Continuous Eigenpaths as a method of finding infinitely many solutions on the nonlinear Laplacian

Jeremy VanHorn-Morris – University of Oregon
and
Jazz Alibalic – Centenary College of Louisiana

July 28, 2000

1 Preface:

This paper is an exploration of the research done by Jazz Alibalic and Jeremy VanHorn-Morris together with John Neuberger Ph.D. at Northern Arizona University's Research Experience for Undergraduates program (by) the National Science Foundation. We examined the eighty-year old problem of the nonlinear laplacian,

$$\Delta u + f(u) = 0$$

for $u : \Omega \to \mathbf{R}$, where Ω is a smooth, bounded subset of $\mathbf{R}^N$, with $u = 0$ on $\partial\Omega$, and $f : \mathbf{R} \to \mathbf{R}$, and f is C^2, subcritical, superlinear, and $f'(0) = 0$.

Specifically we examined how to prove the existence of infinitely many solutions. We continued the work of 1999 NAU REU student Joel Fish, who started the idea of the eigen-flow.

This paper is designed specifically for undergraduates. Knowledge of linear algebra, differential equations, calculus, and some analysis is expected in the reading of this work. Explanations will be given when appropriate and (we hope to have) an extensive appendix available for further references.

2 Preliminaries:

First, we need to talk about the nonlinearity f. We mentioned that f is C^2 which means that f is twice continuously differentiable. We also mentioned that f is subcritical which means the following:

Definition: A function, f: R->R, f is subcritical given that there exists A>0 and p in (1, (N+2)/(N-2)), **[math-type type this]**

$|f(u)| <= A(|u|^p + 1)$ for all u in R.

We don't fully understand the exact details on this definition but we do need f to be subcritical because it will give us a simple first eigenvalue.

Also, f is superlinear, which is formally defined as follows:

$$\lim_{|u| \to \infty} f(u)/u = \infty.$$

This condition gives us the fact that f(u) is in L^2. (????????)

Laplacian (Δ) is a elliptic operator:

$$\Delta u = d^2 u/d x_1^2 + \ldots..+ d^2 u/d x_N^2$$

(for example: if we are in R2, then $\Delta u = u_{xx} + u_{yy}$).

Now we are going to introduce the Function Space: Much of what we have done is in the L^2 normed inner product space. L^2 is defined by the 2-norm, that is: u is in L^2 if integral of u over Ω is bounded **[equation, math-type]**, or another way, the square of the L^2 inner product of u w/itself is bounded. In general L^p is defined by the p-norm:

$$\|u\|_p^p < \infty.$$

The L^2 inner product is defined as:

$$<u,v>_2 = \text{integral}\Omega\ u*v\ dx.$$

The L^2 gradient is NOT defined everywhere. Moreover, the L2 gradient is impractical in many matters of calculation. More specifically we examined the Hilbert Space H = H sub 0, sup 1,2 (Ω). H is the completion of all C2 functions on the closure of Omega with the zero boundary condition. H is a dense subspace of L2(omega). More importantly, though, any and all solutions to our Laplacian lie in H, as well.

In explanation, if u is an element of H, then u has zero Dirichlet boundary condition on Ω, u is differentiable and has a bounded 2-norm.

We define the space H by the Sobolev inner product. For u, v in H the following defines Sobolev inner product:

<u,v>subH = IntegralΩ of (grad u dot grad v).

and the Sobolev norm:

<u,u>sub H = Sqrt[IntegralΩ of (|grad u|)^2].

The First and Second derivative operators Gradient and Hessian of H respectively are derived from the Sobolev inner product (?????????). That is for u in L2,

\nabla u = (partial of u with respect to x1,.,partial of u with respect to xN) in (L2(Ω))^N.

D2 u = N*N matrix of partials derivatives.

Moreover, the H gradient is defined everywhere so we can work in H. Also, both function spaces are infinite dimensional.

We need to talk about one more thing here. We say that u in X is a weak solution if for all v in X the following is true:

Integral Ω ({grad(u).grad(v) – f(u)*v}dx = 0.

Since we are working on a smooth boundary Ω, we can say that weak solutions are twice differentiable functions (or so-called classical solutions).

3 Action Functional:

The first thing that we need to do is to explain what a functional is. Simply put, a functional is almost the same as a function, except for its domain it has functions. In another words, a functional is a function of functions.

Here is a formal definition of the functional:
J is a functional if J is a map J : X -> R where X is a function space.

A very simple example of a functional is f(u) = u^3.
We are not going to specify the function space right now, since we worked on both the L^2 and Sobolev spaces. We will be more precise when we need to clarify whether we are in L^2 or H.

We define the action functional J : X -> R by

$$J(u) = \int_{\Omega} \{\tfrac{1}{2}|\nabla u|^2 - F(u)\}dx \,,$$

where $F(u) = \int_0^u f(s)ds$ is the primitive of *f*.

