Homework 1: Finite Element Method in 1D

Kory Beach
Department of Physics, Astronomy, and Applied Physics,
Rensselaer Polytechnic Institute
`beachk2@rpi.edu`

October 3, 2016

0.1 Introduction

The aim of this report is to detail and compare the efficacy of three methods for solving second-order differential equations. We will do this by obtaining a solution for the equation

$$\frac{d^2y}{dx^2} - 6x^2 = 0 \tag{1}$$

in the range $x \in [0,1]$ with boundary conditions $y(x=0) = y(x=1) = 0$ using (1) an analytical approach, (2) the Gauss-Seidel relaxation method, and (3) a one-dimensional Finite Element method. We will discuss the implementation of these methods using C++ and compare the results for each of the algorithms involved.

0.2 Methods

0.2.1 The Analytic Solution

In order to test the validity of the two algortihmic approaches to solving second order differential equations of this form, we have chosen an example equation whose solution is obtainable by analytical means. By integrating both sides of the function twice one obtains

$$y(x) = \int \frac{dy}{dx}\,dx = \int (2x^3 + c_1)\,dx = \frac{x^4}{2} + c_1 x + c_2 \tag{2}$$

where c_1 and c_2 are constants of indefinite integration.

Given this general solution we can now apply the boundary conditions $y(x = 0) = y(x = 1) = 0$ to find that $c_1 = -\frac{1}{2}$ and $c_2 = 0$. Thus the closed-form solution for this equation given these boundary conditions is:

$$y(x) = \frac{x^4}{2} - \frac{x}{2} \tag{3}$$

0.2.2 The Gauss-Seidel Relaxation Method

The Gauss-Seidel relaxation method is an algorithm that uses the first-order forward and backward difference approximations to discretize the differential equation

$$y''(x) = 6x^2 \approx \frac{y(x+h) - 2y(x) + y(x-h)}{h^2} \tag{4}$$

where h is some small grid spacing. Rearranging for y(x) gives

$$y(x) = \frac{1}{2}(y(x+h) + y(x-h) - 6x^2h^2) \tag{5}$$

which, for a given position x is simply the arithmetic average of the nearest neighbors minus a source term. We can translate this equation to a more directly algorithmic form

$$y_i^k = \frac{1}{2}(y_{i+1}^{k-1} + y_{i-1}^{k} - 6x^2h^2) \tag{6}$$

where i is the index of an array element that contains the value of $y(x_i)$ and k is the iteration number. The insight that the Gauss-Seidel method provides is that it mixes into its calculation both old and (more accurate) new values for the neighbors in y, improving the speed of convergence to the Jacobi method which uses only old values.

In practice, this simply means that, for this one-dimensional problem, we can simply take the average of the left and right array elements to update element i and then move to the right and repeat until convergence.

0.2.3 The Finite Element Method

The finite element method is a technique by which the domain of the function is broken into a grid of many smaller equations that can be solved using matrix algebra. Starting from the differential equation we wish to solve, we can write

$$\frac{d^2y}{dx^2}\Phi(x) = 6x^2\Phi(x) \tag{7}$$

where we have introduced a trial function $\Phi(x)$ which vanishes at the boundaries $\Phi(0) = \Phi(1) = 0$. Integrating by parts on the left hand side yields

$$\int_0^1 \frac{d^2y}{dx^2}\Phi(x)\,dx = \frac{dy}{dx}\Phi(x)\Big|_a^b - \int_0^1 \frac{dy}{dx}\Phi'(x)\,dx = \int_0^1 6x^2\Phi(x)\,dx. \tag{8}$$

Noting that, by definition, the trial function vanishes at a and b we end up with

$$-\int_0^1 \frac{dy}{dx}\Phi'(x)\,dx = \int_0^1 6x^2\Phi(x)\,dx \tag{9}$$

which is known as the weak form of the differential equation.

The main reason for converting the differential equation into this form is so that we can break up the domain into many smaller domains and obtain a global solution that is consistent at the boundaries of all domains. This can be done by approximating the function $y(x)$ to be a linear combination of basis functions $\phi_i(x)$ such that

$$y(x) \approx \sum_{i=0}^{N-1} \alpha_i\phi_i(x) \tag{10}$$

where α_i indicates a set of numerical coefficients and the ϕ_i basis functions can (to a high degree of accuracy) be represented by piecewise-continuous triangle functions, each with a maximum at $\phi_i(x_i)$ =1 and a base width of twice the grid spacing between domain elements x_i.

