A Numerical Investigation of a Nonlinear Elliptic System

by Dennis R. Rice, Jr.

A Thesis
Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in Mathematics

Northern Arizona University
May, 2002

Abstract

A Numerical Investigation of a Nonlinear Elliptic System

Dennis R. Rice, Jr.

We numerically find solutions to the vector Ginzburg-Landau equation with a triple-well potential (as studied by Flores, Padilla, and Tonegawa). We use the Galerkin Newton Gradient Algorithm (by Neuberger and Swift) and bifurcation techniques to find solutions to this problem. With a small parameter, we find a Morse index 2 solution which approximates a pattern formation with triple junction structure whose nodal set is of minimal length and intersects the boundary at right angles.

Acknowledgements

I would like to thank Prof. John M. Neuberger for his encouragement and countless hours of explanation he gave while this work was in progress. Dr. Neuberger's confidence in me was unwavering and often motivated me to work harder to try to live up to his expectations.

I would also like to thank Prof. Pablo Padilla of UNAM for partially funding a trip to Mexico so that we could gain valuable insight into this problem.

This work was partially supported by NSF grant DMS-0074326.

Contents

List of Figures

Chapter 1

Introduction

In this work, we use the Galerkin Newton Gradient Algorithm [7] to numerically study the following system studied by Flores, Padilla & Tonegawa [5]:

$$\begin{aligned} -\epsilon^2 \Delta u + \nabla W(u) = 0 \quad \text{in } \Omega \\ \frac{\partial u}{\partial \eta} = 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{1.1}$$

where $\Omega \subset \mathbf{R}^2$ is a bounded regular (see Definition A.0.1) domain, $u : \Omega \to \mathbf{R}^2$, $W : \mathbf{R}^2 \to [0, \infty)$ is a triple well potential, η is the unit outer normal to the boundary, and ϵ is a small parameter. We will work with sufficiently smooth functions $u \in L^2 \times L^2$, where $L^2 = L^2(\Omega)$. More precisely, we assume that the following conditions from [5] are satisfied:

(i) $W(u) \geq 0$ for all $u \in \mathbf{R}^2$.

(ii) $P_1 = a$, $P_2 = b$ and $P_3 = c \in \mathbf{R}^2$ are three different points such that $W(a) = W(b) = W(c) = 0$, i.e., global minima of W. We will also suppose that these points are nondegenerate, that is, $[D^2W(x)]^{-1}$ exists for $x = a, b$, or c. For simplicity we will take these as the only minima of the potential.

(iii) W is uniformly coercive in u, that is, there exist R_0 and $C > 0$ such that if $|u| \geq R_0$ then $W(u) \geq C|u|^2$. We assume that R_0 was chosen sufficiently large so that the negative gradient flow of W points towards the ball $B_{R_0}(0)$ at points outside the ball.

In our investigation we take $\Omega = (0,1) \times (0,1)$ and $W = \alpha - (u_1^2 + u_2^2) + \zeta(u_1^2 + u_2^2)^2 - \gamma(u_1^3 - 3u_1u_2^2)$, where α, ζ, and $\gamma \in \mathbf{R}^+$ are chosen so that conditions (i) and (ii) are satisfied (see Figure 1.1). Clearly (iii) is satisfied given the dominant fourth order term. Flores, Padilla & Tonegawa use the existence of three Morse index (MI) 1 mountain pass solutions to prove the existence of a nontrivial (nonconstant) MI 2 solution, when ϵ is small.

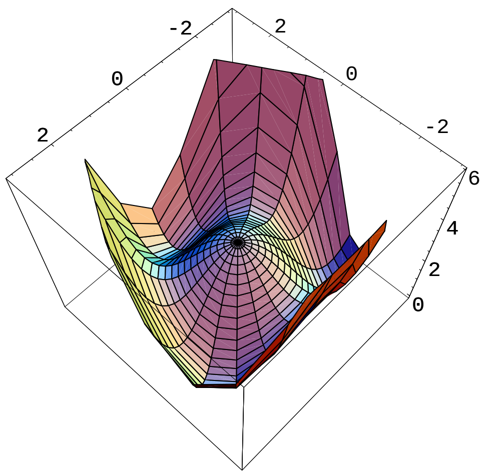
Note in (1.1) that $u : \Omega \subset \mathbf{R}^2 \to \mathbf{R}^2$ can be written as a vector

$$u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}.$$

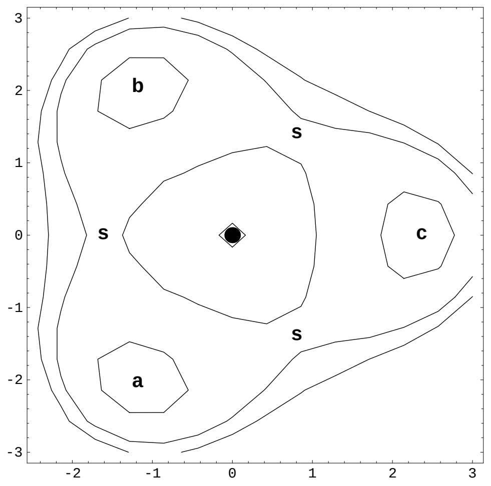
Then (1.1) can be written as the following *system* of equations:

$$\begin{aligned} -\epsilon^2 \Delta u_1 + \frac{\partial W(u)}{\partial u_1} &= 0 \quad \text{in } \Omega \\ \frac{\partial u_1}{\partial \eta} &= 0 \quad \text{on } \partial\Omega \\ -\epsilon^2 \Delta u_2 + \frac{\partial W(u)}{\partial u_2} &= 0 \quad \text{in } \Omega \\ \frac{\partial u_2}{\partial \eta} &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

We use the Gradient Newton Galerkin Algorithm (GNGA, see [7]) in our investigation. Neuberger & Swift apply the algorithm to a single nonlinear elliptic PDE with zero Dirichlet boundary conditions. Here, we apply the algorithm to a system with zero Neumann boundary conditions. In our work we rely on bifurcation theory in order to produce the nontrivial solutions proven to exist by Flores, Padilla & Tonegawa. The proof by Flores, Padilla & Tonegawa and our numerical investigation use variational methods to study (1.1).



(a)



(b)

Figure 1.1: The graph of W with $\alpha \approx 3.23$ and $\zeta \approx \gamma \approx 0.13$. Part (a) is the graph itself and (b) is its contour map. The points **a**, **b**, and **c** are the minima of W, points labeled **s** are saddle points, and the center is a local maximum.

Chapter 2

Implementing GNGA

2.1 Eigenfunctions of the Laplacian

Let Δ be the Laplacian operator. The eigenvalues and eigenfunctions of -Δ on the square with zero Neumann boundary conditions are solutions to the PDE

$$\begin{cases} u_{xx} + u_{yy} + \lambda u = 0 & \text{on } \Omega = (0,1) \times (0,1) \\ \dfrac{\partial u}{\partial \eta} = 0 & \text{on } \partial\Omega. \end{cases}$$

If we let $u = f(x)g(y)$, this problem can be easily solved using separation of variables. The separate equations of f and g become the second-order ODE problems

$$\begin{array}{ll} f'' + \mu^2 f = 0 & g'' + \nu^2 g = 0 \\ f'(0) = f'(1) = 0 & g'(0) = g'(1) = 0. \end{array}$$

The solutions to these problems can be found using characteristic equations and are given by

$$\begin{array}{ll} \mu_m = m\pi & \nu_n = n\pi \\ f_m(x) = \cos m\pi x & g_n(y) = \cos n\pi y \end{array}$$

where $m, n = 0, 1, 2, \dots$.

