THE EVOLUTION OF SOLUTIONS TO BOUNDARY-VALUED PROBLEMS USING FINITE ELEMENTS AND GENETIC ALGORITHMS

By

Matt Deyo-Svendsen

A SENIOR RESEARCH PAPER PRESENTED TO THE DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE OF STETSON UNIVERSITY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF SCIENCE

STETSON UNIVERSITY

2007
0.1 Acknowledgments

Thanks to my advisors Dr. Will Miles and Dr. Dan Plante for their lessons both inside and mostly outside the classroom. Among the other great teachers I have met, they deliver their expertise with clarity and enthusiasm. Without this care for their students, math is one generation away from extinction.

Thanks to my family and friends who energize my day and put up with my bad ones.

A big thanks to the good people at the Coca-Cola corporation for keeping my mind in a constant state of caffeinated bliss.
Contents

0.1 Acknowledgments ................................................................. 2

0.2 Abstract ................................................................................. 8

1 Introduction .............................................................................. 9

1.1 The Finite Element Method ....................................................... 10

1.2 Genetic Algorithms ................................................................. 18

1.2.1 Initialization ........................................................................ 18

1.2.2 Selection .............................................................................. 18

1.2.3 Variation ............................................................................. 19

1.2.4 Replacement ....................................................................... 19

2 Textbook FEM Development ..................................................... 20

3 The Goodness of Good ............................................................. 25

4 Together at Last! ....................................................................... 30

4.1 OneMax Implementation .......................................................... 30

4.2 F.E.M.G.A. modifications ......................................................... 31

4.2.1 The Matrix ......................................................................... 31

4.2.2 SSR Compare ...................................................................... 32
4.2.3 Crossover ........................................... 32
4.2.4 Mutation ........................................... 33
4.3 The Bottom Line ...................................... 34

5 The Metric System ...................................... 37

5.1 Finite Differencing .................................... 38
5.2 An A Posteriori Error Bound ........................ 41
  5.2.1 The $L_2$ Norm .................................. 42
  5.2.2 Derivative Jump .................................. 43

6 Results .................................................. 45

6.1 An Example .......................................... 45
  6.1.1 Genetic Algorithm vs. Uniform Meshes ........ 46
  6.1.2 A Million Years .................................. 47

6.2 Another Example .................................... 50
6.3 Yet Another Example ................................ 52

7 Future Work .......................................... 55
List of Figures

1     Hat functions $\phi_i$ ......................................................... 12
2     A Single Hat Function $h = 0.4$ ........................................... 20
3     A Single Hat Function $h = 0.6$ ........................................... 21
4     A Boundary Hat Function ..................................................... 21
5     Superimposed Hat Functions .................................................. 21
6     The $c_i$’s for $y'' + y = 3x^2$ ............................................. 24
7     The Estimated vs. Actual Solutions A ..................................... 26
8     The Estimated vs. Actual Solutions B ..................................... 26
9     The Elements of A .............................................................. 27
10    The Elements of B .............................................................. 27
11    The Approximation using A ................................................... 28
12    The Approximation using B ................................................... 28
13    The Estimated vs. Actual Solutions A ..................................... 29
14    The Estimated vs. Actual Solutions B ..................................... 29
15    FEMGA UML Diagram .......................................................... 31
16    Multiple runs at 20 iterations .............................................. 34
17    1 iteration ........................................................................... 35
18 2 iterations .......................................................... 35
19 5 iterations .......................................................... 36
20 20 iterations .......................................................... 36
21 Results of a Finite Differencing Approach ..................... 38
22 The Genetic Algorithm Finite Differencing Minimization ...... 40
23 The Actual Residual ............................................... 40
24 10 nodes, 5,000 generations ....................................... 45
25 FEM Approximation Point Residual Error ....................... 45
26 30 nodes, 5,000 generations ....................................... 46
27 32 uniform points with residual error less than or equal to genetic of 10 .... 46
28 30 nodes, 5,000 generations ....................................... 47
29 500,000 Generations, Population Size 30 ....................... 47
30 500,000 Generations - Norm Residual .......................... 48
31 500,000 Generations - Actual Residual ......................... 48
32 500,000 Generations, Population Size 5 ........................ 49
33 500,000 Generations - Norm Residual .......................... 49
34 500,000 Generations - Actual Residual ......................... 50
35 Another Genetic Example .......................................... 51
36 The Modified Mesh ............................................... 51
<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>100,000 Generations</td>
<td>52</td>
</tr>
<tr>
<td>38</td>
<td>Actual $c_i$’s in Norm Calculation on Uniform Mesh Size</td>
<td>53</td>
</tr>
<tr>
<td>39</td>
<td>10,000 Generations</td>
<td>53</td>
</tr>
<tr>
<td>40</td>
<td>Incorrect ratio of $c_i$ in Norm Calculation on Uniform Mesh Size</td>
<td>54</td>
</tr>
<tr>
<td>41</td>
<td>Incorrect ratio of $c_i$ in Norm Calculation on Example DE</td>
<td>54</td>
</tr>
</tbody>
</table>
0.2 Abstract

The Evolution of Solutions to Boundary-Valued Problems
Using Finite Elements and Genetic Algorithms

By
Matt Deyo-Svendsen

May 2007

Advisors: Dr. William Miles, Dr. Daniel Plante

Department: Mathematics and Computer Science

Many implementations of mathematical modeling require solutions to systems of differential equations. The finite element method is a robust technique used to approximate these solutions. The grid chosen determines the accuracy of the finite element method. Hence, different grids yield approximations with different precision relative to the true solution.

The machine learning technique known as genetic algorithms has been implemented to evolve better finite element approximations to an array of one-dimensional boundary-valued problems.
1 Introduction

Many applications of mathematical theory to problems in various domains, such as physics, engineering, biology, and economics, lie in the utilization and solution of differential equations. Yet, due to the complexity of the problem and the mathematics involved, for many of these problems an exact solution may never be found. Therefore, mathematical approximation methods must be used. Modern computing power of many billions of operations performed per second make these numeric solutions manageable and the mathematics define an upper bound on the error for these approximations.

Two typical methods often implemented to solve systems of partial differential equations are the finite differencing method and the finite element method (FEM). Certain classes of problems more naturally lend themselves to FEM. Advantages to the FEM include the robust and adaptive grid layout. Areas of higher change can be given greater focus without a universal change in grid sizing and spacing at the cost of computing time and power.

To make use of this advantage, these high action areas must be known prior to the implementation of the solution. If a system of differential equations is given a random set of grid points and that grid is applied to a machine learning technique, then can a globally optimal grid be attained?