It is rather simple to show that J is twice continuously differentiable. Here is how it works. This is the derivation of the first derivative of J in the v direction:

J'(u)(v) = <grad J(u),v>

= lim t-> 0 1/t (J(u + tv) – J(u))
= lim t-> 0 1/t (integral Ω (.5 |grad(u + tv)|^2 – F(u + tv) - .5 |grad(u)|^2 + F(u) dx
= lim t-> 0 1/t (integral Ω (.5 {|grad(u) + t*grad(v)|^2 - |grad(u)|^2}dx) – 1/t (integral Ω (F(u + tv) – F(u) dx)
= lim t-> 0 1/t (integral Ω (.5 |grad(u)|^2 + t*grad(u).grad(v) + .5 t^2*|grad(v)|^2 - .5 |grad(u)|^2 dx) – integral Ω (f(u)*v dx)
= lim t-> 0 integral Ω (grad(u).grad(v) dx) + integral Ω (t*|grad(v)|^2 dx) – integral Ω (f(u)*v dx)

$$J'(u)(v) = \int_\Omega \{\nabla u \nabla v - vf(u)\}dx.$$

Similarly, $J''(u)(v,w) = \int_\Omega \{\nabla v \nabla w - vwf'(u)\}dx$ is the second derivative in the v, w directions.

It can be easily shown, by integrating by parts, that the grad J(u) = 0 if and only if u is a weak solution to our original problem. From that we can say that the critical points of J are the solutions to the nonlinear laplacian and vice versa. Here is the integration by parts:

$$\begin{aligned} J'(u)(v) &= \int_\Omega \{\nabla u \nabla v + vf(u)\}dx \\ &= \int_\Omega \{-\Delta u \cdot v - vf(u)\}dx \\ &= \int_\Omega -v\{\Delta u + f(u)\}dx. \end{aligned}$$

Given that v is not always 0, $\Delta u + f(u) = 0$, and hence u is a solution.

We are going to call the set S the zero set of J'(u) and define it as follows:

S = {u in H : u ≠ 0 and J'(u)(u) = 0}.

We also know (according to the [CCN] paper) that S is a co-dimension one manifold, it is diffeomorphic to an infinite-dimensional hypersphere.

4 History of the problem:

The problem of the nonlinear laplacian has been around for the past eighty years and many people tried to solve it. Not much progress had been made until the 1970's. It was shown that if $f(0) = 0$ and $f'(0) < \lambda_1$, then there exists a positive solution to the superlinear, subcritical problem. From that, it is just trivial to get the negative solution. In 1988, it was shown that if our Ω is a ball in R^N then there exist infinitely many radially symmetric solutions. We also know that if $N = 1$, then there are infinitely many solutions. In 1995, Dr. Neuberger proved in his [CCN] dissertation that there exists a sign changing solution exactly once on the superlinear, subcritical problem. It has been widely conjectured that there are indeed infinitely many solutions to this problem and the main goal of our research was to try to numerically show that this is indeed true.

5 Volcano:

First, we need to determine what does J functional look like and then we can look into finding the critical points (i.e. solutions to our problem). Since we are working in the infinite dimensional space, we could only come up with one or two-dimensional slices of J. J looks like an infinite dimensional volcano and it is possible to visualize by taking one-dimensional picture and sort of spinning it in infinitely many directions. Here are some of the properties of our volcano:

- it is centered at the origin
- it shoots off to the $-\infty$ in both positive and negative directions as x gets larger
- it is symmetric for f(u) = u^3
- it has a local minimum at the origin (where it is concave down)
- it has two global maximums.

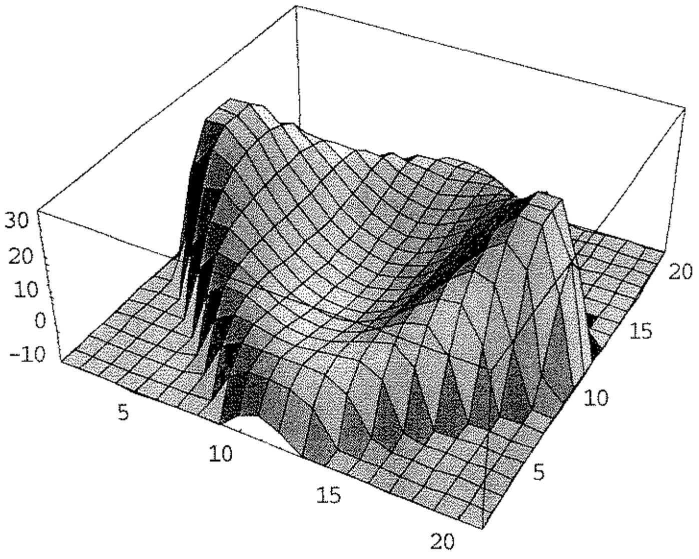
The volcano has a potato-chip rim, which we earlier defined to be in the set S. All of the solutions lie on this potato chip. In fact positive and negative solutions, ω_1 and ω_2 respectively, are the local minimums of J restricted to S (denoted J/S). This was found in the '70s by the use of the Mountain Pass Lemma [CCN]. There is "another" condition for those two solutions to exists and that is $f'(0) < \lambda_1$, where λ_1 is the smallest eigenvalue of the Laplacian on Omega [see eigenstuff of Laplacian later]. As his doctoral thesis, Dr. Neuberger proved the existence of a third nontrivial solution, ω_3, that changes sign exactly once. $\omega_3 \in S$ but we can do even better by defining a new set $S_1 = \{u \in S, u_+ \neq 0, u_- \neq 0, J'(u)(u_+) = 0\}$. Our third nontrivial solution $\omega_3 \in S_1$. We think, but are not certain, that S_1 is a manifold. If this were true, then we could get the fourth nontrivial solution, by another application of the Mountain Pass Lemma (which would be the second sign changing solution exactly once) to the eighty-year old problem. In Dr. Neuberger's proof of the ω_3 solution he used an number of partitions of S. We define them here in the hope that it will benefit future researchers.

$$\hat{S} = \{u \in S: u_+ \neq 0, u_- \neq 0\}, \quad G^+ = \{u \in S: u > 0\}, \quad G^- = \{u \in S: u < 0\},$$