Thus the eigenfunctions of the $-\Delta$ are

$$u_{m,n}(x,y) = \cos(m\pi x)\cos(n\pi y)$$

and the corresponding eigenvalues are

$$\lambda_{m,n} = \mu^2 + \nu^2 = (m^2 + n^2)\pi^2,$$

where $m, n = 0, 1, 2, \ldots$. [6]

2.2 A Basis for $G \subset L^2 \times L^2$

In this work, we will use $\lambda_{m,n} = (m^2 + n^2)\pi^2$ for the (doubly-indexed) eigenvalues of -Δ, and

$$\psi_{m,n}(x,y) = \begin{cases} 1 & \text{for } m = n = 0 \\ \sqrt{2}\cos(m\pi x)\cos(n\pi y) & \text{for } m = 0, n \neq 0 \text{ or } n = 0, m \neq 0 \\ 2\cos(m\pi x)\cos(n\pi y) & \text{for } m \neq 0 \text{ and } n \neq 0, \end{cases}$$

for the eigenfunctions of -Δ, normalized in $L^2 = L^2(\Omega)$. As above, m and n range over all of the nonnegative integers. Since $\{\psi_{m,n}\}$ are eigenfunctions of a self-adjoint (see Definition A.0.7) linear operator, they form a basis for L^2 and $H = H^{1,2}(\Omega)$. (See Theorems A.0.8 and A.0.9 for a sketch of this proof, including Green's First Identity.) We take a finite number $\widetilde{M} \in \mathbf{N}$, and order $\widetilde{M} + 1$ basis elements according to their corresponding (singly indexed) eigenvalues $(0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_{\widetilde{M}})$. This gives us a singly indexed basis, $\{\psi_i\}$. We use the following Galerkin subspace of $H \times H$ and $L^2 \times L^2$:

$$G = \text{Span}\left\{\begin{pmatrix} \psi_i \\ 0 \end{pmatrix}\right\}_{i=0}^{\widetilde{M}} \bigcup \left\{\begin{pmatrix} 0 \\ \psi_i \end{pmatrix}\right\}_{i=0}^{\widetilde{M}},$$

which is of dimension $M = 2(\widetilde{M} + 1)$. Thus, a basis for this set is

$$\left\{\begin{pmatrix} \psi_0 \\ 0 \end{pmatrix}, \begin{pmatrix} \psi_1 \\ 0 \end{pmatrix}, \ldots, \begin{pmatrix} \psi_{\widetilde{M}} \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \psi_0 \end{pmatrix}, \begin{pmatrix} 0 \\ \psi_1 \end{pmatrix}, \ldots, \begin{pmatrix} 0 \\ \psi_{\widetilde{M}} \end{pmatrix}\right\}.$$

For convenience, we use $\{\Psi_i\}_{i=1}^{M}$ to denote this basis. Thus, for $u \in G$, there exist Fourier coefficients $\{a_i\}_{i=1}^{M} \in \mathbf{R}^M$ such that $u = \sum_{i=1}^{M} a_i \Psi_i$.

2.3 The Functional

As previously stated, GNGA uses a variational method to solve (1.1). We define the (energy) functional $J_\epsilon : H \times H \to \mathbf{R}$ as

$$J_\epsilon(u) = \int_\Omega \left(\frac{\epsilon}{2}|\nabla u|^2 + \frac{1}{\epsilon}W(u)\right)dx.$$

For convenience we write J for J_ϵ. It is well known that critical points of J are exactly the classical solutions to (1.1) (see [1], [5], [7], and [8]). To apply GNGA, we need to compute first and second directional derivatives of J. The first directional derivative of J is defined as

$$\begin{aligned}
J'(u)(\Psi) &= \lim_{t\to 0} \frac{J(u+t\Psi)-J(u)}{t} \\
&= \lim_{t\to 0} \int_\Omega \frac{\frac{\epsilon}{2}|\nabla(u+t\Psi)|^2 + \frac{1}{\epsilon}W(u+t\Psi) - \frac{\epsilon}{2}|\nabla u|^2 - \frac{1}{\epsilon}W(u)}{t}\,dx \\
&= \lim_{t\to 0} \int_\Omega \frac{\frac{\epsilon}{2}(|\nabla u|^2 + 2t\nabla u\cdot\nabla\Psi + t^2|\nabla\Psi|^2)}{t}\,dx \\
&\quad + \int_\Omega \frac{\frac{1}{\epsilon}W(u+t\Psi) - \frac{\epsilon}{2}|\nabla u|^2 - \frac{1}{\epsilon}W(u)}{t}\,dx \\
&= \lim_{t\to 0} \int_\Omega \frac{\epsilon t\nabla u\cdot\nabla\Psi + \frac{\epsilon}{2}t^2|\nabla\Psi|^2 + \frac{1}{\epsilon}[W(u+t\Psi)-W(u)]}{t}\,dx \\
&= \int_\Omega \epsilon\nabla u\cdot\nabla\Psi\,dx + \lim_{t\to 0}\int_\Omega \left[\frac{\epsilon}{2}t|\nabla\Psi|^2 + \frac{1}{\epsilon}\frac{W(u+t\Psi)-W(u)}{t}\right]dx.
\end{aligned}$$

Then by the Lebesgue Dominated Convergence Theorem (Theorem A.0.2), we move the limit inside the integral to get

$$\begin{aligned}
J'(u)(\Psi_j) &= \int_\Omega \left[\epsilon\nabla u\cdot\nabla\Psi_j + \frac{1}{\epsilon}\Psi_j\cdot\nabla W(u)\right]dx \\
&= \int_\Omega \left[\epsilon\nabla\left(\sum_{i=1}^M a_i\Psi_i\right)\cdot\nabla\Psi_j + \frac{1}{\epsilon}\Psi_j\cdot\nabla W(u)\right]dx \\
&= \epsilon a_j\lambda_j + \frac{1}{\epsilon}\int_\Omega \Psi_j\cdot\nabla W(u)\,dx.
\end{aligned}$$

Similarly, the second directional derivative is computed to be

$$\begin{aligned}
J''(u)(\Psi_j,\Psi_k) &= \int_\Omega (\epsilon\nabla\Psi_j\cdot\nabla\Psi_k + \frac{1}{\epsilon}D^2W(u)\Psi_j\cdot\Psi_k)\,dx \\
&= \epsilon\lambda_j\delta_{jk} + \frac{1}{\epsilon}\int_\Omega D^2W(u)\Psi_j\cdot\Psi_k\,dx,
\end{aligned}$$

where

$$\delta_{jk} = \begin{cases} 0 & \text{for } j \neq k \\ 1 & \text{for } j = k \end{cases}$$

is the Kronecker delta function. The algorithm, GNGA, consists of performing Newton's method iterations on the Fourier coefficients to find zeros of ∇J.