To answer this question, two areas of research are necessary. First, genetic algorithms (GAs) will be utilized to generate a program capable of learning and adapting to the mesh parameters. Since the ultimate goal is to succeed with problems where little is known, random grids must be mutated from random permutations as genetic algorithms can offer. Second, mathematical methods for measuring the “goodness” of a solution will need to be developed. How can one grid be compared to another? Will the system converge to a specific set of grid points? How many iterations will yield an adequate solution?

This project aims to develop a novel adaptive grid generation solution method for solving differential equations. One-dimensional finite elements were implemented and compared to
closed form solutions of differential equations. Hence, mathematical measurements for the “goodness” of an approximation can be constructed and precisely compared.

1.1 The Finite Element Method

The method used to approximate the solution of the boundary value problems discussed in this paper is called the *Finite Element Method*. To better understand the FEM, we will follow an example differential equation given in [1][4]. The problem begins with an ordinary second-order, linear differential equation on $[\alpha, \beta]$ in the form

$$ y'' + Q(x)y = F(x) \tag{1.1} $$

$$ y(\alpha) = y_1; y(\beta) = y_2. $$

A problem such as this might arise in the study of the deflection of a string on an elastic foundation or of the temperature distribution in a rod[1].

We can express (1.1) in the form

$$ y'' + Q(x)y - F(x) = 0 \tag{1.2} $$

Assuming (1.2) holds then

$$ (y'' + Q(x)y - F(x)) \omega(x) = 0 \tag{1.3} $$

for any continuous, differentiable function $\omega(x)$.

Integration of (1.3) on our interval $[\alpha, \beta]$ gives

$$ \int_{\alpha}^{\beta} (y'' + Q(x)y - F(x)) \omega(x)dx = 0 \tag{1.4} $$
Suppose that $\omega(\alpha) = \omega(\beta) = 0$. Then

\[
\int_{\alpha}^{\beta} y'' \omega(x) dx = y'\omega|_{\alpha}^{\beta} - \int_{\alpha}^{\beta} y' \omega' dx = -\int_{\alpha}^{\beta} y' \omega' dx
\]

Thus (1.4) becomes

\[
\int_{\alpha}^{\beta} -y' \omega' + Q(x)\omega y - F(x)\omega dx = 0 \quad (1.5)
\]

This is known as the *weak formation* of (1.2).

As we have stated, the finite element method produces an approximation for $y$, call it $Y$. $Y$ is comprised by a finite-dimensional subspace whose basis functions have local support. This means the basis function is nonzero on a closed interval. We will use a subspace of continuous, piecewise polynomials. This paper makes use of continuous, piecewise linear functions on $[\alpha, \beta]$. Higher order polynomials give better approximations but, as we will see, produce a less sparse matrix. Therefore, they are harder to solve.

Our approximation can be expressed as the sum

\[
\sum_{i=1}^{n} c_i \phi_i(x) \quad (1.6)
\]

where the $\phi_i$’s are the basis functions for subspace of continuous, piecewise linear functions on $[\alpha, \beta]$. Further detail regarding the basis functions will be given later in this section. Substitute (1.6) into (1.5) to approximate $Y$.

\[
\int_{\alpha}^{\beta} \sum_{i=1}^{n} -c_i \phi_i'(x) \omega' + Q(x)\omega \sum_{i=1}^{n} c_i \phi_i(x) - F(x)\omega dx = 0
\]

\[
\int_{\alpha}^{\beta} \sum_{i=1}^{n} -c_i \phi_i'(x) dx + \int_{\alpha}^{\beta} Q\omega \sum_{i=1}^{n} c_i \phi_i dx = \int_{\alpha}^{\beta} F\omega dx \quad (1.7)
\]

The function $\omega(x)$ is called the *weight function* or *test function.*
To determine the exact subspace to be used, $[\alpha, \beta]$ is divided into $(n - 1)$ subintervals determined by $x_0 = \alpha < x_1 < x_2 < \ldots < x_n = \beta$.

The basis functions are hat functions $\phi_i \ni$

$$\phi_i(x_i) = 1$$
$$\phi_i(x_{i-1}) = 0 = \phi_i(x_{i+1})$$

Note that $\int_\alpha^\beta \phi_i \phi_j dx = 0$ provided $|i - j| > 1$.

This condition is known as Galerkin orthogonality. Using these basis functions, (1.7) can be written as

$$\sum_{i=1}^{n} \int_\alpha^\beta -c_i \phi'_i \omega' + c_i Q \omega \phi_i dx = \int_\alpha^\beta F \omega dx \quad (1.8)$$

Now choose $n$ different $\omega$’s: $\omega_1, \omega_2, \ldots, \omega_n$. Then for each $\omega_i$, an equation is generated in which $c_1, c_2, \ldots, c_n$ are the unknowns. This paper employs the Galerkin method in which $\omega_j = \phi_j$.

Then (1.8) becomes

$$\sum_{i=1}^{n} \int_\alpha^\beta -c_i \phi'_i \phi'_j + c_i Q \phi_i \phi_j dx = \int_\alpha^\beta F \phi_j dx \quad (1.9)$$
Recall that $\phi_i$ and $\phi'_i$ are nonzero only on the interval $[x_{i-1}, x_{i+1}]$. Then (1.9) may be written as

$$\int_{x_{j-1}}^{x_{j+1}} (-c_{j-1} \phi'_{j-1} - c_j \phi'_j - c_{j+1} \phi'_{j+1}) \phi'_j + Q (c_{j-1} \phi_{j-1} + c_j \phi_j + c_{j+1} \phi_{j+1}) \phi_j \, dx = \int_{x_{j-1}}^{x_{j+1}} F \phi_j \, dx$$

(1.10)

Distribute the $\phi'_j$ and the $\phi_j$ in (1.10) and solve for each of the integrals. Let $h_j = x_j - x_{j-1}$ and $h_{j+1} = x_{j+1} - x_j$. Note that on the interval $[x_{j-1}, x_j]$

$$\phi_{j-1} = \frac{x_j - x}{h_j}$$
$$\phi_j = \frac{x - x_{j-1}}{h_j}$$
$$\phi_{j+1} = 0$$
$$\phi'_{j-1} = \frac{-1}{h_j}$$
$$\phi'_j = \frac{1}{h_j}$$

and on the interval $[x_j, x_{j+1}]$

$$\phi_{j-1} = 0$$
$$\phi_j = \frac{x_{j+1} - x}{h_{j+1}}$$
$$\phi_{j+1} = \frac{x - x_j}{h_{j+1}}$$
$$\phi'_{j} = \frac{-1}{h_{j+1}}$$
$$\phi'_{j+1} = \frac{1}{h_{j+1}}$$

