on the amount of memory you have.

   (a) Solve the PDE from problem 1 using a direct solve. Put timing commands in your code and report the time to solve for a range of mesh spacings. Use SOR to solve on the same meshes and report the time and number of iterations. Comment on your results.

   (b) Repeat the previous part in three spatial dimensions for a range of mesh spacings. Change the right side of the equation to be a three dimensional Gaussian. Comment on your results.

Figure 18: Problem 3

0.6.1 Part(a)

To solve the problem using direct solver, the matrix A is constructed (sparse matrix), and the vector f is evaluated using the given function $f\left(x, y\right)$, this results in an $Au = f$ system, which is then solved using a direct solver.

Recall from problem (1) that the A matrix has the following form (as an example, for 3×3 internal grid, or for $h=0.2$)

$$\begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix} \begin{pmatrix} U_{1,1} \\ U_{2,1} \\ U_{3,1} \\ U_{1,2} \\ U_{2,2} \\ U_{3,2} \\ U_{1,3} \\ U_{2,3} \\ U_{3,3} \end{pmatrix} = h^2 \begin{pmatrix} f_{1,1} \\ f_{2,1} \\ f_{3,1} \\ f_{1,2} \\ f_{2,2} \\ f_{3,2} \\ f_{1,3} \\ f_{2,3} \\ f_{3,3} \end{pmatrix}$$

The matrix A is set up, as sparse for the following set of values

h	internal grid size ($n=\frac{1}{h}-1$)
2^{-5}	31×31
2^{-6}	63×63
2^{-7}	127×127
2^{-8}	255×255
2^{-9}	511×511
2^{-10}	1023×1023

A function is written to evaluate $f\left(x,y\right)$ at each of the internal grid points and reshaped to be column vector in the order shown above. Then the direct solver is called.

Next, the SOR solver is used for each of the above spacings. First ω was calculated for each h using $\omega_{opt}\approx2\left(1-\pi h\right)$ resulting in

h	ω_{opt}
2^{-5}	1.803 7
2^{-6}	1.901 8
2^{-7}	1.950 9
2^{-8}	1.975 5
2^{-9}	1.987 7
2^{-10}	1.993 9

Then the SOR solver which was written in problem (1) was called for each of the above cases. The next section shows the results obtained. The source code is in the appendix.

0.6.2 Result of computation

The following is an image of f(x,y) on the grid

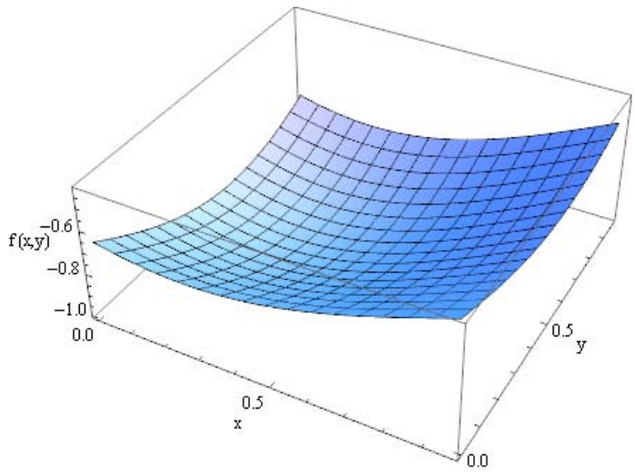

Figure 19: image of $f(x,y)$

And the solution obtained by direct solver on 2D is the following (implemented in Matlab and in Mathematica)

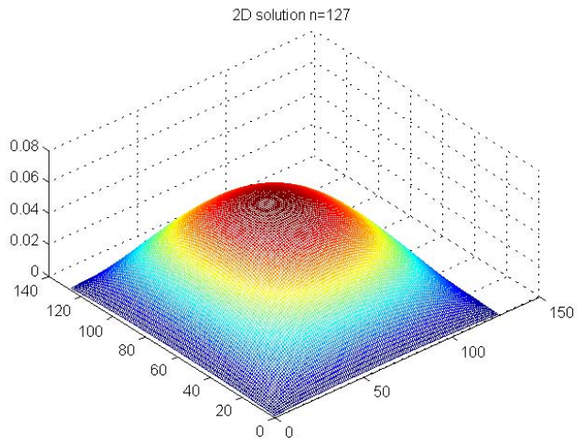

Figure 20: Solution by direct solver, Matlab

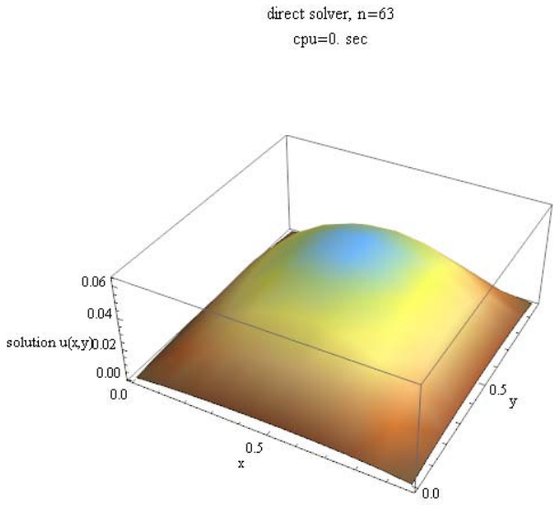

Figure 21: Solution by direct solver using Mathematica

The CPU results below are in seconds. The function cputime() was used to obtain cpu time used in Matlab. For SOR, the cpu time was that of the whole iterative loop until convergence

was achieved. In Mathematica, the command Timing[] which measures the CPU time was used. These are the results obtained using Matlab 2010a and using Mathematica 7.0[2]

In this table, the grid size n represents the number of internal grid points in one dimension. For example, for $n = 31$, the grid size will be 31×31. The number of non zero elements shown in the table relates to storage used by the sparse matrix and was obtained in Matab by calling nnz(A).

h	n	N number of unknowns	number non zero elements	Direct Solver CPU MATLAB	Direct Solver CPU Mathematica	SOR Solver CPU	k SOR number of iterations
2^{-5}	31	961	4,681	0.015	0.016	0	68
2^{-6}	63	3,969	19,593	0.125	0.016	0.094	143
2^{-7}	127	16,129	80,137	0.250	0.063	0.6	306
2^{-8}	255	65,025	324,105	1.544	0.344	5.2	661
2^{-9}	511	261,121	1,303,561	5.538	1.90	48.9	1427
2^{-10}	1023	1,046,529	5,228,553	27.113	14.57	532	3088

These 2 plots illustrate the CPU difference, done on a normal scale and on log scale. (using Matlab results only).