The inner products on L^2 and H are defined as

$$\begin{aligned} < u, v >_2 &= \textstyle\int_\Omega uv\, dx \\ < u, v >_H &= \textstyle\int_\Omega u'v'\, dx. \end{aligned}$$

By the Riesz Representation Theorem (see [1]), for all $u, v \in G$, there exists $z \in L^2$ and $w \in H$, such that

$$J'(u)(v) = < z, v >_2 = < w, v >_H .$$

We define $\nabla_2 J(u) = z$ and $\nabla_H J(u) = w$ (see [8]). So,

$$J'(u)(v) = < \nabla_2 J(u), v >_2 = < \nabla_H J(u), v >_H .$$

Since $\nabla_2 J(u) \in L^2$, there exist coefficients α_i such that

$$\nabla_2 J(u) = \sum_{i=0}^{\infty} \alpha_i \psi_i.$$

Then

$$J'(u)(\psi_j) = < \textstyle\sum_{i=0}^{\infty} \alpha_i \psi_i, \psi_j >_2 \quad = \alpha_j,$$

since the eigenfunctions are orthonormal to each other. Hence,

$$\nabla_2 J(u) = \sum_{i=0}^{\infty} J'(u)(\psi_i)\psi_i.$$

Since $< x, y >_H = < \nabla x, \nabla y >_2$ for all $x, y \in H$, we write

$$\begin{aligned} < \nabla_H J(u), v >_H &= < \nabla\nabla_H J(u), \nabla v >_2 \\ &= \textstyle\int_\Omega \nabla\nabla_H J(u) \cdot \nabla v\ dx. \end{aligned}$$

Then by Green's First Identity (found in Theorem A.0.8),

$$\begin{aligned} \textstyle\int_\Omega \nabla\nabla_H J(u)\cdot\nabla v\ dx &= \textstyle\int_\Omega(-\Delta)(\nabla_H J(u))v\ dx \\ &=< -\Delta\nabla_H J(u), v >_2 . \end{aligned}$$

Hence,

$$\nabla_H J(u) = (-\Delta)^{-1}\nabla_2 J(u).$$

Similarly, we have

$$D_H^2 J(u) = (-\Delta)^{-1} D_2^2 J(u).$$

Then

$$\begin{aligned} [D_H^2 J(u)]^{-1}\nabla_H J(u) &= [(-\Delta)^{-1}D_2^2 J(u)]^{-1}(-\Delta)^{-1}\nabla_2 J(u) \\ &= [D_2^2 J(u)]^{-1}(-\Delta)(-\Delta)^{-1}\nabla_2 J(u) \\ &= [D_2^2 J(u)]^{-1}\nabla_2 J(u). \end{aligned}$$

Hence, we may use the L^2 gradient and Hessian in our implementation of the Newton algorithm, (see [7] and [8] for more details). Henceforth, we drop the "2" subscript on the gradient and the Hessian. Also, it is worth noting that the Sobolev gradient is easily computed to be

$$\nabla_H J(u) = \sum_{i=0}^{\infty}\frac{1}{\lambda_i}J'(u)(\psi_i)\psi_i.$$

We use Fourier coefficients of the term $[D^2 J(u)]^{-1}\nabla J(u)$ as the search direction in our algorithm. In our algorithm, the $M\times M$ matrix A represents $D^2 J(u)$, the elements of the vector $g\in\mathbf{R}^M$ are the Fourier coefficients of $P_G\nabla J(u)$, and the vector $\chi\in\mathbf{R}^M$ is the search direction given by $A\chi = g$. Since the L^2 Hessian is not always invertible, we use a least-squares solver (i.e., we minimize $||A\chi - g||^2$ over all $\chi\in G$) to compute this search direction. Alternatively, one could use a pseudo-inverse (see Definition A.0.5) or, if the Hessian is known to be invertible, one could solve the equation directly. Also it is of use to us to compute the number of negative eigenvalues of A, denoted sig(A). Note that sig(A) corresponds to the MI of u provided that u is a nondegenerate critical point of J and M is sufficiently large.

2.4 Galerkin Newton Gradient Algorithm

As previously stated, this algorithm performs Newton's method on ∇J in coefficient space. This gives us critical points of the functional J which are solutions to (1.1). In our computer code (see Appendix B) we use Simpson's rule to compute all integrals. Additionally, the linear algebra package CLAPACK is used. In particular, 'dgels', a least squares solver, is used to compute the search direction χ and 'dsyev' is used to compute the eigenvalues and eigenvectors of the Hessian matrix A.

The Algorithm

- Choose initial coefficients $a = a^0 = \{a_k\}_{k=1}^M$, set $u = u^0 = \sum a_k \Psi_k$, set δ equal to the desired Newton step size, and set the loop counter $n = 0$.
- Loop:
  1. Calculate $g = g^{n+1} = (J'(u)(\Psi_k))_{k=1}^M \in \mathbf{R}^M$ (gradient vector).
  2. Calculate $A = A^{n+1} = (J''(u)(\Psi_j, \Psi_k))_{j,k=1}^M$ (Hessian matrix).
  3. Compute $\chi = \chi^{n+1} = A^{-1}g$ using a least-squares solver to solve $A\chi = g$ (search direction).
  4. Set $a = a^{n+1} = a^n - \delta\chi$ and update $u = u^{n+1} = \sum_{k=1}^M a_k \Psi_k$.
  5. Increment loop counter.
  6. Calculate the Morse index of u (i.e., sig(A)) and $J(u)$.
  7. Calculate $\sqrt{g \cdot g} = ||P_G \nabla J(u)||$; STOP if sufficiently small.

Chapter 3

Results

The triple-well potential, W, has exactly seven critical points: three global minima (at a, b, and c), three MI 1 saddle points, and a MI 2 local maximum. If u is equivalent to a critical point of W (i.e., u is a constant function), then $\Delta u = 0$, $\nabla W(u) = 0$, and $\frac{\partial u}{\partial \eta} = 0$. Thus for u equivalent to any one of these critical points, u is a trivial solution to

$$-\epsilon^2 \Delta u + \nabla W(u) = 0 \quad \text{in } \Omega$$

$$\frac{\partial u}{\partial \eta} = 0 \quad \text{on } \partial\Omega$$

for all ϵ.

3.1 Morse Index 1 Results

First, we investigate the solutions corresponding to the MI 1 critical points of W. To do this we find solutions of (1.1) for $\epsilon \leq 1$ and plot the value of J versus the value of ϵ. We define a "1-Branch" of this plot to be a branch of constant solutions corresponding to a MI 1 saddle point of W (see Figure 3.1). As one can see, there exists an ϵ^* where the MI of the constant solution changes from 1 to 3. If s is a saddle point between two minima of W and β is the lesser eigenvalue of $D^2W(s)$ then $\epsilon^* = \frac{\sqrt{|\beta|}}{\pi}$. For further details we refer the reader to Chapter 4. For our choice of W, $|\beta| \approx 1.910$ and hence $\epsilon^* \approx 0.430$. At ϵ^* the constant solution, which we call u^*, is degenerate. That is to say $D^2J(u)$ (and also A, if M is sufficiently large) has two zero eigenvalues and is not invertible. Let σ_1 and σ_2 be the eigenfunctions that correspond to these two zero eigenvalues. To find initial guesses that will converge to nontrivial solutions, we look at the curve generated by $J(u)$ when $u = u^* + \rho(\sigma_1 \cos(\phi) + \sigma_2 sin(\phi))$, for $\phi \in [0, 2\pi)$ and some $\rho > 0$ (see Figure 3.2). When collecting the data for this curve, we also compute sig(A) for each ϕ.

Our data indicates that the maxima of this graph correspond to initial guesses that may converge to MI 2 solutions. Similarly, the minima correspond to initial guesses that may converge to nontrivial MI 1 solutions (see Table 3.1). We first choose a ϕ so that our initial guess corresponds to one of these minima. We use this as an initial guess for GNGA at $\epsilon = \epsilon^* - \delta$ for δ small. In fact, with this initial guess, and a small Newton step size, GNGA does converge to a nontrivial MI 1 solution. Using this solution as an initial guess, we decrease ϵ and find solutions along this new branch of nontrivial solutions to complete this portion of the diagram. Similarly, we find MI 2 solutions that also bifurcate off of the 1-Branch at ϵ^*. However, these are not the MI 2 solutions found by Flores, Padilla & Tonegawa [2001].