Note that at the point $x = x_j$ the equations for $\phi_j(x_j) = 1$ on both intervals and $\phi_{j-1}(x_j) = \phi_{j+1} = 0$. Thus our piecewise functions join endpoints on each interval.
Solve the first integral in (1.10) .

\[-c_{x_{j-1}} \int_{x_{j-1}}^{x_{j+1}} \phi'_{x_{j-1}} \phi'_{j} dx = -c_{x_{j-1}} \int_{x_{j-1}}^{x_{j}} \frac{-1}{h_{j}^{2}} dx\]  

(1.11)

Notice the change in the limits of our integral in (1.11) since for \( x > x_{j} \), \( \phi_{j-1} = 0 \).

\[-c_{x_{j-1}} \int_{x_{j-1}}^{x_{j}} \frac{-1}{h_{j}^{2}} dx = c_{x_{j-1}} \frac{x}{h_{j}^{2}} \bigg|_{x_{j-1}}^{x_{j}} = c_{x_{j-1}} \frac{x_{j} - x_{j-1}}{h_{j}^{2}} = c_{x_{j-1}} \frac{1}{h_{j}} \]  

(1.12)

Solve the second integral in (1.10) .

\[-c_{j} \int_{x_{j-1}}^{x_{j+1}} \phi'_{j} \phi'_{j} = -c_{j} \left( \int_{x_{j-1}}^{x_{j}} \frac{1}{h_{j}^{2}} dx + \int_{x_{j}}^{x_{j+1}} \frac{1}{h_{j+1}^{2}} dx \right) = -c_{j} \left( \frac{1}{h_{j}} + \frac{1}{h_{j+1}} \right) \]  

(1.13)

Similar to (1.12), solve for the third integral in (1.10) to find

\[-c_{j+1} \int_{x_{j-1}}^{x_{j+1}} \phi'_{j+1} \phi'_{j} dx = c_{j+1} \frac{1}{h_{j+1}} \]  

(1.14)

For the remaining integrals in (1.10), we can remove our differential equation functions \( Q \) and \( F \) by averaging the value on each interval.

where \( Q_{av,j} = Q \left( \frac{x_{j+1} + x_{j}}{2} \right) \). So for the fourth integral in (1.10)

We found that using the standard midpoint rule for approximating these functions as described in many finite element derivations was a large source of error in the final finite element approximation. Any error in the finite element approximation is propagated and amplified when running genetic algorithm code. Hence, we deviate slightly from the textbook finite
element derivations by employing Simpson’s approximation for our integrals.

Our approximation for $\int_{x_j}^{x_{j+1}} Q_j$ is given by

$$Q_{av,j} = \frac{x_{j+1} - x_j}{6} \left[ Q(x_j) + 4Q \left( \frac{x_{j+1} + x_j}{2} \right) + Q(x_{j+1}) \right].$$

Then

$$Q_{av,j}c_{j-1} \int_{x_{j-1}}^{x_{j+1}} \phi_{j-1} \phi_j dx = Q_{av,j}c_{j-1} \int_{x_{j-1}}^{x_j} \phi_j \left( \frac{x - x_{j-1}}{h_j} \right) \left( \frac{x_j - x}{h_j} \right) dx$$

$$= Q_{av,j}c_{j-1} \frac{1}{h_j^2} \int_{x_{j-1}}^{x_j} -x^2 - x_{j-1}x_j + x_{j-1}x + x_jx dx$$

$$= Q_{av,j}c_{j-1} \frac{1}{h_j^2} \left[ -\frac{x^3}{3} - x_{j-1}x_jx + \frac{x_{j-1}^2x_j}{2} + \frac{x_j^2x_j}{2} \right]_{x_{j-1}}^{x_j}$$

$$= Q_{av,j}c_{j-1} \frac{1}{h_j^2} \left( \frac{x_j^3 - 3x_{j-1}x_j^2 + 3x_{j-1}^2x_j - x_{j-1}^3}{6} \right)$$

$$= Q_{av,j}c_{j-1} \frac{1}{6h_j^2} (x_j - x_{j-1})^3$$

$$= Q_{av,j}c_{j-1} \frac{h_j}{6}$$ \hspace{1cm} (1.15)

The fifth integral in (1.10) is

$$c_j \int_{x_{j-1}}^{x_{j+1}} Q_{av} \phi_j \phi_j dx = c_j \left[ \int_{x_{j-1}}^{x_j} Q_{av,j} \left( \frac{x - x_{j-1}}{h_j} \right)^2 dx + \int_{x_j}^{x_{j+1}} \left( \frac{x_{j+1} - x}{h_{j+1}} \right)^2 dx \right]$$ \hspace{1cm} (1.16)

If we look at the first integral in (1.16) we find

$$\int_{x_{j-1}}^{x_j} \left( \frac{x - x_{j-1}}{h_j} \right)^2 dx = \frac{1}{h_j^2} \int_{x_{j-1}}^{x_j} x^2 - 2x_{j-1}x + x_{j-1}^2 dx$$

$$= \frac{1}{h_j^2} \left[ \frac{x^3}{3} - x_{j-1}^2 + \frac{x_{j-1}^2x_j}{2} \right]_{x_{j-1}}^{x_j}$$

$$= \frac{1}{3h_j^2} (x_j^3 - 3x_{j-1}x_j^2 + 3x_{j-1}^2x_j - x_{j-1}^3)$$

$$= \frac{1}{3h_j^2} (x_j - x_{j-1})^3$$

$$= \frac{h_j}{3}$$
Similarly in (1.16), the second integral is equivalent to $\frac{h_{j+1}}{3}$. This yields

$$Q_{av,j}c_j \int_{x_{j-1}}^{x_{j+1}} \phi_j \phi_j dx = Q_{av,j}c_j \frac{h_j}{3} + Q_{av,j+1}c_j \frac{h_{j+1}}{3}$$  \hspace{1cm} (1.17)$$

Finally, the last integral in (1.10) is solved similar to (1.15).

$$Q_{av,j+1}c_{j+1} \int_{x_{j-1}}^{x_{j+1}} \phi_{j+1} \phi_j dx = Q_{av,j+1}c_{j+1} \frac{h_{j+1}}{6}$$  \hspace{1cm} (1.18)$$

If we solve the right hand side of (1.10) we find

$$\int_{x_{j-1}}^{x_{j+1}} F \phi_j dx = \frac{F_{av,i}h_j}{2} + \frac{F_{av,i+1}h_{j+1}}{2}$$  \hspace{1cm} (1.19)$$

