Object-Oriented Numerical Modeling I Final Report
Dirichlet Problem: Laplace

Andrea Smith
Brian Yadamec

Statement of Problem

We want to create a class that is able to handle the solving of a given Dirichlet partial differential equation through either the Jacobi or the Gaussian method. The class should be able to handle a variety of Dirichlet problems and should take steps to optimize the creation of the solution vector. It should be simple to change the mesh size of the problem solution and reasonable to change the boundary functions.

Justification

The Dirichlet problem can be represented as the matrix problem Ax = b. The size of the matrix and vectors is determined by an N value that represents the accuracy of the approximation to be made. There is a mapping pattern that can be used to populate the value of the b vector, given four boundary functions and an error function. Similarly, the matrix follows a pattern in its values that can be calculated.

Once these values are created, then normal matrix solution methods can be used, such as Gaussian Elimination and the Jacobi Iterative method. Because the patterns of the matrix and b vector are known, some modifications to the methods can be made more efficient.

Methodology

In order to solve the Dirichlet problem, we created a specialized matrix class. This matrix class was optimized for the Dirichlet problem since the matrix follows a pattern. In order to enforce the integrity of the matrix, we are able to restrict the ability to change the values in the matrix. This also removes the necessity to actually store the entire matrix. Since the values of the matrix follow a predictable pattern, the value at any given point can be calculated when it is needed. This means that the only data that the matrix must hold is the N value.

To actually handle the solving of the Dirichlet problem is a Solver class. This class takes five callback functions, which represent the four boundary functions and the g function. These five equations then represent the problem that is to be solved. This class has two member functions for solving the matrix equation using Gaussian Elimination and Jacobi Iteration. Both of these functions take an N parameter for the granularity of the

mesh size to use while solving. The Jacobi method then creates a Dirichlet matrix of the appropriate size and populates the b vector of the equation, using the supplied callback functions. It then passes the matrix and vector to an instance of the Jacobi functor to solve. The Gaussian member function works similarly. It creates the b vector from the callback functions in the same manner; however, the Gaussian matrix requires the ability to modify the matrix, which means that the created Dirichlet matrix gets copied into a dense matrix before being passed to a Gaussian functor.

The Jacobian functor created a specialized instance for when it was working with a Dirichlet matrix. We took out operations that are redundant while working with Dirichlet matrices. This included removing the multiplication of the large quantity of zeros that exist in a Dirichlet matrix, as well as removing the multiplication of the matrix diagonal by a vector, since the diagonal is similar to the identity matrix, and would thus return the vector itself.

Solution

We ran three trials of the program at four different levels of N: 10, 30, 60, and 100. The times reported from the chrono clock was identical at N set to 10, but the difference in performance was immediately apparent at N of 30. The Jacobian method ran four times longer than the Gaussian method did at all levels of N greater than 10. A chart of the average runtimes and a graph for each level of N for both Jacobi and Gauss exists in Appendix A.

Viability

The Gaussian Elimination method appears to have a similar growth rate as the Jacobian method as N gets larger, but also starts at a much lower base than the Jacobian method does. This means that in all examined cases, the Gaussian method executed in a shorter timeframe than Jacobi did. The two methods did return similar values, but when they were graphed it became apparent that they were not identical. Gaussian Elimination tended to create a bulge in the center of the field, while Jacobi tended to create a dip in the middle of the graph. Jacobi also has another mark against it as a viable solution, as when N was 100, the method produced a discontinuity in the graph, dropping an edge down to nearly zero, rather than the expected value of nearly one.

