# MECH 510

# PROGRAMMING ASSIGNMENT 1

Prepared for:

Prof. Carl Ollivier-Gooch

CEME 2050, UBC

Prepared by:

Nicholas Earle

UBC # 21943600

Date:

October 10, 2018

## OBJECTIVE:

The objective of this assignment was to write a program that executes the Point Gauss Seidel (PGS) method for solving partial differential equations (PDEs) on a specified domain. The program was then modified to perform sequential over-relaxation (SOR). Initially both programs were used to solve a domain for which the solution was known both to see if the program works and to see how the discretization error changes when user finer meshes. Finally, this program was tested by applying to a different problem given different initial conditions as well as a source term to see if it indeed converged on a solution. And then estimate the error and uncertainty of the answer.

## PART 1:

Solving a Laplace equation. For the initial formulation a two-dimensional domain with the dimensions 1 x 1 was used. This domain had the boundary conditions given below:

**TABLE 1: BOUNDARY CONDITIONS**

| | |
|---|---|
| Top (y = 1) | $T = \cos(\pi x)$ |
| Bottom (y = 0) | $T = 0$ |
| Left (x = 0) | $\frac{\partial T}{\partial x} = 0$ |
| Right (x = 1) | $\frac{\partial T}{\partial x} = 0$ |

To implement these boundary conditions a series of ghost cells were used. The values of the ghost cells were then calculated using one of the two ways below (each modified for the boundary being calculated):

**TABLE 2: GHOST CELL VALUES**

| | |
|---|---|
| For BC: T | $T = \frac{T_{i,j} + T_{i,j-1}}{2}$ |
| For BC: $\frac{\partial T}{\partial n}$ | $\frac{\partial T}{\partial n} = \frac{T_{i,j} - T_{i,j-1}}{\Delta n/2}$ |

Following this each of the interior cells can now be solved first by using just PGS as follows:

**EQUATION 1: POINT GAUSS SEIDEL**

$$\bar{T}_{i,j}^{k+1} = \frac{\Delta y^2}{2(\Delta x^2 + \Delta y^2)}\left(\bar{T}_{i+1,j}^{k} + \bar{T}_{i-1,j}^{k+1}\right) + \frac{\Delta x^2}{2(\Delta x^2 + \Delta y^2)}\left(\bar{T}_{i,j+1}^{k} + \bar{T}_{i,j-1}^{k+1}\right) - S_{i,j}\frac{\Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)}$$

For this situation where $\Delta x = \Delta y$, this equation can be simplified to simply the average of all of the surrounding cells plus the source term. However, in the code, it was left as is to experiment with changing the cell sizing and dimensions. Next, the sequential over-relaxation was incorporated into the code as follows:

**EQUATION 2: POINT GAUSS SEIDEL W/ OVER RELAXATION**

$$\delta T_{i,j}^{k+1} = \frac{\Delta y^2}{2(\Delta x^2 + \Delta y^2)}\left(\bar{T}_{i+1,j}^{k} + \bar{T}_{i-1,j}^{k+1}\right) + \frac{\Delta x^2}{2(\Delta x^2 + \Delta y^2)}\left(\bar{T}_{i,j+1}^{k} + \bar{T}_{i,j-1}^{k+1}\right) - S_{i,j}\frac{\Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)} - \bar{T}_{i,j}^{k}$$

$$\bar{T}_{i,j}^{k+1} = \bar{T}_{i,j}^{k} + \omega \delta T_{i,j}^{k+1}$$

These two equations were used to calculate the solutions for a series of meshes which were then compared to the exact solution given by:

**EQUATION 3: EXACT SOLUTION**

$$T(x,y) = \frac{\cos(\pi x)\sinh(\pi y)}{\sinh(\pi)}$$

## RESULTS:

The first test case was to run the program on a 10 x 10 mesh starting with an initial solution of zero and running until the maximum change in any given cell is less that $10^{-7}$. Comparing this solution to the exact solution produced the error as shown by Figure 1, below:

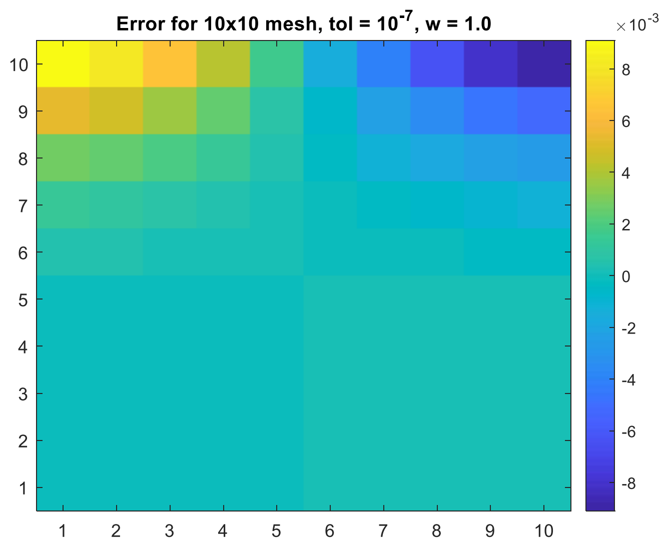


**FIGURE 1: ERROR FOR 10X10 MESH (NO SOR)**

This solution converged in 210 iterations with an $L_2$ norm of the error of 0.0023. Looking at the image above, the error is much greater in the top left and right corners. Although, this is to be expected as the exact solution is close to zero all along the bottom half, so starting from an initial solution of zero everywhere, these cells are already very close to the exact solution and just get refined with every iteration while the top cells start much further away.

The second test case incorporated sequential over-relaxation into the code, with an over-relaxation factor of $\omega = 1.5$, which is expected to significantly decrease the number of iterations until the solution converges. The error for this test is shown in Figure 2, below:

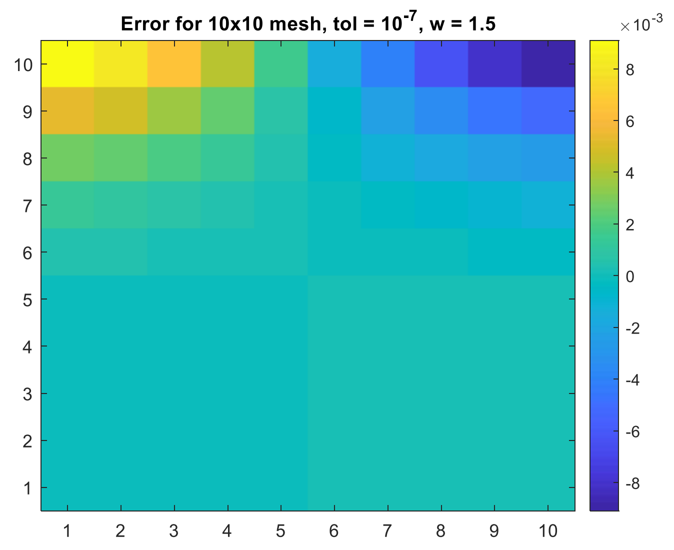


**FIGURE 2: ERROR FOR 10X10 MESH, W/ SOR, W=1.5**

And as expected the number of iterations dropped to 93, taking less than half the time to complete, producing almost identical result. That being said, the $L_2$ norm also did not change, with the significance at which it was calculated.

The third test was to run the program for a 20 x 20 mesh altering the value of $\omega$ to 1.0, 1.3, 1.5. This was done to see how the over-relaxation factor affects how the solution is changing during each iteration. Figure 3, below, shows the maximum change of any cell in the mesh for each iteration, up to a maximum difference of $10^{-7}$. It is shown on a semi-log plot so that it is clear that as the over-relaxation factor is increased the solution converged much faster. Although, this is not necessarily true as $\omega$ increases indefinitely. In the handout notes it is stated that the limit on $\omega$ is 2, however upon investigation, the practical limit of this code is $\omega \sim 1.8$. After this point the solution became unstable, running many thousand times until it was satisfied. As for $\omega = 1.8$, the solution converged in 148 iterations, almost twice as quickly as the 290 iterations needed for $\omega = 1.5$.