In explicit form, when the grid spacing is even for the entire domain (which suffices in our case),

$$\phi_i(x) = \begin{cases} 0, & \text{if } x < x_{i-1} \text{ or } x > x_{i+1} \\ \frac{x-x_{i-1}}{h}, & \text{if } x_{i-1} \le x \le x_i \\ \frac{x_{i+1}-x}{h}, & \text{if } x_i \le x \le x_{i+1} \end{cases} \tag{11}$$

meaning that for a given x_j, $\phi_i(x_j) = \delta_{ij}$. The conveninece of this formulation becomes obvious when we plug this back in to Equation 10 to find that

$$y(x_j) = \sum_{i=0}^{N-1} \alpha_i \phi_i(x_j) = \sum_{i=0}^{N-1} \alpha_i \delta_{ij} = \alpha_j, \tag{12}$$

so the coefficients of the linear basis set are identically the value of the function for their respective regions.

Since Equation 9 is true for any continuous function $\Phi(x)$ that obeys the boundary conditions, we can now write down a system of N equations that by the uniqueness theorem should agree with each other, each time substituting in a different basis function ϕ_i such that $\Phi(x) = \phi_0, \phi_1, \phi_2, ...\phi_{N-1}$. Combining Equations 9 and 10 with this basis function substitution we have the system of equations

$$-\int_0^1 \frac{d}{dx}\left(\sum_{j=0}^{N-1} \alpha_j \phi_j(x)\right) \frac{\phi_i}{dx}\, dx = \int_0^1 6x^2 \phi_i(x) \tag{13}$$

where $i = 0, 1, 2, ...N-1$. Using the expression for $\phi_i(x)$ in Equation 11, we can compute the derivatives in Equation 13:

$$\frac{d\phi_i(x)}{dx} = \begin{cases} 0, & \text{if } x < x_{i-1} \text{ or } x > x_{i+1} \\ \frac{1}{h}, & \text{if } x_{i-1} \leq x \leq x_i \\ \frac{-1}{h}, & \text{if } x_i \leq x \leq x_{i+1} \end{cases} \tag{14}$$

which alllows us to evaluate the integral

$$\int_0^1 \frac{d\phi_i(x)}{dx}\frac{d\phi_j(x)}{dx}\, dx = \begin{cases} 0, & \text{if } |i-j| > 1 \\ \frac{2}{h}, & \text{if } |i-j| = 0 \\ \frac{-1}{h}, & \text{if } |i-j| = 1 \end{cases} \tag{15}$$

for the cases where the integral includes pairs of non-adjacent neighbors, identical elements, and nearest neighbors respectively.

Bringing the whole picture together, we have a system of N equations of the form,

$$\alpha_0 \int_0^1 \phi_0'\phi_0'\, dx + ... + \alpha_{N-1} \int_0^1 \phi_0'\phi_{N-1}'\, dx = \int_0^1 6x^2 \phi_0(x)\, dx \tag{16a}$$

$$\alpha_0 \int_0^1 \phi_1'\phi_0'\, dx + ... + \alpha_{N-1} \int_0^1 \phi_1'\phi_{N-1}'\, dx = \int_0^1 6x^2 \phi_1(x)\, dx \tag{16b}$$

$$\vdots$$

$$\alpha_0 \int_0^1 \phi_{N-1}'\phi_0'\, dx + ... + \alpha_{N-1} \int_0^1 \phi_{N-1}'\phi_{N-1}'\, dx = \int_0^1 6x^2 \phi_{N-1}(x)\, dx \tag{16c}$$

which when evaluated by Equation 15 take the form

$$2\alpha_0/h - \alpha_1/h + 0 + 0 + ... + 0 = \int_0^1 6x^2\phi_0(x)\,dx \tag{17a}$$

$$-\alpha_0/h + 2\alpha_1/h - \alpha_2/h + 0 + ...0 = \int_0^1 6x^2\phi_1(x)\,dx \tag{17b}$$

$$\vdots$$

$$0 + 0 + ... + 0 + -\alpha_{N-2}/h + 2\alpha_{N-1}/h = \int_0^1 6x^2\phi_{N-1}(x)\,dx \tag{17c}$$

