Newton's Method for Solution Finding to Nonlinear Semi-Elliptic PDE's

Northern Arizona University REU Program 2012*

Rachel E. Bachman❀

❀ Dept. of Mathematics and Computer Science, Clarkson University, Potsdam, NY

July 19, 2012

*This paper details the results of work done at Northern Arizona University under the guidance of REU Mentors Drs. John M Neuberger and James W Swift. Primarily work was shared with Nate Veldt of Wheaton College, and Jacob Clark of Arizona State University. This paper details my specific contributions to the project. A fourth partner, Robert Ravier of Cornell University, also shared in research on a related topic, and produced an independent paper detailing more specific results of his work. The program ran from May 29th through July 20th and was graciously funded by the National Science Foundation.

Abstract

This research seeks solutions $u : \Omega \to \mathbb{R}$ to a family of nonlinear semielliptic partial differential equations (PDE), $\Delta u + f(u) = 0$ by applying variational methods. Solutions to these equations have applications in many areas including physics and engineering. Here, the *Gradient Newton Galerkin Algorithm* (GNGA) developed by Drs. Neuberger and Swift of Northern Arizona University is applied and analyzed. This algorithm relies heavily on Newton's method to find critical points of the PDE. Since the original implementation of the algorithm is computationally expensive and slow when applied to fine grids with large numbers of modes, a faster method which was computationally less expensive is needed. The most expensive aspect of GNGA is the calculation of the Hessian matrix at each Newton iteration, as it requires looping over a high number of grid points. With inspiration from related research, a new approach for this step was developed and implemented into the GNGA. The new method is devoid of loops, and instead uses simple matrix multiplication, which proved much more efficient. Comparison and analysis to pre-existing code was then conducted using MATLAB .

It is likely that the faster calculation detailed here will enable solution finding in complex or large systems, as well as other similar families of equations, such as Schroedinger's Equation. The method has proven useful in initial implementation in one and two-dimensional problems, even with high numbers of modes.

1 Introduction

[1][2] The primary concern of this work was to find solutions to non-linear elliptic partial differential equations, in particular:

$$\begin{cases} \Delta u + f(u) = 0 & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases} \tag{1}$$

on a piecewise smooth and bounded region $\Omega \subset \mathbb{R}^N$. The boundary used is $u = 0$ indicating zero Dirichlet conditions. Here, Δ is the Laplacian operator, and we consider f to be a superlinear, subcritical function[4]. Since f satisfies the criteria of superlinear, we have

$$f'(u) > \tfrac{f(u)}{u} \text{ for } u \neq 0 \text{ and } \lim_{|u|\to\infty} \tfrac{f(u)}{u} = \infty.$$

I.e., f increases in the positive direction faster than a line, meaning its average slope $\frac{f(u)}{u}$ is strictly less than its instantaneous slope, $f'(u)$.

Before we define our problem, we review concepts of metric spaces, Newton's Method and the Secant method, and define terms such as *action functional*, *Morse index* and *bifurcation*.

[1] In this paper we are assuming an audience with an undergraduate mathematics background, i.e., basic exposure to concepts of differential equations, linear algebra, and calculus.

[2] Much of this introduction is shared verbatim in the paper "The Critical Exponent Problem for Radially Symmetric Solutions to a Nonlinear PDE" by Nathaniel Veldt and Jacob Clark. The full paper was divided into independent reports for brevity and specificity.

1.1 Preliminary Review

The first topic to be reviewed is *metric space*. A metric is a way to describe the distance between two entities of a set, such as points or subsets, among other things. Another concept necessary for understanding our arguments is the definition of a *complete* set. A subset *S* of a metric space is said to be complete if every Cauchy sequence $(x_n)_{n\in\mathbb{N}}$ in *S* converges in *S*. In other words, the sequence (x_n) has a limit in *S*. Here, a sequence is called a *Cauchy sequence* if it is a sequence $(x_n)_{n=1}^{\infty}$ in a metric space with metric *d*, if for every $\epsilon > 0$ there exists some $N \in \mathbb{N}$ such that

$$d(x_m, x_n) < \epsilon \text{ for all } m, n \in \mathbb{N}, m, n > N.$$

There are several specific cases of complete metric spaces. One in particular, called a *Banach space*, is complete normed vector space. A *vector space* is a set of vectors paired with the binary operations of addition and scalar multiplication such that the group axioms hold. Therefore, in a Banach space there exists a norm function that assigns a length in $\mathbb{R}^+$ to each vector, and length zero to the zero vector. Another example of such a space is a *Hilbert Space*, or a space that is complete under an inner product. More specifically, we use a Hilbert space called a *Sobolev space* with inner product $\langle u, v\rangle = \int_\Omega \nabla u \cdot \nabla v \, dx$.

Next, we define an *action functional*. A functional is a map from a vector space, into the corresponding scalar field. It can be thought of as a "function acting on functions." The functional we study maps the space H into the real numbers, i.e., $J : H \to \mathbb{R}$ and is defined by:

$$J(u) = \int_\Omega \left(\frac{|\nabla u|^2}{2} - F(u) \right) dx, \tag{2}$$

where ∇u is the vector gradient of u and F is the anti-derivative of f with $F(0) = 0$, from above. This functional is important to us because it has been shown that critical points of J are solutions to the PDE. Therefore, we are searching for functions u such that the directional derivative $J'(u)(v) = 0$ for all linearly independent directions v[5].

We can plot slices of this functional to help us visualize where we are searching for solutions. The domain of J is an infinite-dimensional Hilbert space H where solutions to (1) reside. The space is spanned by the eigenfunctions of the negative Laplacian. Since we are unable to plot in infinite dimensions, we must be content with visualizing a single two or three dimensional slice. For example, let $q : \mathbb{R} \to \mathbb{R}$ defined by:

$$q_1(t) = J(tu) = t^2 \int \frac{|\nabla u|^2}{2} dx - \frac{t^4}{4} \int u^4 dx,$$

and consider $q_2(s, t) = J(s\psi_i + t\psi_j)$ displayed in the following figures.

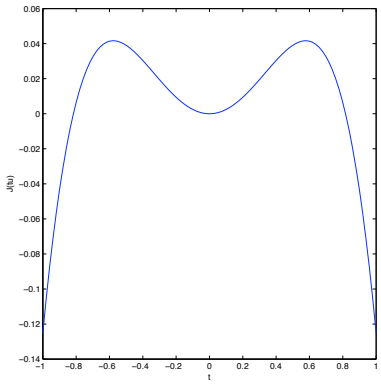


Figure 1: Two-dimensional slice $J(t\psi_1)$

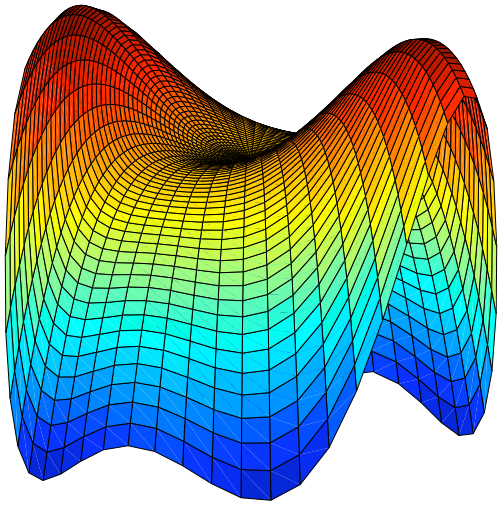


Figure 2: Three-dimensional cross-section of functional, $J(s\psi_1 + t\psi_2)$

Note in Figure 1 that the points at the top of the humps and the bottom of the valley indicate where the first derivative is zero and thus are critical points. We can also see in Figure 2 that $J''(0)(u,u) > 0$. Since we know $J'(0)(u) = 0$ for all $u \in H$, we have that the origin is a local minimum. Also, for solutions $u \neq 0$ we have $J''(u)(u,u) < 0$ indicating that solutions to the PDE are saddle points, as expected. We also observe that since our function $f(u)$ is superlinear, $F(u)$ is superquadratic and therefore larger than the gradient term. Thus $J(tu) \to -\infty$ as $t \to \infty$, so the minimum at the origin is unique.

1.2 Morse Index, Bifurcation, and Solution Finding

Another term we define is *Morse Index*, or *signature*. The Morse Index (MI) of a nondegenerate solution u is the number of linearly independent directions v in the function space in which J "curves down", i.e., $J''(u)(v,v) < 0$.[3] Thus, a MI = 0 solution indicates a local minimum, and a MI = 1 solution indicates a saddle point. We determine this by looking at the signature, or number of negative eigenvalues, of the Hessian matrix, whose arguments are the second partial derivative terms of J.

