MECE 5397 Scientific Computing for Mechanical Engineers

Project A – Poisson Equation

Version: APc2-1

Johnathan Sandoval

1186823

Class session: Mon, Wed @ 1:00pm - 2:30 pm

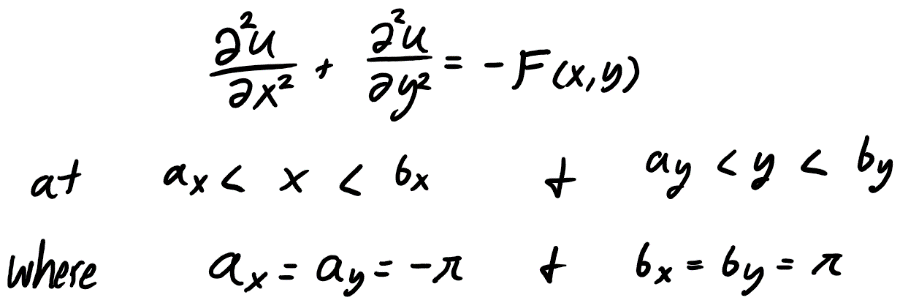
Date: April 30, 2017

Abstract:

The complexity in solving the 2-D Poisson Equation iteratively can vary largely due to many factors that one has to fully understand before the solution begins to even make sense. Because of this complexity, in order to derive a concrete solution one must write a MATLAB script that takes into account all of these conditions that occur at the boundaries and applies them to an adaptive iterative process. Although it took nearly 700 lines of code, the solution was successfully generated using both the Gauss-Seidel method and the SOR (Successive-Over-Relaxation) Method with mixed boundary conditions.

The reason the code has so many lines is because it implements several features which allow the code to ask for different criteria for solving the Poisson equation. The code includes safe guards so even the least mathematically adept person can use it, it has restart capabilities, customizable parameters, an implemented exit method within a selections menu, built in check pointing and it also has many graphs for both methods of solving. A main menu was implemented as well so that the UI felt the most natural, and to allow the user to maximize efficiency.

The result of this code is a solution with a total error smaller than , where X is equal to a value between 1 through infinity. By giving the user the ability to manipulate this value, the code can in turn produce results that can be extremely reliable depending on the value chosen. Although this code was written for a case wherein the x and y are both proportionally the same, a small amount of lines can simply be edited in this code to be repurposed for a scenario in which the x and y are not proportional. The code also generates a case where the user can simultaneously observe the results of both mathematical approaches at the same time and determine which method was best by seeing the amount of time steps were taken to solve both and their respective results. Overall the code solves the equation fast, using both mathematical methods when F is equal to both something, and 0, and it does it by asking for the least amount of effort from the user which is something that every program should strive for.

Mathematical Statement of the problem and discretization:

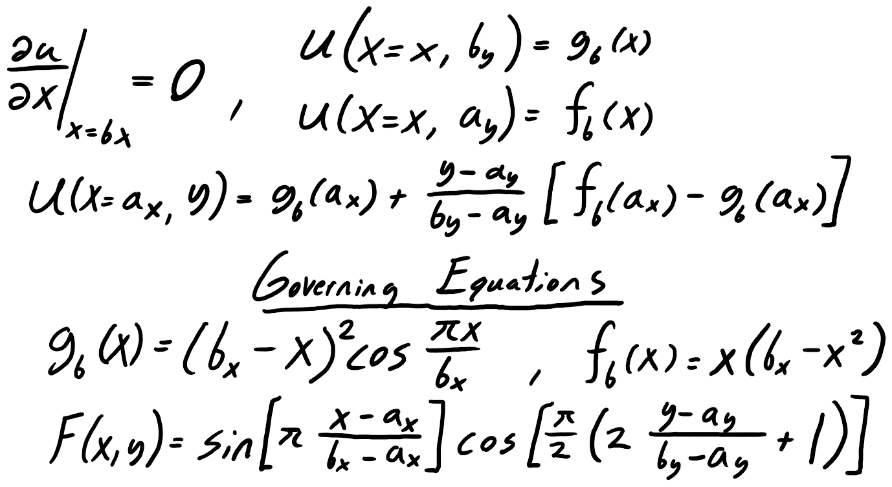
The equation to the left in Figure 1 is the renown Poisson’s equation with the respective x and y limits for the problem at hand. This equation is a generalization of Laplace’s equation (Same equation but F = 0 instead) which is also one that is frequently used in the mechanical engineering field. In this specific project we were tasked with solving the two dimensional Poisson Equation with 3 Dirichlet boundary conditions, and 1 Neumann boundary condition. The equations that dictate the behavior of this function at the boundaries are largely,, and F(x,y). These equations play a major role on the boundaries of our mesh since the boundary conditions call upon these equations at specific locations in the X-Y coordinate grid.