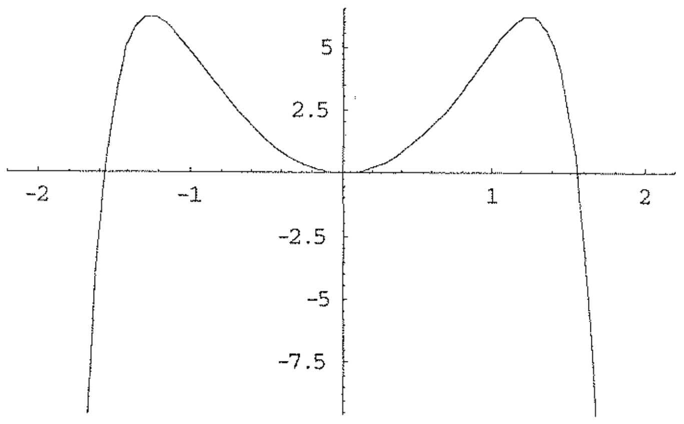
$$\hat{S}^+ = \{u \in S: J'(u)(u_+) < 0, \quad \hat{S}^- = \{u \in S: J'(u)(u_-) < 0, \quad W^+ = G^+ + \hat{S}^+, \quad W^- = G^- + \hat{S}^-$$

Now, S can be partitioned from $G^+ \cup \hat{S} \cup G^-$. Also $\hat{S} = \hat{S}^+ \cup S_1 \cup \hat{S}^-$. Both the positive and the negative solutions are in $G^+ \cup G^-$.

A cross section of J on the Ψ_1 Ψ_2 axes:



A cross section of J on the Ψ_1 axis:



Generic notes: using X as our generic function space. L2 and H Hessian dependent on the Basis chosen? That is, do we have the D2 Hessian functional operator in closed form, without regarding our basis? Need to Standardize vector notation, consistent capitalization of terms like Fourier, Gradient, Hessian, Laplacian…

6 Eigenfunctions of the Laplacian on Ω as an orthogonal basis for the function spaces L^2 and H:

Given that the Laplacian operator is Self-Adjoint, the eigenfunctions of the Laplacian on Ω form an orthogonal basis for the domain-space of the Laplacian. They form an orthogonal basis for both $L^2(\Omega)$ and $H(\Omega)$.

????Definition: Orthogonality ?????

(either define here or in an appendix)

????Definition: Normality ??????

For $\Omega = [0,1]$, the Laplacian operator is just the standard second derivative. That is, for this Ω,

$$\Delta u = u''$$

In order to keep the eigenvalues positive, we look at the eigenfunctions of the $-\Delta$. If we examine the equation $-u'' - \lambda u = 0$ we see that the logical choices for solutions are sines and cosines. It turns out that for the region $\Omega = [0,1]$, the eigenfunctions of the Laplacian are sines and cosines. Specifically they are of the form

$$e^{\alpha x}(c_1 \cos(\beta x) + c_2 \sin(\beta x)).$$

With our given boundary conditions, the eigenfunctions of the Laplacian on Ω are

$$\sin(k\pi x) \quad \text{for } k \in \mathbf{N}$$

with corresponding eigenvalues

$$k^2\pi^2$$

We can normalize this basis using the L^2 inner product to get an orthonormal basis for our function space:

$$\Psi_k = \sqrt{2}\sin(k\pi x)$$

With corresponding eigenvalues denoted as

$$\lambda_k = k^2\pi^2$$

Note that while the Ψ basis is orthonormal in L^2 it is only orthogonal in H. However, the eigenvalues, λ, give us an "isomorphism" between spaces. While Ψ_k is of length 1 in L^2, it is of length λ_k in H. This way any and all information generated in either space can be translated into the other.

The eigenfunctions of the Laplacian give you an orthogonal basis regardless of your choice of Ω. For $\Omega = [0,1]\times[0,1]$ the eigenfunctions of the Laplacian are again sines and cosines. Given our boundary conditions we, the proper functions are

$$\sin(j\pi x)\sin(k\pi y)$$

with eigenvalues

$$(j^2+k^2)\pi^2.$$

Normalized, our orthogonal basis for L^2 is

$$\Psi_{j,k} = 2\sin(j\pi x)\sin(k\pi y).$$

Using the Gauss-Sidel algorithm we can find eigenfunctions of the Laplacian on Ω for any Ω, regardless of shape or symmetry. For example the eigenfunctions of the Laplacian when Ω is a disk in $\mathbf{R}^2$ are the Bessel functions. Hence, regardless of Ω, we can find an orthonormal basis for our function space $L^2(\Omega)$.

Given this basis, we can treat our functions as Fourier series, with the coefficients of the Ψ functions being the coordinates of our function with respect to the Ψ basis. In doing so we can take a truncated approximation of our function, as with a Fourier approximation, and reduce our function space to n dimensions. This is what makes computational aspects possible. For most of our experimentation we used 10 dimensions, with the first 10 Ψ functions as our basis, $\{\Psi_1, \ldots, \Psi_{10}\}$.

Given our basis of the eigenfunctions of the Laplacian it is relatively simple to show how we use the eigenvalues of the Laplacian to switch between the function spaces L^2 and H. This is just a matter of comparing the lengths of the basis functions.