ϕ	$J(u)$	sig(A)
0.0	5.102	1
0.8	5.336	2
1.7	5.148	1
2.3	5.335	2
3.1	5.103	1
4.1	5.316	2
4.7	5.134	1
5.5	5.336	2
6.2	5.108	1

Table 3.1: Values of $J(u, \phi)$ when $u = u^* + (\sigma_1 \cos(\phi) + \sigma_2 sin(\phi))$.

3.2 Morse Index 2 Results

Next, we investigate the solutions to (1.1) corresponding to the MI 2 local maximum of W. In our experiment, we find that there is an $\epsilon^* \in \mathbf{R}$ for which sig(A)=2 for $\epsilon^* < \epsilon \leq 1$, but sig$(A) \geq 6$ for $0 < \epsilon < \epsilon^*$. In Chapter

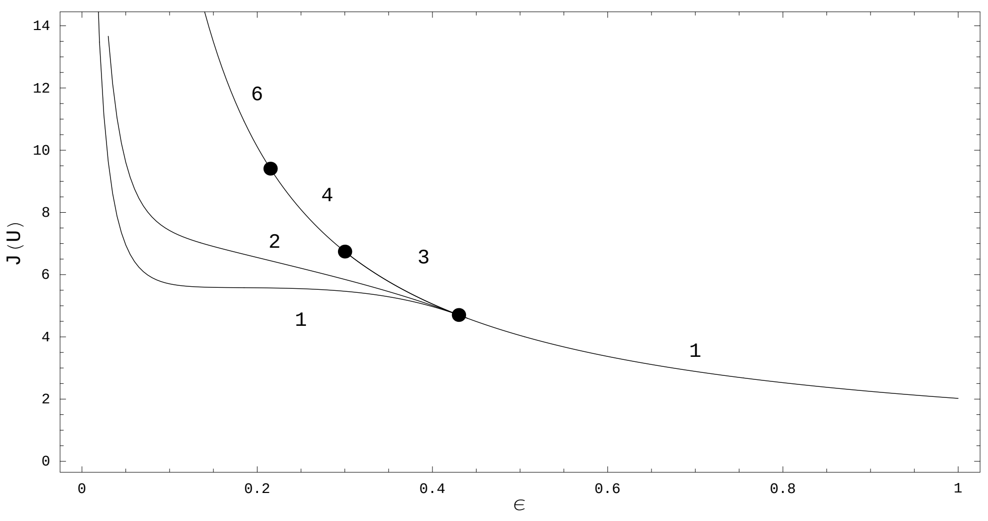


Figure 3.1: Bifurcation diagram containing Morse index 1 solutions. The numbers above or below each branch indicate the Morse index. Further bifurcation is expected at the other bifurcation points. The value of ϵ for every bifurcation point can be computed exactly, $\epsilon = \sqrt{\frac{|\beta|}{\lambda_i}}$ for $i = 1, 2, 3, ...$ For the first bifurcation point, $\epsilon \approx \sqrt{\frac{|1.91|}{\pi^2}} \approx 0.44$, the second is at $\epsilon \approx \sqrt{\frac{|1.91|}{2\pi^2}} \approx 0.31$, and the third is at $\epsilon \approx \sqrt{\frac{|1.91|}{4\pi^2}} \approx 0.22$. Note that the second bifurcation point corresponds to a simple eigenvalue, thus the MI only changes by 1. The first and third eigenvalues correspond to double eigenvalues, thus the MI changes by 2. See Chapter 4 for more details.

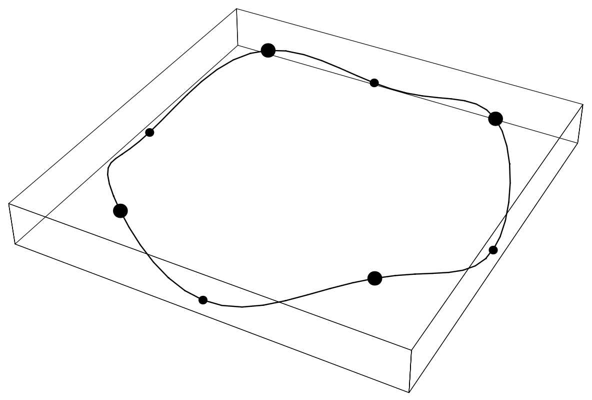


Figure 3.2: Graph of $J(u)$ versus ϕ for $u = u^* + \rho(\sigma_1 \cos(t) + \sigma_2 sin(t))$ with $\rho = 1$. The large dots represent maxima and the small dots represent minima.

4 we calculate ϵ^* for our choice of W to be $\epsilon^* = \frac{\sqrt{2}}{\pi}$. See Figure 3.3 for the bifurcation diagram with $J(u)$ dependent upon the bifurcation parameter ϵ. We define the "2-Branch" of our bifurcation diagram to be the branch of constant solutions corresponding to the MI 2 local maximum of W. Again, at ϵ^*, u is degenerate. In fact, $D^2J(u)$ has four zero eigenvalues with four corresponding eigenfunctions: σ_1, σ_2, σ_3, σ_4. Thus, to search for an initial guess which converges to a MI 2 solution, we look at linear combinations of σ_1, σ_2, σ_3, and σ_4 by using spherical coordinates in 4 dimensions:

$$u = u^* + \rho(\sigma_1 \cos\phi + \sigma_2(\sin\phi\cos\theta) + \sigma_3(\sin\phi\sin\theta\cos\tau) + \sigma_4(\sin\phi\sin\theta\sin\tau))$$

$$\rho > 0,\ \phi \in (-\pi, \pi),\ \theta, \tau \in [0, 2\pi).$$

In practice, there is some trial and error experimentation with values of $\rho \in (0, 1]$ and $\epsilon < \epsilon^*$ required to find a suitable initial guess. Once an initial guess has been found, we use it and the appropriate value for ϵ to find a nontrivial MI 2 solution. As done for the 1-Branch, we decrease ϵ and use the previous solution in turn as an initial guess to complete the nontrivial MI 2 branch of solutions as depicted in Figure 3.3.

Four types of branches of solutions bifurcate off of the 2-Branch at ϵ^*. These four branches correspond to solutions of MI 2, 3, 4, and 5. As ϵ decreases, these branches get very close together. Following these four branches is straightforward for ϵ near ϵ^*, but becomes challenging as $\epsilon \to 0$. There appears to be secondary and perhaps tertiary bifurcation at smaller values of ϵ. Verifying these finer features requires skill and patience. However, by setting $\epsilon = 10^{-4}$, using an initial guess from the bifurcated MI 2 branch, and setting a small Newton step-size, one can obtain the desired MI 2 solution. This solution, proven to exist by Flores, Padilla & Tonegawa [2001], approximates a step-function with triple junction structure (see Figure 3.4). The nodal set of this step-function has minimal length and intersects the boundary at right angles, as expected by Flores, Padilla & Tonegawa. Since we use a Fourier series to approximate a step-function, M must be quite large and a certain amount of error is expected. In Figure 3.4, $\widetilde{M} = 225$, so $M = 452$.