So now substitute (1.12), (1.13), (1.14), (1.15), (1.17), (1.18), and (1.19) into (1.10). We find

$$c_{x_{j-1}} \frac{1}{h_j} - c_j \left( \frac{1}{h_j} + \frac{1}{h_{j+1}} \right) + c_{j+1} \frac{1}{h_{j+1}} + Q_{av,j}c_{j-1} \frac{h_j}{6} + c_j \left( Q_{av,j} \frac{h_j}{3} + Q_{av,j+1} \frac{h_{j+1}}{3} \right) + Q_{av,j+1}c_{j+1} \frac{h_{j+1}}{6} = \frac{F_{av,i}h_j}{2} + \frac{F_{av,i+1}h_{j+1}}{2}$$

$$= \left( \frac{1}{h_j} + Q_{av,j} \frac{h_j}{6} \right) c_{j-1} + \left( - \frac{1}{h_j} - \frac{1}{h_{j+1}} + Q_{av,j} \frac{h_j}{3} + Q_{av,j+1} \frac{h_{j+1}}{3} \right) c_j + \left( \frac{1}{h_{j+1}} Q_{av,j+1} \frac{h_{j+1}}{6} \right) c_{j+1}$$

$$= \frac{F_{av,i}h_j}{2} + \frac{F_{av,i+1}h_{j+1}}{2}$$  \hspace{1cm} (1.20)$$

The system of equations for $1 < j < n$ of (1.20) and the inclusion of the boundary conditions at $x = \alpha$ and $x = \beta$ results in an $n \times n$ matrix which is tridiagonal. So the unknown coefficients are found by solving $[K] \{c\} = \{b\}$ where $b_i$ is the weighted average of $F(x)$. 


The finite element approximation is then given by

\[ Y = \sum_{i=1}^{n} c_i \phi_i(x) \]  

(1.21)

To summarize, we must know the domain of our problem, the "nonhomogenous" piece of the differential equation, the coefficients on various derivatives, and the boundary values. We will assume these are smooth and well-behaved. This will be used in the formation of our approximation.
1.2 Genetic Algorithms

A genetic algorithms is an optimization technique based on genetic theory in biology. A population of potential solutions to a problem are mated and their offspring evolve to some higher level. Maintaining a population of solutions (rather than a single solution) has several advantages[6]. A population of solutions allows a simultaneous exploration of multiple promising solutions. In addition, it allows for statistical decision making based on the entire sample of promising solutions and enables the use of learning techniques to identify problem regularities.

We will now discuss a genetic algorithm in detail.

1.2.1 Initialization

An initial population is generated. These candidate solutions are generated at random out of the set of all possible solutions.

1.2.2 Selection

Selection begins by choosing parents for a new generation. As in genetic evolution, we hope to see the stronger members of the population mate while the weaker members do not (author’s note: this is not an ethics paper). Hence we hope to generate stronger offspring than the current generation. We will briefly cover various selection techniques.

Tournament selection - Candidate solutions are randomly selected as individuals players to compete in a tournament. The members of this subset are played against each other. Based on the comparision criteria, the winners form the next generation of contestants. Larger tournaments will create more contests for members to compete and consequently raise the pressure on the quality of each solution [6]. The size of the tournament need not exhaust the possibilites for potential tournaments between all of the candidate solutions. However,
duplicate contests are replaced.

_Tr truncation selection_ - Candidate solutions are ordered and the lesser members are eliminated. Remaining members are duplicated until the original population size is retained.

1.2.3 Variation

This is the fun part of evolution. As in genetics, genes will swap among the members of an earlier generation to create the next generation. We create this generation by _crossover_ and _mutation_. Crossover is a process where individual genes are swapped in hopes that different combinations will create a stronger offspring. There are many ways to implement this procedure. If we represent the genes of both parents in one long string of information, then we can implement a one-point procedure by choosing a random point in the genetic code and swapping every gene that appears after it. We can also implement uniform crossover by randomly swapping individual genes between parents to create two dissimilar offspring. Mutation applies small modifications to offspring’s genetic code in hopes to explore the surrounding solutions. This rate of mutation will help avoid local optimal solutions.

1.2.4 Replacement

With our new set of mutated creatures, we now repeat the algorithm from the selection step unless we have reached our stopping criteria. Either the population converges to a singleton, the population contains a good enough solution, or an upper bound on the number of iterations has been reached[6]. GAs will converge in probability to a global optimum[7].
2 Textbook FEM Development

goodness of the member. We measure the element in this way. I wrote a matrix solver.

To begin, a finite element solver was written based on textbook examples with known solutions. Algorithms were built in Mathematica because of the accessibility to matrix solvers and graphics. Further, by following textbook examples, accuracy could be checked throughout the software development.

The first such example was Example 6-5 in [4]. It is of the general form discussed earlier. The equation to solve was $y'' + y = 3x^2$ with initial conditions $y(0) = 0$ and $y(2) = 3.5$. For analysis we used the unequally spaced set of base points of $\{0, 0.4, 0.7, 0.9, 1.1, 1.3, 1.6, 2.0\}$. This set will give us $n = 7$ elements.

A method was written for calculating hat functions. Any point $x$ within our initial conditions is defined by each hat function. If $x_{i-1} < x < x_i$ (the increasing portion of the hat function in element $i$), the function is defined by $\frac{x}{h_i} - \frac{x_{i+1}}{h_i}$ where $i$ is the base element and $h_i$ is the distance from $x_{i-1}$ to $x_i$. If $x_i < x < x_{i+1}$ (the decreasing portion of the hat function in element $i + 1$), the function is defined by $-\frac{x}{h_{i+1}} + \frac{x_i}{h_{i+1}}$ where $h_{i+1}$ is the distance between $x_i$ and $x_{i+1}$. If $x$ is not within the interval $x_{i-1}$ to $x_i$, the function is defined as 0.

![Figure 2: A Single Hat Function $h = 0.4$](image-url)
Notice the variation in the length of $h$ between our diagrams in Figure 2 and Figure 3.

Figure 3: A Single Hat Function $h = 0.6$

Figure 4 shows a boundary hat function. This is actually the increasing function of element $n$. The decreasing section is in the previous hat function figure. A plot of all of our elements on a single graph is provided in Figure 5.

Figure 4: A Boundary Hat Function

Figure 5: Superimposed Hat Functions
Notice the hats reach a value of 1 before descending. These hat functions tell us little to nothing about the actual equation we are modeling, except for the intervals we are approximating. Therefore we populate a matrix based on the value of our initial conditions and the function evaluated at various portions of our intervals. This matrix will lead to actual scaling factors which make our estimation possible.