Now, we discuss the subject of *bifurcation*. Bifurcation theory concerns changes in the structure of solutions of the partial differential equations in question based on variations in a chosen parameter. A bifurcation is said to occur when a small smooth change to the parameter causes a sudden behavioral change in the system. In our case, the bifurcation parameter chosen is s, and the locations of bifurcations are based on the locations where

the signature of the Hessian changes. We plot diagrams of these diagrams as $||u||_2$ vs. s to display possible values of the PDE according to the parameter s. We find that at points where s is an eigenvalue, a branch bifurcates from the s-axis. We also find that branches may have multiple bifurcation points at which a new branch begins. Lastly, we review the basic root finding algorithms of Newton's method and the secant method. Given a function in C^2 such that $g : \mathbb{R} \to \mathbb{R}$, Newton's method can approximate roots by generating a sequence from an initial guess, and the value of the function and its derivative at that guess. The sequence is generated recursively as follows:

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}.$$

Iterations are continued until the value of x_n is sufficiently close (within an indicated tolerance) of a zero of g. A proof that Newton's method converges to a solution under appropriate conditions can be found in the appendix.

Similarly, the secant method can also be used to approximate zeros of the gradient. Here, a succession of secant lines are drawn between two function values, and the root is calculated respectively. It can be thought of as a finite difference approximation of Newton's method. Instead, the sequence generated here begins with two initial guesses, as follows:

$$x_n = x_{n-1} - f(x_{n-1})\frac{x_{n-1}-x_{n-2}}{f(x_{n-1})-f(x_{n-2})}.$$

The drawbacks to these methods lies in the necessity of a good initial guess. Without this, neither method is guaranteed to converge (again see proof).

1.3 Galerkin Expansion

It is known that solutions to the PDE (1) lie in the Sobolev space H, a complete vector space that has an inner product for $u, v \in H$ s.t.

$$\langle u, v \rangle_H = \int_\Omega \nabla u \cdot \nabla v dx,$$

with corresponding norm for measuring functions $u \in H$

$$||u||_H^2 = \int_\Omega (\nabla u)^2 dx.$$

Note that $H \subset L_2$, where

$$L_2 = L_2(\Omega, \mathbb{R}) = \{u : \Omega \to \mathbb{R} \mid \int_\Omega u^2 < \infty\}.$$

Therefore, we also make use of the L_2 inner product and norm:

$$\langle u, v \rangle_2 = \int_\Omega uv dx,$$

$$||u||_2^2 = \int_\Omega (u)^2 dx.$$

We observe that $H = \text{span}\{\psi_i\}_{i\in\mathbb{N}}$, where ψ_i are an orthonormal set of eigenfunctions in L_2 of $-\Delta$ under zero Dirichlet boundary onditions with corresponding eigenvalues λ_i. We have that:

$$-\Delta\psi_i = \lambda_i\psi_i \text{ on } \Omega.$$

Thus, $\langle\psi_i, \psi_j\rangle_2 = 0$ if $i \neq j$ and $\|\psi_i\|_2^2 = 1$.

We take the Galerkin space G to be the finite dimensional subspace of H, defined by:

$$G = \text{span}\{\psi_1, \psi_2, \ldots, \psi_{\mathrm{m}}\} \text{ where } \mathrm{m} \in \mathbb{Z}^+.$$

A Galerkin expansion moves the PDE from a continuous function space to a discrete space. Solutions to the PDE u are written as a truncated generalized Fourier series:

$$u = \sum_{k=1}^{m} a_k\psi_k,$$

where a_k values are the Fourier coefficients of the expansion.

2 Using GNGA to Find Solutions

The solution finding algorithm we propose was introduced by our advisors, Drs. Neuberger and Swift, and is known as the Gradient Newton Galerkin Algorithm, or GNGA . In basic terms, the algorithm finds critical points of the action functional of the PDE using Newton's method. Since we calculate the Newton direction with the gradient vector and the Hessian matrix filled with second partial derivatives, we step until we find zeros of the gradient which indicate locations that are local extrema and thus are solutions.

The basic GNGA is outlined below[4]. Note that the orthonormal basis is generated by the eigenfuncitons such that under zero Dirichlet boundary conditions $\psi_k(x) = \sqrt{2}\sin(k\pi x)$ for the unit interval $\Omega = [0, 1]$. We choose the number of modes $m \in \mathbb{Z}^+$, and set the number of grid points n such that $n = 2m + 1$. We know that this number of grid points allows us to compute exact numerical integration in the region used here. Proof is given in the Section E of the appendix.

Algorithm 1 The GNGA

Require: Region Ω and nonlinearity f
  Orthonormal basis $\{\psi_k\}_{k=1}^m$ for a sufficiently large subspace $G \subset H$
  Choose initial coefficients $a = a^0 = \{a_k\}_{k=1}^m$
  $u \leftarrow u^0 = \Sigma a_k \psi_k$
  $j = 0$
  while $\|g\|_2 >$ tolerance **do**
    $g = g^{j+1} = (J'(u)(\psi_k))_{k=1}^m \in \mathbb{R}^m$ (Gradient vector)
    $A = A^{j+1} = (J''(u)(\psi_j, \psi_k))_{j,k=1}^m$ (Hessian matrix)
    $\chi = \chi^{j+1} = A^{-1} g$
    $a^{j+1} = a^j - \chi$, $u^{j+1} = \sum_{k=1}^m a_k^{j+1} \psi_k$
    Calculate $sig(A(u))$ and $J(u)$ if desired.
    Calculate $\|g\|_2$, approximation of $\|\nabla J(u)\|$.
    $j \leftarrow j + 1$
  end while

3 Speeding Up the GNGA

One hindrance that is inherent to the GNGA originally developed by Drs. Neuberger and Swift in [4] is the calculation of the Hessian matrix at each Newton step. Since we are forced to approximate numerical integration at numerous grid points, it is a computationally expensive process, especially if the algorithm is extended to higher dimensions.

In hopes of avoiding this, or at minimum reducing the computational expense, other Newton's method based algorithms were explored. In particular, we took interest in the work by Zhou and Wang [6] who detailed a similar solution finding algorithm which used both a minimax and Newton approach on similar equations. In their work, they also sought to numerically examine the qualitative behavior or positive and nodal solutions to similar semilinear elliptic PDE problems. In our work, as well as theirs, we can see that the critical points of the action functional $J(u)$ are weak solutions to the PDE displayed in (1).

For $v, w \in H$, the first derivative of the action functional in the v direction can be defined as

$$\begin{aligned} J'(u)(w) &= \lim_{t \to \infty} \frac{J(u + tw) - J(u)}{t} \\ &= \int_\Omega (\nabla u \cdot \nabla w - f(u)w)\, dx \end{aligned}$$

Consider the first term of the integral, $\int_\Omega \nabla u \cdot \nabla w$. We now apply integration by parts.

$$\begin{aligned} \text{Let } r = \nabla u, \quad ds = \nabla w\, dx \\ dr = \Delta u\, dx, \qquad s = w \end{aligned}$$

$$\text{So we have } \nabla uw\Big|_0^1 - \int_\Omega w\Delta u.$$

Since $w = 0$ on the boundary, $\nabla uw\Big|_0^1 = 0$ we are left with only $-\int_\Omega w\Delta u$. Now, we rewrite $J'(u)(w)$ such that

$$J'(u)(w) = -\int_\Omega (\Delta u + f(u))\, w\, dx \tag{3}$$

Using this derivation, the conjecture that critical points of $J(u)$ are solutions can be seen in the following sketch of the reverse direction of the proof. The forward direction is very difficult, and has been omitted.

Proof. Let the twice differentiable function u be a solution to $\Delta u + f(u) = 0$. Then $J'(u)(w) = \int_\Omega (\Delta u + f(u))v = 0$ for all $v \in H$. Since the gradient of the action functional is zero, we have that u is a critical point. Thus, we have shown that solutions are critical points. □

Using the same techniques, we define the second directional derivative as:

$$\begin{aligned} J''(u)(v,w) &= \int_\Omega (\nabla v \cdot \nabla w - f'(u)vw)\, dx \\ &= \int_\Omega (\Delta w - f'(u)w)\, v\, dx \end{aligned}$$

Note that since we are concerned with local extrema and saddle points, the zeros of the first derivative and the sign of the second are of importance.