In L^2 the inner product of Ψ_j, Ψ_k is: $\langle \Psi_j, \Psi_k \rangle_2 = \int_\Omega \Psi_j \Psi_k \, dx = \delta_{jk}$

In H the inner product of Ψ_j, Ψ_k is:

$$\langle \Psi_j, \Psi_k \rangle_H = \int_\Omega \nabla\Psi_j \nabla\Psi_k \, dx = \int_\Omega -\Delta\Psi_j \Psi_k \, dx$$
$$= \int_\Omega \lambda_j \Psi_j \Psi_k \, dx = \lambda_j \delta_{jk}$$

Where $\delta_{jk} = 0$ if $j \neq k$ and 1 if $j = k$.

Similarly, were we to compute the coordinates of a function, they would be the same in both L^2 and H. However, they would have differing lengths. To compute the coordinates of a specific function, we just project the function onto the basis vectors. If want just the n dimensional approximation of our function, we project onto only the first n basis elements. In practice, we generate functions within our n dimensional space by choosing the coordinates of the point we want in our n dimensional vector space, and then generating our function. For example, supposing that $\{a_i\}$ is our coordinate vector, then our corresponding function, u is

$$u = \sum_{i=1}^{n} a_i \Psi_i .$$

One final note regarding the Laplacian. If we examine the matrix representation of the positive Laplacian operator with respect to the eigenfunction basis, the matrix is diagonal with entries corresponding to the negatives of the $-\Delta$ eigenvalues, λ.

7 The L^2 and H differential operators: Gradient and Hessian

$$\nabla_2 J(u) = -\Delta u - f(u)$$

$$\nabla_H J(u) = u + \Delta^{-1} f(u)$$

define the L^2 and H gradient. Note that

$$\nabla_2 J(u) = -\Delta\{\nabla_H J(u)\} .$$

(question: do we have an explicit representation of the D2 J(u) functional in either H or L2?????)

Similarly we can compare the second derivative, Hessian functionals, $D_2^2 J(u)$ and $D_H^2 J(u)$.

$$D_2^2 J(u) = -\Delta\{D_H^2 J(u)\}$$

Now, given that our basis is composed of the eigenfunctions of $-\Delta$, we can describe what the Sobolev gradient and Hessian of *J(u)* are compared to the L^2 gradient and Hessian, with respect to our Ψ basis. This becomes very important when looking at our truncated vector space. For example, let $\{g_i\}_2$, *i* = 1, …, *n*, be the L^2 gradient vector of *J(u)*. Then, the vector of the H gradient of *J(u)*, $\{g_i\}_H = \left\{\frac{g_i}{\lambda_i}\right\}_2$, *i* = 1, …, *n*.

Similarly,
if the matrix $\lfloor h_{i,j} \rfloor_2$, *i, j* = 1, …, *n*, is the L2 Hessian matrix in our truncated vector space, then the matrix of the Sobolev Hessian, $\lfloor h_{i,j} \rfloor_H = \left[\frac{h_{i,j}}{\lambda_j}\right]_2$ *i, j* = 1, …, *n*. (????????? Right notation?!?!!??!??!?). Thus we have a simple way of numerically, or sometimes, otherwise, calculating the Sobolev gradient and hessian. And, while it turns out that for Newton's method, both the Sobolev and L^2 operators generate the same algorithm [see Newton and Euler algorithms later in the paper], for Euler's method they do make a

"??(significant)??" difference {make sure there is a reference to this in Newton's method}.

8 Eigenstuff: The hessian matrix, Morse Index/signature, eigenpaths, flows, and finding the solutions…

Given that the D^2 Hessian operator is Self-Adjoint, all of its eigenvalues are real and the collection of eigenfunctions forms an orthonormal basis for the domain space of the Hessian. For computational purposes, this means that the matrix of the hessian of *J(u)* is symmetric, all eigenvalues are real, and the eigenvectors form an orthonormal basis. For our truncated vector space, the hessian is a $n\times n$ symmetric matrix. It turns out that at the origin, the matrix of the L^2 Hessian of *J(u)* is diagonal with respect to the Ψ basis, with diagonal entries corresponding to the eigenvalues of the minus Laplacian. Actually, at the origin, the matrices of the minus laplacian and the L^2 Hessian are the same. As an explanation let's examine second derivative of *J* at the origin, $J''(0)$.

$$J''(0)(\Psi_j,\Psi_k)=\int_\Omega\{\nabla\Psi_j\nabla\Psi_k-\Psi_j\Psi_k f'(0)\}dx$$

Since one of the conditions on *f* is that $f'(0)=0$, we can simplify this to.

$$J''(0)(\Psi_j,\Psi_k)=\int_\Omega\nabla\Psi_j\nabla\Psi_k dx=\int_\Omega-\Delta\Psi_j\Psi_k dx=\lambda_j\delta_{jx}$$

One can now see that, since this represents the *j,k* entry in the L^2 Hessian matrix, the hessian matrix is diagonal and has entries corresponding to the eigenvalues of the minus Laplacian. Thus, the minus Laplacian is identical to the L^2 Hessian of *J* at the origin, as both matrices and operators.

9 Newton's Method for finding solutions to the Laplacian:

Newton's Method is a basic concept from Calculus. The purpose is to find the zero of some continuous, differentiable function f with an initial guess x_0. If our initial guess is close enough to the actual zero of the function, then the Newton's method should be able to converge to that zero.

