FEEG6002 Advance Computational Method Coursework 2015/16: PDE methods.

Name : Alan Tan Kay Meng

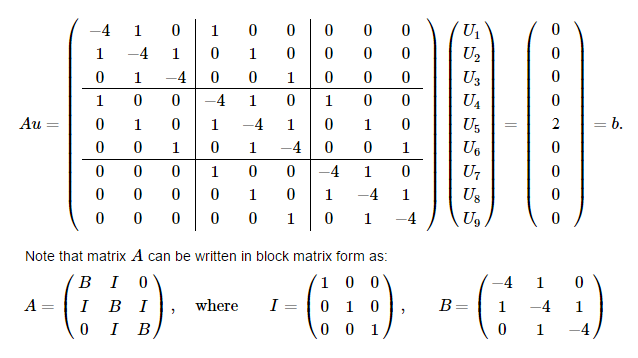
Student ID : 25816322

Link to Bitbucket :

Question 5 : Give a short write-up for the codes you have written in questions 1-4.

Q1:

In Question 1 consist of 3 main function which is get\_A, get\_rho, and laplace2d. Function get\_A and get\_rho returns matrix A and matrix b. Boundary condition is given 2 for ρ(0.5,0.5) and zeros everywhere else for the PDE ∇2*u* ρ,which is the b vector created by function get\_rho. The figure below extracted from FEEG6002 lecture notes numerical method 05 shows the arrangement of the A matrix, b matrix and the equation to solve PDE (1).



The origin of the B matrix is from the second derivative of the first central difference

Where the mesh separation is h. Thus, if N = number of mesh grid, then h = 1 / (N-1).Thus the function laplace2d for question 1, A = get\_A(n) \* (1/(h\*\*2)).

The code in the next line U = sp.linalg.solve(A, b) is where the build in solver solve for A.U = b to solve for vector U row by row. After solving for the U, dot product of Matrix A and the solved vector U is computed to check if the answer of PDE (1) at ρ(0.5,0.5) is equal to 2 in the code line 78, CheckU = np.dot(A,U).

The function laplace2d is coded to print the value of CheckU at midpoint of vector in code line 90.

“ Q1: Value of the dot product A.u1 is 2.000 at (0.5,0.5). “

Q2:

Question 2 code consist of 2 functions, iterate() and gauss\_seidel(). In the iterate function, A matrix is extracted from get\_A() and is solved with boundary condition written within the code. This means that the function iterate() is made specifically to solve PDE (1). By using Gauss-Seidel method with successive over relaxation given by the below equation,

where is the relaxation parameter. The matrix is solved row by row by giving the iterate function an initial guess for x of length N2. For example, the first row in PDE(1) , . By using the above equation, .

After the matrix is solved row by row, the process is repeated until the solution of x converged. This is done in the function gauss\_seidel which copy the previous x vector, compute a new x vector and compare the difference between the two vectors. If the difference of x\_old and x\_new is less than the tolerance, which in this case 1.0E-09 the x vector is said to be converged. Number of iteration required to converge the vector is recorded to be 50 for N = 9.

Default value for the relaxation parameter ω is set to be equal to 1. In the code line 219 is where the function calculating the optimal value of ω using the below equation when number of iteration, ‘niter’ is equal to parameter k = 10 set in the function.

Two differences of x vector, dx1 and dx2 is extracted at 10th and 11th iteration for the gauss\_seidel function to calculate the optimal value for ω. Thus the above equation can be rewritten for our case in the following form.

To investigate the effect of relaxation factor on the number of iteration to converge, function x\_conv() in code line 226 is created. The function looped from omega = 1.1 to omega = 1.9 recording the value of x at each and every iteration while converging the solution. A graph of no. of iteration VS x value is plotted to show the effect of relaxation factor on the converging speed of the code could be found in the next page.

It was found that number of iteration needed to converge the PDE is actually higher when value increased from 1.1 to 1.9. It was due to a very small tolerance at 1E-09 value where high omega function are suffering from minor vibration and are not considered converged. However, if we were to reduce the residual tolerance down to 1E-04 the purple line showing omega = 1.5 will be converged before 20 iteration.