In the original `MATLAB` implementation of the `GNGA` code, we use numerical integration lies in approximating $J''(u)$, our Hessian matrix, after each iterate. This calculation was done by looping over each of the upper triangular grid points and performing matrix multiplication at each step.

$$H_{i,j} = (\lambda_i - s)\, \delta_{i,j} - B(:,i)^T .* B(:,j)^T (3u^2)/n \tag{4}$$

Here ".$*$" is component-wise multiplication, and $B(:,i)$ is the i^{th} column of the orthonormal basis B, as is `MATLAB` convention. We then solved the system $H\chi = g$ where χ is the search direction used in Newton's method. Thus solving $(D^2J(u))^{-1}\nabla J(u) = H^{-1}g$ where H is invertible or the pseudo-inverse of $D^2J(u)$ multiplied by $J'(u)$ when it is not. Note that we do not actually calculate H^{-1}; rather, we rely on methods from linear algebra to solve an augmented matrix. `MATLAB` finds the most efficient method to do so.

3.1 The Riesz Representation Theorem

We now pause to recall the *Riesz Representation Theorem* which connects a Hilbert space to a continuous dual space by a linear mapping. More precisely, it states that every linear operator on a Hilbert space can be represented by a unique element in the dual space. Given $T_1 : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$, there exists a unique $M \in M_{n\times n}$ such that $T_1(w, z) = Mw \cdot z$ for all vectors $w, z \in \mathbb{R}^n$.

It also holds that given a map $T_2 : \mathbb{R}^n \to \mathbb{R}^n$, there exists a unique $M \in M_{n\times n}$ such that $T_2(x) = Mx$ for all $x \in \mathbb{R}^n$.

We use this approach to derive the equation (3.3) in the paper by Zhou and Wang[6]. Consider the second directional derivative:

$$\begin{aligned} J''(u)(v, w) &= -\int_\Omega (\Delta v + v\, f'(u))\, w\, dx \\ &= \langle D_2^2 J(u)v, w\rangle_2. \end{aligned}$$

We have that for some v:

$$D_2^2 J(u)v = \nabla J(u).$$
$$\text{So, } J''(u)(v, w) = J'(u)(w),$$

$$\text{recalling that } J'(u)(w) = \int_\Omega (\nabla u - \ f(u))\, w\, dx.$$

Since the Hessian matrices are self-adjoint bilinear operators, using the *Riesz Representation Theorem* we can solve for a vector v to satisfy this equality:

$$\langle D_2^2 J(u)v, w\rangle = \langle \nabla J(u), w\rangle. \tag{5}$$

From related work by Zhou and Wang in [6], this equivalent to Equation (3.3).

3.2 Solving Zhou's Equation for Newton Direction

As derived above, setting $\langle J'(u), w\rangle = \langle J''(u)v, w\rangle$ allows us to determine the directional vector v by weakly solving the Equation 6 as is done in [6].

$$\begin{cases} -\Delta v(x) - f'_u(x, u(x))v(x) = -\Delta u(x) - f(x, u(x), & x \in \Omega \\ v(x) = 0, & x \in \partial\Omega \end{cases} \tag{6}$$

Consider solving for the vector of unknowns, v, and note that provided with an initial guess for u and its respective discretized Laplacian, we can generate at a linear system $\mathbf{A}\vec{v} = \vec{G}$. On the left side of our system, we have that $A_{i,k} \simeq (-\Delta - f'(u))$. Using discrete approximations, this equates to $A_{i,k} = B_{i,k}(\lambda_k - (s - 3u_i^2))$ where B is the basis of orthonormal eigenfunctions of the Laplacian in L_2 and λ are eigenvalues. On the left

hand side of the system, we have that $-\Delta u - f(u)$ is approximated as $B_{i,k}\, g_i = G_i$, the basis multiplied with our gradient vector from traditional GNGA .

Now, we have n equations for the m unknowns of the vector $\vec{v}$, an overdetermined and simple to compute system. Our basis or eigenfunctions will not change, requiring only one computation, and the Hessian need not be computed. The Newton direction we determined can be used to update our coefficient vector a, allowing us to then compute a new u, and G until we hone in on a solution. However, this still forces us to find a least squares solution to $\mathbf{A}\vec{v} = \vec{g}$ which becomes very slow as the size of the grid in increased, thus losing its benefit. To correct this, we left multiply the system by B^T/n, knowing that $B^T B/n = I_{mxm}$. So we have:

$$B_{i,k}^T B_{i,k}/n\ [\lambda_k - (s - 3u_i^2)]v_k = B_{i,k}^T B_{i,k}/n\ [a_i(\lambda_i - s) - B_{i,k}^T \frac{(su_i + u_i^3)}{n}]$$
$$I_{mxm}[\lambda_k - B_{i,k}^T (s + 3u_i^2) B_{i,k}]v_k = I_{mxm}[a_i(\lambda_i - s) - B_{i,k}^T \frac{(su_i + u_i^3)}{n}]$$
$$[\lambda_k - B_{i,k}^T (s + 3u_i^2) B_{i,k}/n]v_k = [a_i(\lambda_i - s) - B_{i,k}^T \frac{(su_i + u_i^3)}{n}]$$

Note that the right hand side is simplified to our original gradient vector. Essentially, by solving the equation in the Zhou paper we have a method of calculating the Hessian without any loops that does not require solving an overdetermined system [6]. Simply,

$$A_{i,k} = \lambda_k - B_{i,k}^T (s + 3u_i^2) B_{i,k}/n,$$
$$g = a_i(\lambda_i - s) - B_{i,k}^T \frac{(su_i + u_i^3)}{n},$$

and $v = A^{-1}g$. We now refer to this search direction as *FHS* for *Faster Hessian Solution*.

Implementing this into GNGA we have the following:

Algorithm 2 The GNGA After Implementing *FHS*

Define Ω, f, L, and δ.
Construct vector of eigenvalues λ and orthonormal basis $B = \{\Psi\}_{k=1}^{M}$
Choose initial coefficient vector a^0, and set $u \leftarrow u^0 = \Sigma a_k \psi_k$
Set $j = 0$
while $\|g\|_2 >$ tolerance **do**
  Calculate the gradient vector $g_i = a_i(\lambda_i - s) - B_{i,k}^T \frac{(su + u_i^3)}{n}$
  Calculate the Hessian matrix, $A_{i,k} = \lambda_k - B_{i,k}^T (s - 3u_i^2) B_{i,k}/n$
  Determine Newton direction such that $v = A^{-1}g$
  Update vector a as $a^{n+1} = a^n - v$, then calculate u, and g
  $j = j + 1$
end while

4 Results of Comparing *FHS* with the Traditional Hessian Calculation

4.1 Results on the Unit Interval

In order to determine the usefulness of the new algorithm, it's speed was compared to the original `GNGA` using `MATLAB` . The program was initially run on $m = 100$ modes and a grid size $n \times n$ where $n = 2m+1$. It uses Newton's method to converge to a solution with an initial guess $a(1) = 1$ and a tolerance of 10^{-9} then plots the norm of the resulting solution of u. Examples can be seen in Figures 3 and 4. It also outputs the vector, χ in `GNGA` or v in *FHS* , so that we can be sure that our Hessian calculations are valid. At each experiment, we found equality between the two.