The following algorithm finds zeroes using the Newton's method:

```
- make an initial guess x_0
- Loop until f(x[[i]]) < \epsilon
- Loop
- x[[i+1]] = x[[i]] – f(x[[i]]) / f '(x[[i]])
```

- End Loop

Newton's method works in most of the cases, even when our initial guess is not right next to the actual root of the function. Newton's method only fails to converge if we are really close to the horizontal asymptote or a local minimum. Even though Newton's seems to be a rather simple algorithm, it is a great tool to find zeroes of a function in the higher dimensions. It will find the roots of the function f: R^N -> R^N.

Since there are two different Newton's methods, first we need to figure out which one of we need to use to generate our algorithm. The first method is called the continuous Newton's method. What it does is it calculates x'(t) by the following algorithm:

$$x'(t) = -f(x(t)) / f'(x(t)), \text{ where } x(0) = x_0.$$

The second method is called the discrete Newton's method. It does not calculate the derivative. Instead, it calculates the x_{k+1} using this algorithm:

$$x_{k+1} = x_k - \text{delta}*f(x_k) / f'(x_k), \text{ where } 0 < \text{delta} < 1.$$

We decided to use the discrete Newton's method and from now on we will call it simply the Newton's method.

We used the following algorithm for the ODE and the PDE case:

```
- given a function f, calculate the L2 gradient and the hessian
- make an initial guess (double indexed in PDE case)
- Loop While[(norm>tol) && (num<maxits)]
- Then a = a – delta * PseudoInverse[hessian].grad
    o Note: we use the Pseudoinverse in the case that the hessian of J has one or
      more zero eigenvalues. Pseudoinverse diagonalizies the hessian and it
      calculates the inverse of a nondegenerate (invertible) matrix.
```

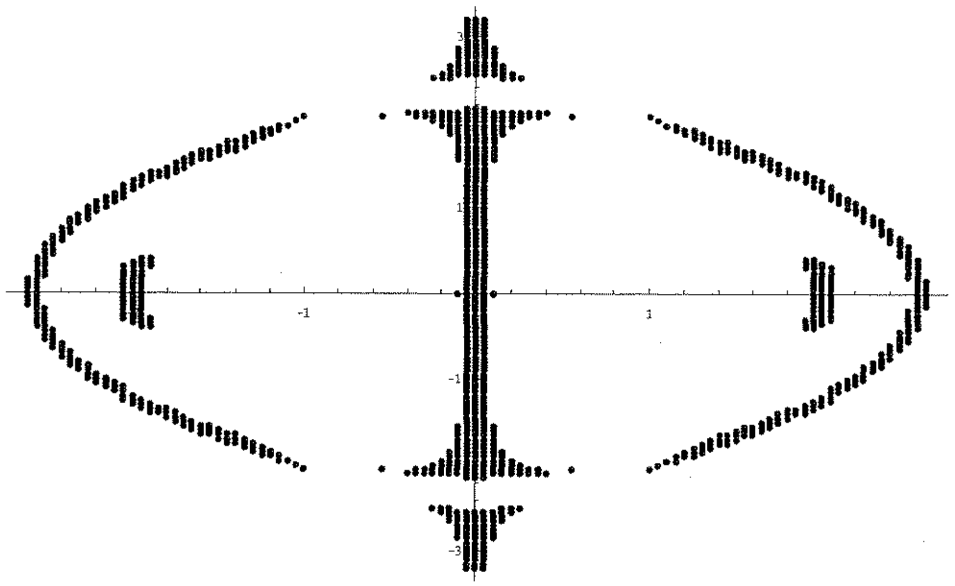
According to John Neuberger's [CCN] paper, we know that the first and the second derivatives of our functional J indeed exist. Furthermore, if we use \nabla J to be the first derivative operator and the hessian to be the second derivative operator, then we can find the critical points of J (i.e. points where \nabla J = 0) which are the solutions to our ODE and PDE. Basically, we are looking for zeroes of the \nabla J and we are using the Newton's method to find the roots which will be the solutions to our ODE / PDE.

10 Gamma Sets, $\Gamma_S, \Gamma_1, \Gamma_2, \ldots, \Gamma_i, \ldots$:

Definition: Loosely, we define Γ_i to be the collection of points, u, where $\nabla J(u)$ is orthogonal to the ith eigenfunction of the hessian, $D^2 J(u)$.

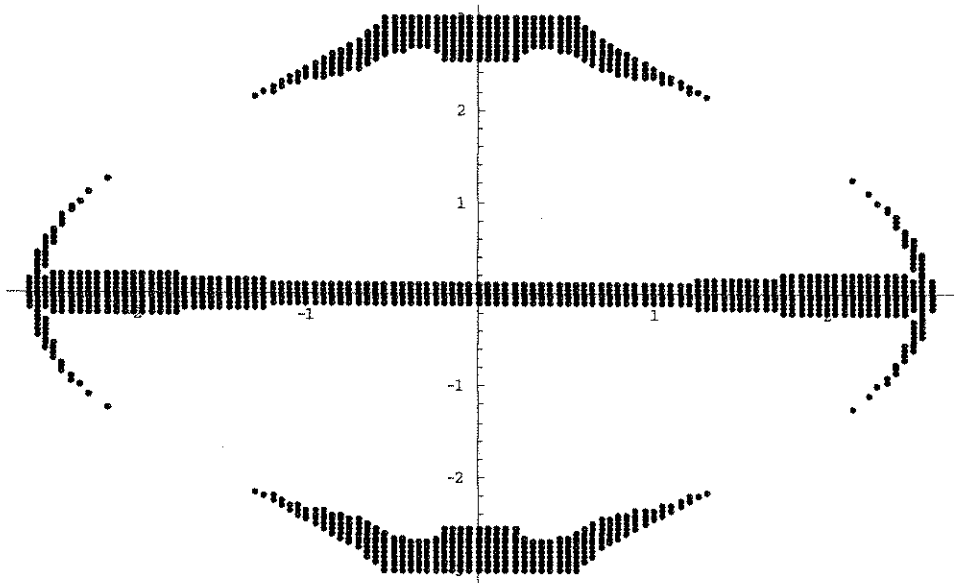
Ideally, we would like to include in Γ_i only the points that form codimension 1 manifold, diffeomorphic to a hyperplane. There will also be extra points that lie upon what we hope to be a manifold diffeomorphic to a hypersphere. This hypersphere is a collection of subsets of all the other Γ_i's. This is what we want to be Γ_S. Ideally, the *i*th solution is the intersection of all the Γ_j's save the *i*th one.