3.3 Other Patterns

Other patterns are depicted in Figure 3.5. These patterns correspond to the MI 3, 4 and 5 bifurcation branches in Figure 3.3. Even though the MI

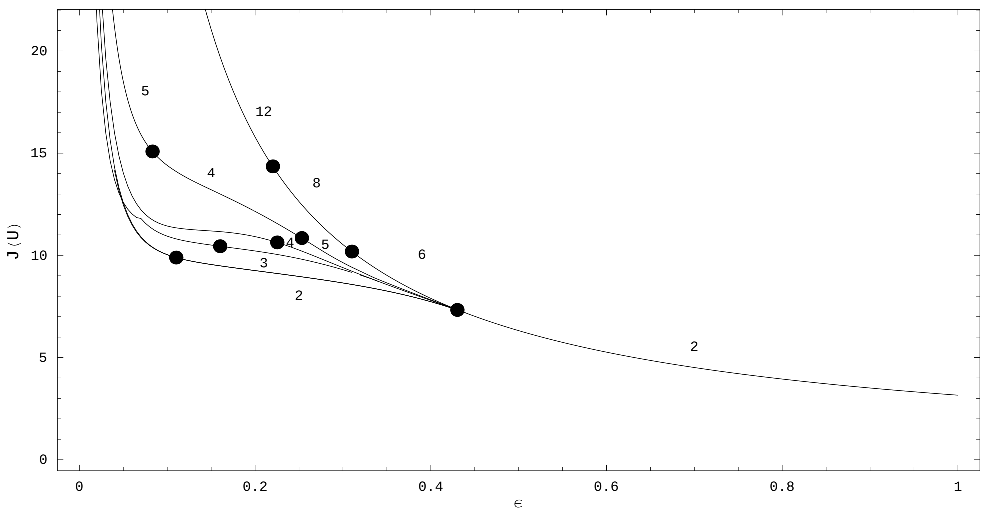
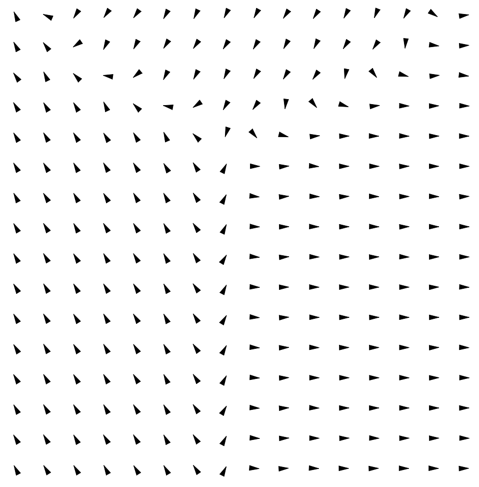
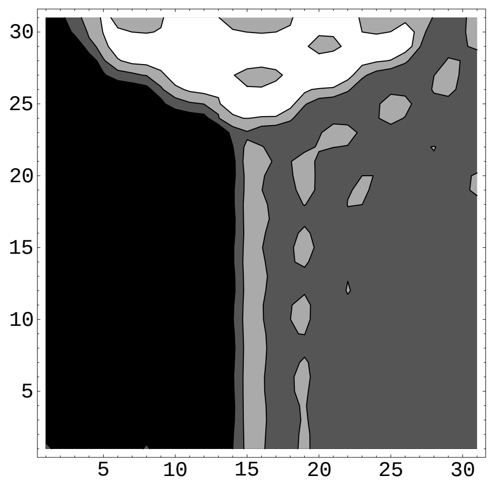


Figure 3.3: Bifurcation diagram containing Morse index 2 solutions. The numbers above or below each branch indicate the Morse index. Further bifurcation is expected at the other points (where the constant solution is degenerate.) Here $\beta = -2$ is a double eigenvalue of $D^2W(0)$. Thus the value of ϵ at the bifurcation points is $\epsilon = \sqrt{\frac{|-2|}{\lambda_i}}$ for $i = 1, 2, 3, ...$ For the first bifurcation point, $\epsilon = \sqrt{\frac{|-2|}{\pi^2}} = 0.45$, the second is at $\epsilon = \sqrt{\frac{|-2|}{2\pi^2}} = 0.32$, and the third is at $\epsilon = \sqrt{\frac{|-2|}{4\pi^2}} \approx 0.22$. Since β is a double eigenvalue of $D^2W(0)$, the jump in MI across bifurcation points is double that of the corresponding Morse indices in Figure 3.1. See Chapter 4 for more details. Note the discontinuity in the MI 3 branch near $\epsilon = 0.075$, this is an example of the "branch-jumping" that can occur when numerically attempting to follow a given branch. Also, notice that the MI 5 branch changes to MI 4 and then back to MI 5, a feature worthy of further study. This diagram is incomplete for $\epsilon < 0.32$ where other bifurcations off the main 2-Branch lead to more features. Additional, similar branches of MI 2, 3, 4 and 5 bifurcate from the main 2-Branch at $\epsilon^* = \frac{\sqrt{|-2|}}{\pi}$. These additional branches correspond to rotations and permutations of solutions found here.



(a)



(b)

Figure 3.4: Approximation of a pattern formation with triple junction structure at $\epsilon = 10^{-4}$ on the MI 2 nontrivial branch. This is the solution expected by Flores, Padilla & Tonegawa [2001]. Each region corresponds to a minima of W: $a = (-1.1, -2.0)$, $b = (-1.1, 2.0)$, and $c = (2.3, 0)$. Figure (a) is a vector field representation of u. If $u = (u_1, u_2)^T$, then Figure (b) is a contour plot of $u_1 \cdot u_2$.

does change along each branch, the patterns do not change substantially. As $\epsilon \to 0$ the boundaries become more defined, but the basic shape of the pattern stays the same. For example, a pattern found on the MI 4 bifurcation branch at $\epsilon \approx 0.24$ has the same"vertical stripes" shape as the pattern found at $\epsilon \approx 0.01$. At $\epsilon \approx 0.24$ the MI of the pattern is 4, but at $\epsilon \approx 0.01$ the MI is 2.

More patterns may be found by carefully following the other bifurcation points on Figure 3.3. We also believe rotations and permutations of these patterns can be found. These rotations and permutations would be found by carefully selecting different linear combinations of eigenfunctions σ_1, σ_2, σ_3, and σ_4.

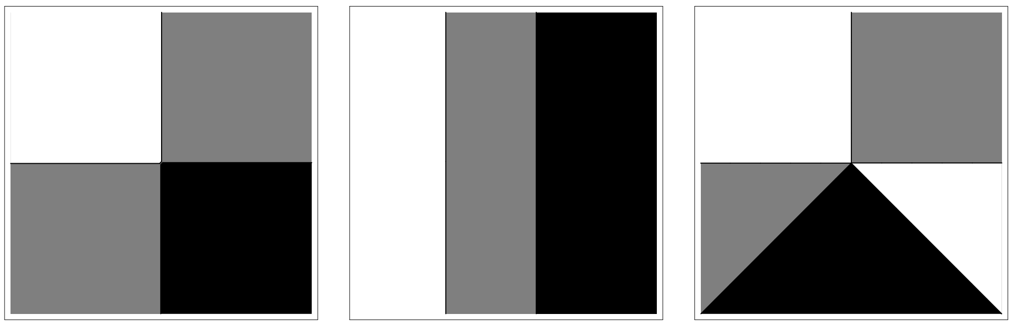


Figure 3.5: Patterns found corresponding to the MI 3, 4 and 5 bifurcation branches in Figure 3.3. These patterns are not found with actual data, but are fair representations of the patterns that would be found for larger values of M and smaller values of ϵ.

Chapter 4

Existence Proof of Bifurcation Points

In our numerical experiments in this paper and elsewhere, we have observed bifurcation in almost all instances where the Hessian $D^2J(u)$ is not invertible and this singularity is isolated. We do not know of existing theorems that will prove the existence of all such bifurcations, but let us see that we may at least apply known theory to account for bifurcation when the zero eigenvalues are simple. Furthermore, one may arrange for the existence of many such simple bifurcation points.