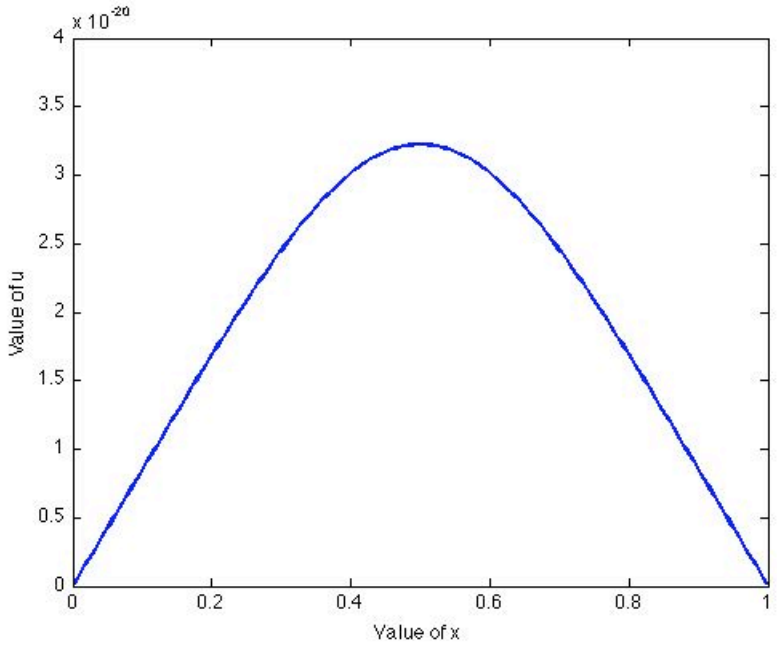


Figure 3: The ψ_1 solution to the ODE

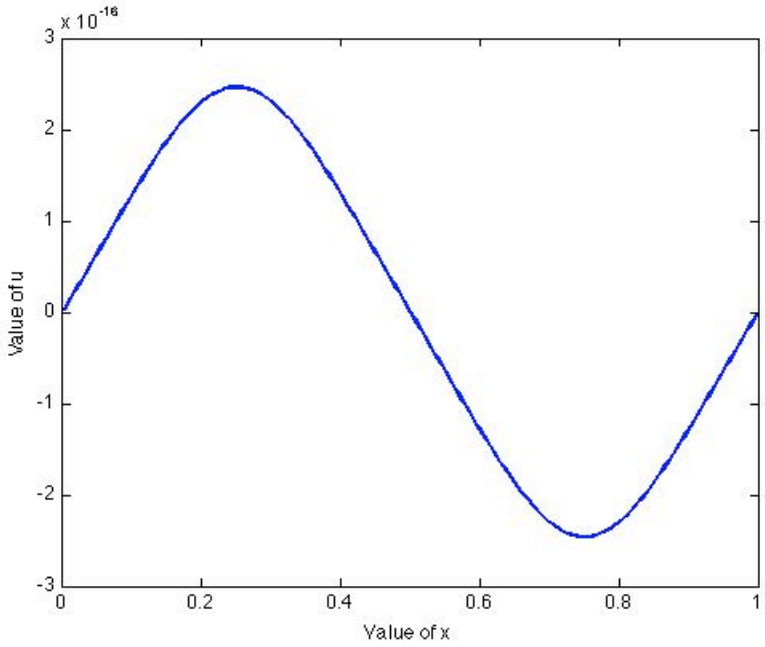


Figure 4: The ψ_2 solution to the ODE

At first, it appeared that the two algorithms were comparable and that *FHS* did not necessarily speed things up. However, if we weight the step size by a factor $\kappa = .1$, was introduced into the Newton step for smoothness, the result was much more promising. Before the addition of κ, traditional `GNGA` took 0.58 seconds and *FHS* took 0.25 seconds. With the weighting factor, *FHS* took 0.94 seconds compared to 18.94 seconds in `GNGA` . This is roughly a 95% difference in time, or a factor of 20. This change in speed gives us hope that we may be able to find new solutions in higher dimensions since computation time is reduced.

Table 1: Comparison of Calculation Time for ODE (in seconds)

m	50	100	150	200	250	300	350	400	450	500
Original	0.08	0.31	0.75	1.41	2.31	3.55	5.26	7.09	9.50	12.67
FHS	0.02	0.02	0.04	0.09	0.14	0.21	0.34	0.45	0.62	.86
Factor	4.17	14.86	18.54	15.90	16.75	16.74	15.43	15.78	15.30	14.66

What we did not account for was the amount of time needed to generate a plot in both cases. As m increased, this time became less important, and the factor by which *FHS* was faster increased for larger m to an approximate average value. So instead we looked at the time needed simply to converge to a solution in each case by varying the grid size by changing $m = 50$ to $m = 500$ by increments of 50, then comparing the elapsed times. Clearly, the new method is significantly faster than the traditional approach, suggesting that fewer calculations may be necessary.

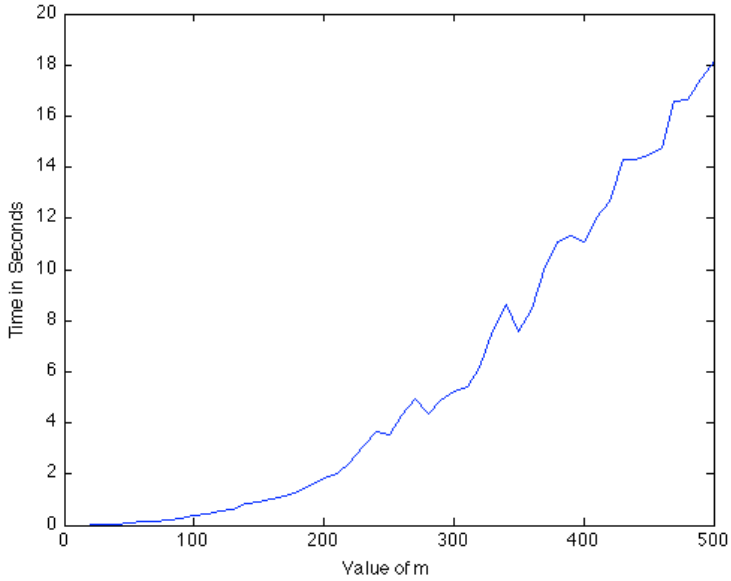


Figure 5: Plot of time vs m on the ODE using traditional GNGA

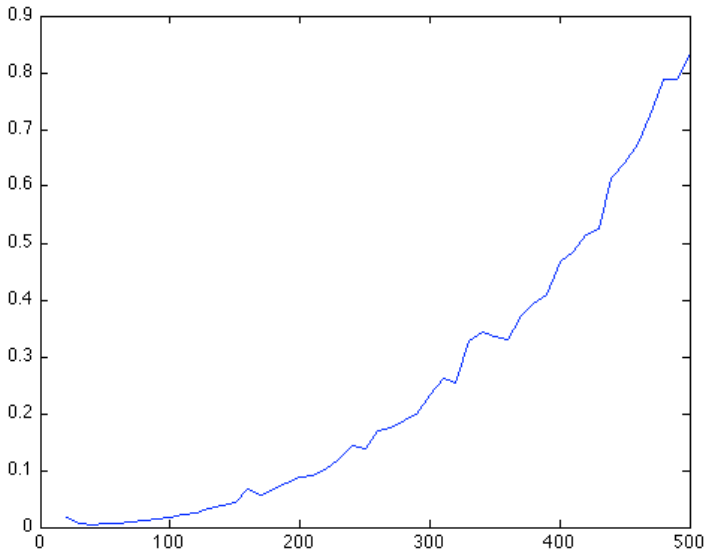


Figure 6: Plot of m vs time on the ODE using *FHS*

Another thing we observe is the relationship between the time needed for calculation and the variable m in the ODE. By Figures 5 and 6 we see that although the relationship

is not completely uniform, it does seem to follow a linear path in both cases for $m \geq 200$. The nonlinearity of $m \leq 200$ may be caused by overhead in `MATLAB` calculations. The curves are similar in shape, however the rate of growth is significantly less using the ***FHS*** method. This can be seen by looking at the time scale in both figures.

4.2 Implementing *FHS* on the Square

The next step in comparing the two methods of calculating a Hessian was to move into the next dimension and test the two algorithms on the square. In order to do this, a new two-dimensional basis was needed.

Before generating a basis of eigenfunctions, we require a new list of eigenvalues. In one dimension, we had $\lambda_p = p^2\pi^2$ for an integer $p \in \{1, 2, \ldots, m\}$. On the unit square we must index by two integers p, q from the same set. This is done by looping over the elements q for each element p to generate a list of eigenvalues such that $\lambda_{p,q} = (p^2 + q^2)\pi^2$ concatenated with columns that store the corresponding p, q ordered pair. We then sort the table by the column of eigenvalues $\lambda_{p,q}$ in ascending order, retaining the values of p and q. A truncated table is shown below.

$\lambda_{p,q}$	p	q
$2\pi^2$	1	1
$5\pi^2$	2	1
$5\pi^2$	1	2
$8\pi^2$	2	2
$13\pi^2$	3	2

Next, to generate the square basis, we calculate a one-dimensional $n \times n$ basis as before, then take the outer product of column p with column q for all p, q. At each calculation, we get an $n \times n$ matrix which is then identified as a $n^2 \times 1$ column vector using the `MATLAB` command `reshape` and concatenated to the right of the previous column vector. Thus, our new two-dimensional basis is $n^2 \times m^2$ rather than $n \times m$. We must scale our basis by n^2 so that our eigenfunctions are still normalized. Here, we expect to find solutions that display symmetry, as seen in the plots of solutions in Figures 7 - 10 where we display contours of $||u||$ on an $n \times n$ grid.