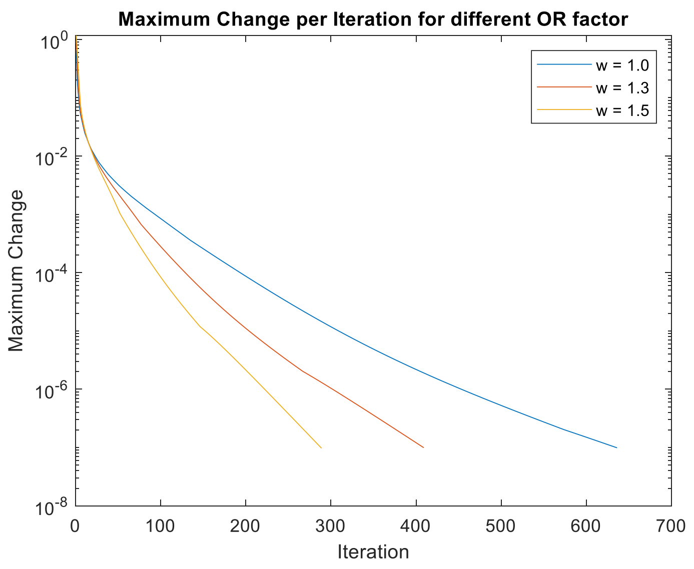


**FIGURE 3: MAX CHANGE PER ITERATION, DIFFERENT OR FACTOR**

The fourth and final test case was to run the program for additional meshes of 40 x 40 and 80 x 80, then tabulate the solutions and errors to find the actual order of accuracy for the code. The following table shows the $L_2$ norm for each test case, along with some other parameters. The ones in bold are used below to calculate the order of accuracy. These were all calculated to a tolerance of $10^{-10}$.

**TABLE 3: SOLUTION RESULTS FOR DIFFERENT MESHES**

| Mesh Size | Tolerance | OR | Iterations | $L_2$ Norm |
|---|---|---|---|---|
| 20 x 20 | $10^{-7}$ | 1.0 | 637 | 6.2322e-04 |
| | $10^{-10}$ | 1.5 | 487 | **6.2323e-04** |
| 40 x 40 | $10^{-7}$ | 1.0 | 1888 | 1.6066e-04 |
| | $10^{-10}$ | 1.5 | 1649 | **1.5924e-04** |
| 80 x 80 | $10^{-7}$ | 1.0 | 5854 | 5.9165e-05 |
| | $10^{-7}$ | 1.5 | 2548 | 4.8341e-05 |
| | $10^{-10}$ | 1.5 | 5546 | **4.0154e-05** |

The order of accuracy can be found by calculating the slope of the line made by the $L_2$ norms on a log-log plot. That is:

**EQUATION 4: ORDER OF ACCURACY**

$$k = \frac{log(L_{2_2}/L_{2_1})}{log(\Delta x_2/\Delta x_1)} = \frac{log(4.0154e-05/1.5924e-04)}{log(\frac{1}{80}/\frac{1}{40})} = \mathbf{1.9876}$$

So, for all intents and purposes, the code is second order accurate, which is as expected based on the error analysis and Taylor expansions done in class. The images shown in Figure 4, below, show how the resolution of the solution improves as the mesh becomes finer all compared to the exact solution on an 80x80 grid.