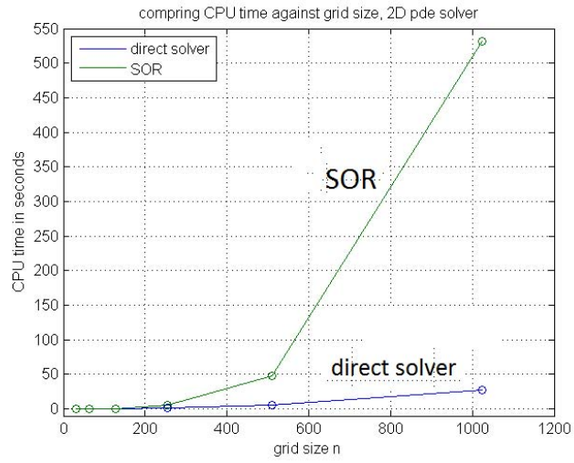

Figure 22: prob3 part a compare CPU normal scale

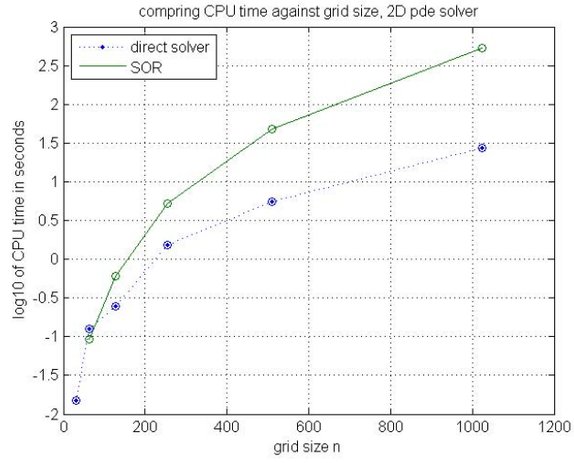

Figure 23: plot prob3 part a compare CPU

Comments on results obtained

[2]Matlab 2010a, on windows 7, 64 bit OS, intel i7 930, with 8 GB installed physical RAM.

CPU performance for SOR is given by

$$work = number\ iterations \times work\ per\ iteration$$

The number of iterations depends on the constant used for tolerance. Let k be the number of iterations, and let the tolerance be Ch^2 where h is the spacings. Hence

$$k = \frac{\log\left(Ch^2\right)}{\log \rho_{sor}} = \frac{\log C + 2\log h}{\log(1-2\pi h)} \approx \frac{\log C + 2\log h}{-2\pi h} \approx O\left(h^{-1}\log h\right)$$

But $h = O\left(\frac{1}{n}\right)$ where n is the number of grid points in one dimension. Therefore

$$k = O(n \log n)$$

And since there are n^2 unknowns, the work per iteration is $O\left(n^2\right)$, hence for SOR performance, work becomes

$$CPU_{sor} = O\left(n^3 \log n\right)$$

Expressing this in terms of the unknowns $N = n^2$ gives

$$CPU_{sor} = O\left(N^{\frac{3}{2}} \log N\right)$$

For direct solver, the work is proprprtional to (Nb) where b is the banwidth (when using nested dissection method)[3]. The bandwidth is n, hence for direct solver on 2D using sparse matrices, the performance is n^3

$$CPU_{direct} = O\left(N^{\frac{3}{2}}\right)$$

In summary

method	CPU (in terms of number of unknowns N)	CPU in terms of n
SOR 2D	$O\left(N^{\frac{3}{2}} \log N\right)$	$O\left(n^3 \log n\right)$
direct solver 2D	$O\left(N^{\frac{3}{2}}\right)$	$O\left(n^3\right)$

For small number of unknowns, SOR was very competitive with direct solver but when the number of unknowns became larger than about $N \approx 100$, the direct solver is faster as the effect of the $\log n$ factor starts to take effect on the performance of SOR. The results shown in the plots above confirmed this performance analysis.

0.6.3 Part(b)

The goal is to solve

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = -\exp(-(x-0.25)^2 - (y-0.6)^2 - z^2)$$

On the unit cube. Referring to the following diagram made to help in setting up the 3D scheme to approximate the above PDE

[3] See textbook, page 68.

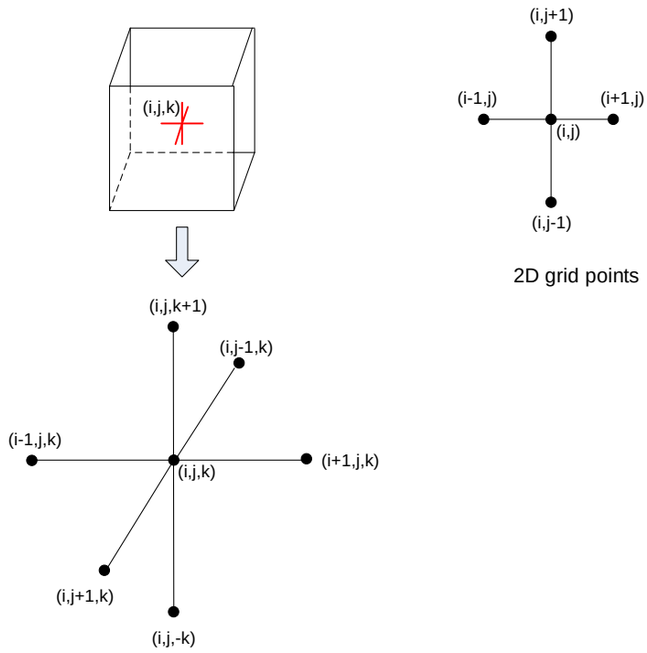

Figure 24: 3D axis

The discrete approximation to the PDE can be written as

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = \frac{1}{h^2}\left(U_{i-1,j,k} + U_{i+1,j,k} + U_{i,j-1}, k + U_{i,j+1,k} + U_{i,k,k-1} + U_{i,j,k+1} - 6U_{i,j,k}\right)$$

Hence the SOR scheme becomes

$$U_{i,j,k}^{[k+1]} = \frac{\omega}{6}\left(U_{i-1,j,k}^{[k+1]} + U_{i+1,j,k}^{[k]} + U_{i,j-1,k}^{[k+1]} + U_{i,j+1,k}^{[k]} + U_{i,j,k-1}^{[k+1]} + U_{i,j,k+1}^{[k]} - h^2 f_{i,j}\right) + (1-\omega)\, U_{i,j,k}^{[k]}$$

For the direct solver, the A matrix needs to be formulated. From

$$\frac{1}{h^2}\left(U_{i-1,j,k} + U_{i+1,j,k} + U_{i,j-1}, k + U_{i,j+1,k} + U_{i,k,k-1} + U_{i,j,k+1} - 6U_{i,j,k}\right) = f_{i,j,k}$$

And solving for $U_{i,j,k}$ results in

$$U_{i,j,k} = \frac{1}{6}\left(U_{i-1,j,k} + U_{i+1,j,k} + U_{i,j-1}, k + U_{i,j+1,k} + U_{i,k,k-1} + U_{i,j,k+1} - h^2 f_{i,j,k}\right)$$

To help make the A matrix, using an example with $n = 2$, the following diagram is made with the standard numbering on each node