Γ_1 cross section as defined loosely above.



Γ_1 cross section on the Ψ_1, Ψ_2 plane using the Sobolev operators.

Γ_2 cross section as defined loosely above.



Γ_2 cross section on the Ψ_1, Ψ_2 plane using the Sobolev operators.

11 Eigenflows:

Morse Index or signature:

Definition: Morse Index is the number of negative eigenvalues of the Hessian, or in other words, it is the number of "down" directions at the saddle point.

We need to mention that if nondegenerate, the positive and negative solutions have Morse Index 1 and the sign changing solution exactly once has Morse Index 2.

Inflection Set:

Definition: The inflection set I is the set where the signature changes.
Last year's REU student, Joel Fish, defined it to be

$$I = \{p|\lambda_i = 0;\ i \in \{1,2,\ldots,N\}\}$$

where

$$\lambda = \begin{pmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & 0 & \vdots \\ \vdots & 0 & . & 0 \\ 0 & \ldots & 0 & \lambda_N \end{pmatrix}$$

Now we need to get back to Newton's method and try to determine whether a path generated by the Newton's method would go across the inflection set. If we consider the basins of attraction graph, we see that if we are following the path generated by the continuous Newton's method, we should converge to the solution with the same signature as the original guess. Unless we are at the point where continuous Newton's method is not defined, we should not change signature following this path.

The eigenflows:

We define the eigenflows to be the paths following a given eigenvector. Two things are required to define a given eigenflow. An initial point, w_0, and a function, $P(t)$, representing a magnitude, parameterized with respect to t, loosely referred to as time. The basic idea is to use an eigenfunction of the hessian of $J(w)$, or eigenvector in our truncated vector space, as the direction of the gradient of our path at w. As such, the parameterization of our path, $F_i(w)$, must solve the differential equations:

$$F_i(0) = w_0$$
$$\nabla_X F_i(t) = P(t) E_i(w)_X$$

and,

$$w = F_i(t)$$

where $E_i(w)_X$ refers to the *i*th eigenfunction, or eigenvector, of the X hessian of *J(w)*.

This definition requires only that the eigenfunctions, or eigenvectors, change continuously, ((see Dr. Neuberger about finding something that shows this)), which should be applicable given the symmetry in the hessian matrix, and that *J* is C^2.

More important, though, is a specific type of eigenflow which I am calling the eigenpath. The eigenpaths have a given initial condition that $F_i(0)=0$, where the output 0 refers to the zero function, or vector if you are in the truncated vector space. There is a very specific *P* that will define a path such that the X gradient of *J(w)* is collinear to the *i*th eigenfunction, or eigenvector, of the X hessian of *J(w)*. Along this path there is a unique, continuous collection of *w*'s such that the gradient of *J(w)* lies solely in the *i*th eigenspace of the hessian. Given that this path is continuous, the gradient is orthogonal to all but dimension of our function space. Numerically we have evidence that this path will always lead to a critical point of *J* and a solution to the nonlinear Laplacian.

AAAAAAAAAAAAAAAAAAAAAAAHHHHHHHHHHHHHHHHH!!!!!!!

Maybe anone of this is necessary, maybe the magnitude is just the magnitude of the gradient….Maybe all we're doing is following the right gradient flow!!!!!!!!!!!!!!!!!!!!!!

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAHHHHHHHHHHHHHHH!!!!!

Talk to Neuberger about all this….

Eigenpaths correspond to collections of points where the X gradient is collinear to an eigenfunction of the X hessian of *J*. We can use Euler's method to estimate these paths.

12 Euler's shooting method:

Euler's method is yet another simple concept from calculus. It is one of the simplest ways to solve differential equations. The basic idea is that we take our original equation, calculate the derivative of the function f(y , t) using the forward difference. Here is the basic rundown:

We start with $dy/dt = f(y, t)$.
Then $(y(t_{n+1}) - y(t_n)) / h = f(y, t)$ where h is the step size.

Finally, $y(t_{n+1}) = y(t_n) + h^* f(y_n, t_n)$.

Since we want to follow the Eigenflows to get to our solutions, we had to come up with an algorithm that would do that for us. Well, Euler's method did just that for us. The following algorithm was used for following the Eigenflows:

- given a function f (we used $f(u) = u^3$ to be our function), calculate the gradient and the hessian matrix (L_2 vs. H)
- make an initial guess (double indexed in the PDE case)
- loop While [(norm > tolerance) && (number < iterations)]
- Calculate the E_k as the eigenvector of the k-th sorted eigenvalue
- Project the gradient on E_k
- Take a delta step in the projected gradient direction
- End loop.