Figure 1: Poisson's equation with limits

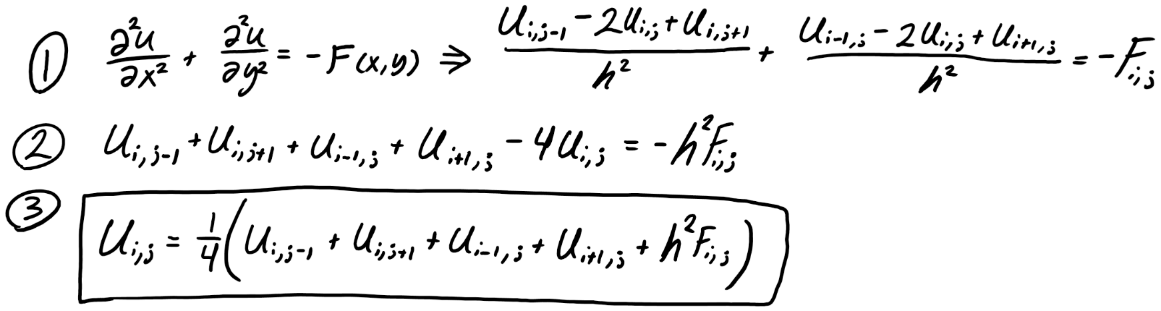
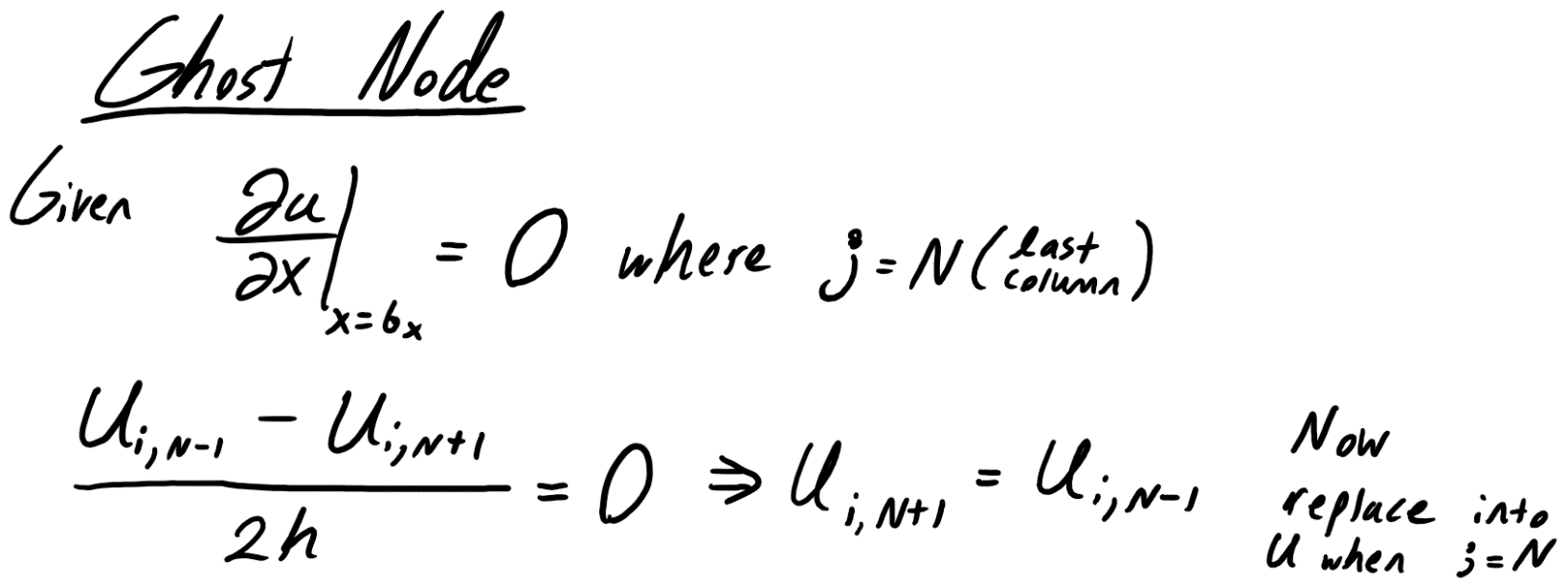
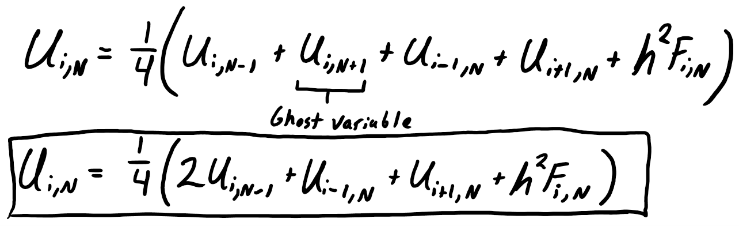
To begin tacking a problem of this sort with an iterative method such as the Gauss-Seidel, or SOR, one must first be able to discretize the partial differential function into its respective matrix variation using the i and j notation. By using the 2nd order central finite difference we can split the left side of the PDF as seen in step 1 of Figure 3. Since the problem calls for a domain where the length of X and Y are proportionally the same, then the assumption can be made that the distance from one node to another is the same, and thus the step length, h, is the same in both the X and Y directions. Next the equation needs to be simplified further so the common factor between the denominator of both U values, is moved to the right side as seen in step 2. Lastly, since we are solving this in an ordered iterative manner we really only care for the point that has the exact i, and j values of the point we need to solve, which in this instance is . So after isolating we can further simplify the equation by dividing out the 4 in front of and thus leave us with the discretized Poisson equation.

Figure 2: Boundary conditions and Governing equations

Figure 3: Discretization of the 2D Poisson Equation

If we had only Dirichlet conditions then the rest of this would just be trivial and essentially be ready to solve, but in this instance we also have a single Neumann boundary condition. Applying a Neumann Boundary condition is trickier than in sounds because if tackled in an improper way, then the solution may not converge on a final value. In order to mitigate this issue a technique must be applied where the points on this boundary are properly accounted for when solving for their true value. The solution to this is called a “Ghost Node”, which is just a way of using a node which doesn’t exist on the grid, but can be used for deriving the proper arithmetic for accounting values at this range on the boundary. The application of the ghost node is much simpler when the methodology is understood, as we can see below in Figure 4. Assuming you are given a 1st degree Neumann condition, you begin by taking the central difference of the derivative at the point of your boundary. For this specific problem we had the boundary on the right side of the X-Y grid, along all values of Y, so for this instance the value of j would be equal to N for all

instances. By simplifying the central difference we can see that we end up with = which is not directly applicable yet since N+1 would give an error within MATLAB for asking for values which don’t exist. Next this equivalence needs to be applied to our discretized equation for when j = N, and then the central difference equivalence can be substituted into the equation at . The result from all this an equation only valid for the boundary on the far right side, but it is an equation that can properly calculate the boundary values of the problem. Applying this once at the end of every row results in properly accounted boundary conditions, which behave also as expected when analyzed later in the report.

Figure 4: Discretizing the ghost node and deriving new boundary equation.

Description of Numerical Method:

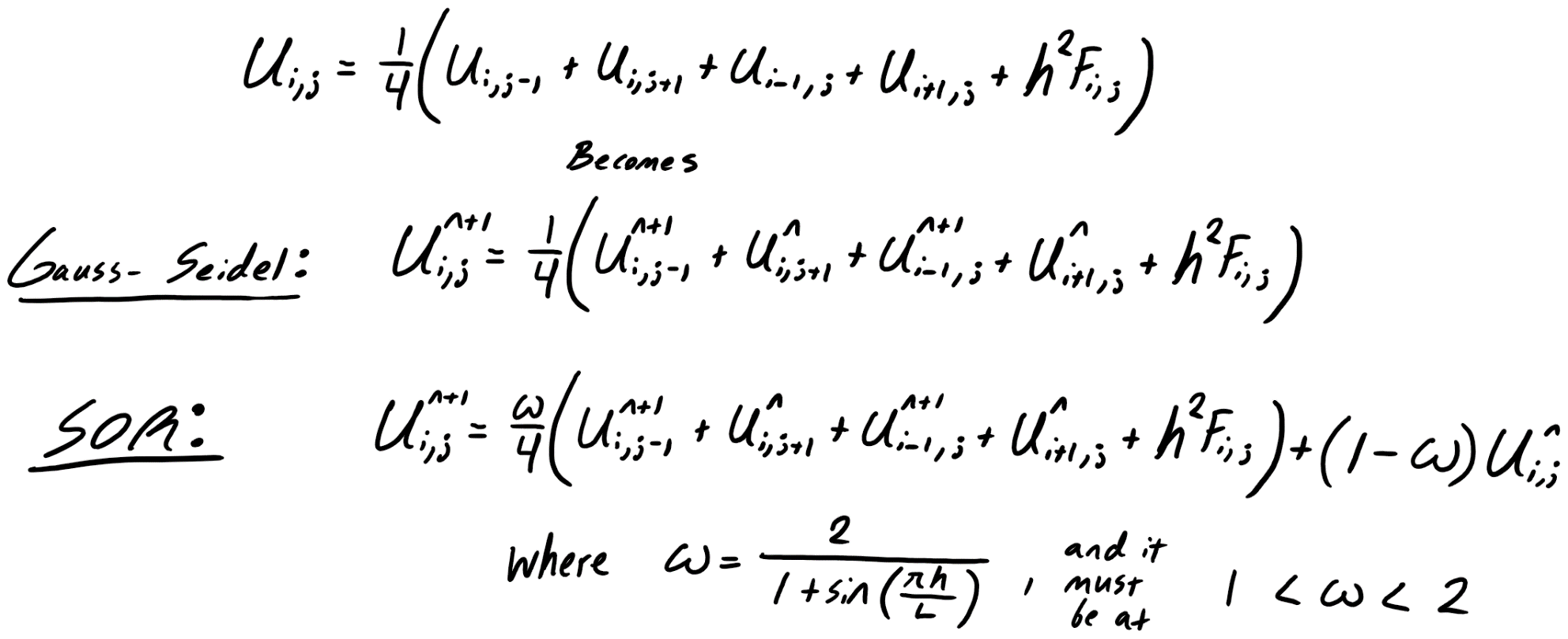


Figure 5: Gauss-Seidel and SOR equations with time step constants added

To solve this problem two different iterative methods are employed so that the solutions could be compared and observed for differences. The two methods in are the ones above in Figure 5, they are the Gauss-Seidel and the Successive Over Relaxation methods. Both methods have their own distinct advantages but the rate at which they converge is different.

The Gauss-Seidel method employs an iterative methodology that is extremely similar to the Jacobi method which was mentioned many times before in class. The key difference between the two being that Gauss-Seidel computes the next time step using only elements from that new time step that have already been calculated. By doing so, if this was used computationally then not much memory storage would be needed for keeping new values as the old ones would simply be over written. The draw back with using this method would be that it is extremely computationally heavy when narrowing the error down to a small value (). The Gauss-Seidel method was used in this code by first assigning a “guess” matrix for the first time step of U, and then solving for the values of from left to right, top to bottom using the algorithm. Although at higher error accuracies the answers were grid independent, it did require a large amount of time to fully run, which is something that would hamper research progress if this were a larger project.

The SOR method on the other hand performed much faster than the Gauss-Seidel method did, and not surprisingly either. The SOR method is actually the same equation as the Gauss-Seidel equation, the only difference being that the SOR method includes a constant called the over-relaxation factor, or just ω. If you set ω = 1, then the result you get is the Gauss-Seidel equation exactly, but for solutions using this method you have to test different values of ω in order to come upon an answer that converges fast. In order to even meet this convergence criteria, first ω must be a value that is 1 < ω < 2, if not then the equation will just simply never converge. Many equations exist for getting a decent value for this equation but the best one used for this project is:

Using this equation for values, the result was ω = 1.95, which not only fits the convergence criteria but also gave for results that converged extremely fast and reliably. The best feature about the SOR method is that once the Gauss-Seidel methodology is understood, then applying the other SOR constants becomes trivial since it’s essentially the same equation.

Technical specifications of computer used:

Computational power was not used sparingly as most of this code was written on a homebuilt computer made for computationally heavy programs/applications. To handle any visual rendering inside of MATLAB and to assist with computations, an overclocked NVIDIA GeForce GTX 1080 (ASUS Strix series) with 4GBs of DDR5 video memory was used. Although this may be overkill for most users, having a top of the line GPU is something that definitely improves performance overall when dealing with simulations and rendering. To store and access MATLAB files, an SSD was used for its reliability and speed. SSD’s offer impeccable speeds when reading or writing a file in a computer which is always a benefit, especially if you are using checkpoints.

Figure 6: Johnathan's Computer Specs

The only part of this computer setup that has questionable specifications is the RAM since it is clocked at a relatively low speed compared to most other setups. A respectable clock rate for most RAM nowadays is starting at around 2100MHz, but my setup is clocked at 666MHz. The clock rate in the end is not a huge bottleneck, only when excessively large matrices are being dealt with (when a matrix is equal to 2 million x 2 million or more).

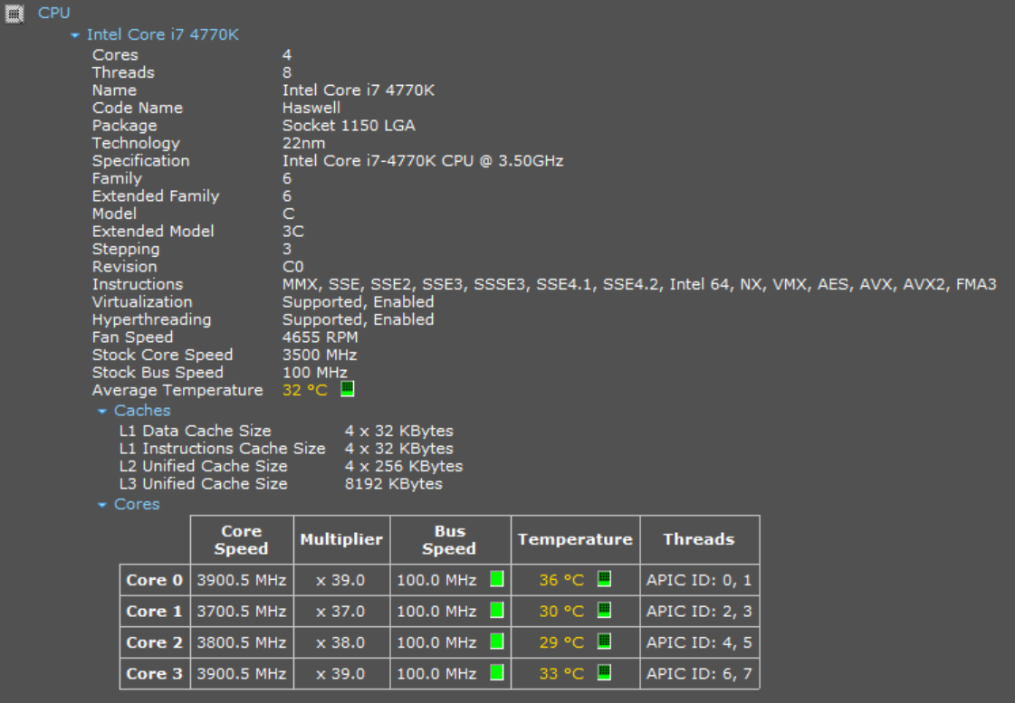
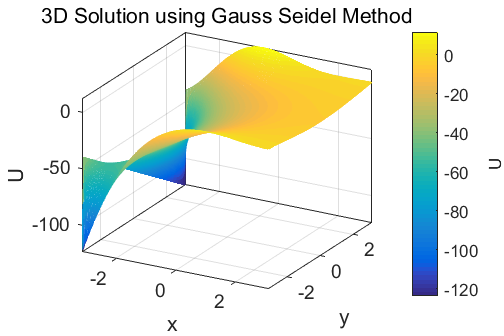
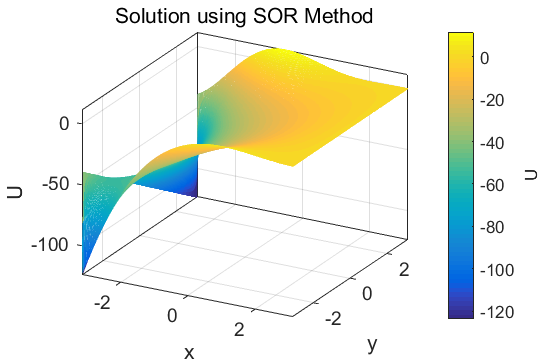
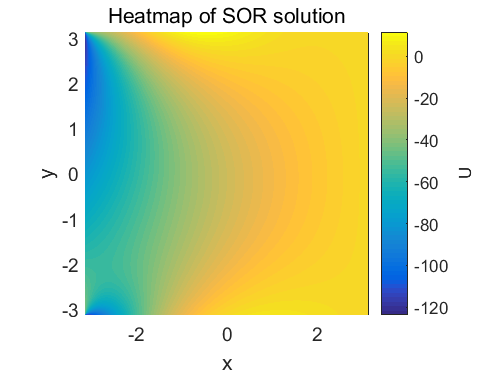
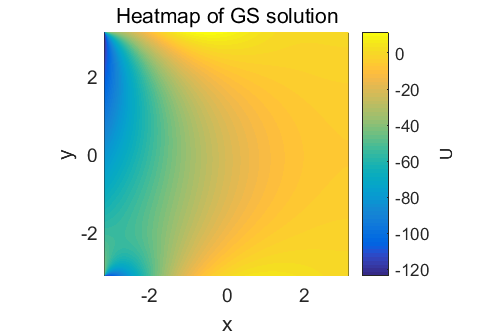
The most important part of this entire setup is the CPU, or the brain of the computer in a less accurate but simpler analogy. The CPU in question is an Intel 4th Generation i7 4770k, clocked stock at 3.50GHz, and also mounted under a water cooling unit. This quad core CPU is capable handling multitasking environments extremely well without getting remotely hot, so as long as the software or application supports hyper threading, and multicores.