which lends itself quite easily to the matrix form $\mathbf{Ay} = \mathbf{b}$ where

$$\mathbf{A} = \begin{pmatrix} 2/h & -1/h & 0 & \cdots & 0 \\ -1/h & 2/h & -1/h & \cdots & 0 \\ 0 & -1/h & 2/h & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 2/h \end{pmatrix}, \mathbf{y} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_{N-1} \end{pmatrix} \tag{18}$$

and $\mathbf{b}$ is a vector of size $N-2$ cotaining the right hand side intergrals:

$$\mathbf{b} = \begin{pmatrix} \int_0^1 6x^2\phi_1(x)\,dx + \alpha_0/h \\ \int_0^1 6x^2\phi_2(x)\,dx \\ \int_0^1 6x^2\phi_3(x)\,dx \\ \vdots \\ \int_0^1 6x^2\phi_{N-1}(x)\,dx + \alpha_N/h \end{pmatrix}. \tag{19}$$

It is worth noting that the result from Equation 12 means that the vector $\mathbf{y}$ contains the value of the function $y(x)$ for every element within the domain excluding the boundaries. The boundary conditions are then specified in the first and last elements of vector $\mathbf{b}$, which in our case would be $\alpha_0 = y(x_0) = \alpha_N = y(x_N) = 0$.

The integrals in $\mathbf{b}$ can be evaluated explicilty by using the expression for $\phi_i(x)$ in Equation 11. Performing the integration one finds the general form for our case

$$\int_0^1 6x^2\phi_i(x)\,dx = \frac{1}{2h}\left(6x_i^4 + x_{i+1}^4 + x_{i-1}^4 - 4x_i^3x_{i+1} - 4x_i^3x_{i-1}\right) \tag{20}$$

which simplifies to

$$\int_0^1 6x^2\phi_i(x)\,dx = h^3(6i^2+1) \tag{21}$$

when one notes that, since the domain is from $x = 0$ to $x = 1$, every x_i can simply be rewritten as $x_i = ih$ where i is the index in the array and h is the grid spacing.

With this mathematical framework in place, the problem is reduced to a simple matrix inverse problem $\mathbf{y} = \mathbf{A^{-1}b}$ where $\mathbf{b}$ and $\mathbf{A}$ are populated as specfied by Equations 18 and 21.

0.3 Implementation

Both the Gauss-Seidel relaxation and the finite element method were implemented using C++ in Visual Studio 2013. For the Gauss-Seidel Method, an array of 150 elements was defined for the domain $0 \leq x \leq 1$ with an even grid spacing of $h = 1/150$ in between array elements. Since it was necessary to start the algorithm with some initial "guess" function for $y(x)$, the quadratic function $y = x^2 - x$, which adheres to boundary conditions $y(x = 0) = y(x = 1) = 0$, was chosen for the starting condition of array y.

The enirety of the relaxation process takes place within a nested loop that successively applies Equation 6 by iterating through iteration number k and array index i in the following manner:

```
for (int k = 1; k < ITER; k++) {
    for (int i = 1; i < N - 1; i++) {
        y[i] = 0.5*(y[i - 1] + y[i + 1] - 6*pow(x[i], 2)*pow(h, 2));
     }
    storearray(y)
}
```

where **ITER** is some iteration maximum and **storearray**() here is some function that stores the current array y for future use.

It is also useful to create a convergence threshold to keep track of the rate and behavior of self-consistent convergence. For this simple one-dimensional model this threshold for a given iteration k can be expressed by

$$\Delta_k = \frac{1}{N} \sum_{i}^{N} \frac{y_i^k - y_i^{k-1}}{y_i^k} \tag{22}$$

where we have taken the arithmetic average of normalized changes in each array element i for every iteration k. It is sufficient for our purposes to say that self-consistent convergence is achieved when $\Delta_k \approx 10^{-5}$.