Let us consider the case where u is the constant solution $u \equiv 0$ corresponding to the MI 2 local maximum of W. In our example the two eigenvalues of $D^2W(0)$ are both equal to -2, but with a slight modification of W one can force these two negative eigenvalues to be distinct. Thus, assume that $\beta_2 < \beta_1 < 0$ are these two values, with corresponding eigenfunctions $\sigma_1, \sigma_2 \in \mathbf{R}^2$, with $|\sigma_i| = 1$. For ease of notation, let $\beta = \beta_2$ and $\sigma = \sigma_2$. Let $\lambda = \lambda_i$ be a simple eigenvalue of the scalar negative Laplacian problem (with zero Neumann boundary condition), and take $\psi = \psi_i$ to be the corresponding eigenfunction, with $\int_\Omega \psi^2 \, dx = 1$. Certainly such simple eigenvalues abound for our case $\Omega = (0,1) \times (0,1)$, and for general regions they are the rule rather than the exception. Now let $\epsilon^* = \sqrt{|\beta|/\lambda}$.

We claim that $x_0 = \sigma\psi$ is an eigenfunction of $D^2J(0)$ corresponding to a simple zero eigenvalue. Indeed,

$$\begin{aligned}\langle D^2J(0)x_0, x_0\rangle &= J''(0)(x_0, x_0)\\ &= \textstyle\int_\Omega(\epsilon^*|\sigma|^2|\nabla\psi|^2 + \frac{1}{\epsilon^*}(D^2W(0)\sigma\cdot\sigma)\psi^2)\,dx\\ &= \epsilon^*\lambda + \tfrac{1}{\epsilon^*}\beta \;=\; 0.\end{aligned}$$

Note that $\{\sigma_j\psi_k\}$ forms an orthonormal basis for $H\times H$, as well as for $L^2\times L^2$. Since $\langle D^2J(0)v, v\rangle < 0$ for $v = \sigma_j\psi_k$, $k < i$, $j = 1, 2$, and $\langle D^2J(0)v, v\rangle > 0$ for $v = \sigma_j\psi_k$, $k > i$, $j = 1, 2$ or $k = i$, $j = 1$, we see that along the 2-Branch MI$\leq 2(i-1)$ for $\epsilon > \epsilon^*$ and MI$\geq 2i - 1$ for $\epsilon < \epsilon^*$. Similarly, we can provide for many simple zero eigenvalues along the 1-Branch. In that case we need only look for simple eigenvalues of $-\Delta$, since the Hessian of W automatically has simple eigenvalues at saddle points.

Let us see that in this simple eigenvalue case, bifurcation can be proven to exist. We will apply Theorem A.0.4 at some ϵ^* where $x_0 = \sigma\psi$ is an eigenfunction of $D^2J(0)$ with a corresponding simple zero eigenvalue. We need to establish that the four hypotheses of that theorem hold. Let $F(t, x) = \nabla J_{\epsilon^*+t}(x)$. Clearly F, F_x, and F_{tx} all exist and are continuous, and $F(t, 0) = 0$ for all $t > 0$, which are the first two conditions. Since the Hessian $F_x(t, x)$ is self-adjoint, it follows that $N = N(F_x(0, 0))$ and $R = (R(F_x(0, 0)))^\perp$ are the same 1-dimensional subspace of $H\times H$, spanned by x_0, which is the third condition. Since $w = F_{tx}(0, 0)x_0 \in R(F_x(0, 0))$ if and only if $\langle w, v\rangle = 0$ for all $v \in N$, we need only show that $\langle w, x_0\rangle \neq 0$ to conclude that the final condition of Theorem A.0.4 is satisfied. Since $F_{tx}(0, 0) = \frac{\partial}{\partial t}F_x(t, x)|_{(0,0)}$ and

$$J''_\epsilon(0)(v, v) = \int_\Omega(\epsilon|\nabla v|^2 + \frac{1}{\epsilon}D^2W(0)v\cdot v)\,dx,$$

we see that

$$\langle w, x_0\rangle = \lambda - \frac{1}{(\epsilon^*)^2}\int_\Omega((D^2W(0)\sigma\cdot\sigma)\psi^2)\,dx = \lambda - \frac{\beta}{(\epsilon^*)^2} > 0.$$

We conclude that all four conditions are satisfied, and Theorem A.0.4 implies that there is bifurcation wherever (at least) $D^2W(0)$ has a simple eigenvalue $\beta < 0$, $\lambda = \lambda_i$ is simple, and we choose $\epsilon^* = \sqrt{\frac{|\beta|}{\lambda}}$. A similar proof shows the existence of bifurcation points at the simple zero eigenvalues of $D^2J(u)$ when u is one of the constant solutions belonging to one of the three 1-Branches.

Chapter 5

Conclusion

Our method works very well as long as ϵ is not too small. Extremely small values of ϵ yield solutions that are close to step-functions thus our basis of cosine functions has some limitations. One could consider using an alternate choice of basis for $L^2 \times L^2$ and $H \times H$. In fact, at the time of this writing the use of a basis of wavelets is being investigated. Such a basis would be useful for studying our problem as well as the following Ginzburg-Landau equation with Dirichlet boundary conditions from [2] and [3]

$$\begin{aligned} -\Delta u - \frac{1}{\epsilon^2} u(1-|u|^2) &= 0 \text{ in } \Omega, \\ u &= g \text{ on } \partial\Omega, \end{aligned}$$

where u is a complex valued function and $g \in \mathbf{R}$. With proper boundary conditions and choice of Ω, this equation models vortices, which perhaps would be better handled by other bases such as wavelets. In any case, the change in boundary condition necessitates the use of a modified basis. See [3] for more information on Ginzburg-Landau vortices.

Further investigation may be done by following bifurcation branches from the other bifurcation points. These branches may lead to other pattern formations, (especially for bifurcations off of the 2-Branch). Another direction one may wish to pursue is investigating the effect of having a nonsymmetric region Ω or a nonsymmetric triple-well potential. We have already developed and tested algorithms for computing a basis of eigenfunctions of the Laplacian for general regions. Thus implementing our method on a nonsymmetric region would not be difficult. The properties of the resulting pattern formations may be interesting and useful.

There is also opportunity to do theoretical work with this problem. Using known theorems, one might analytically prove the existence of multiple bifurcation branches. This would be similar to what was done in Chapter 4, but would consist of looking at multiple zero eigenvalues as opposed to simple zero eigenvalues. Also a more thorough study of the numerics could be done to make the algorithm more efficient. For example, as of this writing, the use of a backtracking line search to find a better Newton step-size has been implemented.

Bibliography

[1] Adams, R. A. [1975] *Sobolev Spaces*, (Academic Press, New York).

[2] Almeida, L. & Betheul, F. [1998] "Topological Methods for the Ginzburg-Landau Equations," *J. Math. Pures Appl.* **77**, 1-49.

[3] Bethuel, F., Brezis, H. & Heléin, F. [1994] *Ginzburg-Landau Vortices* (Birkhauser, Boston).

[4] Crandall, M. G. & Rabinowitz, P. H. [1971] "Bifurcation from Simple Eigenvalues," *Journal of Functional Analysis* **8**, 321-340.

[5] Flores, G., Padilla, P. & Tonegawa, Y. [2001] "Higher Energy Solutions in the Theory of Phase Transitions: A Variational Approach," *J. Differential Equations* **169**(1), 190–207.

[6] McOwen, R. C. [1995] *Partial Differential Equations: Methods and Applications*, (Prentice-Hall, New Jersey).

[7] Neuberger, J. M. & Swift, J. W. [2001] "Newton's Method and Morse Index for Semilinear Elliptic PDEs," *Internat. J. Bifur. Chaos Appl. Sci. Engrg.* **11**(3), 801–820.