Figure 7: Specs of CPU used (i7 4770k)

Although this entire computer setup was originally designed for video games, the hardware inside of it is more than qualified for handling undergraduate level computer simulations. The only improvements that could be made here to improve overall MATLAB performance would be the following.

* Overclock CPU
* Overclock GPU (Assuming MATLAB really needs a higher GPU clock speed)
* Higher RAM clock speed (Upgrading from DDR3 to DDR4 RAM)

**Results:

**

(a) (b)

Figure 8: (a) Shows solution of U in a 3D mesh and also its heat map using the Gauss-Seidel method

**(b) Shows solution of U in a 3D mesh and also its heat map using the SOR method**

The resulting solution for this project is an interesting one, because simply having a numerical value for an output was not enough to feel confident about the result. As seen above in Figure 8, the results for the solutions with both methods hardly differ from one another. The 3D mesh gives an idea of how the solution would look in a three dimmensional space, but it does not give a full image of what is truly occuring. Using the heatmaps generating using countourf( ), a much clearer image is generated that is more useful when observing the behavior of the results.

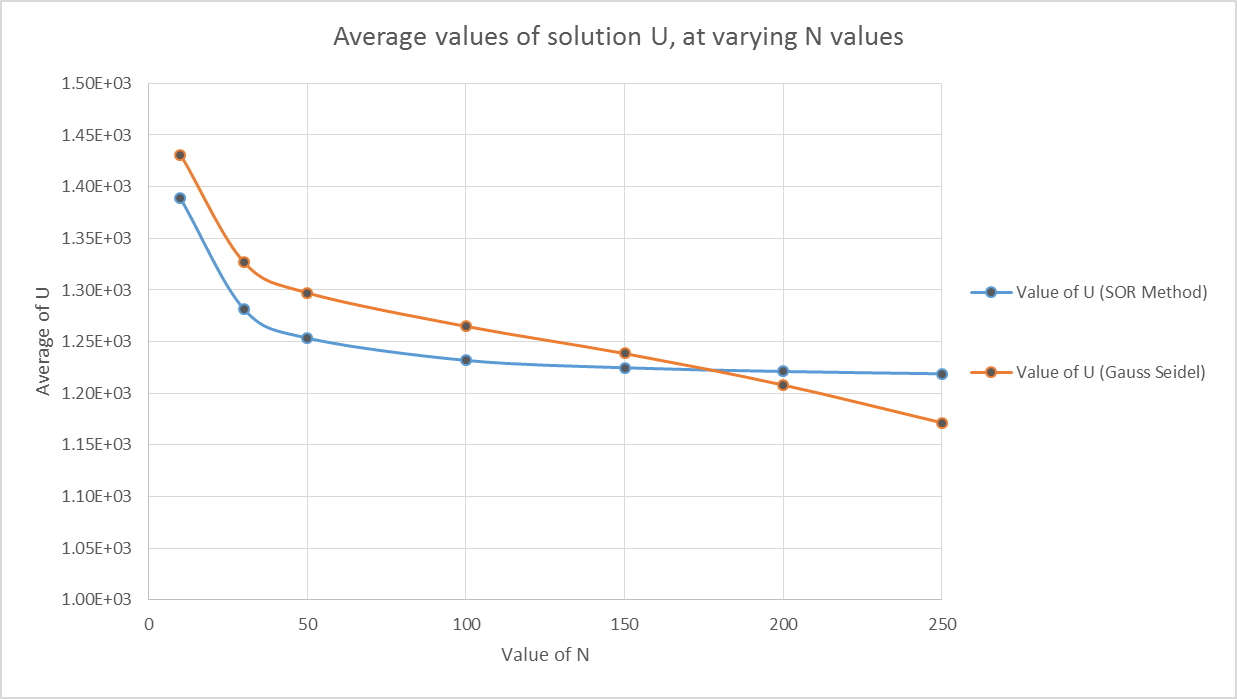
Using these images we can see that the Neumann Boundary condition was met since the slope at the far right boundary of the plot is equal to zero. The initial guess for this problem assumed the points in that area to be zero and thus the conditions were all met. The behavior of the function can also be further verified by looking for the values of the respective governing functions ,, and *F(x,y)* at the points where the appropriate boundaries are located. Since these values correspond correctly, then the boundaries can be assumed to have been properly applied. Once solutions were being generated consistently with the same results the answer could be assumed to be on the right path but the reason that the answer wasn’t entirely valid yet was because another study had to be performed on it to ensure that the results for this type of problem did not vary largely with changing grid sizes. Initially that wasn’t the case when applying the Gauss-Seidel method, where the answer would immediately deviate to 0 meaning that something wrong was happening with the internal nodes. This issue was easily fixed with minor code changes and the results began to look identical to those of SOR .Then a grid independence test was done where both problems were solved at different values of N, and compared to see how the answers would change based on increasing grid points. The result was definitely reassuring as the answers converged at a relatively low value of N, as seen above in Figure 9 and also numerically in Figure 10. One key thing that was being looked for here was that the values of the answer did not vary much regardless of the value of N. If the value remained constant then the answer could be deemed grid independent, which in the case of the SOR method it was. The values of the Gauss-Seidel method varied largely not due to the grid size, but rather the allotted error prescribed at the beginning. If an error of was used then the problem would eventually reach an answer that was not correct at a high grid count, but as that error was increased to and above, then the answer began to look much better but at the cost of more computational power and time. If the error was left at it is predicted that the results would be essentially identical but the amount of time such computations would take at a high grid count would more than excessive. So at a high error value ( for max accuracy) then the solutions become grid independent for most values, most specially for the Gauss-Seidel method.