We must create a $m \times n$ sized matrix, where $m$ is the number of base points and $n = m + 1$ in all cases. In this case, we have $m = 8$. We have a square matrix and an additional column for our $c$ approximations.

For the first row of the matrix, we define $k_{11}$ as 1 and our first endpoint value in the last column $k_{1n}$. For the last row of the matrix, we define $k_{mm}$ as 1 and our last endpoint value in the last column $k_{mn}$. The next step is to populate the rest of the matrix. We begin with populating the square between $k_{11}$ and $k_{mm}$. We will create a sparse matrix with a band length of 3. For any row $i$ such that $1 < i < m$, our matrix entries are defined as

$$k_{i(i-1)} = -\frac{1}{h_i} - Q_{av,i} \frac{h_i}{6}$$

$$k_{ii} = \left(\frac{1}{h_i h_{i+1}}\right) - \left(Q_{av,i} \frac{h_i}{3} + Q_{av,i+1} \frac{h_{i+1}}{3}\right)$$

$$k_{i(i+1)} = -\frac{1}{h_{i+1}} - Q_{av,i+1} \frac{h_{i+1}}{6}$$
As in the example we are following, our code generates the following banded matrix.

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2.567 & 5.6 & -3.383 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -3.383 & 8.167 & -5.033 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -5.033 & 9.867 & -5.033 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -5.033 & 9.867 & -5.033 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -5.033 & 8.167 & -3.383 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -3.383 & 5.6 & -2.567 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

If we reduce this matrix to row reduced echelon form, we get this matrix.

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{bmatrix}
\]
Figure 6 shows a graph of our approximation against our analytic solution of 
\[ y = 3x^2 + 6 \cos x - 0.00343 \sin x - 6. \]

![Graph of solution](image)

Figure 6: The \( c_i \)'s for \( y'' + y = 3x^2 \)

We have successfully implemented a finite element solver. How good is our approximation?
3  The Goodness of Good

To test how well our FEM approximation works, let us look at a slightly more complicated example. We will analyze the equation

\[ y = \frac{100}{x^2} \sin \left( \frac{10}{t} \right) \]

We can put it in the form acceptable to a finite element code, namely \( y' + Qy = F \). Hence we get the equation

\[ y' + y = \frac{100 \sin \left( \frac{10}{x} \right)}{x^2} - \frac{200 \sin \left( \frac{10}{x} \right)}{x^3} - \frac{1000 \cos \left( \frac{10}{x} \right)}{x^4} \]

Let us impose the boundary values of \( y(1) \) and \( y(5) \).

In reality, sometimes little is known about our exact equation. Let us try to catch the spike on the left half of the interval in two cases. Although we will be able to see the actual solution, it would not necessarily be obvious which set was better in practice.

Choose the base points \( A = \{1.0, 1.1, 1.2, 1.35, 1.5, 1.75, 2.0, 3.0, 4.0, 5.0\} \) and run it through our code.

Since we have engineered this example, we can plot our estimation versus the actual solution in Figure 7.
Figure 7: The Estimated vs. Actual Solutions $A$

We can see there is too much activity for a perfect fit to our known solution since it would require too many points.

But what if we were to choose another set of base points identical in size of $A$? Let $B = \{1.0, 1.05, 1.1, 1.15, 1.25, 1.5, 1.75, 2.5, 3.5, 5.0\}$. The estimation is graphed in Figure 8.

Figure 8: The Estimated vs. Actual Solutions $B$

We can see in Figure 7 and Figure 8 that our first estimation $A$ more closely catches the behavior of the analytic solution!
The first natural question is how much better is this estimation than the other? Let us look at the sum of the squared residuals. To do this we can measure the goodness of our base point set by

$$\frac{1}{n} \sum_{i=0}^{n} (\hat{y}_i - y(x_i))^2$$  \hspace{1cm} (3.22)

where $n$ is the number of elements, $\hat{y}_i$ is the estimated value of the differential equation, and $y(x_i)$ is the actual value.

![Figure 9: The Elements of A](image1)

![Figure 10: The Elements of B](image2)
Figure 11 and Figure 12 are the approximations of the function on our domain obtained by interpolating along each hat function.

Figure 11: The Approximation using $A$

Figure 12: The Approximation using $B$
The absolute error of each approximations are illustrated in Figure 13 and Figure 14.

![Figure 13: The Estimated vs. Actual Solutions A](image1)

![Figure 14: The Estimated vs. Actual Solutions B](image2)

If we sum the squared residuals of $A$, we get a value of 6.39156. The sum of the squared residuals for $B$ is 35.999. This confirms our thoughts from Figure 7 and Figure 8.

We now have a measure for the goodness of a set of base points. The next natural question is is there an even better approximation? Is there a “best” approximation?

To answer this we turn to genetic algorithms.
4 Together at Last!

4.1 OneMax Implementation

We break the uncongenial bonds of Mathematica and turn to the euphoric embrace of Java. To test the implementation of a basic genetic algorithm, we build the OneMax problem given in [6] (this exact example is called MaxOne in [7]). In essence, the members of our population are strings of equal length randomly comprised of 1’s and 0’s. The strongest of the species is the member who most closely sums to the gene length of the population. Hence, there can be many strongest candidates (such as 10101 and 01110 for gene length 5). Members compete in tournaments to reveal the strongest living candidates and those are mated using crossover techniques. To avoid local maxima, we apply mutation based on probabilities. Therefore, eventually our population will mutate to the optimal specimen summing to our gene length.

Our genetic algorithm was coded using the object-oriented paradigm. Our OneMax objects were simply a string of 1 and 0 integers. This OneMax object contains a compareTo method where OneMax A and B are summed and the higher valued OneMax is returned (ties are arbitrarily broken).

The OneMax objects are held within a Generation Vector in a Tournament object. The Tournament randomly selects OneMax players for Home and Away competition. The Home and Away teams are compared and the victorious individuals are given to a Crossover object. Here the Crossover object performs uniform crossover on the two parents. These are then sent to a Mutation object which slightly modifies the 1s and 0s in our OneMax objects. This completes a cycle of the genetic algorithm. The next Generation is ready for competition unless our stopping criteria (are there any OneMax objects containing all 1s?) has been met.
4.2 F.E.M.G.A. modifications

In general, we have a genetic algorithm complete with tournaments, crossover and mutation. We also have a set of base points for finite elements with a comparison technique. Thus, with slight modifications to our OneMax example, we can host finite element tournaments.

![FEMGA UML Diagram](image)

Figure 15: FEMGA UML Diagram

Now we look at some details of our modifications.