Appendix A

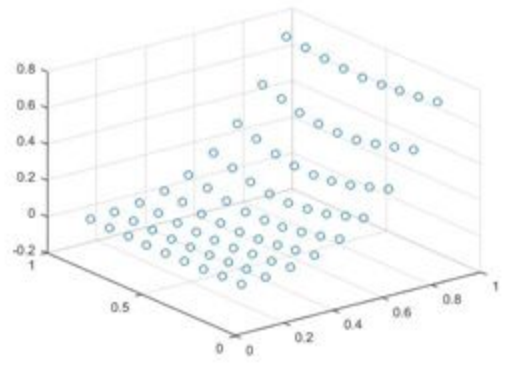

Figure 1: Gaussian N=10

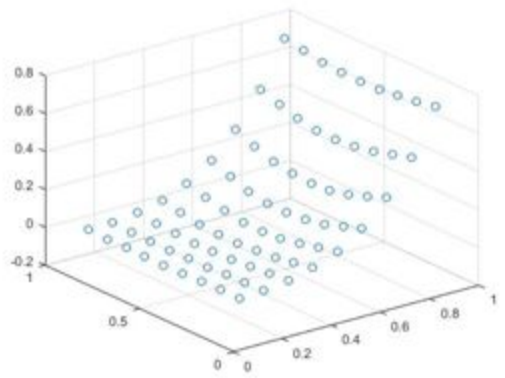

Figure 2: Jacobi N=10

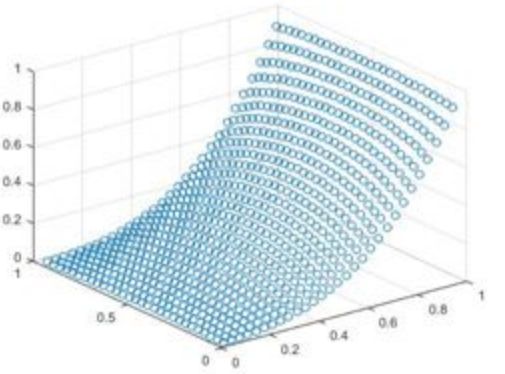

Figure 3: Gaussian N=30

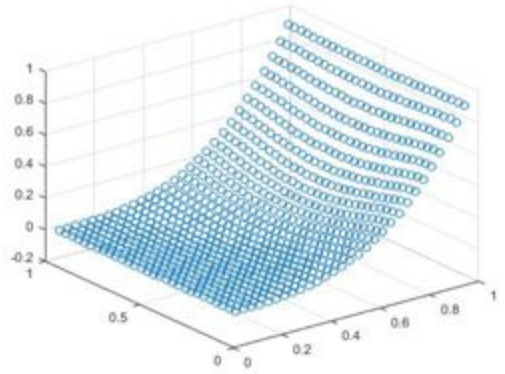

Figure 4: Jacobi N=30

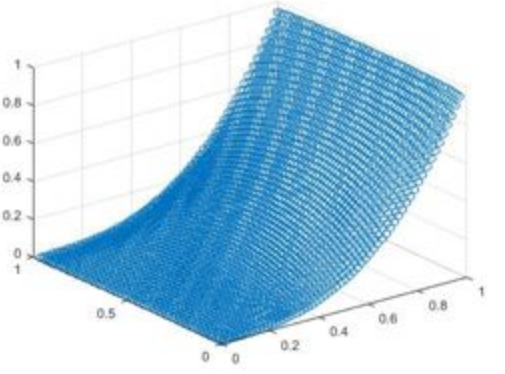

Figure 5: Gaussian N=60

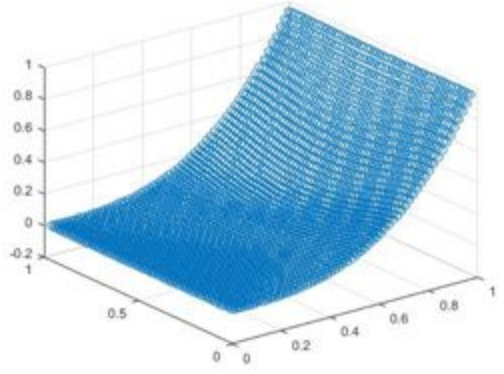

Figure 6: Jacobi N=60

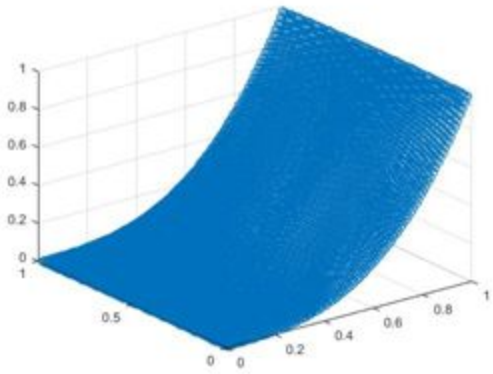

Figure 7: Gaussian N=100

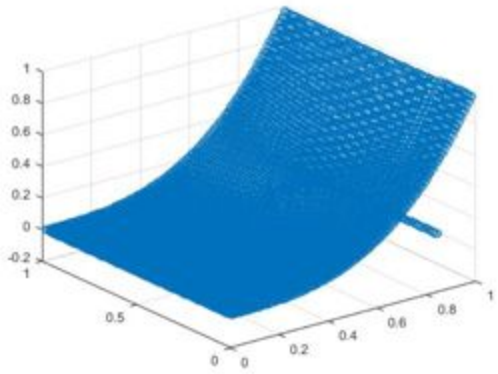

Figure 8: Jacobi N=100