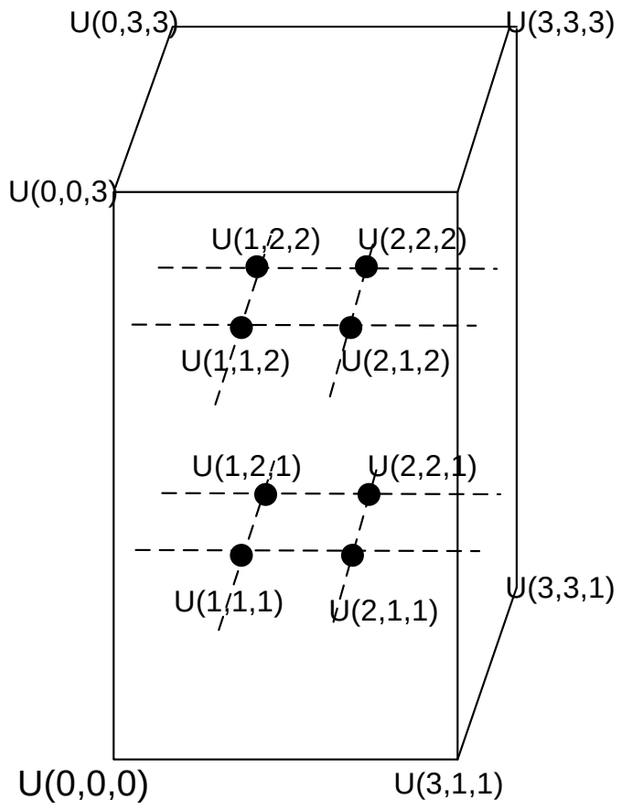

Figure 25: 3d grid example

By traversing the grid, left to right, then inwards into the paper, then upwards, the following A matrix results

$$\begin{pmatrix} -6 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -6 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & -6 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & -6 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -6 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & -6 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & -6 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & -6 \end{pmatrix} \begin{pmatrix} U_{1,1,1} \\ U_{2,1,1} \\ U_{1,2,1} \\ U_{2,2,1} \\ U_{1,1,2} \\ U_{2,1,2} \\ U_{1,2,2} \\ U_{2,2,2} \end{pmatrix} = h^3 \begin{pmatrix} f_{1,1,1} \\ f_{2,1,1} \\ f_{1,2,1} \\ f_{2,2,1} \\ f_{1,1,2} \\ f_{2,1,2} \\ f_{1,2,2} \\ f_{2,2,2} \end{pmatrix}$$

Figure 26: repating A sructure n2

One can see the recursive pattern involved in these A matrices. Each A matrix contains inside it a block on its diagonal which repeats n times. Each block in turn, contain inside it, on its diagonal, smaller block, which also repeats n times.

It is easier to see the pattern of building A by using numbers for the grid points, and label them in the same order as they would be visited, this allowed one to see the connection between each grid point to the other much easier. For example, for $n = 2$,

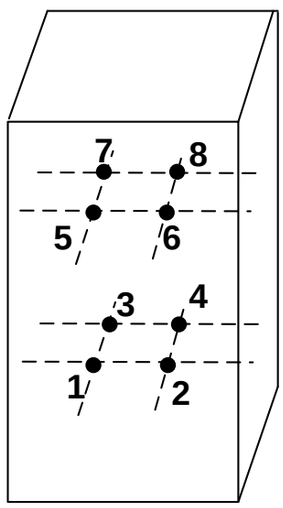

Figure 27: 3d grid n2 numbers

One can see now more easily the connections. grid point 1 has connection to only $2, 3, 5$ points. This means when looking at the A matrix, there will be a 1 in the first row, at columns $2, 3, 5$. Similarly, point 2 has connections only to $1, 4, 6$, which means in the second row, there will be a 1 at columns $1, 4, 6$. Extending the number of points to $n = 3$ to better see the pattern of A results in

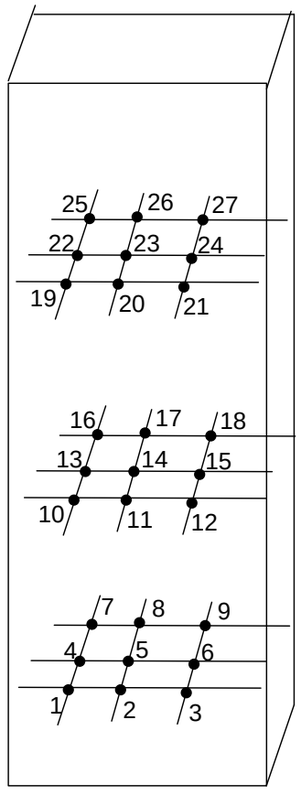

Figure 28: 3d grid n3 numbers

From the above, one can see clearly that, for example, point 1 is connected only to $2, 4, 10$ and 2 is connected to $1, 3, 5, 11$ and so on. The above shows that each point will have a connection to a point which is numbered n^2 higher than the grid point itself. n^2 is the size of the grid in each surface. Hence, the general A matrix, for the above example, can now be written as

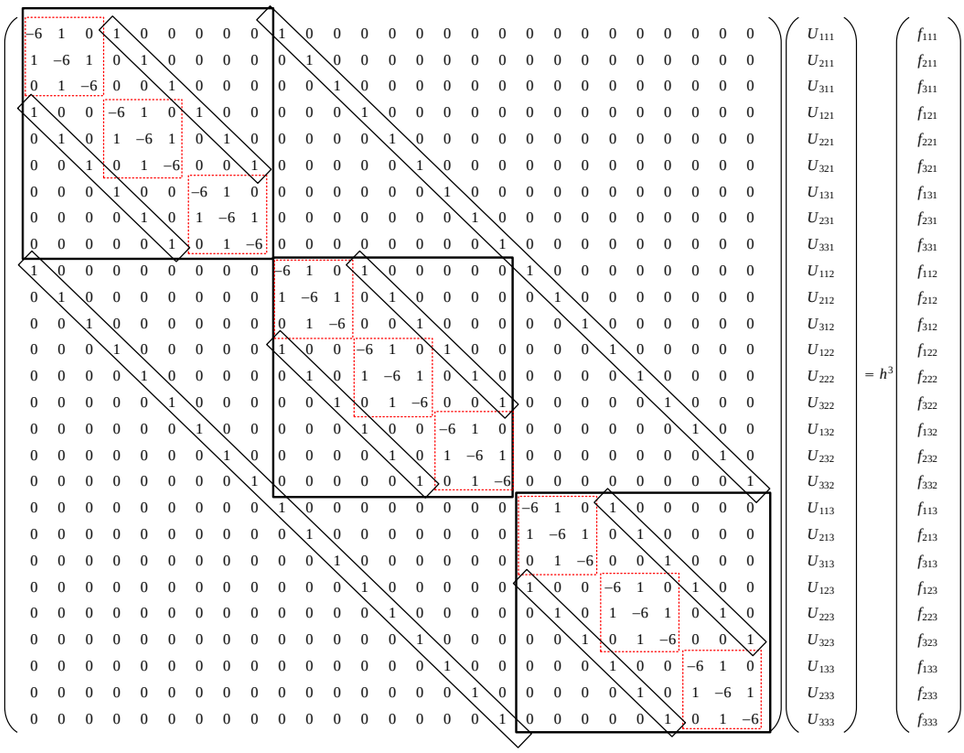

Figure 29: A sructure 3D

One can see the recursive structure again. There are $n = 3$ main repeating blocks on the diagonal, and each one of them in turn has $n = 3$ repeating blocks on its own diagonal. Here $n = 3$, the number of grid points along one dimension.