4.2.1 The Matrix

To fit with the OneMax genetic algorithm the Mathematica version was reimplemented in Java. All of the algorithms are identical except an additional MatrixSolver was written. Recall that our matrix is populated
We can solve this matrix by Gaussian elimination in the following manner. Subtract the first row $k_{21}$ times from the second row to eliminate $k_{21}$. Divide the second row by $k_{22}$ to leave a 1 in the center of the band. Repeat this process to eliminate the left entries from the band. Do the same from the end of the matrix to eliminate the entries to the right of the center of the band. The final result will be a matrix in row reduced echelon form (i.e. the weights for our finite element model).

4.2.2 SSR Compare

For this example we know the analytic solution so we can compute the sum of the squared residuals for our element approximations. With this matrix solver, we can implement a FiniteElement object which calculates the weights of a BasePoints object. After our solution has been found, we send the BasePoints object to the Compare object. Our first implementation is an SSRCompare.

4.2.3 Crossover

Uniform Crossover - Originally, we planned to handle Crossover in our BasePoints objects by randomly distributing the points of the parent BasePoints between the children. This was achieved by creating a global list of possible points and then randomly picking points for each child until the global set is exhausted. For the most part, this method worked. Yet, occasionally we would run out of points to distribute. For example, if we had offspring
A = \{1,2,3,4\}, \; B = \{1,2,5,3,5,4\}, \text{ and } C = \{1,2,3,4\}. \text{ Duplicate sets like } A \text{ and } C \text{ were frequent, because if the original parent was a strong candidate it would dominate in any tournament it competed in. \text{ We have global points } G = \{2,2,2,5,3,3,3,5\}. \text{ So there is a possibility with the global methods to generate crossover offspring } A = \{1,2,5,3,4\}, \text{ } B = \{1,3,3,5,4\} \text{ and } C = \{1,2,2,4\}. \text{ Clearly, the duplicate nodes in } C \text{ are an issue. This exception was handled by checking for duplicate nodes within the mesh, but if we were down to the last set, as in the example, we don’t have much else to choose from. This case was then handled by discarding the duplicate set and copying a previous set at random. }

This global crossover method appeared to be too random. If a single point is “good,” this is due to the nodes surrounding it. In other words, the element is a good approximation, not the single node. With mutation implemented, we were not preserving any of the structure we had generated in previous generations. Hence, this method was abandoned.

_Mutated Crossover - _This method of crossover collects a global list from the offspring of the previous generation. The list is ordered sequentially. If we have \( n \) members in our generation, the first \( n \) nodes in the global list are randomly distributed (with duplicate nodes possibly going to many offspring, unlike the previous uniform crossover method). This process is repeated for the next \( n \) nodes in the global list for the subsequent spots remaining in our offspring. This method hopes to preserve some of the structure we have generated by grouping nodes by value.

### 4.2.4 Mutation

We set a range of mutation and randomly change digits with a certain probability within our BasePoints objects. Hence no mutation is possible as well as much mutation.
4.3 The Bottom Line

Here are some preliminary results. Our code is a fast learner!

Consider the function

\[
y = -\frac{1}{6} e^{-x/100} \left[ 3000 - 3000 e^{x/100} + 75 e^{x/100} \cos\left(\frac{x}{2}\right) \right]
\]

There are so many features that we will have trouble capturing them all but this function provides a good example of the genetic algorithm reacting to such features. We randomly generate 6 meshes. We have tournaments of 6 competitions. Like our earlier examples, we have 11 nodes each. We used MutatedCrossover and Mutation with a probability of mutation of \( \frac{1}{W} \) and a possible range of mutation of -5 to +5. We plot the average SSR in the first 20 iterations in Figure 16. Some trials began with SSR values in the thousands.

![Graph showing multiple runs at 20 iterations](image)

Figure 16: Multiple runs at 20 iterations
Here are the plots from a single run like the ones in our graph. We can see that the points are randomly distributed but begin to grow into much better solutions.

![Figure 17: 1 iteration](image1)

Notice the spike in Figure 17 at $x = 100$ and $x = 200$ because of the closely space nodes. Our element near $x = 200$ is virtually nonexistent.

![Figure 18: 2 iterations](image2)

We quickly (1 iteration) lost our spike at $x = 200$ between Figure 17 and Figure 18. Bad genes, apparently.
By Figure 19, the spikes have diminished due to the slight variations in crossover and mutation methods between generations.

By Figure 20 our spikes have drastically diminished and the overall fit is much improved. Given enough time, our mesh will smooth. Although this is one example, the global populations did evolve towards similar regions.
5 The Metric System

Two questions essentially drive this research project. The first is how are we to capture the internal behavior between each of our finite element approximation points? This question is really a subset of the larger problem: how do we compare FEM approximation meshes without explicit knowledge of the actual solution? The tournaments and other selection methods of the genetic algorithm rides on this metric.

Up to this point, we have attempted a proof of concept by testing approximations versus the actual solution. Yet if the actual solution is known, approximating techniques are somewhat trivial. Ideally we can apply this technique to problems of unknown solutions. However we then lose our goodness test (3.22) as we have been measuring the error against the actual solution at each FEM approximation point. It should be noted that this goodness test is only measuring how close the approximation points are to the actual solution. We are left to assume that all of the points interior to each element are also converging to the actual solution.

One quick way to alleviate this problem is to expand our metric to some number of internal points along each interval. Recall that the finite element method produces a set of basis functions. Hence we have a function to find every point in our domain.

We would like to avoid adding more points to the mesh. Although it is a logical next step to finding better approximations, it would prevent us from refining the current method for a fixed number of points. In a sense, this simulates the computational complexity of larger, unsolved problems.
5.1 Finite Differencing

Recall that the problem we are trying to solve is given by (1.1). We can express this second-order linear differential equation as a finite differencing scheme given by

\[ \delta = \frac{y(x_{i+1}) - y(x_i)}{\Delta x} + Qy(x_i) - F \]  

(5.23)

where \( x_{i+1} \) and \( x_{i-1} \) are equally spaced by \( \Delta x \) from \( x_i \). Here, \( y \) is our finite element approximation and \( \delta \) is the approximation error from expressing the differential equation in this form. Hence our task in the genetic algorithm becomes that of minimizing \( \delta \).

Although we have a set of functions to test the internal points of each element, we cannot cross the boundary between elements. We are taking a derivative in our residual metric and our functions are discontinuous between elements. This results in an infinite error no matter where our estimation nodes lie. Hence, we can only measure the error internal to the finite elements but not across the elements.

Figure 21 is a test run of this implementation for the finite differencing representation residual minimization.