Figure 9: Plot showing progression of Grid independence test for both methods

Another interesting result was the amount of time steps that it took both methods to solve at different values of N. For the Gauss-Seidel method the amount of time steps increased very largely and went up to 16000 time steps for a 250x250 grid, whereas the solution for the SOR method just took a mere 414 time steps for the same grid. This showed just how much more optimized the SOR method truly is when compared against Gauss-Seidel. The results from this testing are seen above in Figure 10, and they also show just how much the time steps really did increase for minor changes in the value of N. Although these values were calculated for when the error was set to the results were consistent enough to show expected behavior for other scenarios.

Figure 0: Data from Grid Independence test with time steps required to finish solution for both methods

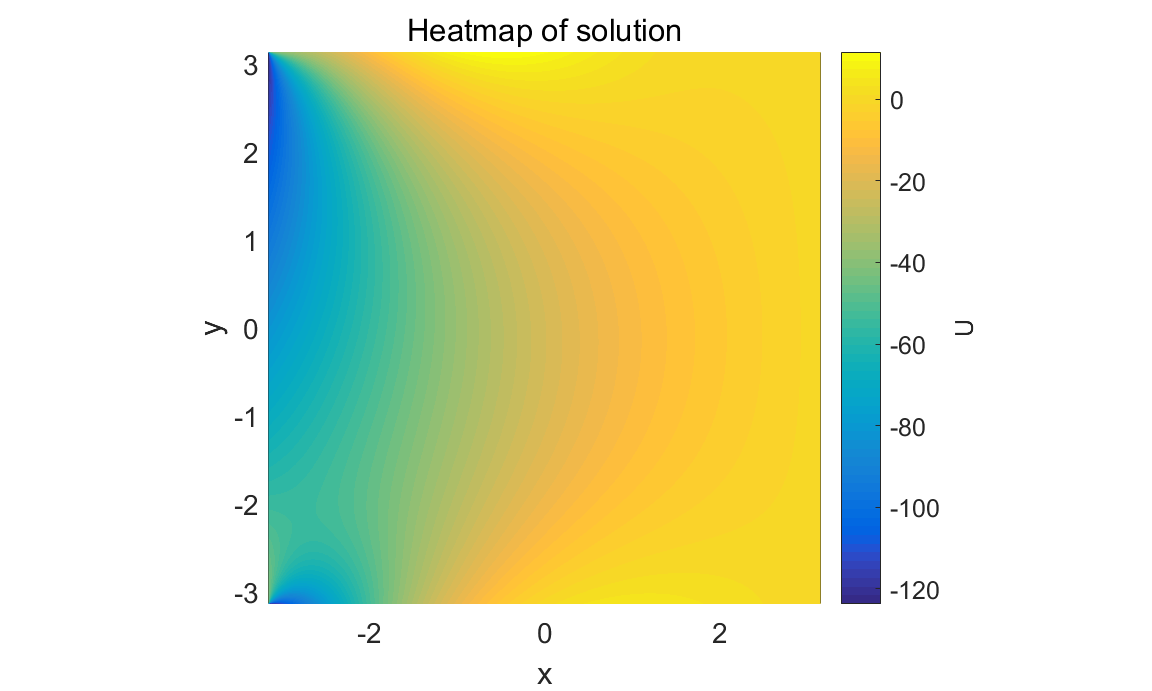
The last task in the entire project was to also observe the results of the equation for when the value of F is set equal to zero. This small change would in turn make the equation more generalized and we end up with the Laplace equation which we have used before in many other courses. The results of setting this equal to zero were a bit disappointing because the change in the results was extremely small, and actually almost negligible, as seen in Figure 11 to the right. It was a result should have been more expected because the equation for F varied very little since it was a sin and cos function, which have max values of 1.That coupled with the fact it was multiplied with , the length of a single node, inside the equation which exponentially decreased its value. The heat map above when compared to the other ones also shows very little difference in results but it does shed some light on how minimal the effects of F were on the equation. The situation did not change when applied to both methods thus it can said with confidence that F(x,y) = 0 did not significantly change the results of this project.

Figure 11: Heat map for Poisson’s equation when F = 0

Summary and Conclusion:

The applications of solving a 2D Poisson equation vary largely since it is just a generalization of the Laplace equation. It is because of this that the applications can go from being used in electromagnetism for studies on how electric fields work, or they can be used in fluid dynamics to study fluid potentials. So although this project did not specify any units of measurement, the end result from is now feeling much more comfortable handling such large equations inside of computer enviroments and also being able to apply these equations to other fields past just mechanical engineering.

Although initially expected for an increasing value of N, it was proven that the SOR method was singlehandedly the fastest method for solving this sort of problem. While the Gauss-Seidel method took thousands of time steps to solve the equation, the SOR method breezed through in 1/50th of the steps. To test whether the derived answers were correct, and also grid independent, several test were done to see if the answer would converge to a specific value. This test showed that at around a grid size of N = 150-200, results converged and the answers were identical. This result reassured the credibility of the final result in this project. Small improvements could have been made for this project such as having a few more resources when initially attempting the discretization of the Poisson equation. Other improvements such as class lessons on checkpointing would have also helped significantly. In the end this project was highly informative, and is one that will most definitely assist me in my coming graduate studies here at the University of Houston.

References:

[1] - <https://people.eecs.berkeley.edu/~demmel/cs267/lecture24/lecture24.html>

[2] - [http://12000.org/my\_courses/UC\_davis/fall\_2010/math\_228a/HWs/HW3/Neumman\_BC/Neumman\_BC.htm#x1-80002](http://12000.org/my_courses/UC_davis/fall_2010/math_228a/HWs/HW3/Neumman_BC/Neumman_BC.htm" \l "x1-80002)

[3] <http://www3.nd.edu/~gtryggva/CFD-Course2010/2010-Lecture-11.pdf>

Code used to calculate values (Also on https://github.com/sleep0holic/Project):

%Johnathan Sandoval

%Student ID# 1186823

%Computing For Mechanical Engineers

%4/28/2017

%Solution for Poisson Equation, version "APc2-1"

%Notes before running this code!

%This code simulates the 2 required linear solving methods of Gauss-Seidel

%and Successive Over Relaxation.

%NOW with built in CHECK POINTING

%----------------------------------------------------------------------------

%When initiated you will be asked for the method you intend on

%using/seeing.

% 1) The first method is the Gauss-Seidel with the solution for

%when F = a value and also for when F = 0 (Laplace equation)

% 2) The second method uses the SOR method with a carefully chosen

% over-relaxation factor (w) and also for when F = 0

% 3) The third method takes both methods and compares them with their own plots so

%that they can be compared for differences.

% 4) The fourth method allows you to reload a checkpoint in the event that

% you were not able to finish your calculation. All progress is saved when half

% of the code is done. If the code executes completely then the checkpoint

% is also conveniently deleted so you don't have to worry about that.