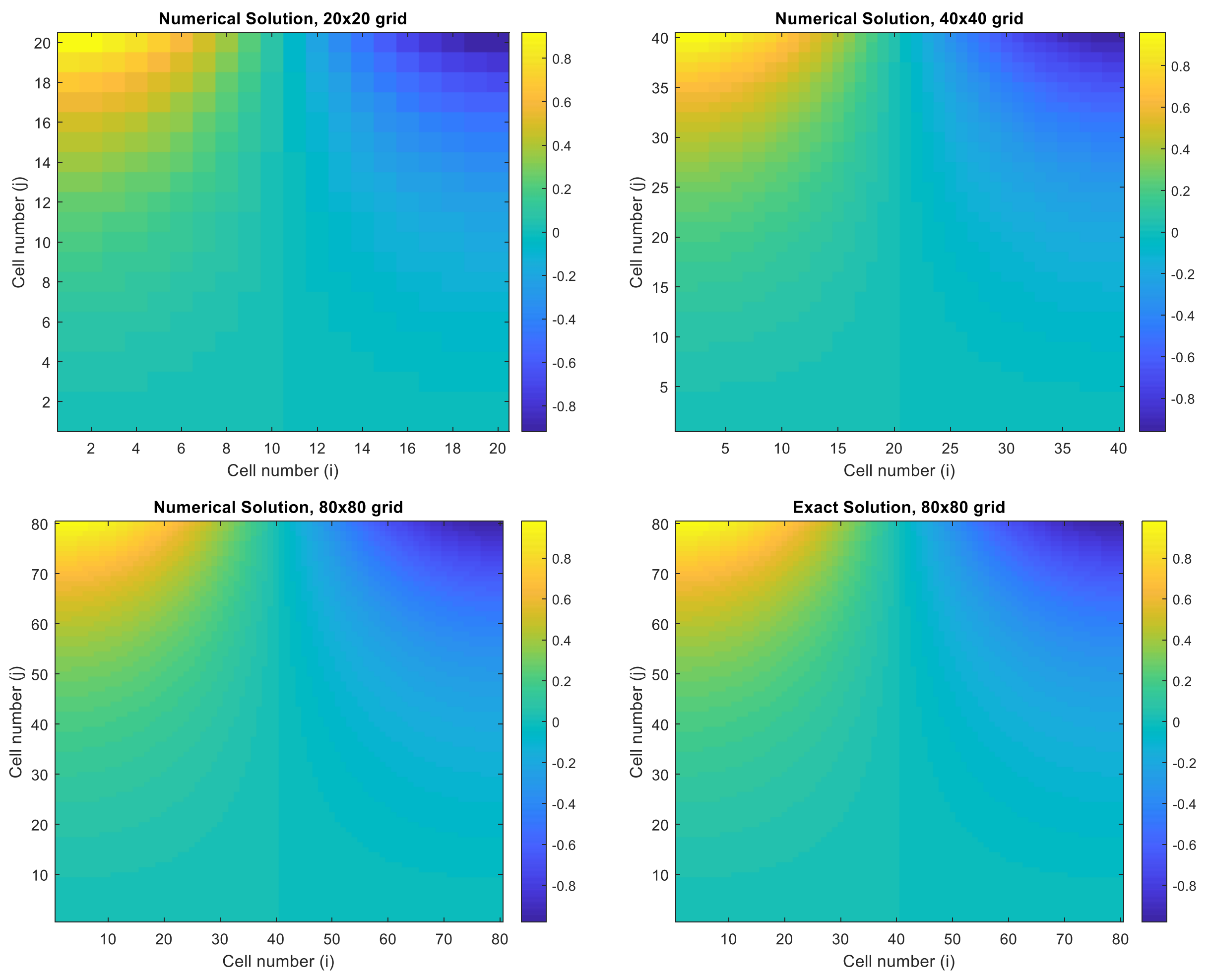


**FIGURE 4: SOLUTION RESOLUTION**

## PART 2:

Solving a Poisson equation. Taking the code as was formulated in part 1, it was modified to suit the given Poisson equation. That is a source term was added, and the parameter in question was changed from temperature to pressure. The equation to be solved was:

**EQUATION 5: POISSON**

$$\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} = -\left(\left(\frac{\partial u}{\partial x}\right)^2 + 2\frac{\partial u}{\partial x}\frac{\partial v}{\partial y} + \left(\frac{\partial v}{\partial y}\right)^2\right)$$

With the following velocities:

**EQUATION 6: VELOCITIES**

$$u = x^3 - 3xy^2$$
$$v = -3x^2y + y^3$$

Substituting $u$ and $v$ into the equation above, then expanding and cancelling, reduces to:

**EQUATION 7: POISSION W/ SOURCE**

$$\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} = -18(x^2 + y^2)^2$$

The full calculation can be found in the appendix. The boundary conditions for the domain are as follows:

**TABLE 4: POISSON BOUNDARY CONDITIONS**

| | |
|---|---|
| Top (y = 1) | $P = 5 - \frac{1}{2}(1 + x^2)^3$ |
| Bottom (y = 0) | $\frac{\partial P}{\partial y} = 0$ |
| Left (x = 0) | $\frac{\partial P}{\partial x} = 0$ |
| Right (x = 1) | $P = 5 - \frac{1}{2}(1 + y^2)^3$ |

The ghost cell values can be calculated using the formulas found in Table 2, above.

## RESULTS:

With the parameters outlined above the following results were calculated. Figure 4, below, shows a plot of the solution on an 80 x 80 grid.

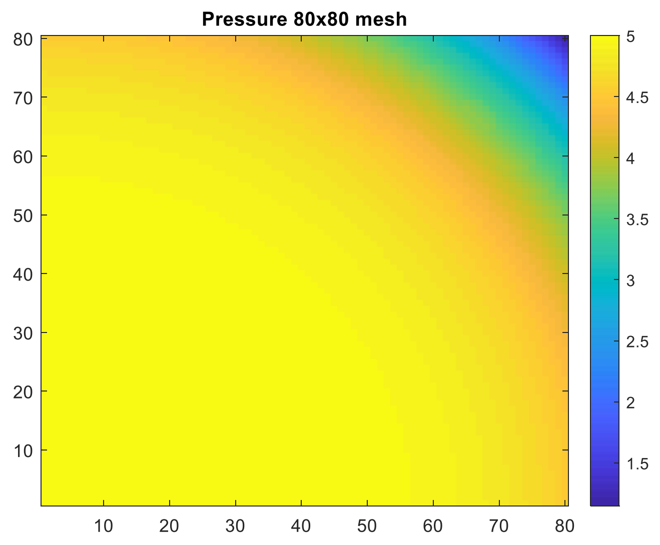


**FIGURE 5: PRESSURE 80X80**

Given the source term having the form of an upside-down circular paraboloid centred at the origin, this solution is as expected, where the origin is the high point and all the surrounding values decrease as the radius increases.

Next the value of P(1/2,1/2) was calculated on each grid. The point in each case is located at the corner of four cells, so the value was taken to be the average of all the surrounding cells.

**TABLE 5: P(1/2,1/2)**

| Grid | P(1/2,1/2) |
|---|---|
| 10 x 10 | 4.81406 |
| 20 x 20 | 4.88991 |
| 40 x 40 | 4.91645 |
| 80 x 80 | 4.92758 |

Next, using the method outlined in the JFE handout, an error bound was calculated for this value. For these three values were needed so the value found on the 10x10 grid was excluded. The following table outlines the results of this with the appropriate error bounds.