[8] Neuberger, J. W. [1997] *Sobolev Gradients and Differential Equations*, Lecture Notes in Mathematics **1670**, (Springer-Verlag, Berlin).

[9] Rudin, W. [1976] *Principles of Mathematical Analysis*, (McGraw-Hill, New York).

[10] Taylor, A. E. [1967] *Introduction to Functional Analysis*, (John Wiley & Sons, New York).

Appendix A

Definitions and Theorems

Definition A.0.1 *Suppose all one-point sets in X are closed. The set X is* **regular** *if for each pair consisting of a point x and a closed set B disjoint from x, there exist disjoint open sets containing x and B, respectively.*

Theorem A.0.2 (Lebesgue Dominated Convergence Theorem, see page 17 of [1]) *Let $A \subset \mathbf{R}^n$ be measurable and let $\{f_n\}$ be a sequence of measurable functions converging to a limit pointwise on A. If there is a function $g \in L^1(A)$ such that $|f_n(x)| \leq g(x)$ for every n and all $x \in A$, then*

$$\lim_{n\to\infty} \int_A f_n(x)\,dx \; = \int_A \left(\lim_{n\to\infty} f_n(x) \right)\, dx.$$

Definition A.0.3 *Let $f : X \to Y$.*
The **range** *of f is $R(f) = \{y \in Y \mid f(x) = y \text{ for some } x \in X\}$.*
The **null space** *of f is $N(f) = \{x \in X \mid f(x) = 0 \in Y\}$.*

Theorem A.0.4 (Crandall and Rabinowitz, Theorem 1.7 of [4]) *Let X and Y be Banach spaces, V a neighborhood of 0 in X and*

$$F : (-1, 1) \times V \to Y$$

have the properties

(a) *$F(t,0)=0$ for $|t| < 1$,*

(b) *The partial derivatives F_t, F_x and F_{tx} exist and are continuous,*

(c) *$N(F_x(0,0))$ and $Y|R(F_x(0,0))$ are one-dimensional.*

(d) $F_{tx}(0,0)x_0 \notin R(F_x(0,0))$, *where* $N(F_x(0,0))$ =*span*$\{x_0\}$.

If Z is any complement of $N(F_x(0,0))$ *in X, then there is a neighborhood U of (0,0) in* $\mathbf{R} \times X$*, an interval (-a,a), and continuous functions* $\phi : (-a,a) \to \mathbf{R}$, $\psi : (-a,a) \to Z$ *such that* $\phi(0) = 0$, $\psi(0) = 0$, *and*

$$F^{-1}(0) \cap U = \{(\phi(\alpha), \alpha x_0 + \alpha\psi(\alpha)) : |\alpha| < a\} \cup \{(t,0) : (t,0) \in U\}.$$

Definition A.0.5 *Let A be an* $n \times n$ *matrix such that* $Ae_i = \beta_i e_i$, *where* e_i *and* β_i *are corresponding eigenvectors and eigenvalues of A, respectively. Let* $\tau > 0$ *be some tolerance and*

$$\gamma_i = \begin{cases} \frac{1}{\beta_i} & if|\beta_i| > \tau \\ 0 & if|\beta_i| \leq \tau \end{cases}$$

Let A^+ *denote the* **Pseudo-Inverse** *of matrix A. Then we can compute*

$$A^+ g = \sum_i^n \gamma_i (g \cdot e_i) e_i.$$

The **Pseudo-Inverse** *can also be found by computing*

$$A^+ = (A^T A)^{-1} A^T.$$

Definition A.0.6 (See Section 6.11 of [10]) *A linear operator A with domain and range in the inner-product space X is called* **symmetric** *if*

$$< Ax, y >_X = < x, Ay >_X$$

for all x and y in the domain of A.

Definition A.0.7 (See Section 6.2 of [10]) *Let X be a complete inner-product space and A a continuous linear operator on X into X. The adjoint of A,* A^* *is defined by the relation*

$$< Ax, y >_X = < x, A^* y >_X \quad x,\ y \in X$$

A is **self-adjoint** *if* $A^* = A$. *Clearly a self-adjoint operator is symmetric.*

Theorem A.0.8 *The Laplacian operator is self-adjoint in L^2, that is,*

$$< \Delta u, v >_2 = < u, \Delta v >_2 .$$

Proof:

We use Green's First Identity (page 297 of [9]) which states

$$\int_\Omega f \Delta g \; dV = \int_{\partial\Omega} f \frac{\partial g}{\partial \eta} \; dA - \int_\Omega \nabla f \cdot \nabla g \; dV.$$

Then,

$$\begin{aligned} < \Delta u, v >_2 &= \textstyle\int_\Omega \Delta u v \; dV \\ &= \textstyle\int_{\partial\Omega} v \frac{\partial u}{\partial \eta} \; dA - \int_\Omega \nabla v \cdot \nabla u \; dV \\ &= -\textstyle\int_\Omega \nabla v \cdot \nabla u \; dV \\ &= -\textstyle\int_{\partial\Omega} u \frac{\partial v}{\partial \eta} \; dA + \int_\Omega u \Delta v \; dV \\ &= \textstyle\int_\Omega u \Delta v \; dV \\ &= < u, \Delta v >_2 . \end{aligned}$$

Theorem A.0.9 (See Sections 6.11 and 6.2 of [10]) *Let A be a self-adjoint continuous linear operator on a complete inner-product space X. Then A has real eigenvalues λ_i and eigenfunctions corresponding to distinct eigenvalues are orthogonal.*