% 5) The final choice allows the user to simply exit the program without

% issue. Added simply to make the life of the grader easier. Thank me

% later.

%After selecting the method you will be asked for the amount of steps you

%would like to use (Value of N, note that its proportional in both X and Y directions

% so choose the amount steps you feel your computer can safely handle).

%Lastly you will be asked for the type of error resolution you are aiming to

%achieve. The error resolution is scaled as 10^-X, where X is a number between 1 to infinity.

%Again, this value can be taxing on certain computer setups so use high

%values with caution.

%------------------------------------------------------------------------------------------

%This is a fairly lengthy code but the features that it offers make it

%one of the most versatile codes available for this type of problem/project. There

%is no need to change any values inside of this code.

%------------------------------------------------------------------------------------------

clear all; close all;clc

lock = 0; flag = 0; main = 0;marker = 0; marker1 = 0; t = [];

while main ~= 1;

while lock ~= 1;

while flag ~= [1:5]

disp('Please select one of the three cases you would like to try:')

disp('Case #1: Gauss-Seidel solution')

disp('Case #2: SOR Method solution')

disp('Case #3: Both Solutions Compared')

disp('Case #4: Reload Checkpoint')

disp('Case #5: Exit program')

disp(' ')

flag = input('Case #'); checkp = flag;

if flag ~= [1:5];

disp(' ')

disp('HAHA okay that was funny, now seriously....select a case this time!')

disp(' ')

elseif flag == 4;

break

elseif flag == 5;

return

end

end

if checkp == 4;

if exist( 'checkpoint.mat','file' )

if t == 2;

t = 2;

elseif t == 3;

t = 2;

else

t = 1;

end

fprintf('Checkpoint file found - Loading\n'); disp(' ');

load('checkpoint.mat')

else

disp('There is no checkpoint file here, next time look for "Saving checkpoint"');

return

end

else

t = 1;

disp(' ')

disp('Please input the value of N you would like to use. N is proportional in the X and Y direction')

disp('Recommendation: 200 is best for a fast but ideal solution')

disp(' ')

Nx = abs(input('Value of N = '));

disp(' ')

disp(' ')

disp('Lastly, Please input the 10^-X accuracy youd like to reach')

disp('Recommendation: 7 at minimum for a consistent solution. Raise at your own risk')

disp(' ')

X = abs(input('Selection: '));

disp(' ')

disp('Loading....')

disp(' ')

disp(' ')

lock = 1;

end

end

XX = (10^-X);

%Setting up the number of XY grid points

Ny = Nx; step = 1/Nx; H = 1/(step^2);

%Setting up length of X and Y regions

ax = -pi; bx = pi; ay = ax; by = bx;

Lx = 2\*pi; Ly = 2\*pi;

%Setting up XY coordinates

minX1 = ax; maxX1 = bx;

minY1 = ay; maxY1 = by;

x = linspace(minX1,maxX1,Nx);

y = linspace(minY1,maxY1,Ny);

[xx,yy] = meshgrid(x,y);

yy = flipud(yy);

hx = x(2) - x(1); hy = y(2) - y(1);

uw = (((bx-ax).^2).\*cos(pi.\*ax/bx))+(((yy-ay)/(by-ay))\*((ax.\*(bx - ax).^2) - (((bx-ax).^2).\*cos(pi.\*ax/bx))));

gb = ((bx-xx).^2).\*cos(pi.\*xx/bx);fb = (xx.\*(bx - xx).^2);

%Defining my Boundary Conditions and Initial Conditions (Have 2 sets)

U = zeros(Ny,Nx);

U(1,2:Nx-1) = gb(1,2:Nx-1);

U(Ny,2:Nx-1) = fb(Ny,2:Nx-1);

U(2:Nx-1,1) = uw(2:Nx-1,1);

%%

switch flag

%Gauss-Seidel Code

case 1

while t ~= 3;

if t == 1;

F = sin(pi.\*(xx-ax)/(bx-ax)).\*cos((pi/2).\*(2.\*((yy - ay)./(by - ay))+1));

elseif t == 2;

F = zeros(Ny,Nx);

U = zeros(Ny,Nx);

U(1,2:Nx-1) = gb(1,2:Nx-1);

U(Ny,2:Nx-1) = fb(Ny,2:Nx-1);

U(2:Nx-1,1) = uw(2:Nx-1,1);

end

%Main loop that evaluates the remaining Neumann BC and also deals with the corners

%by taking a mean of the two boxes that touch that the corner.

bound = 1; n = 0;

e = 1; Q = (69.833\*Nx - 1164.3)/3; tic;

while e > XX;

Up = U;

for i = 2:Nx-1;

for j = 2:Ny-1;

if bound == 1;

bound = bound +1 ;

U(1,1)= (U(1,2)+U(2,1))/2; %These are used to even out my irregular spikes on the corners

U(1,Nx)= (U(1,Nx-1)+U(2,Nx))/2;

U(Ny,1)= (U(Ny-1,1)+U(Ny,2))/2;

U(Ny,Nx)= (U(Ny,Nx-1)+U(Ny-1,Nx))/2;

Up = U;

end;

if j == Ny-1;

U(i,Nx) = (1/4)\*(2\*U(i,Ny-1)+U(i-1,Ny)+U(i+1,Ny)+(hx^2)\*F(i,Ny));

end

U(i,j) = (0.25)\*(Up(i+1,j)+U(i-1,j)+Up(i,j+1)+U(i,j-1)+(hx^2)\*F(i,j));

end

end

U(1,1)= (U(1,2)+U(2,1))/2;

U(1,Nx)= (U(1,Nx-1)+U(2,Nx))/2;

U(Ny,1)= (U(Ny-1,1)+U(Ny,2))/2;

U(Ny,Nx)= (U(Ny,Nx-1)+U(Ny-1,Nx))/2;

E = U - Up;

e = mean(mean(E(2:Nx-1,2:Nx-1).^2));

n = n+1;

[Uxx, Uyy] = gradient(U,hx,hy);

Uxx = -Uxx; Uyy = -Uyy;

Ut = sqrt(Uxx.^2 + Uyy.^2);

end

if t == 1;

disp(['The error is ',num2str(e)])

disp(['Using the Gauss Seidel Method, this problem took ',num2str(n),' iterations to solve.'])

disp(' ')

figure(1) % VECTOR FIELD U

set(gcf,'units','normalized','position',[0.02 0.52 0.3 0.32]);

%surf(xx,yy,U');

mesh(xx,yy,U);

xlabel('x'); ylabel('y'); zlabel('U');

title('3D Solution using Gauss Seidel Method','fontweight','normal');

set(gca,'fontsize',14);

rotate3d

box on

axis tight

h = colorbar;

h.Label.String = 'U';

view(55,49);

figure(2)

set(gcf,'units','normalized','position',[0.33 0.52 0.3 0.32]);

contourf(xx,yy,U,16);

pcolor(xx,yy,U);

shading interp

xlabel('x'); ylabel('y');

title('Heatmap of GS solution','fontweight','normal');

set(gca,'fontsize',14)

box on

h = colorbar;

h.Label.String = 'U';

axis square

box on

figure(3)

set(gcf,'units','normalized','position',[0.65 0.52 0.3 0.32])

contourf(xx,yy,Ut,20);

shading interp

xlabel('x'); ylabel('y');

title('Gradient of U using GS','fontweight','normal');

set(gca,'fontsize',14)

box on

h = colorbar;

h.Label.String = '| Up |';

axis equal

else

disp(['The error is when F=0 is ',num2str(e)])

disp(['Using the Gauss Seidel Method, this problem took ',num2str(n),' iterations to solve.'])