**TABLE 6: ERROR BOUND RESULTS**

| | $\phi = pressure\ at\ (1/2\,,1/2)$ |
|---|---|
| $N_1, N_2, N_3$ | $6400, 1600, 400$ |
| $r_{21} = r_{32}$ | $2$ |
| $\phi_1, \phi_2, \phi_3$ | $4.92758, 4.91645, 4.88991$ |
| $p$ | $1.25371$ |

| | |
|---|---|
| $\phi_{ext}^{21}$ | 4.93871 |
| $e_a^{21}$ | 0.0022587 |
| $e_{ext}^{21}$ | 0.0022536 |
| $GCI_{fine}^{21}$ | 0.0028233 |

So, for this problem it can be said that for the 80 x 80 grid, the solution of P(1/2,1/2) is 4.92758 ±0.28%. Looking at the change in the solution from grid to grid, this small error bound does make sense as it seems the solution is converging very near the exact solution, whatever that may be (running with a finer mesh until crashing the computer never exceeded ~4.93).

## CONCLUSION

To conclude, while this program is a very simple example of what the engine of a CFD solver looks like, it does do a good job at representing how solutions are calculated and what can be expected as results. On top of that, it also shows that many times, what one would hope to be the converged solution is no where near the actual solution. This can be caused by any number of things especially a simple error in problem input or simply running a loop on the wrong bounds. For both the Laplace test cases and the Poisson example, the code was able to produce solutions that converged well and as expected. While, this code is still very elementary, with some more tweaks hopefully it can become much more robust with the ability to solve some more realistic problems.

## Appendix

$$\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} = -\left(\left(\frac{\partial u}{\partial x}\right)^2 + 2\frac{\partial v \partial u}{\partial x \partial y} + \left(\frac{\partial v}{\partial y}\right)^2\right)$$

$u = x^3 - 3xy^2$

$V = -3x^2y + y^3$

$$\frac{\partial u}{\partial x} = 3x^2 - 3y^2 \qquad \frac{\partial u}{\partial y} = -6xy$$

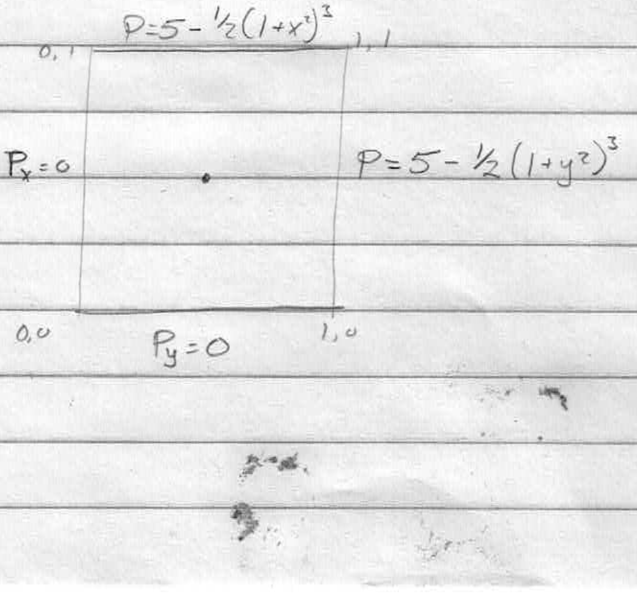
$$\frac{\partial v}{\partial x} = -6xy \qquad \frac{\partial v}{\partial y} = -3x^2 + 3y^2$$

$$\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} = -\left(\left(3x^2 - 3y^2\right)^2 + 2\left(-6xy\right)\left(-6xy\right) + \left(-3x^2 + 3y^2\right)^2\right)$$

$$= \left(\left(9x^4 - 18x^2y^2 + 9y^4\right) + 72x^2y^2 + \left(9x^4 - 18x^2y^2 + 9y^4\right)\right)$$

$$= -18\left(x^4 + 2x^2y^2 + y^4\right)$$

$$S \quad = -18\left(x^2 + y^2\right)^2$$



$P = 5 - \frac{1}{2}(1+x^2)^3$
0, 1
1, 1
$P_x = 0$
$P = 5 - \frac{1}{2}(1+y^2)^3$
0, 0
1, 0
$P_y = 0$