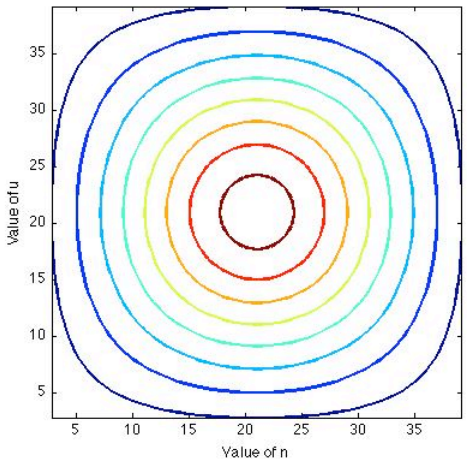


Figure 7: The ψ_1 solution to the PDE

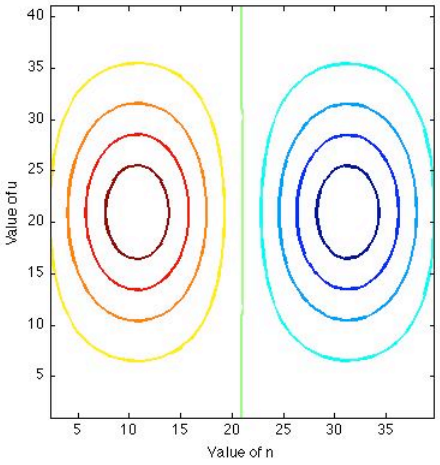


Figure 8: The ψ_2 solution

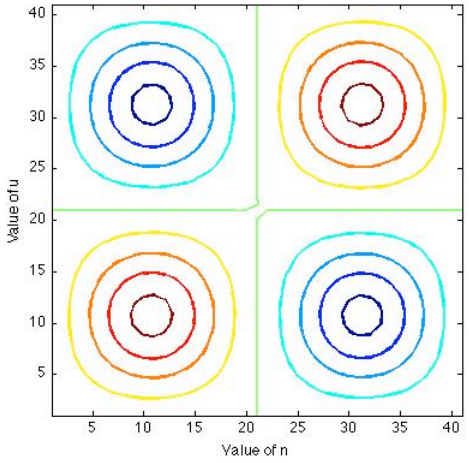


Figure 9: The ψ_4 solution

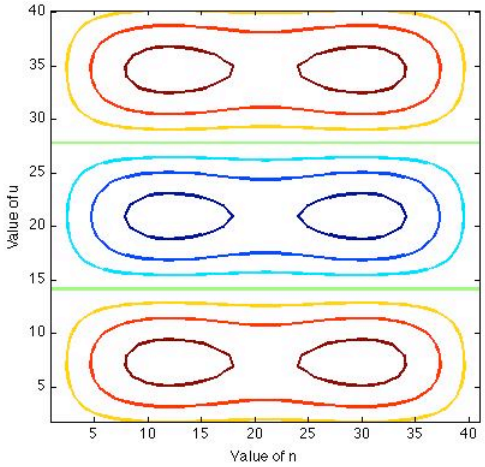


Figure 10: The ψ_6 solution

Now that we have a basis of eigenfunctions with corresponding eigenvalues, as well as a Hessian and a gradient, `GNGA` can be run as before. We again can vary the grid size by changing m to compare the time each calculation takes. Since the number of modes and grid points become squared, the size of the matrices increase quadratically, so we use smaller m for our comparison. Here, we take $m \in \{5, 10, \ldots, 25\}$. Also, since the calculation is already on a larger grid, we did not need to introduce a κ to weight the search direction in Newton's method.

Table 2: Comparison of Calculation Time for PDE on the Square (in seconds)

m	5	10	15	20	25
Original	0.0829	0.3389	2.6889	13.5182	151.2837
FHS	0.0388	0.0797	0.6247	3.2891	11.8090
Factor	2.1341	4.2511	4.3040	4.1099	12.8109

As expected, using *FHS* reduced the time required to solve the system (see Table 2). To show this on an even larger grid where the factor of time would be higher, the system was also tested with a basis that was $n^2 \times m^2 = 1681 \times 1600$, or $m = 40,\ n = 81$. Using *FHS* the `GNGA` took 3.12 minutes compared to 32.25 minutes with the traditional calculation. This is a reduction in time by a factor of 10.3.

5 Counting Calculations in Each Approach

We seek to explain not only which method of calculating the Hessian is faster, but also what causes the time difference. In order to do so one would need some knowledge of how `MATLAB` handles matrix calculations when executing commands such as `diag` or `spdiags`, `MATLAB` commands for creating diagonal and sparse diagonal matrices, respectively. Experimentation concerning efficiency was done and is included in the appendix, however, finding the difference in the order of calculations necessary only requires counting operations in simple matrix multiplications.

In the original method of calculating the Hessian, numerical integration is approximated for the upper half of a triangular matrix. Since we know the matrix is symmetric, the lower triangle is then filled with values from the upper half. Consider the following `MATLAB` code to calculate one entry of the Hessian:

$$(\lambda_i - s)\delta_{i,j} - B(:,i)'.*B(:,j)'*(3*u.^2)/n$$

If we break this up into its components and determine how many operations happen at one grid point, we can determine how many it takes for the whole matrix. Recall that there are $\frac{m(m+1)}{2}$ grid points in the upper triangle, and $\frac{m(m-1)}{2}$ in the lower.

Table 3: Number of Calculations in Each Component for Traditional Hessian

$\lambda_i - s$	$(\lambda_i - s)\delta_{i,j}$	$B(:,i)'.*B(:,j)'$	$(3*u_i.^2)/n$	$\vec{B'}*\vec{f'(u)}/n$	Subtraction
1	1	n	3n	2n-1	1

Now, if we consider that these $6n + 2$ calculations are done at each of the upper triangular entries, we have that $(6n+2)\frac{m(m+1)}{2}$ calculations happen in the upper triangle. These are then transfered to the lower triangle of the matrix adding $\frac{m(m-1)}{2}$ calculations. Since we

have that $n = 2m + 1$, we find that in total, $6m^3 + \frac{21}{2}m^2 + \frac{7}{2}m$ calculations are required to calculate the Hessian. So it is of the order $6m^3$.

Using ***FHS*** , we expect that since the time is greatly reduced that we are doing fewer calculations. We do not loop over grid points, rather we rely on matrix multiplication. So, in `MATLAB` :

$$\texttt{diag}(\lambda - s) - B' * \texttt{diag}(3 * u.^2) * B/n.$$

Again, we can break this down component-wise to determine the expense. Recall that the multiplication of an $a \times b$ matrix with a $b \times c$ matrix requires $ac(2b - 1)$ calculations: abc multiplications and $ac(b - 1)$ additions.

Table 4: Number of Calculations in Each Component for ***FHS***

$\lambda - s$	$3(u.^2)$	$A = B' * \texttt{diag}(3 * u.^2)$	$A * B$	$/n$	Subtraction
m	2n	mn(n-1)	$m^2(n-1)$	m^2	m^2
Total: $m^2n + m^2 + mn^2 - mn + m + 2n$					

Again, we can use the relationship between m and n. We have that in total, ***FHS*** takes $6m^3 + 4m + 5m + 2$, so it is of the same order as the original method, contrary to our hypothesis. Note that we do not understand how `MATLAB` generates a diagonal matrix, only how it handles it during computation. At most, it would seem that it would take m^2 calculations for $\texttt{diag}(\lambda - s)$ and n^2 for $\texttt{diag}(3 * u.^2)$. However, these additions would not significantly affect the difference in computations needed.

Since this result was surprising, we now seek to determine why ***FHS*** is still faster. First, we ran more naive matrix experiments in `MATLAB` to compare the time it takes for the built in multiplication function versus manually computation through loops. We populated an $m \times n$ matrix $\mathbf{A}$ with random values and multiplied it with its transpose. Next, we performed the same calculation by looping over every grid point and manually summing the components, in other words hard coding the calculation with three nested loops. Lastly, we looped over the columns and used the built in dot product function. As was expected, with both moderate and large grids ($m = 30, n = 50$ and $m = 300, n = 500$), we found that the built in function was fastest, followed by using the dot product in nested loops, and finally the hard-coded version.