![Figure 21: Results of a Finite Differencing Approach](image)
Although the decisions were not based on any knowledge of the actual solution, the code was unable to capture the behavior of the solution. Notice how the large error in capturing the spike on [1, 2] propagates through the rest of the approximation.
Figure 22 is a plot of the finite differencing error which is minimized by the genetic algorithm. The residual error is on the order of $10^8$ and does not change much over the evolution of this species.

![Graph of finite differencing error](image1)

**Figure 22: The Genetic Algorithm Finite Differencing Minimization**

To better explain the convergence of the algorithm, Figure 23 shows the progression of the old metric (3.22) during the same test as Figure 22.

![Graph of actual residual](image2)

**Figure 23: The Actual Residual**

The task now becomes understanding why $\delta$ is not an appropriate error predictor. (5.23) must contain finite elements that have at least the same degree as the differential equation. Otherwise, approximating the error will include zero terms when we ask for too many derivatives from our approximating function. With this metric we are confined to the number of internal test points.
5.2 An A Posteriori Error Bound

Up to this point, we have been trying to solve this problem in an *a priori* manner. Another form of problem solving is in an *a posteriori* manner. These philosophies are referred to analytic and synthetic, respectively [3].

An analytical statement is verified through logical steps from a set of axioms. For example, the statement “There are infinitely many prime numbers” cannot be verified through experimentation (at the current level of computing power...but someday, you never know!). Similarly, the finite differencing derivation for a metric to drive the genetic algorithm towards a suitable finite element approximation was achieved (unsuccessfully) through no knowledge about any specific solution to the differential equation.

On the other hand, synthetic statements are verified through concrete examples. The statement “There is a prime number larger than 17” can be verified with the number 19. To do so, we must prove that 19 is in fact a prime number.

As [3] states, numerical methods are usually compared to a Taylor series expansion to account for the round-off error of the approximation (*an a priori* approach). However, this relies on knowing the exact solution. For this problem, we do not have that luxury. Therefore, we must move to the *a posteriori* problem solving approach.

Essentially, we must test a set of known functions and then extrapolate our findings to the larger set of unknown problems in the same form.
5.2.1 The $L_2$ Norm

A norm assigns a length to a vector in whatever dimensions it may reside. In two dimensions, the norm gives us our intuitive measure of distance by taking the square root of the sum of the squares of straight lines.

The $L_2$ norm measures the size of a function. The notation is given by:

$$||u|| = \int_{\Omega} u^2 dx$$

where $u$ is a function of $x$ and $\Omega$ is the domain of $u$.

Recall that the finite element method produces a vector space for its solution. A norm of $x$ on a vector space, denoted $||x||$ has the following properties:

$$||x|| \geq 0$$

$$||x|| = 0 \iff x = 0$$

$$||\alpha x|| = |\alpha||x||$$

$$||x + y|| \leq ||x|| + ||y||$$  (5.24)

where $\alpha$ is a scalar and $y$ is another vector in the vector space.

Equation (5.24) is known as the triangle inequality. According to this property, the length of one side of a triangle cannot exceed the sum of the lengths of the other two sides [5].

Recall that our differential equation is of the form (when the function $Q$ in front of $u$ is equal to 1).

$$u' + u = f$$  (5.25)
Similar to our idea with finite differencing, we will have some residual error \( R \) if we modify (5.25).

\[
R = u' + u - f
\]  
(5.26)

In this case, \( u \) is the set of functions generated by the finite element approximation.

### 5.2.2 Derivative Jump

Further FEM theory indicates that the change in derivatives between elements contributes to the error bound. We will call \( J \) the jump. We define an error bound as the \( L_2 \) norm as well as the jumps between derivatives. Then the error is given by \( c_1 ||h^a R|| + c_2 ||h^{a-1} J|| \) where \( a \) is some power to scale \( h \) to match the order of the polygon in the finite element approximation [8].

For our specific problem, through experimentation we find:

\[
|u - U| \leq c_1 ||hR|| + c_2 ||J||
= c_1 \sum_i ||h_i R_i|| + c_2 \sum_p ||h_p^0 J_p||
\]

where \( i \) is the number of elements, \( p \) is the number of internal element approximation points, \( h \) is the element size, \( R \) is the residual given in (5.26), \( J \) is the jump between derivatives [8]. Notice that \( h \) is of one less order in the jump (because we drop from a line to a point). For the purposes of this paper, we can experiment with the power on the \( h \)'s knowing that they will be one or possibly two degrees apart.

Therefore, in the genetic algorithm we will use the error bound, which does not depend on any knowledge of the actual solution, as an error estimate. When we use an error bound we find different challenges than using an error estimate of each mesh. The error bound does
not require the actual solution to the differential equation but it does require constants on each piece of the total error. For our simulations, we seek constants that bound the equation as closely as possible. In other words, we wish to push that bound as close to the solution as possible. If we are too far away, we may not be representing the function correctly. If we are too close, we run the risk of no longer bounding the error.

For a useable measure, we must solve for $c_1$ and $c_2$. This was achieved by generating a set of uniform grids for varying numbers of points and a set of random points for varying numbers of points with a known solution. This data was used in a linear solver package to find the constants which when multiplied to each norm would be equal to the actual error. We will use $c_1 = 4.2158 \times 10^{-4}$ and $c_2 = 5.487 \times 10^{-11}$. 
6 Results

6.1 An Example

To compare our genetic algorithm result to the problem we used as an example in Figure 11 and Figure 12, consider the plot in Figure 24 using 10 nodes. The test ran for 5,000 generations.

![Figure 24: 10 nodes, 5,000 generations](image)

![Figure 25: FEM Approximation Point Residual Error](image)

For comparison, the residual error at our approximation points sums to 4.15337 using the original residual equation (3.22).
6.1.1 Genetic Algorithm vs. Uniform Meshes

It would take 32 uniformly spaced points to get a residual error less than that of the genetic algorithm placed points.

Figure 26: 30 nodes, 5,000 generations

Figure 27: 32 uniform points with residual error less than or equal to genetic of 10

For this genetic algorithm simulation of 30 finite element nodes over 5,000 generations, it would take 65 uniformly spaced points to have a comparable residual error.
norm unchanged.

6.1.2 A Million Years

Now consider a test run of 500,000 generations with a population of size 30. Like the others in our results, this genetic algorithm test was done with no knowledge of the actual solution. We can see in Figure 29 that the actual solution has been completely eclipsed by the FEM approximation! As we would expect, we have more points in the areas of increased behavior.
For comparison with earlier approximations, this mesh has residual error of \(2.11687 \times 10^{-3}\).