Now that the A structure is understood, the Matlab code for filling the sparse matrix is modified for the 3D case as follows

```
function L3 = lap3d(n)

L2=lap2d(n,n);
e=ones(n^3,1);
L=spdiags([e e],[-n^2 n^2],n^3,n^3);
Iz=speye(n);

L3=kron(Iz,L2)+L;
end

%-------
function L2 = lap2d(nx,ny)

Lx=lap1d(nx);
Ly=lap1d(ny);

Ix=speye(nx);
Iy=speye(ny);

L2=kron(Iy,Lx)+kron(Ly,Ix);
end

function L=lap1d(n)
e=ones(n,1);
L=spdiags([e -3*e e],[-1 0 1],n,n);
end
```

.

To test, for example, for $n = 2$

```
EDU>> full(lap3d(2))
ans =
    -6     1     1     0     1     0     0     0
     1    -6     0     1     0     1     0     0
     1     0    -6     1     0     0     1     0
     0     1     1    -6     0     0     0     1
     1     0     0     0    -6     1     1     0
     0     1     0     0     1    -6     0     1
     0     0     1     0     1     0    -6     1
     0     0     0     1     0     1     1    -6
```

.

Using the above function, the solution was found using direct solver.

Result of computation

The results for the 3D solver are as follows. In this table, n represents the number of grid points in one dimension. Hence $n = 10$ represents a 3D space of [10,10,10] points. The number of non zero elements in the table relates to the sparse matrix used for the direct solver and was obtained using Matab call nnz(A).

h	n	N total number unknowns (n^3)	make sparse CPU time	A\f CPU time	Total Direct Solver CPU	Total SOR Solver CPU	SOR number iterations
0.090909	10	1,000	0.047	0	0.047	0	23
0.047619	20	8,000	0.062	0.44	0.502	0.078	44
0.032258	30	27,000	0.016	3.90	3.90	0.405	65
0.027778	35	42,875	0.359	8.70	8.80	0.75	77
0.024390	40	64,000	0.328	21.20	21.50	1.29	88
0.021739	45	91,125	0.296	39.80	40.00	2.11	100
0.019608	50	125,000	0.624	84.20	84.80	3.24	112
0.017857	55	166,375	0.421	157.30	157.70	4.9	125
0.016393	60	216,000	0.889	244.10	244.20	7.17	138

For the direct solver, Matlab ran out of memory at $n = 65$ as shown below

```

EDU>> nma_HW3_problem_3_partB_direct_solver
.....

******************************
grid is 3D [60,60,60]
h=0.016393
cpu time for making sparse matrix=1.060807 seconds
dimensions of A (sparse matrix) is [216000,216000]
nnz(A)= 1490400
cpu time for direct solver=240.693943 seconds
******************************
grid is 3D [65,65,65]
h=0.015152
cpu time for making sparse matrix=0.826805 seconds
dimensions of A (sparse matrix) is [274625,274625]
nnz(A)= 1897025
??? Error using ==> mldivide
Out of memory. Type HELP MEMORY for your options.

Error in ==> nma_HW3_problem_3_partB_direct_solver at 54
    u = A\f;
```

.

This plot illustrates the CPU difference table on a log scale

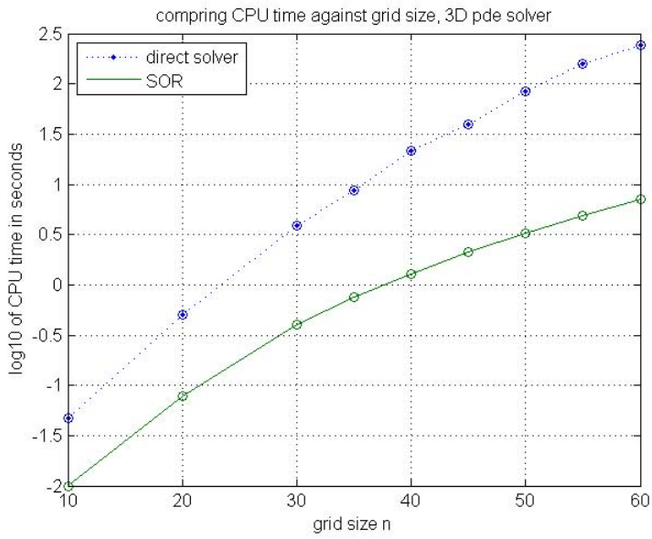

Figure 30: prob3 part B compare CPU log scale

Comments on results obtained

CPU performance for SOR is given by

$$\text{work} = \text{number iterations} \times \text{work per iteration}$$

The number of iterations depends on the constant used for tolerance. Let k be the number of iterations, and let the tolerance be Ch^2 where h is the spacings. Hence

$$k = \frac{\log\left(Ch^2\right)}{\log \rho_{sor}} = \frac{\log C + 2\log h}{\log\left(1 - 2\pi h\right)} \approx \frac{\log C + 2\log h}{-2\pi h} \approx O\left(h^{-1}\log h\right)$$

But $h = O\left(\frac{1}{n}\right)$ where n is the number of grid points in one dimension. Hence

$$k = O\left(n \log n\right)$$

And since there are n^3 unknowns (compared to n^2 in 2D), then work per iteration is $O\left(n^3\right)$, hence for SOR performance becomes

$$CPU_{sor} = O\left(n^4 \log n\right)$$

Expressing this in terms of $N = n^3$ as the number of unknowns, gives

$$CPU_{sor} = O\left(N^{\frac{4}{3}} \log N\right)$$

For direct solver, the work is proprprtional to (Nb) where b is the banwidth (when using nested dissection method)[4]. The bandwidth is n^2 in this case and not n as was with 2D. Hence the total cost is

$$\begin{aligned} CPU_{direct} &= O\left(N \times n\right) \\ &= O\left(n^5\right) \\ &= O\left(N^{\frac{5}{3}}\right) \end{aligned}$$

Hence

method	CPU (in terms of N)	CPU in terms of n
SOR 3D	$O\left(N^{\frac{4}{3}} \log N\right)$	$O\left(n^4 \log n\right)$
direct solver 3D	$O\left(N^{\frac{5}{3}}\right)$	$O\left(n^5\right)$

[4]See textbook, page 68.