For the finite element method the same parameters for N and h were used, but instead of arrays, vectors as defined by the Numerical Recipes 3 library were used for x and y. Since matrix $\mathbf{A}$ as shown in Equation 18 is a tridiagonal matrix a sparse matrix function **tridag.h**, which requires an input of three vectors, each containing one of the center matrix diagonals, was used to solve $\mathbf{y} = \mathbf{A^{-1}b}$:

```
for (int i = 0; i < n-2; i++) {
    middiag[i] = 2 / h;  //Middle diagonal
}
for (int i = 0; i < n - 3; i++) {
    ldiag[i+1] = -1 / h; // Left-bottom diagonal
    rdiag[i] = -1 / h;   // Top-right diagonal
}
for (int i = 1; i < n-1; i++) {
    b[i-1]=-1*pow(h, 3) * (6 * pow(i, 2) + 1); //Source integrals
}
tridag(ldiag, middiag, rdiag, b, y) // Outputs Vector y
```

where we have used the results given in Equations 18 and 21 for the input values.

0.4 Results and Discussion

After performing the two methods and comparing their convergence with the analytic solution, it was clear that the finite element method was both faster and more efficient for the same parameter space. As shown in Figure 1, after 1000 iterations of the Gauss-Seidel method, neither convergence with the analytic solution nor self-consitent Δ_k convergence was achieved. It was only after $\sim$ 5000 iterations that a self-consistent convergence threshold of 10^{-5} was reached for the Gauss-Seidel relaxation. In contrast, the finite element method, due to the combination of rapid sparse matrix algebra and the elegant simplicity of basis functions in the initial algorithm, yielded a highly accurate result nearly instantly.

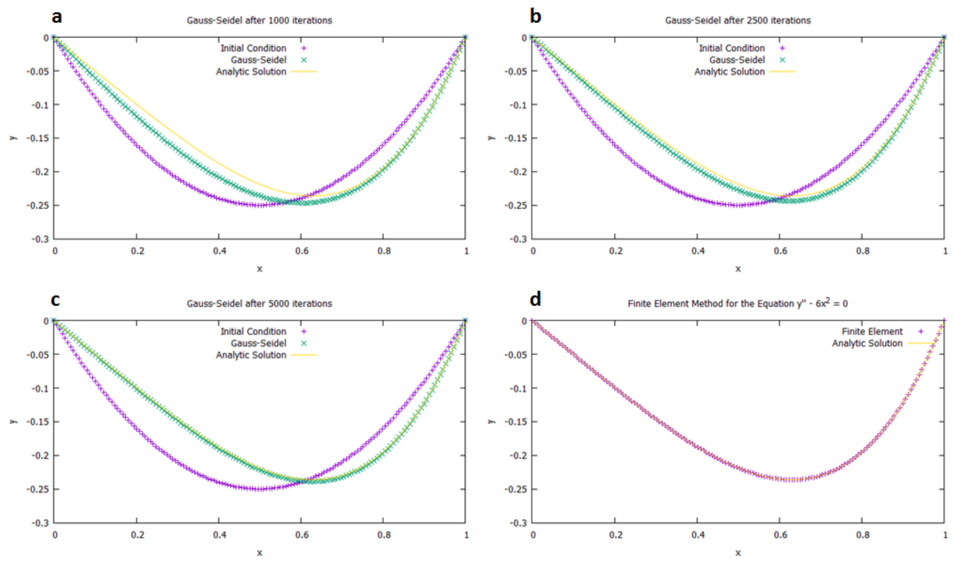

Figure 1: Plots for y(x) using the Gauss-Seidel (a-c) and finite element (d) methods. Gauss-Seidel plots show the initial "guess" function in purple, the Gauss-Seidel output after n iterations in green, and the analytic solution in yellow. Finite element shows excellent convergence with a single matrix calculation.

The significant difference in performance of the two algorithms can perhaps be best attributed to the fundamental problem that each algorithm is trying to solve. The Gauss-Seidel method starts with an approximation and then seeks to build a self consistent picture by successive application of an arithmetic average, whereas the finite element method seeks a global solution that by its very nature as a system of equations is mathematically required to be a self-consistent parameter space. Thus, in terms of scalability, the finite element method is far superior because once the work of determining the input parameters has been done for a given source term and boundary conditions, the rest is simply a single matrix inverse calculation.

To conclude, by solving this example differential equation we have shown not only how to perform Gauss-Seidel and finite element calculations for any given one-dimensional system, but also that the finite element method is superior to the Gauss-Seidel method in terms of both computational efficiency and mathematical rigor.

0.5 References

Advanced Computational Physics Lecture Slides; Lecture 3: Module 1 (Finite Element, Part 1), Dr. Vincent Meunier