Recall that Figure 30 uses an error bound while Figure 31 uses knowledge of the actual solution. In Figure 30 and Figure 31, the measures do not track one to one. This is due to the fact that the genetic algorithm will not make the same decisions on which specimen is better.

Our next 500,000 generation test was with a population of size 5. It ran for 15 minutes on a single thread. By comparison, the 30 member population ran for more than 7.5 times as long! The norm residual from (3.22) for this mesh is \(2.12212 \times 10^{-3}\). See Figure 32.
Figure 32: 500,000 Generations, Population Size 5

Figure 33 shows the norm residual over time.

Figure 33: 500,000 Generations - Norm Residual
Figure 34 shows the actual residual over time.

![Graph showing actual SSR over generations](image)

**Figure 34: 500,000 Generations - Actual Residual**

### 6.2 Another Example

Consider another example. To mimic the unknown equations we wish to solve, we will leave the constants in our norm unchanged. Let \( y = e^{\sin(20\sqrt{x})} \). Therefore,

\[
y' + y = e^{\sin(20\sqrt{x})} + \frac{10e^{\sin(20\sqrt{x})}\cos(20\sqrt{x})}{\sqrt{x}}
\]

Running the genetic algorithm for 5,000 generations with 30 nodes yields a graph (Figure 35) with norm error \( 9.073 \times 10^{-4} \) and actual residual \( 3.594 \times 10^{-2} \).
One question raised during research was can we improve the presented mesh by hand since we can see the actual equation? The answer is not necessarily. Consider the FEM point at 0.987. We can see that the interval $[0.95, 1.0]$ is well defined whereas the interval $[0.1, 0.15]$ is not. Can we simply move the point to better define that region (and therefore the overall approximation)? In this case, the answer is no! Figure 36 is the plot of the modified mesh.

Error has propagated throughout the mesh. Our residuals agree with what we can see. The norm error is $1.6898 \times 10^{-3}$ and the actual residual is $6.6382 \times 10^{-2}$.
6.3 Yet Another Example

For this example, consider the following equation for the analytic solution:

\[ y = \cos^3(1.2x) \sin^2(1.6x) \]

Our differential equation then becomes:

\[ y' + y = 3.2 \cos^3(1.2x) \cos(1.6x) \sin(1.6x) + \cos^3(1.2x) \sin^2(1.6x) - 3.6 \cos^2(1.2x) \sin(1.2x) \sin^2(1.6x) \]

First, let us run the genetic algorithm for 100,000 generations. The result is Figure 37.

![Figure 37: 100,000 Generations](image)

Next, let us test the equation using the constants in the norm calculation that we have found to be correct. Recall they are \( c_1 = 4.2158 \times 10^{-4} \) and \( c_2 = 5.487 \times 10^{-11} \). Figure 38 shows the varying number of uniform mesh points on the \( x \)-axis and the corresponding residuals on the \( y \)-axis. They have been scaled to lie close to one another. The trend of the residual as the number of points increases is more important than the scale between them. Although we have seen that the genetic algorithm may not make the same decisions, the trend will the the same over successive generations. Refer to Figure 33 and Figure 34 as an example.
Figure 38: Actual $c_i$'s in Norm Calculation on Uniform Mesh Size

Using the experimental constants results in Figure 39 after 10,000 generations.

Figure 39: 10,000 Generations

Now let us run the same test but with incorrect constants, say $c_1 = 1$ and $c_2 = 1$. These constants will be way too high to properly bound the actual error. In other words, the relationship between the norm and the jump in derivatives will be skewed towards one or the other. Figure 40 shows the change in concavity due to the incorrect ratio in constants and results in the residuals not tracking one another.
Figure 40: Incorrect ratio of $c_i$ in Norm Calculation on Uniform Mesh Size

This error will propagate through the genetic algorithm each generation. Figure 41 shows the exact same simulation as Figure 39 but with the incorrect constant ratio.

Figure 41: Incorrect ratio of $c_i$ in Norm Calculation on Example DE

We have completely missed an entire feature of our actual solution! Although the norm residual does not depend on the actual solution, we can see that the accuracy depends on finding these constants. Hence, we must test a set of known functions to gain confidence in the norm constants.
7 Future Work

We have sucessfully found and implemented a metric to minimize error in the finite element approximation by using a combination of norms. Calculating an upper bound on the error and then minimizing that bound was the key.

There will always be work (and play) to do on modifying parameters in the genetic algorithm such as the number of specimens in a population and the probability of mutation. Optimizing these parameters will be vital to this method becoming an efficient numeric technique.

One direction this research could take would be to raise the level of accuracy in the hat functions by using quadratics. By definiton, this will decrease error in all of the finite element approximations. This will require rederiving the finite element approximation. The finite element matrix will then be a 5-banded sparse matrix rather than the current 3-banded sparse matrix we get from linear hat functions. Therefore, another matrix solver will need to be implemented and optimized. This change will also affect the norm’s value of \( h \). (WHAT IS \( h \)?)

In addition, we are confined to a one-dimensional problem. The concept of mapping in-between a system of equations and nodes within the approximation remains the same, but in a two-dimensional FEM a surface is divided into triangular regions [2]. We lose some of the sparseness in the matrix as well.

Many open questions remain. Does the genetic algorithm ever reach a global minimum? Although tests have been run for hundreds of thousands of generations, the genetic algorithm continues to find better solutions. The element approximation points are on the real domain and therefore there is an infinite set of possible solutions.

Another open question is how does the technique handle nonlinear differential equations? This will also require a rederivation of the finite element solver.
References


Biographical Sketch

My name is Matt Deyo-Svendsen and I am a senior at Stetson University. My parents decided to hyphenate their names and have regretted it ever since they noticed it doesn’t fit in the boxes for most standardized paperwork. I am majoring in both Mathematics and Computer Science because it reminds me of peanut butter and jelly - it just fits.

I regularly volunteer for Habitat for Humanity. There is good theology and good therapy in swinging a hammer every Saturday. I love to ride my bike. Things become simpler if you have to carry everything you need on your back. I’m learning to play the guitar by spending time with my friends. We play intramural sports. I enjoy Bill Cosby comedy acts and the Simpsons. I also enjoy the occasional Diet Coke. If you really know me, then you’ll know why that last sentence is a lie.

I’m a lucky guy. My parents value only one thing more than education and that is their kids. My sister is my best friend but I’ve got great runner-ups. I’m fortunate that I get to work, study and play with them all so much.

This summer I will be traveling to Zambia to build homes with the Habitat for Humanity Global Village Program.

This research project will complete Phase III of my plot to take over the world.