The above shows that SOR is faster than direct solver performance. The results shown in the plots above confirmed this analytical performance prediction showing SOR to be faster. To verify the above, a plot was made using the above complexity cost measure to determine if the resulting shape matches the one obtained from the actual runs above. The following plot shows the complexity cost made on sequence of values that represent n

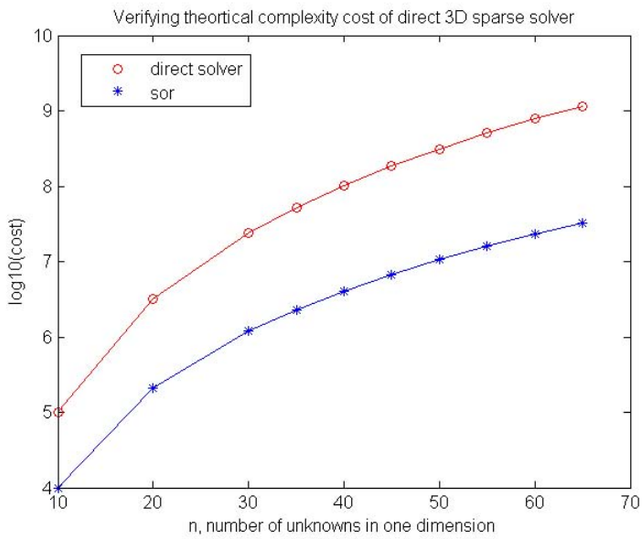

Figure 31: verify cost of 3D

The following matlab code was in part used to generate the above

```
n=[10 20 30 35 40 45 50 55 60 65];
plot(n,log10(n.^5),'ro',n,log10(n.^4.*log10(n)),'*');
```

It can be seen that the cost curves matches those produced with the actual runs, but for a scaling factor as can be expected.

Therefore one can conclude that in 3D SOR is faster than direct solver. This result was surprising as the expectation was that the direct solver will be faster than SOR in 3D as it was in 2D. Attempts were made to find any errors in the code that can explain this, and none were found.

0.7 Problem 4

3. In this problem we compare the speed of SOR to a direct solve using Gaussian elimination. At the end of this assignment is MATLAB code to form the matrix for the 2D discrete Laplacian. The code for the 3D matrix is similar. Note that with 1 GB of memory, you can handle grids up to about 1000×1000 in 2D and $40 \times 40 \times 40$ in 3D with a direct solve. The range of grids you will explore depends on the amount of memory you have.

   (a) Solve the PDE from problem 1 using a direct solve. Put timing commands in your code and report the time to solve for a range of mesh spacings. Use SOR to solve on the same meshes and report the time and number of iterations. Comment on your results.

   (b) Repeat the previous part in three spatial dimensions for a range of mesh spacings. Change the right side of the equation to be a three dimensional Gaussian. Comment on your results.

Figure 32: Problem 3

Periodic boundary conditions mean that the solution must be such that $u'(0) = u'(1)$ and $u(0) = u(1)$. As an example, the following is a solution to $u''(x) = f(x)$ with Periodic boundary conditions just for illustration

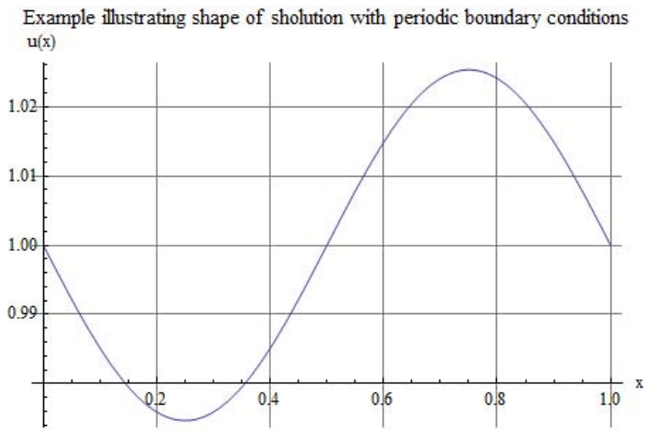

Figure 33: example periodic BC

0.7.1 part(a)

Using the standard numbering system

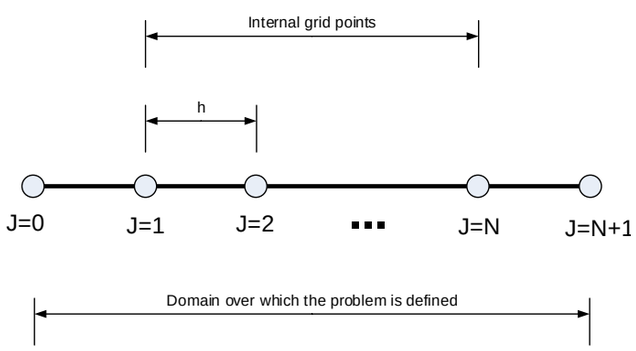

Figure 34: problem 4 part a scheme

In the above diagram, u_0 represents u at $x = 0$ and u_{N+1} represents u at $x = 1$. The 3 point discrete Laplacian for 1-D at x_0 is given by

$$u_0'' = \frac{u_{-1} - 2u_0 + u_1}{h^2} \tag{1}$$

where x_{-1} is an imaginary grid point to the left of x_0 in the diagram above.

Expanding u_{-1} about u_0 by Taylor results in $u_{-1} = u_0 - hu_0'$, hence

$$u_0' = \frac{u_0 - u_{-1}}{h} \tag{2}$$

Similarly, by Taylor expansion of u_N about u_{N+1} results in

$$u_N = u_{N+1} - hu_{N+1}'$$

Hence

$$u_{N+1}' = \frac{u_{N+1} - u_N}{h} \tag{3}$$

But $u_0' = u_{N+1}'$ from boundary conditions, hence (2)=(3) which results in

$$\frac{u_0 - u_{-1}}{h} = \frac{u_{N+1} - u_N}{h}$$

Solving now for u_{-1} from the above gives

$$u_{-1} = u_0 + u_N - u_{N+1}$$

But $u_{N+1} = u_0$, also from the boundary conditions, hence the above results in

$$u_{-1} = u_N$$

Use the above value of u_{-1} in (1) gives

$$u_0'' = \frac{u_N - 2u_0 + u_1}{h^2}$$

Similarly the derivation for u''_{N+1} results in

$$u''_{N+1} = \frac{u_N - 2u_{N+1} + u_1}{h^2}$$

For every other internal grid point $i = 1 \cdots N$ the standard 3 point central difference is used

$$u''_i = \frac{1}{h^2}(u_{i-1} - 2u_i + u_{i+1})$$

Therefore, the following set of equations are obtained

$$\begin{aligned}
&\tfrac{1}{h^2}(u_N - 2u_0 + u_1) = f_0 && i = 0\\
&\tfrac{1}{h^2}(u_{i-1} - 2u_i + u_{i+1}) = f_i && i = 1 \cdots N\\
&\tfrac{1}{h^2}(u_N - 2u_{N+1} + u_1) = f_{N+1} && i = N+1
\end{aligned}$$

And the system can now be put in the form $Au = f$ resulting in

$$\frac{1}{h^2}\begin{pmatrix}
-2 & 1 & 0 & 0 & \cdots & 1 & 0\\
1 & -2 & 1 & 0 & \cdots & 0 & 0\\
0 & 1 & -2 & 1 & 0 & \cdots & 0\\
0 & 0 & 1 & -2 & 1 & 0 & \vdots\\
0 & 0 & 0 & \cdots & \ddots & \cdots & 0\\
0 & 0 & 0 & \cdots & 1 & -2 & 1\\
0 & 1 & 0 & \cdots & 0 & 1 & -2
\end{pmatrix}\begin{pmatrix}
u_0\\ u_1\\ u_2\\ u_3\\ \vdots\\ u_N\\ u_{N+1}
\end{pmatrix} = \begin{pmatrix}
f_0\\ f_1\\ f_2\\ f_3\\ \vdots\\ f_N\\ f_{N+1}
\end{pmatrix} \tag{4}$$

The above A matrix is singular since $A\boldsymbol{b} = 0$ for $\boldsymbol{b}$ the vector 1^T. Hence the null space of A contains a vector other than the 0 vector meaning that A is singular.