5.1 How `MATLAB` Optimizes Matrix Multiplication

With some research into how `MATLAB` actually handles these calculations, it was found that it uses a library of built in functions called LAPACK. This library uses block algorithms which handle and evaluate multiple columns of a given matrix at a time. The speed of LAPACK is based on its use of Basic Linear Algebra Subroutines (BLAS) which provide the speed by allowing it to work on larger portions of matrices during calculations. Essentially, the addition of LAPACK to `MATLAB` added a significant number of paths to optimize calculations based on characteristics of the matrices being used. I.e. whether the

matrix contains real or complex values, whether it is symmetric, and what properties or calculations is being asked by the user[2].

This tells us that in the original calculation for the Hessian, the time difference was generated in the loops that were used to find the value at each grid point. `MATLAB` could not optimize the calculation since it was confined to the grid point in question rather than being able to call LAPACK paths for an optimized matrix multiplication. We were only allowing for `MATLAB` to work on part of the matrices at a time. In *FHS* , we allow `MATLAB` to optimize the calculation through its subroutines and block operators, which speeds up the calculation as a whole.

6 Conclusions and Future Work

The results found in this experiment satisfied the desire of our mentors for a method of calculating the Hessian in one line of code. The *FHS* method outlined here is not only more concise in coding, but is also handled faster computationally in `MATLAB` in one and two dimensions. We find that although the *FHS* and the traditional calculation are of the same order of computations in theory, *FHS* relies more on built in `MATLAB` functions which can be optimized for speed. It is our hope that the implimentation of this solution to the `GNGA` will enable solution finding on large grids or more complex PDE, as well as in higher dimensions in future work.

Some work done this summer that is not included in the body of this paper includes using Newton's method to plot basins of attraction, which is detailed in section B of the appendix. We also implemented *FHS* into the modified mountain pass algorithm (`MMPA`) outlined in the thesis work of Dr. Neuberger in [1]. Using the `MMPA`, we only have enough constraints find solutions for the PDE (1) when the Morse index of the hessian is less than three. This was verified in the implementation experiment. It is possible that in future work we may be able to enforce additional constraints to find solutions with higher Morse index.

Another place I would like to implement the *FHS* is into the research of my partners, Nate Veldt and Jake Clark, on the PDE with spherical coordinates. The necessary alterations to *FHS* are minor; we must implement a different volume element for integration, and generate a basis of eigenfunctions based on the spherical Bessel functions. The *FHS* could also be implemented to study the Swift-Hohenberg equation $-(\Delta+k^2)^2u+su-u^3 = 0$, or in (1) using various other Sobolev exponents.

7 Acknowledgements

I would like to thank Drs. John Neuberger and Jim Swift for introducing me to the area of variational methods for PDE's, in particular for the guidance in implementing their algorithms. This program taught me a lot, and I enjoyed the work, despite frustration at times.

I would also like to thank my partners, Nate Veldt, Jake Clark, and Rob Ravier for their contributions and companionship. I appreciated working together and learning from each other throughout the program.

Also, thank you to Northern Arizona University for the hospitality and opportunity, and to the National Science Foundation for funding our project and enabling our experience.

References

[1] Alfonso Castro, Jorge Cossio, and John M Neuberger. A sign-changing solution for a superlinear dirichlet problem. 2003.

[2] Cleve Moler. Matlab incorporates lapack: Increasing the speed and capabilities of matrix computation. `http://www.mathworks.com/company/newsletters/news_notes/clevescorner/winter2000.cleve.html`, Winter 2000.

[3] John M. Neuberger, Nándor Sieben, and James W. Swift. Automated bifurcation analysis for nonlinear elliptic partial difference equations on graphs. *Internat. J. Bifur. Chaos Appl. Sci. Engrg.*, 19(8):2531–2556, 2009.

[4] John M. Neuberger and James W. Swift. Newton's method and Morse index for semilinear elliptic PDEs. *Internat. J. Bifur. Chaos Appl. Sci. Engrg.*, 11(3):801–820, 2001.

[5] Paul H. Rabinowitz. *Minimax methods in critical point theory with applications to differential equations*, volume 65 of *CBMS Regional Conference Series in Mathematics*. Published for the Conference Board of the Mathematical Sciences, Washington, DC, 1986.

[6] Zhi-Qiang Wang and Jianxin Zhou. A local minimax-Newton method for finding multiple saddle points with symmetries. *SIAM J. Numer. Anal.*, 42(4):1745–1759, 2004.

Appendix

A Proof of the Convergence of Newton's Method

Definition 1. *If $g : \mathbb{R} \to \mathbb{R}$ satisfies $g(p) = p$, then we call p a fixed point.*

Theorem 1. *If $g : [a, b] \to^c [a, b]$ then g has a fixed point.*

Proof. Let $h(x) = x - g(x)$. Also, assume $g(a) \neq a$ and $g(b) \neq b$. Now, since the range of g is $[a, b]$, we have that $g(a) > a$ and $g(b) < b$. This tells us that $h(a) < 0$ and $h(b) > 0$, and since h is continuous, being the difference of two continuous functions, by the $IntermediateValueTheorem$ there exists a point $p \in [a, b]$ such that $h(p) = 0$. Thus, $h(p) = 0 = p - g(p)$ and we have shown that $\exists\, p \in [a, b]$ such that $g(p) = p$. □

Theorem 2. *If also, $\exists\, \sigma \in (0, 1)$ with $|g'(x)| \leq \sigma$ on $[a, b]$. Then:*

1. *The fixed point on the interval is unique.*
2. *The fixed point iteration, $p_{k+1} = p_k - \frac{f(p_k)}{f'(p_k)}$, converges to p.*

Proof. Let p, q be fixed points over $[a, b]$ Now, the slope of the secant between p and q is $|\frac{g(p)-g(q)}{p-q}|$. However, since p and q are fixed points, we have that the slope is $|\frac{p-q}{p-q}| = 1 \not\leq 1$. Thus contradicting our assumption that $\sigma < 1$. Therefore, there can only be one fixed point over$[a, b]$.

Next, let $p_0 \in [a, b]$, and $p_{k+1} = g(p_k)$. By the $MeanValueTheorem$, there exists a value $c \in [a, b]$ such that for some $r, s \in [a, b], |\frac{g(r)-g(s)}{r-s}| = |g'(c)|$. Using this, it can be seen that:

$$\begin{aligned} |p - p_1| &= |g(p) - g(p_0)| \\ &= |g'(c_0)(p - p_0)| \\ &= |g'(c_0)||p - p_0| \\ &\leq \sigma|p - p_0| \\ &< |p - p_0| \end{aligned}$$

Thus, $|p - p_1| < |p - p_0|$, and p_1 is no futher from p than p_0. To show convergence of

fixed point iteration, we must show $\lim_{n\to\infty} |p-p_k| = 0$. Observe that $|p-p_k| \leq \sigma^k |p-p_0|$ for k = 1. Now, suppose $|p-p_{k-1}| \leq \sigma^{k-1}|p-p_0|$. From earlier, we know that:

$$|p-p_k| \leq \sigma|p-p_{k-1}|$$

and,

$$|p-p_{k-1}| \leq \sigma^{k-1}|p-p_0|$$

Thus,

$$|p-p_k| \leq \sigma^k|p-p_0|$$

Since $0 < \sigma < 1$, we have that as $n \to \infty$, $\sigma^n \to 0$. Thus, $0 \leq lim_{k\to\infty}|p-p_k| \leq \lim_{k\to\infty} \sigma^k|p-p_0| = 0$. Because $|p-p_0| \to 0|$, it follows that $p_k \to p$. Thus, we have shown that $g(p_{k-1}) = p_k \to p$. □

Theorem 3. *If $f : \mathbb{R} \to \mathbb{R}$ is continuous and twice differentiable, and $f(p) = 0$, $f'(p) \neq 0$, then $\exists\ \delta > 0$ such that if $p_0 \in B_\delta(p)$ where B is a ball with radius δ, then Newton's Method converges.*

Proof. Let us again define $p_{k+1} = g(p_k) = p_k + \frac{f(p_k)}{f'(p_k)}$. Thus g is a generating function for the sequence P_k with $p_k \to p$ as $k \to \infty$ by $\mathbf{Theorem 2}$.