After 50 iteration, the converged x value is substituted into A.X to validate value of ρ(0.5,0.5). The single line result printed for question 2 in the code is as follow:

“Q2: Value of the dot product A.x2 is 2.000 at (0.5,0.5). niter = 50, optimal omega = 1.53462 “

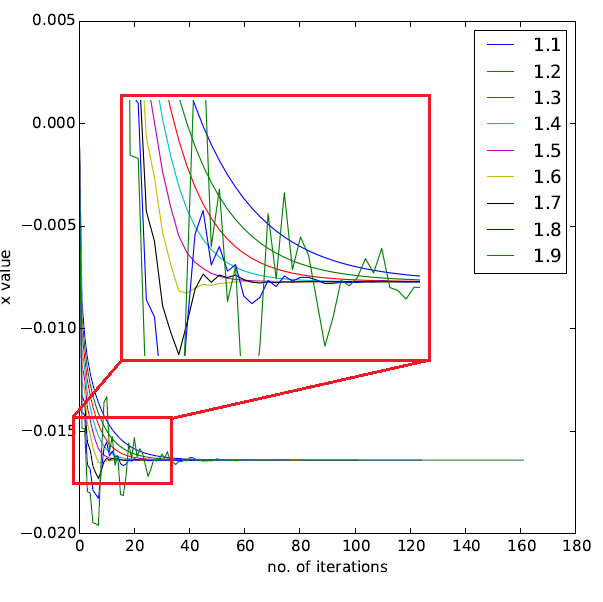


Figure 1: No. of iterations VS x value graph

Q3:

By using central difference method in question 1 and substitute the given stencil which is of higher order yields the following equation,

where and are the mesh size in the x and y directions respectively. Assuming a uniform grid

We can discretize the continuous Laplace equation in the bounded into a discrete eq:

The new A matrix generated using the same method used in Q1 is shown in the next page with N = 5 grid.

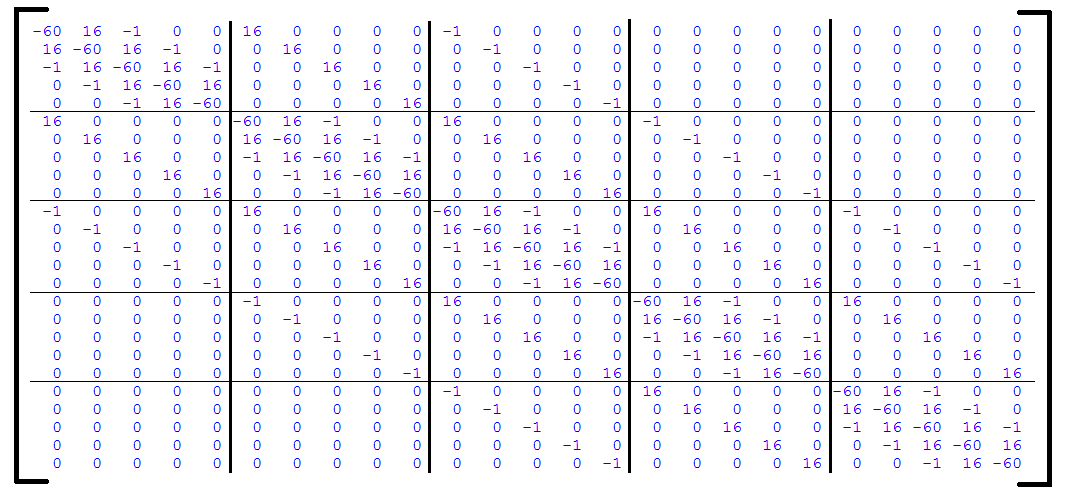


Figure 2: A matrix generated using new stencil

In the code line 386 is where the PDE is solved with the laplace2dq3() function which is similar to the solver used in Q1. To investigate the effect of the new stencil on solving the mesh, the U vector, which is the solved PDE(2) is reshaped into a matrix and are compared with U vector in Q1. Since the solver is the same, different result yield will be due to the change of the stencil.