disp(' ')

figure(4) % VECTOR FIELD U

set(gcf,'units','normalized','position',[0.02 0.1 0.3 0.32]);

%surf(xx,yy,U');

mesh(xx,yy,U)

xlabel('x'); ylabel('y'); zlabel('U');

title('Solution using Gauss-Seidel Method when F = 0 (Laplace Eqn)','fontweight','normal');

set(gca,'fontsize',14);

rotate3d

box on

axis tight

h = colorbar;

h.Label.String = 'U';

view(55,49);

figure(5)

set(gcf,'units','normalized','position',[0.33 0.1 0.3 0.32]);

contourf(xx,yy,U,16);

%pcolor(xx,yy,U);

shading interp

xlabel('x'); ylabel('y');

title('Heatmap of GS solution','fontweight','normal');

set(gca,'fontsize',14)

box on

h = colorbar;

h.Label.String = 'U';

axis square

box on

figure(6)

set(gcf,'units','normalized','position',[0.65 0.1 0.3 0.32]);

contourf(xx,yy,Ut,20);

shading interp

xlabel('x'); ylabel('y');

title('Gradient solution','fontweight','normal');

set(gca,'fontsize',14)

box on

h = colorbar;

h.Label.String = '| Up |';

axis equal

end

fprintf('Saving checkpoint');

t = t+1;

save('checkpoint.mat');

disp(' ')

disp(' ')

end

disp(' ')

disp('Would you like to select another case or quit!?')

disp(' # 1) to quit')

disp(' # 2) to select another case')

zz = abs(input('Selection # '));

if zz == 1;

main = 1;

break

elseif zz == 2;

lock = 0;

flag = 0;

marker1 = 1;

marker = 1;

disp(' ')

disp(' ')

disp(' ')

disp(' ')

t = 0;

end

case 2

while t ~= 3;

if t == 1;

F = sin(pi.\*(xx-ax)/(bx-ax)).\*cos((pi/2).\*(2.\*((yy - ay)./(by - ay))+1));

elseif t == 2;

F = zeros(Ny,Nx);

U = zeros(Ny,Nx);

U(1,2:Nx-1) = gb(1,2:Nx-1);

U(Ny,2:Nx-1) = fb(Ny,2:Nx-1);

U(2:Nx-1,1) = uw(2:Nx-1,1);

end

%Round two at loops

bound = 1; n = 0;

e = 1;

w = 2/(1+sin(pi\*hx/(2\*pi))); %it is less than 2 thus okay

while e > XX;

Up = U;

for i = 2:Nx-1;

for j = 2:Ny-1;

if bound == 1;

U(2:Ny-1,Nx) = (1/4)\*(2\*U(2:Ny-1,Ny-1)+U([2:Ny-1]-1,Ny)+U((2:Ny-1)+1,Ny)+(hx^2)\*F((2:Ny-1),Ny));

bound = bound +1 ;

U(1,1)= (U(1,2)+U(2,1))/2;

U(1,Nx)= (U(1,Nx-1)+U(2,Nx))/2;

U(Ny,1)= (U(Ny-1,1)+U(Ny,2))/2;

U(Ny,Nx)= (U(Ny,Nx-1)+U(Ny-1,Nx))/2;

end;

U(i,j) = (1-w)\*Up(i,j)+(w/4)\*( Up(i+1,j)+U(i-1,j)+ Up(i,j+1)+ U(i,j-1) + (hx^2)\*F(i,j));

end

end

E = U - Up;

e = mean(mean(E(2:Nx-1,2:Nx-1).^2));

n = n+1;

[Uxx, Uyy] = gradient(U,hx,hy);

Uxx = -Uxx; Uyy = -Uyy;

Ut = sqrt(Uxx.^2 + Uyy.^2);

end

if t == 1;

disp(['The error is ',num2str(e)])

disp(['Using the SOR Method, this problem took ',num2str(n),' iterations to solve.'])

disp(' ')

figure(1) % VECTOR FIELD U

set(gcf,'units','normalized','position',[0.02 0.52 0.3 0.32]);

mesh(xx,yy,U);

xlabel('x'); ylabel('y'); zlabel('U');

title('Solution using SOR Method','fontweight','normal');

set(gca,'fontsize',14);

rotate3d

box on

axis tight

h = colorbar;

h.Label.String = 'U';

view(55,49);

figure(2)

set(gcf,'units','normalized','position',[0.33 0.52 0.3 0.32]);

contourf(xx,yy,U,16);

pcolor(xx,yy,U);

shading interp

xlabel('x'); ylabel('y');

title('Heatmap of SOR solution','fontweight','normal');

set(gca,'fontsize',14)

box on

h = colorbar;

h.Label.String = 'U';

axis square

box on

figure(3)

set(gcf,'units','normalized','position',[0.65 0.52 0.3 0.32])

contourf(xx,yy,Ut,20);

shading interp

xlabel('x'); ylabel('y');

title('Gradient solutionusing SOR','fontweight','normal');

set(gca,'fontsize',14)

box on

h = colorbar;

h.Label.String = '| Up |';

axis equal

else

disp(['The error when F = 0 is ',num2str(e)])

disp(['Using the SOR Method when, this problem took ',num2str(n),' iterations to solve.'])

disp(' ')

figure(4) % VECTOR FIELD U

set(gcf,'units','normalized','position',[0.02 0.1 0.3 0.32]);

mesh(xx,yy,U);

xlabel('x'); ylabel('y'); zlabel('U');

title('Solution using SOR Method when F = 0 (Laplace Eqn)','fontweight','normal');

set(gca,'fontsize',14);

rotate3d

box on

axis tight

h = colorbar;

h.Label.String = 'U';

view(55,49);

figure(5)

set(gcf,'units','normalized','position',[0.33 0.1 0.3 0.32]);

contourf(xx,yy,U,16);

pcolor(xx,yy,U);

shading interp

xlabel('x'); ylabel('y');

title('Heatmap of solution','fontweight','normal');

set(gca,'fontsize',14)

box on

h = colorbar;

h.Label.String = 'U';

axis square

box on

figure(6)

set(gcf,'units','normalized','position',[0.65 0.1 0.3 0.32]);

contourf(xx,yy,Ut,20);

shading interp

xlabel('x'); ylabel('y');

title('Gradient solution','fontweight','normal');

set(gca,'fontsize',14)

box on

h = colorbar;

h.Label.String = '| Up |';

axis equal

end

fprintf('Saving checkpoint');

t = t+1;

save('checkpoint.mat');

disp(' ')

disp(' ')

end

if marker1 ~= 1;

disp(' ')

disp('Would you like to select another case or quit?')