To determine the dimension of the null space, the rank of A is determined. Removing the last column and the last row of A results in an $n-1$ by $n-1$ matrix

$$A_{n-1} = \begin{pmatrix}
-2 & 1 & 0 & 0 & \cdots & 1\\
1 & -2 & 1 & 0 & \cdots & 0\\
0 & 1 & \ddots & 1 & 0 & \cdots\\
0 & 0 & 1 & -2 & 1 & 0\\
0 & 0 & 0 & 1 & -2 & 1\\
0 & 0 & 0 & \cdots & 1 & -2
\end{pmatrix}$$

The square matrix inside of A_{n-1} that extends from the first row to the one row before the last row is of size $n-2$

$$A_{n-2} = \begin{pmatrix}
-2 & 1 & 0 & 0 & \cdots\\
1 & -2 & 1 & 0 & \cdots\\
0 & 1 & \ddots & 1 & 0\\
0 & 0 & 1 & -2 & 1\\
0 & 0 & 0 & 1 & -2
\end{pmatrix}$$

And this matrix is a full rank as it is the A matrix used for 1-D with Dirichlet boundary conditions and this matrix is known to be invertible (same one used in HW2).

Therefore, the rank of A can not be less than $n-2$ where n is the size of A.

In other words, the size of the null space of A can at most be 2. To determine if the size of the null space of A can be just one, the matrix A_{n-1}shown above has to be invertible as well.

One way to show that A_{n-1} is invertible, is to show that the last column of A_{n-1} is linearly independent to any of the remaining columns of A_{n-1}.

The last column of A_{n-1} is $c_{n-1}\begin{pmatrix} 1\\ 0\\ \vdots\\ 0\\ 1\\ -2 \end{pmatrix}$ and this column is linearly independent with the first

column of A_{n-1} which is $c_1 = \begin{pmatrix} -2 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$ since $a \times c_{n-1} + b \times c_1 = 0$ only when a, b are both zero. The same can be shown with all the other columns of A_{n-1}, they are all linearly independent to the last columns of c_{n-1}. Since all the other $n-2$ columns of A_{n-1} are linearly independent with each others (they make up the Dirichlet matrix known to be invertible) then A_{n-1} is invertible. This shows that the rank of A is $n-1$, hence the null space of A has dimension 1 only. In other words, only the and only the vector 1^T in the null of A. (Since A is symmetric, the null space of the adjoint of A is the same).

0.7.2 part (b)

In terms of looking at the conditions of solvability, the continuous case is considered and then the conditions are translated to the discrete case.

The pde $u''(x) = f(x)$ with periodic boundary conditions has an eigenvalue which is zero (the boundary conditions $u'(1) = u'(0)$ results in this), hence

$$0 = \int_0^1 f(x)\, dx$$

Is the solvability condition which results from the $u'(1) = u'(0)$ boundary conditions. (same argument that was carried out in part (d), problem 1, HW1 which had Neumann boundary conditions is used). Now, what solvability conditions does $u(0) = u(1)$ add to this if any?

Since

$$u''(x) = f(x)$$

Then integrating once gives

$$u'(1) - u'(0) = \int_0^1 f(x)\, dx + C_1$$

But since $u'(1) = u'(0)$, then the above implies that

$$0 = C_1$$

And integrating twice the PDE results in

$$u(1) - u(0) = C_2$$

But $u(1) - u(0) = 0$, hence $C_2 = 0$. So the only solvability condition is based on the fact that an eigenvalue is zero, which implies

$$\int_0^1 f(x)\, dx = 0$$

This is the same as was the case with Neumann boundary conditions. In the discrete case, this implies that solvability condition becomes the discrete integration (Riemman sum)

$$h \sum_{j=0}^{j=N+1} f\left(jh\right) = 0$$

For 2D, by extension of the above, there will be 2 eigenvalues of zero values, hence the discrete solvability condition becomes

$$h^2 * \sum_{i=0}^{i=N+1} \left(\sum_{j=0}^{j=N+1} f\left(ih, jh\right) \right) = 0$$

0.7.3 Part(c)

Since this is an iff problem, then the following needs to be shown

1. If v is in the null space of A then v is an eigenvector of T with eigenvalue 1
2. If v is an eigenvector of T with eigenvalue 1 then v is in the null space of A

Solving part(1)

Since v is in null space of A, then by definition

$$Av = 0$$

But $A = M - N$, hence the above becomes

$$(M - N)\, v = 0$$
$$Mv - Nv = 0$$

Since M is invertible by definition, then M^{-1} exists. Premultiply both sides by M^{-1}

$$M^{-1}Mv - M^{-1}Nv = M^{-1}0$$

But $M^{-1}0 = 0$ then the above becomes

$$Iv - M^{-1}Nv = 0$$
$$M^{-1}Nv = v$$
$$Tv = v$$

Therefore v is an eigenvector of T with an eigenvalue of 1.

Solving part(2)

Since v is an eigenvector of T with eigenvalue 1 then

$$Tv = \lambda v$$

With $\lambda = 1$, and since $T = M^{-1}N$, then the above becomes

$$M^{-1}Nv = v$$

Multiply both sides by M

$$Nv = Mv$$

Therefore

$$Mv - Nv = 0$$
$$(M - N)\, v = 0$$

Hence

$$Av = 0$$

Therefore v is in the null space of A.