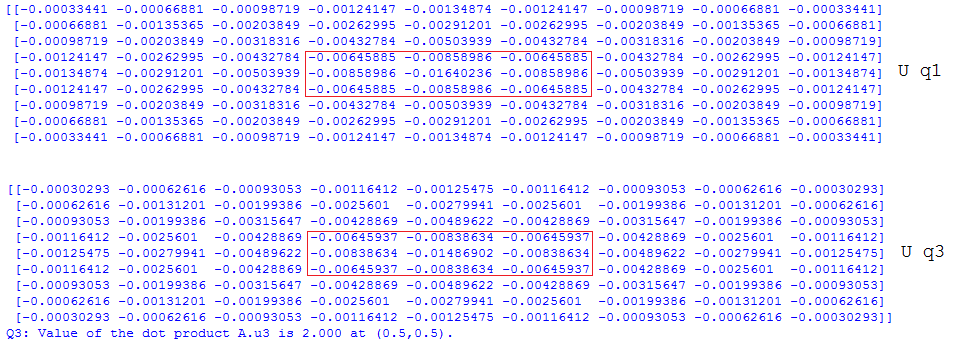


Figure 3: Comparison of Solved U vector of different stencil.

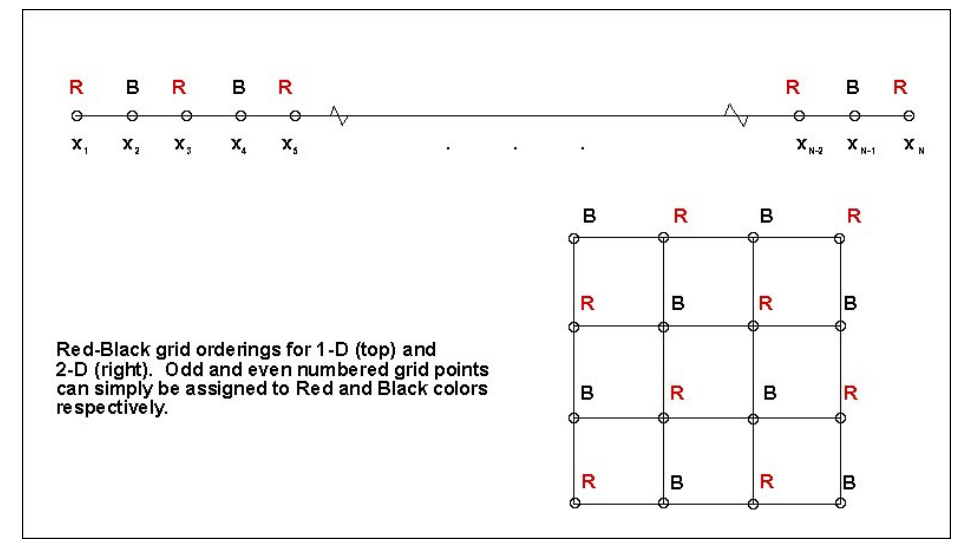
From Figure 3 above where two rectangular red box located are the position (0.5 , 0.5) of the reshaped U vectors and the values around it. Difference of value between them is not big but still significant.

Dot product of the A matrix and solved U3 is computed and result is printed to the display.

“Q3: Value of the dot product A.u3 is 2.000 at (0.5,0.5). ”

Q4:

A red-black matrix is often useful when trying to compute an inherently sequential problem in parallel. For example, by using a red black coloring scheme for grid points, the Gauss-Seidel method can be vectorized to compute the solution on all Red grid points simultaneously, followed by all Black grid points. To create the red-black matrix, one should use a corresponding red-black grid which chooses alternating grid points in such a manner that odd grid points are colored red and even numbered grid points are colored black or vice-versa. The graph below shows a red-black ordering for 1D and 2D grids, this idea can also be extended similarly to 3D.



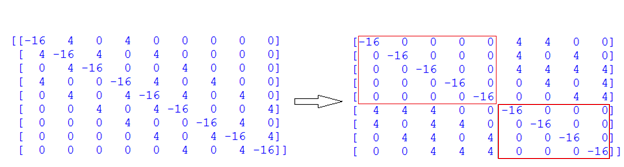
A red-black matrix should take the block form

Where D1 and D2 are strictly diagonal matrices and C is diagonally sparse. This would in general correspond to the solution of the linear system.