First, let us explain our loop conditions. The norm of the gradient is just the square root of the grad dot grad. So if our norm ever gets smaller then the tolerance (we use 10^(-5) to be our tolerance), then that means that our gradient is basically 0 and that we are at the solution. The second condition on the loop simply stops it if we use more than x number of iterations. It should not take more than 200 iterations with a small delta step size to converge to the solution and we did not want the loop to run forever if we ever reach a point where the gradient is orthogonal to the flow. We were not quite sure in the beginning whether we should use the Sobolev or the L_2 gradient and the hessian. We did some trial and error and it seemed that if we are in L_2, then we could only follow the E_1 flow to the solution. When we tried to follow the E_2 flow, we would stay on the ridge for a while, but then we would start falling off the ridge as we would get closer to the solution since the Ψ_1 coordinate started to get away from 0 and we went somewhere not close to any solutions. Then we decided to try to use the Sobolev gradient while everything else would still be in L_2. This seemed to be working better, but we would not converge to the solution. What happened was that we would come to the point where the gradient is orthogonal to the E_i flow and we would just stay at that point. Those points were rather close to the actual solution (which we checked using the Newton's method). Then we decided to work in the Sobolev space which seemed to be working the best so far. If we used the H gradient and hessian, and we tried to follow the E_i flow, we would converge to a solution for the first two Eigenflows, and it would overshoot just a bit for the other flows and then stay at the point where the gradient is orthogonal to the flow. Basically, this makes us believe that we indeed do have the right algorithm for following the flows that will take us to two solutions for each flow if we have simple eigenvalues.

13 Projection Algorithm:

However, it would be nice to know just how close to the eigenpaths we are. Better yet, we would like to be able to project ourselves onto the eigenpaths. That way we can take large steps in Euler's method and still be guaranteed that we can follow an eigenpath. Our first attempt to do this was a failure.

Since while on an eigenpath the gradient is collinear to a single eigenfunction of the hessian, it is also necessarily orthogonal to the rest. The idea was to define the function(al)

$$\{\hat{g}(u)\}_j = \begin{cases} 0 & \text{for } j = i \\ \nabla J(u).E_j(u) & \text{for } j \neq i \end{cases}$$

where we are following the ith eigenpath, then use Newton's method to find a zero of the function. The idea itself is valid. If we zero our function $\hat{g}$, then, ideally, the gradient should be orthogonal to each of the eigenvectors, save the ith. Given this, we would then be on the ith eigenpath. Unfortunately, Newton's method requires the use of the derivative of the given function to find zeros. While we can use the Chain Rule and find half the derivative, the second half, $\nabla J(u).E_j'(u)$ does not have a closed form representation. Thus we attempted to numerically find the derivative of $E_j(u)$ by calculating the change in the jth eigenvector as we changed u in each of the Ψ directions, thus giving us an effective numerical gradient. While the idea still seems sound, the implementation was apparently flawed and our algorithm did not work. [toss in a the algorithm here?*******] However, a short time later, Dr. Neuberger had another brilliant flash of inspiration. What we actually are trying to do is find a place where the gradient is zero in all the eigenfunction directions, save that corresponding to the eigenpath. Given that the hessian is Self-Adjoint, the eigenfunctions of the hessian form an orthonormal basis for the domain space, X. So, all we really need to do, is compute the gradient in basis of the eigenfunctions. Since the derivative of the gradient is the hessian, which is the diagonal matrix of eigenvalues in the basis of eigenvectors, it is not only well defined and in closed form, it is also incredibly easy to deal with computationally. It turns out to be very easy to compute the Newton search direction with respect to the basis of eigenfunctions, and similarly simple to then convert to the coordinates of the Ψ basis. The algorithm turns out to works very well in all but a few specific areas. Here we present an outline of the eigenpath projection algorithm. As with all algorithms we have used, all computations have been done in the n dim truncated vector space. Unless otherwise stated, all inner products and norm operations are the standard Euclidean operations. In explanation, a will denote the vector of coefficients for our function u.

-Projecting onto the Eigenpaths:

terms:
norm: length of the Newton search direction
grad: Gradient of J at a
Egrad: Gradient of J at a, in terms of the eigenvector basis
hess: Hessian matrix of J at a

evects: Matrix of eigenvectors of *hess* (in Mathematica sorted by eigenvalue, in rows)
evals: List/vector eigenvectors of *hess* (in Mathematica sorted my magnitude)
conv: Convergence minimum to determine when we have finished the projection
δ: Step size for Newton's Method.

-choose an initial point, and a guess of *i*th eigenvector, e_i

```
Loop
    While norm>conv,

        -Compute grad, hess, evects, evals

        -Determine which eigenvector is the appropriate one
            compute pos = Max[evects.e_i], this gives the position of the
appropriate eigenvector in the list evects.
        -Set e_i to be the ith eigenvector
            e_i = evects[[pos]]
```

-Compute the Newton Search direction $[hess^{-1}.grad]_e$. Since the hessian is just the diagonal eigenvalue matrix, and the *i*th coordinate of the gradient in the eigenbasis is just *grad.evects*[[*i*]], our Newton Search vector is

$$search_e = [grad.evects[[i]] * \eta_i]_i$$

in the eigenbasis, where η_i is the Pseudoinverse eigenvalue of the hessian, or defined as follows.

$$\eta_i = \begin{cases} \frac{1}{\beta_i} & \text{if } \beta_i \neq 0 \\ 0 & \text{if } \beta_i = 0 \end{cases}$$, where β_i is the *i*th eigenvalue of the hessian.

