James Hahn
CS1645
High Performance Computing

Verification

Below are the results of the verification test with source term $S(x, y) = xe^y$ and boundary conditions $T(0, y) = 0$, $T(2, y) = 2e^y$, $T(x, 0) = x$, and $T(x, 1) = xe$. The exact solution is $T = xe^y$. The contour map (Figure 1) can be found below plotted on the domain $0 \leq x \leq 2$ and $0 \leq y \leq 1$. A convergence rate of 10^{-12} was used on an iteration-by-iteration basis. These are the results from the Jacobi algorithm.

Grid Size	**L-∞ Error**
5x5	1.0929×10^{-3}
11x11	1.8886×10^{-4}
21x21	4.8004×10^{-5}
41x41	1.2026×10^{-5}
51x51	7.6982×10^{-6}
101x101	1.9238×10^{-6}
201x201	4.7511×10^{-7}

The discretization is verified since the above tests were ran on successively finer grids. The algorithm was also tested on odd-dimension grids, such as 11x5, 21x11, and 41x21. The algorithm still worked and converged.

Figure 1
Contour Map of verification test

Problem Solving

With boundary conditions T(0, y) = 0, T(1, y) = 0, T(x, 0) = 0, and T(x, 1) = 0, as well as source term S(x, y) = 0.2 on the domain $0 \leq x \leq 1$ and $0 \leq y \leq 1$, the below contour map for the problem solving portion of this project can be found below in Figure 2. Results are from the Jacobi algorithm.

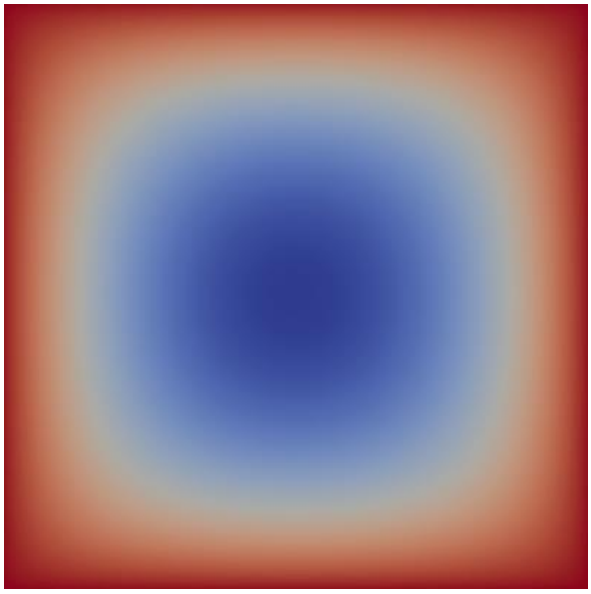

Figure 2
Contour Map of problem solving test

To show convergence, the point x = 0.1, y = 0.1 was monitored on a step-by-step process with convergence rate of 10^{-12}. Below, you can find the table of values for the coordinate (0.1, 0.1) for each grid size.

Grid Size	**Value of (0.1, 0.1)**
5x5	-0.00859375
11x11	-0.00256262
21x21	-0.00260024
41x41	-0.00261068
51x51	-0.00261197
101x101	-0.00260244

Although I do not have the computational resources available to me to continue increasingly higher grid sizes (at the cost of waiting for much larger grid sizes to finish), the above values of (0.1, 0.1) have briefly showcased the convergence of its true value to around -0.0026, or very slightly above it.

Performance

As grid size increases, so did the time taken to solve Poisson's equation for the verification test. Times were not recorded for the problem solving example because we only desire to monitor the algorithm and its effect on time as the grid size increases regardless of the problem. The grid sizes were pushed to a decent amount. Initially, only grid sizes up to 201x201 were requested. The below table displays additional grid sizes up to 50001x50001 and a jump from ~20 seconds at 10001x10001 to ~560 seconds at 50001x50001 was observed. No further grid sizes were observed because the predicted time will most likely surpass an hour.

Grid Size	Time (seconds)
5x5	0.00
11x11	0.01
21x21	0.12
41x41	1.41
51x51	3.25
101x101	46.51
201x201	681.19
301x301	4109.85

Algorithm Presentation

Clearly, the project handout contains a brief, general, vague description of the algorithm and Poisson equation. After further investigation, the Jacobi algorithm was implemented for this project to solve any solution given the source term, domain constraints, and boundary conditions. Later on, the Gauss-Seidel algorithm was implemented. Results are shown in the next section comparing the two algorithms.

As noted in the handout, the second derivative can be used to solve the true values of the solution. It turns out, the following equation, after solving for $T_{i, j}$, can be used to approximate the desired value:

$$(T_{i-1, j} - 2T_{i, j} + T_{i+1, j}) / (\Delta x)^2 + (T_{i, j-1} - 2T_{i, j} + T_{i, j+1}) / (\Delta y)^2 = S_{i, j}$$

Which ends up being:

$$[(\Delta y)^2 T_{i-1, j} + (\Delta y)^2 T_{i+1, j} + (\Delta x)^2 T_{i, j-1} + (\Delta x)^2 T_{i, j+1} - (\Delta y)^2 (\Delta x)^2 S_{i, j}] / (2(\Delta y)^2 + 2(\Delta x)^2) = T_{i, j}$$

So, this above equation was used to approximate the value of $T_{i, j}$ on each iteration, based on its neighbors and underlying source term, which was always provided for all coordinates.

An important note is this algorithm does not iterate over the boundaries because they are important boundary conditions to keep the algorithm stable. As such, $T_{i, j}$ is only a concern for all interior points of the grid.

In the end, to find the error between the 'solved' grid and the 'true' grid, given in the verification test, we calculated the L-∞ norm (maximum difference between 'solved' value and 'true' value in the grid).

Satisfactory results were generated, as the grid both converged and produced useful contour maps related to the underlying solution values.

Jacobi vs. Gauss-Seidel (Bonus)

In this project, Jacobi was initially implemented, but Gauss-Seidel was implemented later on. The main difference between the two algorithms is the use of memory. In Jacobi, the algorithm stores an array A, then updates all the values from iteration i to find all the new values for iteration i+1. In Gauss-Seidel, all values in iteration i are generated, but in iteration i+1, all values are constantly updated, so the matrix is a mixture of both old and new values. Gauss-Seidel is in-place memory and Jacobi is not in-place.

The below table showcases their differences in times on various grid sizes:

Grid Size	**Jacobi time (seconds)**	**Gauss-Seidel time (seconds)**
5x5	0.00	0.00
11x11	0.01	0.05
21x21	0.12	0.06
41x41	1.41	0.72
51x51	3.25	1.62
101x101	46.51	26.21