disp(' # 1) to quit')

disp(' # 2) to select another case')

zz = abs(input('Selection # '));

if zz == 1;

main = 1;

return;

elseif zz == 2;

lock = 0;

flag = 0;

disp(' ')

disp(' ')

disp(' ')

disp(' ')

t = 0;

marker = 1;

end

elseif marker1 == 1;

flag = 0; lock = 0;

marker = 1;

marker1 = 0;

end

%%

case 3

%Gauss-Seidel goes first

F = zeros(Ny,Nx);

F = sin(pi.\*(xx-ax)/(bx-ax)).\*cos((pi/2).\*(2.\*((yy - ay)./(by-ay))+1));

%Round two at loops

bound = 1; n = 0;

e = 1;

while e > XX;

Up = U;

for i = 2:Nx-1;

for j = 2:Ny-1;

if bound == 1;

bound = bound +1 ;

U(1,1)= (U(1,2)+U(2,1))/2;

U(1,Nx)= (U(1,Nx-1)+U(2,Nx))/2;

U(Ny,1)= (U(Ny-1,1)+U(Ny,2))/2;

U(Ny,Nx)= (U(Ny,Nx-1)+U(Ny-1,Nx))/2;

end;

if j == Ny-1;

U(i,Nx) = (1/4)\*(2\*U(i,Ny-1)+U(i-1,Ny)+U(i+1,Ny)+(hx^2)\*F(i,Ny));

end

U(i,j) = (0.25)\*(Up(i+1,j)+U(i-1,j)+Up(i,j+1)+U(i,j-1)+(hx^2)\*F(i,j));

end

end

E = U - Up;

e = mean(mean(E(2:Nx,2:Nx).^2));

n = n+1;

[Uxx, Uyy] = gradient(U,hx,hy);

Uxx = -Uxx; Uyy = -Uyy;

Ut = sqrt(Uxx.^2 + Uyy.^2);

end

disp(['The error is ',num2str(e)])

disp(['Using the Gauss Seidel Method, this problem took ',num2str(n),' iterations to solve.'])

disp(' ')

figure(1) % VECTOR FIELD U

set(gcf,'units','normalized','position',[0.02 0.52 0.3 0.32]);

mesh(xx,yy,U);

xlabel('x '); ylabel('y '); zlabel('U');

title('Solution using Gauss Seidel Method','fontweight','normal');

set(gca,'fontsize',14);

rotate3d

box on

axis tight

h = colorbar;

h.Label.String = 'U ';

view(55,49);

figure(2)

set(gcf,'units','normalized','position',[0.33 0.52 0.3 0.32]);

contourf(xx,yy,U,16);

shading interp

xlabel('x '); ylabel('y ');

title('Heatmap of solution','fontweight','normal');

set(gca,'fontsize',14)

box on

h = colorbar;

h.Label.String = 'U';

axis square

box on

figure(3)

set(gcf,'units','normalized','position',[0.65 0.52 0.3 0.32])

contourf(xx,yy,Ut,20);

shading interp

xlabel('x '); ylabel('y ');

title('Gradient solution','fontweight','normal');

set(gca,'fontsize',14)

box on

h = colorbar;

h.Label.String = 'U';

axis equal

fprintf('Saving checkpoint');

t = t+1;

save('checkpoint.mat');

disp(' ')

disp(' ')

%SOR Method

U = zeros(Ny,Nx);

U(1,2:Nx-1) = gb(1,2:Nx-1);

U(Ny,2:Nx-1) = fb(Ny,2:Nx-1);

U(2:Nx-1,1) = uw(2:Nx-1,1);

F = zeros(Ny,Nx);

F = sin(pi.\*(xx-ax)/(bx-ax)).\*cos((pi/2).\*(2.\*((yy - ay)./(by-ay))+1));

%Round two at loops

bound = 1; n1 = 0;

e = 1;

w = 2/(1+sin(pi\*hx/(2\*pi)));

while e > XX;

Up = U;

for i = 2:Nx-1;

for j = 2:Ny-1;

if bound == 1;

bound = bound +1 ;

U(1,1)= (U(1,2)+U(2,1))/2;

U(1,Nx)= (U(1,Nx-1)+U(2,Nx))/2;

U(Ny,1)= (U(Ny-1,1)+U(Ny,2))/2;

U(Ny,Nx)= (U(Ny,Nx-1)+U(Ny-1,Nx))/2;

end;

if j == Ny-1;

U(i,Nx) = (1/4)\*(2\*U(i,Ny-1)+U(i-1,Ny)+U(i+1,Ny)+(hx^2)\*F(i,Ny));

end

U(i,j) = (1-w)\*Up(i,j)+(w/4)\*( Up(i+1,j)+U(i-1,j)+ Up(i,j+1)+ U(i,j-1) + (hx^2)\*F(i,j));

end

end

E1 = U - Up;

e = mean(mean(E1(2:Nx,2:Nx).^2));

n1 = n1+1;

[Uxx, Uyy] = gradient(U,hx,hy);

Uxx = -Uxx; Uyy = -Uyy;

Ut = sqrt(Uxx.^2 + Uyy.^2);

end

disp(['The error is ',num2str(e)])

disp(['Using the SOR Method, this problem took ',num2str(n1),' iterations to solve.'])

disp(' ')

figure(4) % VECTOR FIELD U

set(gcf,'units','normalized','position',[0.02 0.1 0.3 0.32]);

mesh(xx,yy,U);

xlabel('x '); ylabel('y '); zlabel('U');

title('Solution using SOR Method','fontweight','normal');

set(gca,'fontsize',14);

rotate3d

box on

axis tight

h = colorbar;

h.Label.String = 'U [ U ]';

view(55,49);

figure(5)

set(gcf,'units','normalized','position',[0.33 0.1 0.3 0.32]);

contourf(xx,yy,U,16);

shading interp

xlabel('x '); ylabel('y ');

title('Heatmap of solution','fontweight','normal');

set(gca,'fontsize',14)

box on

h = colorbar;

h.Label.String = 'U [ U ]';

axis square

box on

figure(6)

set(gcf,'units','normalized','position',[0.65 0.1 0.3 0.32]);

contourf(xx,yy,Ut,20);

shading interp

xlabel('x '); ylabel('y ');

title('Gradient solution','fontweight','normal');

set(gca,'fontsize',14)

box on

h = colorbar;

h.Label.String = '| Up |';

axis equal

z = n-n1;

disp('Based off these values....')

disp('')

disp(['The SOR method took ',num2str(z),' less iterations to solve'])

disp(' ')

end

if marker ~= 1;

disp(' ')

disp('Would you like to select another case or quit?')

disp(' # 1) to quit')

disp(' # 2) to select another case')

zz = abs(input('Selection # '));

if zz == 1;

main = 1;

return;

elseif zz == 2;

lock = 0;

flag = 0;

disp(' ')

disp(' ')

disp(' ')

disp(' ')

end

elseif marker == 1;

flag = 0; lock = 0; marker = 0;

end

end

%So you don't have to delete it later when you see it on the desktop or

%wherever

delete checkpoint.mat