0.8 References

1. Applied Mathematica by David Logan, chapter 8

0.9 Source code

```
function nma_build_HW3()

list = dir('*.m');

if isempty(list)
    fprintf('no matlab files found\n');
    return
end

for i=1:length(list)
    name=list(i).name;
    fprintf('processing %s\n',name)
    p0 = fdep(list(i).name,'-q');
    [pathstr, name_of_matlab_function, ext] = fileparts(name);

    %make a zip file of the m file and any of its dependency
    p1=dir([name_of_matlab_function '.fig']);
    if length(p1)==1
        files_to_zip =[p1(1).name;p0.fun];
    else
```

```
        files_to_zip =p0.fun;
    end

    zip([name_of_matlab_function '.zip'],files_to_zip)

end

end
```

```
function nma_cpu_plot_2D()
%
% to plot O(n^4) vs. O(n^3 log(n))
% used to verify the cost complexity for problem 3, part (b), HW3
% Math 228A
% Nasser M. Abbasi
%

close all;
n=10:1:40;
plot(n,n.^2,'r',n,n.^3.*log(n));
legend('direct solver','sor')

figure
close all;
n=10:1:4000;
plot(n,n.^(5/2),'r',n,n.^(3/2).*log(n));
legend('direct solver','sor')

figure
close all;
n=[10 20 30 35 40 45 50 55 60 65];
plot(n,log10(n.^5),'ro',n,log10(n.^4.*log10(n)),'*');
legend('direct solver','sor')
hold on
n=[10 20 30 35 40 45 50 55 60 65];
plot(n,log10(n.^5),'r-',n,log10(n.^4.*log10(n)),'-');
title('Verifying theortical complexity cost of direct 3D sparse solver');
xlabel('n, number of unknowns in one dimension');
ylabel('log10(cost)');

end
```

```
function  nma_HW3_prob3_parta_SOR()
% file name: nma_HW3_prob3_parta_SOR.m
%
% This solves the SOR for 2D for HW3, problem 3, parta
% Math 228A, Fall 2010, UC Davis
% Nasser M. Abbasi
%

DOPLOTS = true; %set to false if do not want to see plots

% setup the spacings needed for the problem
%spacings = [2^-5 2^-6 2^-7 2^-8 2^-9 2^-10];
spacings = [2^-5 2^-6 2^-7 ];
omega    = arrayfun(@(i) 2*(1-pi*spacings(i)),1:length(spacings));

for m = 1:length(spacings)

    h = spacings(m);

    % evaluate f(x,y) on grid
```

```
    [X,Y] = meshgrid(0:h:1, 0:h:1);
    f     = -exp(-(X-0.25).^2-(Y-0.6).^2);
    nPoints = size(f,1);
    fv    = reshape(f,nPoints^2,1); % use grid vector norm
    normf = sqrt(h) * norm(fv,2);

    w = omega(m);

    fprintf('******************\n');
    fprintf('gridsize=[%d,%d]\n',nPoints-2,nPoints-2);

    % initialize space (grid) for residual calculation and for solution
    resid = zeros(nPoints,nPoints);
    u     = zeros(nPoints,nPoints);
    unew  = u;

    done      = false; %flag set to true in loop below when it converges
    tolerance = h^2;   % set tolerance
    k         = 1;     % initialize iteration counter

    t         = cputime;

    while not(done)
        for i = 2 : nPoints-1
            for j = 2 : nPoints-1

                resid(i,j)= f(i,j) - 1/h^2 * ( u(i-1,j) + u(i+1,j) + ...
                    u(i,j-1) + u(i,j+1) - 4*u(i,j) );

                unew(i,j) = w/4* ( unew(i-1,j) + u(i+1,j) + unew(i,j-1)+...
                    u(i,j+1) - h^2*f(i,j)) + (1-w)*u(i,j);
            end
        end

        u      = unew;
        residv = reshape(resid,nPoints^2,1); % use grid vector norm
        normResidue = sqrt(h) * norm(residv,2);

        if ( normResidue / normf) <tolerance
            done = true;
        else
            k = k+1;
        end

        if DOPLOTS
            subplot(1,2,1);
            mesh(X ,Y ,resid );

            subplot(1,2,2);
            mesh(X ,Y ,u );

            drawnow;
        end

    end


    fprintf('cpu time for SOR=%f seconds\n',cputime-t);
    fprintf('number of iterations = %d\n',k);

end

end
```

```
function nma_HW3_prob3_partb_3d_SOR()

% file name: nma_HW3_prob3_partb_3d_SOR.m
%
% This does the SOR solver for 3D for HW3, problem 3, partB
% Math 228A, Fall 2010, UC Davis
% SOR 3D solver
%
% Nasser M. Abbasi
%

DOPLOTS = true; %set to false if do not want to see plots

% setup the spacings needed for the problem
%gridsize = [10 20 30 35 40 45 50 55 60 65];
gridsize = [10 20 ];
spacings = arrayfun(@(i) 1/(gridsize(i)+1),1:length(gridsize));
omega    = arrayfun(@(i) 2*(1-pi*spacings(i)) ,1:length(spacings));

fprintf('----- Starting 3D SOR solver ---------\n');

for m = 1:length(spacings)

    h = spacings(m);
    nInternalGridPoints = gridsize(m);
    nPoints = nInternalGridPoints+2;

    fprintf('*******************************\n');
    fprintf('grid is 3D [%d,%d,%d]\n',nInternalGridPoints, ...
        nInternalGridPoints,nInternalGridPoints);

    fprintf('h=%f\n',h);

    [X,Y,Z] = meshgrid(0:h:1, 0:h:1,0:h:1);

    % initialize space (grid) for residual calculation and for solution
    u       = 0.*X+0.*Y+0.*Z;
    unew    = u;
    resid   = u;

    % evaluate f(x,y,z) on grid
    f       = -exp(-(X-0.25).^2-(Y-0.6).^2 - Z.^2);
    normf   = sqrt(h)* norm( reshape(f(2:end-1,2:end-1,2:end-1),...
        nInternalGridPoints^3,1),2);

    w         = omega(m); %optimal w for SOR
    done      = false; %flag set to true in loop below when it converges
    tolerance = 0.1*h^2;   % set tolerance
    k         = 1;     % initialize iteration counter

    t         = cputime;

    while not(done)
        for i = 2 : nPoints-1
            for j = 2 : nPoints-1
                for z = 2 : nPoints-1
                    resid(i,j,z)= f(i,j,z) - 1/h^2 * ( u(i-1,j,z) + ...
                        u(i+1,j,z) + u(i,j-1,z) + u(i,j+1,z) - ...
                        6*u(i,j,z) + u(i,j,z-1) + u(i,j,z+1));

                    unew(i,j,z) = w/6* ( unew(i-1,j,z) + u(i+1,j,z) + ...
                        unew(i,j-1,z) + u(i,j+1,z) + unew(i,j,z-1) + ...
                        u(i,j,z+1)- h^2*f(i,j,z)) + (1-w)*u(i,j,z);
```

```
                end
            end
        end

        u = unew;
        if DOPLOTS
            subplot(1,2,1);
            mesh(X(:,:,nPoints-1),Y(:,:,nPoints-1),resid(:,:,nPoints-1));

            subplot(1,2,2);
            mesh(X(:,:,nPoints-1),Y(:,:,nPoints-1),u(:,:,nPoints-1));

            drawnow;
        end

        % can't do norm on 3D, change to vector
        residv = reshape(resid(2:end-1,2:end-1,2:end-1),...
            nInternalGridPoints^3,1);
        normResidue = sqrt(h) * norm(residv,2);

        if (normResidue/normf) < tolerance
            done = true;
        else
            k = k+1;
        end

    end

    fprintf('cpu time for 3D SOR solver =%f seconds\n',cputime-t);
    fprintf('number of iterations = %d\n',k);
end

end
```

```