-Zero out the eigenpath direction: Since we only want to zero the gradient in all but one direction, we need to prevent Newton's Method from zeroing the gradient with respect to the path eigenvector. To do this, just set $search_e$[[pos]] = 0.

-Switch to Ψ coordinates: To convert to the standard Ψ coordinates we just take a sum. The Newton search direction in Ψ coordinates is

$$search_\Psi = \sum_{k=1}^{n} search_e[[k]] * \Psi_k .$$

-Compute:

$$norm = (search_\Psi \cdot search_\Psi)^{1/2}$$

-Move in the search direction:

$$a = a - \delta * search_\Psi$$

End Loop

This projection algorithm works very well in all but a few areas. We have experienced problems after crossing inflection set, except on the e1 flow. I believe that the reason for this, is that the zero eigenvalue corresponds to the eigenvector of the eigenpath. That is, there are no zero eigenvalues in the Newton Search directions. However, by decreasing our step size, δ, we can minimize the area where the projection algorithm returns bad results. For example, the following gives the regions along the Ψ_2 axis that the projection algorithm returns bad results for different step sizes. Here we use $f(u)=u^3$ and $\Omega = [0,1]$.

Step Size, δ	**Bad region along the Ψ_2 axis**
0.2	1.64 to 2.55
0.05	1.74 to ~ 2.2
0.01	1.77 to 1.85

Since our f is symmetric, there are mirror regions in the negative Ψ_2 direction as well. Along the Ψ_2 axis there is a signature change at 1.78. There is similar evidence along the Ψ_1 axis for the signature 2,3 change, and along the Ψ_3 axis at the signature 0,1 and 1,2 changes. I believe that at the signature change the projection algorithm takes you to Γ_S instead of the eigenflow. However, it seems that the projection of the Ψ_2 axis onto the e2 flow gradually returns to what should be e2 flow as we progress farther into the bad region. I would assume that were we projecting onto Γ_S for the entirety of the bad region we would make a noticeable jump back off the Γ_S set and onto the e2 flow. This does not seem to be the case. For the e2 flow, the projection along the bad region begins with a Ψ_1 coordinate close to 2.5 and then, depending on the step size in our algorithm, slowly goes to 0. Along the e2 flow the Ψ_1 coordinate should be 0 throughout. Since Γ_S doesn't intersect the e2 flow until a solution, about 5.1 in the Ψ_2 coordinate, the projection algorithm cannot be following Γ_S completely either. Also, the bad region only begins at an inflection point, after which all eigenvalues are nonzero. It may just be a numerical inconsistency in the algorithm. Finding what the problem with the algorithm is, though, is essential in determining whether our enthusiasm is justified. On a brighter note, though, there does seem to be evidence that one can follow an eigenflow, each eigenflow, to a solution.

When looking at the PDE case, $\Omega = [0,1]\times[0,1]$, we instantly run into the problem of eigenvalues with multiplicity greater than one. Where we have a non-simple eigenvalue we also have an eigenspace of dimension greater than 1. In this case there is no single direction to follow in order to find a solution. What we have, instead, is, for example, a plane where the gradient lies within the eigenspace of a single, non-simple, eigenvalue. Whether the eigenvalue retains its multiplicity as we follow the "eigenplane" from the origin or it separates into two or more simple eigenvalues we still don't know. This too is a good area for further research. Note that the "eigenplane" should actually be a 2-dimension manifold, not necessarily a plane. However, supposing that it remains planar, then it will eventually intersect the Γ_S set to form a continuous path; circular if Γ_S is continuous. On this path the local minima and maxima will also be critical points, and hence solutions.

14 Conclusion:

The methods given here are numerical methods that provide insight into ways of proving the existence of infinitely many solutions for the nonlinear laplacian. In truth, they may possibly be used to find critical points of any functional. While we have only a small amount of tested data, it seems very much that this method is an appropriate and effective tool, and that with further study analytic results could prove the existence of eigenpaths from the origin which lead to a solution. From the results we have now, we can speculate that for each simple eigenvalue, (at the origin), there are two continuous eigenpaths from the origin to solutions, in the positive and negative eigenfunction directions. Since we believe the existence of infinitely many simple eigenvalues, we have an outline of an existence proof for infinitely many solutions to the nonlinear laplacian. This has been our goal for the research program this summer. We are very pleased with the work we were able to accomplish. Although much more still needs to be done, we hope our work will help other research in this area.

15 Further Research:

-Rigorous definitions of the different Γ sets, specifically that allows for Γ_S; Eigenflows; Eigenpaths

-Projection algorithm for the PDE, likely in a low level language like C or Fortran

-Following a non-simple eigenpath

-Eigenvalues crossings?

-Remove ambiguity in sorting the Eigenvectors

Additions to numerical experimentation:

-Calculating the angle between the gradient and an eigenvector, in various notebooks.

-Use projection algorithm to draw pictures of the Γ sets

15 16 References:

[CCN] J. M. Neuberger. *Existence of a Sign-Changing Solution to a Superlinear Dirichlet Problem.* Doctoral Thesis Dissertation, University of North Texas (1995).

[NEWTON] J. M. Neuberger and J. W. Swift. *Newton's Method and MorseIndex for Semilinear Elliptic PDEs.* Northern Arizona University (2000).

[NUM] J. M. Neuberger. *A NumericalMethod for Finding Sign-Changing Solutions of Superlinear Dirichlet Problems,* Nonlinear World 4, no. 1 (1997), p73-83.

[JF] J. Fish. *The Peanut, Potato Chip, and the Infinite Dimensional Volcano.* Northern Arizona University REU(1999).