Now, if $s \in B_\delta(p)$, then similarly we have that $s \in [p - delta, p + \delta]$. Since $f'(p) \neq 0$, and $g(p) = p$, we have that g is continuous and small near p. Thus, $\exists\ \delta \geq 0$ such that $|g'(x)| \leq \frac{1}{2} \leq \sigma < 1$ on $(p - \frac{\delta}{2}, p + \frac{\delta}{2})$. By the $MeanValueTheorem$, we have that:

$$\begin{aligned} |g'(x)| &= |\frac{g(p+\frac{\delta}{2}) - g(p-\frac{\delta}{2})}{(p+\frac{\delta}{2}) - (p-\frac{\delta}{2})}| \\ &= |\frac{g(p+\frac{\delta}{2}) - g(p-\frac{\delta}{2})}{2\frac{\delta}{2}}| \end{aligned}$$

$$\begin{aligned} \text{So,} \quad & |\delta g'(x)| = |g(p+\tfrac{\delta}{2}) - g(p-\tfrac{\delta}{2})| \\ \text{Therefore,} \quad & \delta\sigma \leq |g(p+\tfrac{\delta}{2}) - g(p-\tfrac{\delta}{2})| \end{aligned}$$

Since we have that $\delta \leq \frac{1}{2}$ and $\sigma \leq 1$, this tells us that initial guesses of points within δ of the fixed point will converge to p. □

B Using Newton's Method to Plot Basins of Attraction

Occasionally when solving equations with Newton's Method, it is helpful and interesting to plot a diagram of the basins of attraction. We do this by taking a two-by-two grid of

points, separated by a step size, and letting each grid point represent an initial guess for the root of a specific function. When we run Newton's method on one of these points, we can discover which root of the equation of interest each initial guess converges to, or if it converges at all. To visualize this convergence, we assign a color to each root, and a shade to depict how many iterations the point takes to converge to the root. We then plot the grid points and their corresponding color. For example, a dark red point and a light red point both indicate initial guesses that converge to the same root, but the darker shading indicates that it took more iterations in order to converge.

A basin of attraction in this sense is a set of points in our resulting diagram that all converge to the same root of the function in question. Namely, all of the points which are the same color, shading aside. These diagrams are often fractals and are pleasing to the eye as well as enlightening as to the behavior of the function.

B.1 Examples

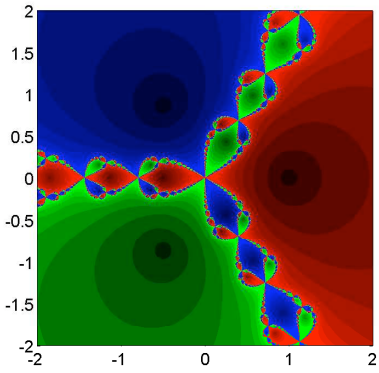
It is easy to plot basins of attraction for simple complex functions such as $g(z) = z^3 - 1$. In this case we simply choose a section of the complex plane for our initial guesses, and run the sequence

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)},$$

until the initial guess converges to one of three roots.

When we run Newton's Method on the points in the complex plane from -5 to 5 on each axis, we obtain Figure 11.

Figure 11: Basins of Attraction for $g(z) = z^3 - 1$



As we can see, there are three distinct roots that each point might possibly converge to. We can do the math to find that these roots are $1 + 0i$, $-\frac{1}{2} + \frac{\sqrt{3}}{2}i$, and $-\frac{1}{2} - \frac{\sqrt{3}}{2}i$, namely three roots of unity. Notice that there is a relatively large area surrounding each root where points converge to the root closest to them. We also note a fractal boundary between roots, where it is harder to tell exactly where each point will converge, given an initial guess.

C Poincaré Inequality

The *Poincaré Inequality* allows us to obtain bounds on the function u using knowledge of its derivatives and defined domain. The classical Poincaré Inequality is as follows:

Poincaré Inequality. *Consider $p \in \mathbb{Z}$ s.t. $1 \leq p \leq \infty$ for a bounded region $\Omega \in \mathbb{R}^n$ with Lipschitz boundary. There exists $C \in \mathbb{Z}$ which depends only on Ω and p s.t. for all functions $u \in W^{1,p}(\Omega)$ - a Sobolev space- we have:*

$$||u - u_\Omega||_{L^p(\Omega)} \leq C||\nabla u||_{L^p(\Omega)}$$

where

$$u_\Omega = \frac{1}{|\Omega|}\int_\Omega u(x)dx$$

is the average value of u over Ω and $|\Omega|$ is the norm in the space.

Generalized for our system, we have that:

$$||u||_H^2 \geq \lambda_1||u||_{L_2}^2$$

where $||u||_H = \int (u')^2$ is the Sobolev norm, and $||u||_{L_2}$ is the L_2 gradient. Thus the H norm is always greater than or equal to the product of the first eigenvalue and the norm in L_2. The proof follows from the Pythagorean theorem.

Proof.

$$\begin{aligned}
||u||_H^2 &= ||\sum u_i\Psi_i||_H^2 \\
&= \sum u_i^2||\Psi_i||_H^2 \\
&= \sum u_i^2\lambda_i \\
&\geq \lambda_i\sum u_i^2 = \lambda_1\sum u_i^2||\Psi_i||_2^2 = \lambda_1||u||_2^2
\end{aligned}$$

□

D Matrix Multiplication Experiments in MATLAB

While trying to track the number of calculations needed to calculate the Hessian matrix in different ways, It was necessary to understand the basics of how `MATLAB` handles matrices. For example, if we were using a diagonal matrix, we were unsure if `MATLAB` recognized this and ignored the zeros during calculation, or if every entry was still used. We also wished to determine the value of using the `spdiags` command to fill a sparse matrix.

Table 5: Elapsed Time for Calculations with 100×100 Matrices and 100×1 Vectors

Calculation	Filled/Filled	Filled/Diag	Filled/Spdiags	Filled/Vector
Time (s)	4.33e-4	4.50e-4	3.97e-3	4.10e-5

The comparisons were done by filling square matrices with random numbers, and calculating the time elapsed for various combinations of multiplications. First, the sizes were 100×100. The multiplications done were: two filled matrices, a matrix with a vector, a filled matrix with a diagonal matrix, and a filled matrix with a diagonal matrix using `spdiags`. The time elapsed for the diagonal matrix and the two filled matrices were of the same order of magnitude after repeated tests, suggesting that `MATLAB` does not treat them distinctly. Also, the `spdiags` multiplication took an order of magnitude longer than regular matrix multiplication. The vector multiplication was the fastest by an order or magnitude.

Table 6: Elapsed Time for Calculations with 1000×1000 Matrices and 1000×1 Vectors

Calculation	Filled/Filled	Filled/Diag	Filled/Spdiags	Filled/Vector
Time (s)	0.1970	0.2623	0.029410	0.0042

Next, the size of the matrices was increased to 1000×1000. Here, the calculation time for the multiplication two filled matrices jumped up in time by two orders of magnitude. The same was true for a filled matrix with a diagonal, and for a matrix with a vector . The calculation with `spdiags` remained of the same order. This tells us that there is some overhead expense to using `spdiags`, however, in bigger calculations it is faster since `MATLAB` recognizes that it is sparse and treats the zero entries differently.

E Sine and Cosine on the Interval using Midpoint and Trapezoid Rule

Exact Numerical Approximation of Sine and Cosine using Midpoint and Trapezoid Rule. *Assume $k, \in \mathbb{Z}^+$ and $n \in \mathbb{Z}^+$.*

The Midpoint and Trapezoid rule give the exact answer for $\int_0^1 \cos(k\pi x)dx$ if and only if k is odd, or k is even and $n \nmid \frac{k}{2}$, and the exact answer for $\int_0^1 \sin(k\pi x)dx$ if and only if k is even.

Proof. Let us begin with the Trapezoid rule. Let us denote the Trapezoid rule for our integral $\int_0^1 e^{i\pi kx}dx$ as

$$(C + iS)^T_{n,k}.$$

We shall denote the midpoint rule similarly.