function nma_HW3_problem_3_part_A_graph_plot()
% This   used to generate plot to compare CPU time of SOR
% and direct solver for 2D problem
% file name nma_HW3_problem_3_part_A_graph_plot.m

close all;
x=[31 63 127 255 511 1023];
directCPU=[0.015 .125 .250 1.544 5.538 27.113];
sorCPU=[0 0.094 0.6 5.2 48 532];

%plot(x,log10(directCPU),':.',x,log10(sorCPU));
plot(x,directCPU,'-',x,sorCPU);

title('compring CPU time against grid size, 2D pde solver');
xlabel('grid size n');
%ylabel('log10 of CPU time in seconds');
ylabel('CPU time in seconds');
hold on;
legend('direct solver','SOR','Location','NorthWest');

grid on
%plot(x,log10(directCPU),'o',x,log10(sorCPU),'o');
plot(x,directCPU,'o',x,sorCPU,'o');
ylim([-10 550]);

end
```

```
function nma_HW3_problem_3_part_B_graph_plot()
```

```
% This  used to generate plot to compare CPU time of SOR
% and direct solver for 3D problem
% file name nma_HW3_problem_3_part_B_graph_plot.m
% Nasser M. Abbasi
% 11/8/2010

x=[10 20 30 35 40 45 50 55 60];
directCPU=[0.047 0.5 3.9 8.8 21.50 40 84 157.7 244.4];
sorCPU=[0.01 0.078 0.405 0.75 1.29 2.11  3.24  4.9  7.17];

plot(x,log10(directCPU),':.',x,log10(sorCPU));
%plot(x,directCPU,'-',x,sorCPU);

title('compring CPU time against grid size, 3D pde solver');
xlabel('grid size n');
ylabel('log10 of CPU time in seconds');
%ylabel('CPU time in seconds');
hold on;
legend('direct solver','SOR','Location','NorthWest');

grid on
plot(x,log10(directCPU),'o',x,log10(sorCPU),'o');
%plot(x,directCPU,'o',x,sorCPU,'o');

end
```

```
function nma_HW3_problem_3_partA()
%
%  name: nma_HW3_problem_3_partA.m
%  purpose: direct solver for HW3, problem 3, part A. 2D
%  UC Davis math 228A
%
%  description of algorithm:
%    This script when called, will find the solution to the problem
%    Au=f as described in the HW, by using sparse matrix and direct
%    solver.  The script will solve the problem for the following h
%    spacings: 2^-5 2^-6 2^-7 2^-8 2^-9 2^-10
%
%    It will find the cpu time used and print to the screen the result
%    for each grid space.
%
% external functions called:
%    This scripts makes calls to nma_lap2d() to geberate
%    the sparse matrices.
%
% date written: 11/5/2010
% by: Nasser M. Abbasi
%

close all; clear all;

DOPLOTS  = true; %set to false if do not want to see plots
%spacings = [2^-5 2^-6 2^-7 2^-8 2^-9 2^-10];
spacings = [2^-5 2^-6 2^-7 ];

gridSize = arrayfun(@(i) 1/spacings(i)-1,1:length(spacings));

for i = 1:length(spacings)

    h = spacings(i);
    n = gridSize(i);

    fprintf('n=%d\n',n);
```

```
38
39      % evaluate f(x,y) on grid and convert to vector
40      [X,Y] = meshgrid(h:h:1-h, h:h:1-h);
41      f     = -exp(-(X-0.25).^2-(Y-0.6).^2);
42      f     = reshape(f,n^2,1);
43
44      t = cputime;
45      A = nma_lap2d(n,n)./h^2;  %make the A matrix
46      fprintf('cpu time for making sparse matrix=%f seconds\n',cputime-t);
47      fprintf('nonzero elements=%d\n',nnz(A));
48
49
50      t = cputime;
51      u = A\f;
52      fprintf('cpu time for direct solver=%f seconds\n',cputime-t);
53
54      % plot solution if needed
55      if DOPLOTS
56          u=reshape(u,n,n);
57          mesh(u);
58          title(sprintf('2D solution n=%d',n));
59          drawnow;
60      end
61
62  end
63
64  end
```

```
1   function mma_HW3_problem_3_partB_direct_solver()
2   %
3   %  script file, name: mma_HW3_problem_3_partB_direct_solver.m
4   %
5   %  purpose: direct solver for HW3, problem 3, part B (3D).
6   %  UC Davis math 228A
7   %
8   %  description of algorithm:
9   %    This script when called, will find the solution to the problem
10  %    Au=f as described in the HW, by using sparse matrix and direct
11  %    solver.  The script will solve the problem for the following h
12  %    spacings: 2^-3, 2^-4, 2^-5 or grid size n=7,15,31
13  %
14  %    It will find the cpu time used and print to the screen the result
15  %    for each grid space.
16  %
17  % external functions called:
18  %    This makes calls to nma_lap3d() to generate
19  %    the sparse matrices.
20  %
21  % date written: 11/5/2010
22  % by: Nasser M. Abbasi
23  %
24  close all; clear all;
25  %gridsize = [10 20 30 35 40 45 50 55 60 65];
26  gridsize = [10 20 30];
27  spacings = arrayfun(@(i) 1/(gridsize(i)+1),1:length(gridsize));
28
29  fprintf('----- Starting 3D direct solver ---------\n');
30
31  DOPLOTS=true;  %set to false if do not see plot of solution
32
33  for i = 1:length(spacings)
34
35      h = spacings(i);
```

```
    n = gridsize(i);

    fprintf('*****************************\n');
    fprintf('grid is 3D [%d,%d,%d]\n',n,n,n);
    fprintf('h=%f\n',h);

    % evaluate f(x,y,z) on grid and convert to vector
    [X,Y,Z] = meshgrid(h:h:1-h, h:h:1-h,h:h:1-h);
    f     = -exp(-(X-0.25).^2-(Y-0.6).^2 - Z.^2);
    f     = reshape(f,n^3,1);

    t = cputime;
    A = nma_lap3d(n)./h^2; % make the A matrix, sparse
    fprintf('cpu time for making sparse matrix=%f seconds\n',cputime-t);
    [nRow,nCol]=size(A);
    fprintf('dimensions of A (sparse matrix) is [%d,%d]\n',nRow,nCol);
    fprintf('nnz(A)= %d\n',nnz(A));

    t = cputime;
    u = A\f;
    fprintf('cpu time for direct solver=%f seconds\n',cputime-t);

    % plot solution if needed
    if DOPLOTS
        u=reshape(u,n,n,n);
        mesh(u(:,:,n-1));
        title(sprintf('3D solution, top surface only, n=%d',n));
        hold on;
    end

end

end
```

```
function nma_nnz_estimate()
% to estimate nnz() as function of n for 2D sparse matrix
% Nasser M. Abbasi
% HW3 math 228A

clear all;
close all;
n=[3 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 200 300 400 500 600];
y=arrayfun(@(i) nnz(lap2d(n(i),n(i))),1:numel(n));
plot(n,y,'r');
hold on;
plot(n,n.^2.25);
title('estimating nnz order for 2D sparse matrix');
xlabel('n, number of grid points in one dimension');
ylabel('nnz, number of non-zero elements');
legend('Matlab nnz()','n^2.2')

end
```

```
function nnz_estimate_3D()
% file name  nnz_estimate_3D.m
% to estimate nnz() as function of n for 3D sparse matrix
% Nasser M. Abbasi
% HW3, Math 228A

clear all;
close all;
n=[3 10 20 30 40 50 60 70 80 100 150 200];
```

```
y=arrayfun(@(i) nnz(lap3d(n(i))),1:numel(n));
plot(n,y,'r');
hold on;
plot(n,n.^3.35);
title('estimating nnz order for 3D sparse matrix');
xlabel('n, number of grid points in one dimension');
ylabel('nnz, number of non-zero elements');
legend('Matlab nnz()','n^3.35')

end
```