The matrix is then solved first for the red points followed by the black points as shown below:

The solver iterates until the solution converged in code line 557.

The matrix A\*(1/h2) from Q2 is rearranged into a red-black matrix using the function redblackA(N) and is shown in the figure below. The boundary condition generating function get\_rho(N) is rearranged with function redblackb(N) and queue to be solved.

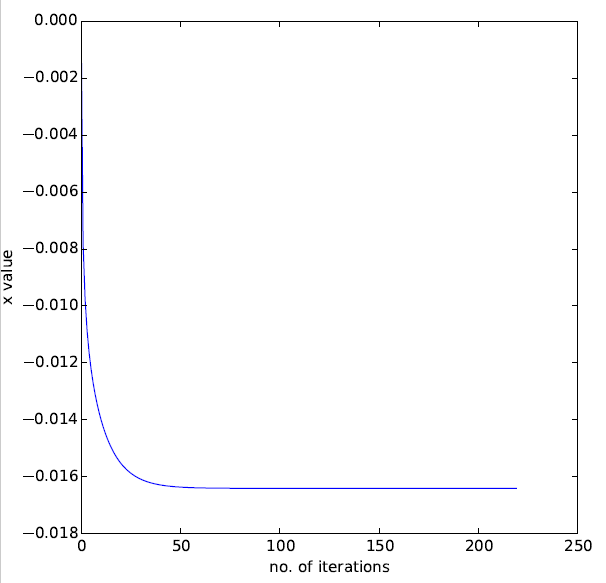


The solved vector x is reverted back to normal sequence with the function redblackb\_rev(x). Similar procedure to validate the value at (0.5,0.5) using dot product of Matrix A and vector x, ( A.X4 ) after reverting vector x back to the original arrangement.

“Q4: Value of the dot product A.xrb is 2.000 at (0.5,0.5). niterRB = 219”

Looking at the number of iteration taken to converge the PDE (1) which is about 4 times larger than the Gauss-Seidel method brings great interests to discover the problems within it. To investigate the converging pattern, similar plot made in Q2 is computed and the result is as shown below.

The solution is actually converged at about 50 iteration but due to the small residual value, 1E-09 the function continue to count until 219 iteration. The converging process has less fluctuation and converge relatively fast at the first few iteration. The central difference method is solving faster for PDE (1) might be due to the boundary condition at center which is favorable to central difference scheme. For a skew or bias boundary condition the red-black grid might be able to solve the PDE faster. To compare performance of different solver in this question the Gauss-Seidel are used to solve the red-black grid. Result of the function is printed to the display as shown below.



“ Q4: Using Gauss-seidel, value of the dot product A.x4 is 2.000 at (0.5,0.5). niter4 = 150, optimal omega = 1.52747 “

The multigrid method is one of two approaches to the use of multigrid techniques. The multigrid method is a means for accelerate the convergence of a traditional iterative method known as relaxation by solving a coarse problem. A global correction of the fine grid solution are approximated from time to time throughout the process rather than solving it. Residual, or the difference between xN and xN+1 are first found and are restricted to the coarse grid using a restriction operator. Then, the simpler and coarser grid are solved exactly. Prolongation operator are used to interpolate the correction on the restriction earlier back to the fine grid and the process is iterate until converged.

REFERENCES

1. From <http://computationalmathematics.org/topics/files/red_black.pdf> (link check Jan 16)

Austen D. “Creating and Using a Red-Black Matrix” (2010)

1. From <http://www.southampton.ac.uk/~feeg6002/lecturenotes/feeg6002_numerical_methods05.html> (link check Jan 16) “Partial Differential Equations II: 2D Laplace Equation on 5x5 grid”
2. From <http://www.nrbook.com/a/bookcpdf.php> (link check Jan 16)

Press, WH, Teukolsky, SA, Vetterling, WT, and Flannery BP. “Numerical Recipes in C”