By the definition of the trapezoid rule, we expand $(C+iS)_{n,k}^T$ as follows.

$$(C+iS)_{n,k}^T = \frac{1}{n}\left(\frac{1}{2}e^0 + \left[\sum_{j=1}^{n-1}\left(e^{\frac{i\pi k}{n}}\right)^j\right] + \frac{1}{2}e^{i\pi k}\right) \tag{7}$$

The Midpoint rule is denoted

$$(C+iS)_{n,k}^M = \frac{1}{n}\sum_{j=0}^{n-1}\left(e^{ik\pi}\right)^{\frac{j+\frac{1}{2}}{n}} \tag{8}$$

For ease of notation, let us define $\omega = e^{\frac{i\pi k}{n}}$. This means that

$$\omega = e^{\frac{i\pi k}{n}} = \cos(k\pi x) + i\cdot\sin(k\pi x)$$

We can now rewrite (7) above as

$$(C+iS)_{n,k}^T = \frac{1}{n}\left(\frac{1}{2}e^0 + \left[\sum_{j=1}^{n-1}(\omega)^j\right] + \frac{1}{2}e^{i\pi k}\right)$$

and write (8) similarly.

To make any progress on the theorem, we must break into cases. Namely, k odd and k even.

Case 1

Now, let us first suppose k is odd. It may be helpful to note that $\omega^{\frac{1}{2}} = e^{\frac{i\pi k}{2n}}$ Now, we can compute the exact integral for ω, namely,

$$\int_0^1 \omega dx = \int_0^1 e^{i\pi kx}dx = \frac{2i}{\pi k}$$

for odd k. Expanding (7), noticing the first and last terms cancel, leaves us with

$$\begin{aligned}
\frac{1}{n}\sum_{j=1}^{n-1}\left(e^{\frac{i\pi k}{n}}\right)^j &= \frac{1}{n}\left(\frac{\omega-\omega^n}{1-\omega}\right) \\
&= \frac{1}{n}\left(\frac{\omega+1}{1-\omega}\right) = \frac{1}{n}\frac{(\omega+1)(1-\bar{\omega})}{1-\omega-\bar{\omega}+1} \\
&= \frac{1}{n}\frac{\left(e^{\frac{i\pi k}{n}}+1\right)\left(1-e^{\frac{-i\pi k}{n}}\right)}{2\left(1-\cos(\frac{k\pi}{n})\right)} \\
&= \frac{1}{n}\frac{2\cdot i\cdot\sin(\frac{k\pi}{n})}{2\left(1-\cos(\frac{k\pi}{n})\right)} \\
&= \frac{1}{n}\frac{i\cdot\sin(\frac{k\pi}{n})}{\left(1-\cos(\frac{k\pi}{n})\right)}
\end{aligned}$$

This number is purely imaginary. Clearly, this demonstrates that the cosine term is exact (namely, 0), and the sine, or rather the imaginary term in our expansion is not exact, though it approaches the exact answer as $n \to \infty$.

For the midpoint rule, and k odd, one would do the following:

$$\begin{aligned}
(C+iS)_{n,k}^{M} &= \frac{1}{n}\sum_{j=0}^{\infty}\left(e^{ik\pi}\right)^{\frac{j+\frac{1}{2}}{n}} \\
&= \frac{1}{n}\sum_{j=0}^{\infty}\left(\omega^{\frac{1}{2}}\omega^{j}\right) \\
&= \frac{1}{n}\left(\frac{\omega^{\frac{1}{2}}-\omega^{\frac{1}{2}}\omega^{n}}{1-\omega}\right) \\
&= \frac{1}{n}\cdot\frac{2\omega^{\frac{1}{2}}}{1-\omega} \\
&= \frac{2}{n}\cdot\frac{\omega^{\frac{1}{2}}\left(1-\bar{\omega}\right)}{\left(1-\omega\right)\left(1-\bar{\omega}\right)} \\
&= \frac{2}{n}\cdot\frac{\omega^{\frac{1}{2}}-\omega^{\frac{-1}{2}}}{1-\omega-\bar{\omega}+1} \\
&= \frac{2}{n}\frac{e^{\frac{ik\pi}{2n}}-e^{-\frac{ik\pi}{2n}}}{2-2\cos\left(\frac{k\pi}{n}\right)} \\
&= \frac{2}{n}\frac{2i\cdot\sin\left(\frac{k\pi}{2n}\right)}{2\left(1-\cos\left(\frac{k\pi}{n}\right)\right)}
\end{aligned}$$

So, finally, for the midpoint rule,

$$(C+iS)_{n,k}^{M} = \frac{2}{n}\frac{i\cdot\sin\left(\frac{k\pi}{2n}\right)}{1-\cos\left(\frac{k\pi}{n}\right)}$$

Thus, $C_{k,n}^{M} = 0$ and $S_{k,n}^{M} = \frac{2}{n}\frac{\sin\left(\frac{k\pi}{2n}\right)}{1-\cos\left(\frac{k\pi}{n}\right)}$ So, we can conclude that Cosine is exact for any odd k, and we can say that Sine is not exact for any odd k. This simply means that, while $\frac{2}{n}\frac{\sin\left(\frac{k\pi}{2n}\right)}{1-\cos\left(\frac{k\pi}{n}\right)} \to \frac{2}{k\pi}$ as $n \to \infty$, which can be verified by a computer algebra system, or other for any finite value of n, $\frac{2}{n}\frac{\sin\left(\frac{k\pi}{2n}\right)}{1-\cos\left(\frac{k\pi}{n}\right)} \neq \frac{2}{k\pi}$

Case 2

Let us suppose k is even. To prove this, we must use exhaustive cases. Let us again denote the midpoint rule for ω as $(C+iS)_{k,n}^{M}$ and the trapezoid rule as $(C+iS)_{k,n}^{T}$.

This leaves us with two cases. The case when $n \mid \frac{k}{2}$ and when $n \nmid \frac{k}{2}$

To make this easy, let us compute the case where the function we are computing the integral of is

$$\omega_1 = e^{\frac{2i\pi k}{n}} = \cos(2k\pi x) + i \cdot \sin(2k\pi x).$$

Let us first look at the Trapezoid rule for $\int_0^1 \omega_1$. Now,

$$(C + iS)_{k,n}^T = \frac{1}{n} \sum_{j=0}^{n-1} \omega_1^j$$

and, in turn the midpoint rule is just shifted by an angle, namely ω. Thus,

$$(C + iS)_{k,n}^M = \frac{1}{n} \sum_{j=0}^{n-1} \omega_1^j \omega.$$

I will cover the case of the Midpoint rule, due to the fact that the trapezoid rule is sufficiently similar.

In the case of the Midpoint rule,

$$(C + iS)_{k,n}^M = \frac{1}{n} \sum_{j=0}^{n-1} \omega_1^j \omega = \frac{\omega}{n} \sum_{j=0}^{n-1} \omega_1^j.$$

Now, we break into two sub cases.

Let us now cover the midpoint rule when k is odd.

Case 1.a: $n \mid k$

This implies $\frac{k}{n} = d \in \mathbb{Z}^+$, which in turn implies that $\omega_1 = e^{2i\pi d}$. Thus, when we put this term into the sum,

$$\begin{aligned} \frac{\omega}{n} \sum_{j=0}^{n-1} \omega_1^j &= \frac{\omega}{n} \sum_{j=0}^{n-1} \left(e^{2i\pi d}\right)^j \\ &= \frac{\omega}{n} \sum_{j=0}^{n-1} 1^j \\ &= \frac{\omega}{n} n = \omega = e^{\frac{i\pi k}{n}} = e^{i\pi d} = (-1)^d \end{aligned}$$

Recall, that we can integrate this function (ω_1) exactly, and the result is 0. Therefore, in this case, the Sine term is exact, but the Cosine term is not.

Case 1.b: $n \nmid k$

This implies,

$$\sum_{j=0}^{n-1} (e^{\frac{2i\pi k}{n}})^j = \frac{(e^{\frac{2i\pi k}{n}})^n - 1}{e^{\frac{2i\pi k}{n}} - 1} = 0$$

Therefore, we can conclude that both sine and cosine are exact in this case.

Now, we observe that for even k, one can rewrite the ω integral $\int_0^1 e^{\frac{i\pi k}{n}} dx$ as $\int_0^1 e^{\frac{2i\pi \frac{k}{2}}{n}} dx$ Through this result, we can conclude that $(S)_{k,n}^{M,T}$ for ω is exact for all even k, and $(C)_{k,n}^{M,T}$ is exact if and only if $n \nmid \frac{k}{2}$

Therefore, the Midpoint and Trapezoid rule give the exact answer for $\int_0^1 \cos(k\pi x)dx$ if and only if k is odd, or k is even and $n \nmid \frac{k}{2}$, and the exact answer for $\int_0^1 \sin(k\pi x)dx$ if and only if k is even.

□