**FINAL PROJECT:**

**SOLVING POISSON’S EQUATION**

Submitted to

Mr. Clayton Price

Professor

Missouri University of Science and Technology

Prepared by

Matt Lindsay and George Mausshardt

May 12, 2012

Contents

[Problem Statement 3](#_Toc356039307)

[Mathematical Background 3](#_Toc356039308)

[Methodologies 5](#_Toc356039309)

[Mathematical Techniques 5](#_Toc356039310)

[Parameters 5](#_Toc356039311)

[Solution and Analysis 5](#_Toc356039312)

[Viability 5](#_Toc356039313)

[Conclusion 5](#_Toc356039314)

[APPENDIX A 5](#_Toc356039315)

# Problem Statement

In a great deal of scientific and mathematical research and discoveries, problems can be solved by using a partial differential equation to represent the entire system. Unfortunately, this means that there are also many independent variables that must be solved to determine the system’s state and solution. The software process described below attempts to simplify this process by estimating differential equations as systems of equations, which can then be solved quickly and effectively.

Although the mathematics behind this estimation will be discussed later, the idea to model a partial differential equation as a system of equations makes the problem easier not only for humans, but for computers as well. In fact, the speed of the algorithm matters so greatly that two different solvers for systems of equations have been implemented and analyzed in the software to test the efficiency of the differing methods. For the purpose of this report, the methods tested are Gaussian Elimination, a pivot method to find a single value to be used in backward substitution, and Gauss-Seidel, an iterative method that guesses and corrects itself ever more closely to the correct solution. These methods, the results of this analysis, and the subsequent comparisons resulting from the analysis are described in detail below.

The problem therefore, is to solve a partial differential equation and the method to be discussed involves defining a system of equations that approximate solutions close enough to the actual values to make the simulation worthwhile.

# Mathematical Background

Although the basics of the software were defined in the problem statement above, the underlying mathematics are somewhat deep and are further explained in the following section. In addition to the two methods used to solve the system of equations, this section also explains how solving a system of equations can effectively provide estimates for the much more complex partial differential equation.

First, the Gauss-Seidel method, as mentioned above, is an iterative method used to solve systems of equations. Although the method is relatively quick, especially since it is already designed as an improvement over Jacobi Iteration, the one major drawback to using Gauss-Seidel is that the method can only be expected to work on a system of equations that is diagonally dominant, although it may work on systems that do not meet this requirement. To clarify, a system of equation is diagonally dominant if the absolute values of the coefficients of the main left-to-right diagonal in the system are greater than or equal to the absolute values of the coefficients of all other entries in the same row of the matrix defining the system.

After ensuring that the system meets this requirement, Gauss-Seidel makes a guess at the values that will solve the system. After verifying that these values are not correct, the estimations received are stored, one at a time, into the unknowns and the process is repeated, overwriting one variable at a time until the solution vector contains data close enough to the real solution that the two are functionally equivalent. The test for “equivalence” is done in this project by comparing vector norms of the last two iterations and declaring the method complete if the two norms differ by no more than a very small value set at the beginning of the Gauss-Seidel method.

The second method used to solve systems of equations is called Gaussian Elimination and belongs to the pivot class of solvers. Not only does this method work on any matrix (as opposed to Gauss-Seidel’s occasional failure on matrices that are not diagonally dominant), but it is fairly easy to understand and can be utilized by anyone with a basic knowledge of Linear Algebra. The method works by row-reducing a matrix into reduced-echelon form, where the major diagonals hold the value 1 and all other entries except the solution vector equal 0. For more information about this process, refer to Appendix A.

Upon the completion of the echelon reduction, the values stored in the solution vector are indeed the solutions to the equation. Note that this is the method used by a human doing the method, but the algorithm explained by this report merely creates an upper-triangular matrix and utilizes backward-substitution to find the values more quickly. This means that the first half of row-reduction is accomplished and the bottom left half of the matrix holds 0 values, but the top half still holds non-zero entries. However, this is all that is needed to solve the system, as the bottom row holds a single value that can be used to find one unknown, which can then be plugged into the row above to achieve a second value, propagating up the entire matrix to complete the algorithm.

Now that the linear system solvers have been described, the definition of the translation from partial differential equation to system of linear equations approximation becomes much easier to describe. Although the derivation of the technique is described in the methodologies section below, the actual process will be defined here.

To begin the description, consider that every equation in mathematics can be graphed and that differential equations can be graphed on a three dimensional plane. The information given in our approximation includes the values on the boundaries of this plane, such that we know for certain what the equation will return when an x or y value equals 0 or 1. Also consider that integration over a number of points can be approximated by evaluating a large number of equally spaced points on the equation and summing them together. With this information in mind, consider a matrix of all of these points in the estimation and realize that another very strong estimation would be to assume that each point is the average of the four points surrounding it in the matrix.

Now create a “mesh” with values in the middle of the mesh numbered starting at 1 in the lower left hand corner and incrementing by one every step to the right, eventually jumping to the next row and starting again at the far left. Using these numbers as indices in the matrix, create a coefficient matrix where each row contains the aggregate of the numbers used to estimate the sum, such that the right diagonal contains only 1’s and every point in the row where a column number matches an index in the mesh contains a -1/4, that point’s part of the estimation. The B Vector simply performs the summation, adding up all values on the outside edge inside of the vector, then multiplying by 0.25, the average, and subtracting off ((h^2) / 4) times the forcing function, the right hand side of the differential equation. Now that the matrix and vector are created, the only task left is to solve a system of equations, an incredibly simple process.

# Methodologies

The following section discusses the techniques used and inputs necessary to make the approximations and linear system solvers work as expected. The first section, Mathematical Techniques, explains how we derived the approximation method, how it works, the techniques used to make it quicker, while the second section describes the parameters and C++ techniques used in the solution described in this report.

## Mathematical Techniques

How did we derive the approximation? What techniques did we use to make this process easier?

## Parameters

The main parameter is the size of the “mesh.” This is passed into the Partial\_DiffEQ constructor, and its value should be (n-1)2. For example, for a mesh that is 4x4, a value of 9 should be passed in.

# Solution and Analysis

What are our results? SHOW GRAPHS! SHOW CHARTS! DO SOME EXCEL PLOTS! Which one is better? Compare iterations/row arithmetic and total runtime using the Analyzer.

# Viability

Why is this solution feasabile? Explain why our estimate is “good enough.”

# Conclusion

Restate the topic paragraph in different words.

# APPENDIX A

Describes the method of row-reduction for a matrix. Just copy and paste from the web (and cite the source!)