Appendix B

Computer Code

All code for this project is written in the C++.

```
/* Newton-pde system solver.
Solves (-epsilon^2)*lap u + grad W(u) = 0 w/ Neumann BC
Compile on Windows with Lapack and Blas DLL’s installed.

Uses dgesv and dsyev.

Note: Since grad and hess are being sent to dgesv,
the indexing goes from 0 to M-1.  This is true for
all vectors/matrices of M length.
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
// f2c.h is used for LAPACK
#include <f2c.h>

// Global functions

double J();
double f(double u, double v);
double fpu(double u, double v);
double fpv(double u, double v);
double W(double u, double v);
double g(double u, double v);
double gpu(double u, double v);
double gpv(double u, double v);
double psi(int i, int j, double x, double y);
int compare( const void *a, const void *b);
void init_lam();
int iii(int j, int k);
int disp_coeff(int i);
int fdisp_coeff(int i);
void fprint_uv(int inc);
void do_evals();
void print_hess(int inc);
void print_grad();
void print_uv(int inc);
void set_uv();
void newline(int num);
void print_evals(int num);
void fprint_evals(int num);
void print_report(int iter, double norm);
void fprint_report(int iter, double norm);
double do_grad();
void do_hess();
void init_values();
double simpsons(int j,int k,int func);
void clear_a();
void print_efuncs(int num);
int sig();
void populate_u();
void set_a();
void fscan_uv(int inc);
void backup_uv();

// Global Constants
int report_switch=0;  // Report to file
  // at each step of Newtons?
const char uname[13]="ugraph39.txt";
const char vname[13]="vgraph39.txt";
const char fname1[12]="nuem.txt";
const int M=7;  // M=6 is used for most experiments.
const int N=30;
const double Pi=3.1415926535;
const int M2=M*M;  // M^2
const double delta1=.15;
const double TOL=.0005;
double epsilon=.0075; // initial value of epsilon
const double morsethresh=0.0;

// Global Variables
// These are global to keep things simple.
double lam[M2];
int lami1[M2], lami2[M2], morse, func_flag=0;
double hess[2*M2][2*M2],grad[2*M2],
a[2*M2],c, evals[2*M2],
u[(N+1)*(N+1)],v[(N+1)*(N+1)];

FILE *fp1;

void main()
{

int i, num, maxits;

double norm, delta, tol, dx;

// This file holds bifurcation data.

FILE *bifur;
bifur=fopen("bifur.txt","w");

delta=delta1;

if ((fp1= fopen(&fname1[0],"w")) == NULL)
{
printf("Error opening write file data.txt\n");
exit(0);
}
```

```
double n=(double)N;
dx=1/n;

// Populate lamda array with eigenvalues and
// corresponding index values.
init_lam();

// Initialize U coeffs.
init_values();
// Compute U=(u,v) based on coeffs.
set_uv();

printf("m=%d, N=%d eps=%f delta=%f\n ",
M,N,epsilon,delta);

fprintf(fp1,"m=%d, N=%d eps=%f delta=%f\n ",
M,N,epsilon,delta);

disp_coeff(6);

fdisp_coeff(6);

newline(1);

// Here we set variables used in the call to dgels.
// This is a little unusuall because of the way
// CLAPACK is set up for MS Windows.
long m=2*M2; // m is used only for the lapack call.
long info=10;   // M2 is used everywhere else.
long one1=1;  //Variable for # of RHS columns.
long lwork=8*M2*M2;
double work[8*M2*M2];

// Save off initial value of epsilon
double initeps=epsilon;

// Set delta_epsilon, how much to change epsilon.
// deleps>0 => epsilon decreases
// deleps<0 => epsilon increases
double deleps= .005;

epsilon += deleps;

// This is the beginning of a loop
// used to draw bifurcation diagrams.
while(epsilon > .004)
{
epsilon -= deleps;

// Start main loop for convergence.
/* Find critical points of gradiant */

delta=delta1; norm=1; tol=TOL; num=0; maxits=2000;
while((num<maxits) && (norm > tol))
{
num++;

// Compute the gradient
norm=do_grad();

// Compute the Hessian
do_hess();

// Evals must be computed before hessian goes to
// dgels_.
do_evals();

//Using dgels_ a least sqrs solver.
dgels_("N",&m, &m, &one1, *hess, &m, grad, &m,
    work, &lwork, &info);
// Error Check!
if (info!=0)
{
printf("\ndgels Error Info= %d\n", info);
exit(0);
}

// Newtons step on co-efficients
// with delta step-size.
for(i=0;i<2*M2;i++) a[i]-=delta*grad[i];

// Compute U
set_uv();

print_report(num, norm);
if (report_switch)
{
fprint_report(num,norm);
fflush(fp1);
}

backup_uv();  //Makes a back-up, just in case.
fprint_uv(1);


}   // Main loop over!

fprint_report(num,norm);
fflush(fp1);
fprintf(bifur,"%f %f %d\n",epsilon,J(),morse);
fflush(bifur);


}
//end of while loop used to draw bifurcation diagrams.

fclose(bifur);

disp_coeff(6);
fdisp_coeff(6);
print_uv(5);

fclose(fp1);
```

```
}
/* End of Main Program  */

// Set W, triple well potential
double W(double u, double v)
{
double fudge=2.95821;

return 0.2714397815791004 - (u*u) - (v*v)
+ 0.13121718850160427*((u*u + v*v)*(u*u + v*v))
- 0.13164822780734486*(u*u*u - 3*u*v*v)+fudge;
}

// First Component of grad W
double f(double u, double v)
{
return -2*u - 0.13164822780734486*(3*(u*u) - 3*(v*v))
+ 0.5248687540064171*u*((u*u) + (v*v));

}

// f prime u
double fpu(double u, double v)
{
return -2 - 0.7898893668440*u + 1.0497375080128*(u*u)
+ 0.5248687540064171*((u*u) + (v*v));
}

// f prime v
double fpv(double u, double v)
{
return 0.7898893668440692*v + 1.0497375080128342*u*v;
}

// Second component of grad W
double g(double u, double v)
{
return -2*v + 0.7898893668440692*u*v
```

```
+ 0.5248687540064171*v*((u*u) + (v*v));
}

double gpu(double u, double v)
{
return 0.7898893668440692*v + 1.0497375080128342*u*v;
}

double gpv(double u, double v)
{
return -2 + 0.78988936684*u + 1.0497375080128*(v*v)
+ 0.5248687540064171*((u*u) + (v*v));
}


// This function returns the value
// of eigenfunction \psi.
// In the future, this will be turned into
// an array of size M+1.
double psi(int i, int j, double x, double y)
{
double ret;

if (i==0 || j==0) ret=sqrt(2)*cos((double)i*Pi*x)
*cos((double)j*Pi*y);

else ret=2*cos((double)i*Pi*x)*cos((double)j*Pi*y);

if (i==0 && j==0) ret=1;
return ret;

}


// Used by sort in init_lam
int compare( const void *a, const void *b)
{
double z;
```

```
z=(*(double *)a - *(double *)b);

if (z<0)  return -1;
if (z==0) return  0;
if (z>0)  return  1;

return((int)z);

}


// Initialize and sort eigenvalues
void init_lam()
{
double tlam[4*M*M][3];
int j,k;
int i=0;
for(j=0;j<2*M;j++)
for(k=0;k<2*M;k++)
{
tlam[i][0]=(j*j+k*k)*Pi*Pi;
tlam[i][1]=j;
tlam[i][2]=k;
i++;
}
// Sort evals using qsort in stdlib.h

qsort(tlam, 4*M*M, sizeof(double)*3, compare);


for(i=0;i<M2;i++)
{
lam[i]=tlam[i][0];
lami1[i]=(int)tlam[i][1];
lami2[i]=(int)tlam[i][2];
}
```

```
// Initialize a and hess arrays.
for(i=0; i<2*M2; i++)
{
a[i]=0;
for(j=0; j<M2; j++)
hess[i][j]=0;
}
}


// Index function.  Takes in indices j,k and
//returns sorted eval number.
int iii(int j, int k)
{
int i,ret;

i=0;
ret=-1;
while(i<M2 && ret==-1)
{
if (lami1[i]==j && lami2[i]==k) ret=i;
else i++;
}

return ret;

}

// Make an i x i display of coefficents.  i<=2*sqrt(M2).
int disp_coeff(int i)
{
if (i>2*sqrt(M2)) return -1;

int j,k;
printf("u coeffs:\n");
for(j=0;j<i;j++)
{
for(k=0;k<i;k++)
if  (iii(j,k) != -1)
printf("%7.3f   ", a[iii(j,k)]);
else printf("000.000   ");
printf("\n");
}

newline(1);
printf("v coeffs:\n");
for(j=0;j<i;j++)
{
for(k=0;k<i;k++)
if  (iii(j,k) != -1)
printf("%7.3f   ", a[iii(j,k)+M2]);
else printf("000.000   ");
printf("\n");
}

printf("\n");
return 0;
}


// fdisp means file display
int fdisp_coeff(int i)
{
if (i>2*sqrt(M2)) return -1;

int j,k;
fprintf(fp1,"\n//u coeffs:\n");
for(j=0;j<i;j++)
{
for(k=0;k<i;k++)
if (iii(j,k) != -1)
fprintf(fp1,"a[iii(%d,%d)]=%.12f;",j,k, a[iii(j,k)]);
else fprintf(fp1,"a[iii(%d,%d)]=0.00000;",j,k);
fprintf(fp1,"\n");
}
```

```
fprintf(fp1,"\n//v coeffs:\n");
for(j=0;j<i;j++)
{
for(k=0;k<i;k++)
if (iii(j,k) != -1)
fprintf(fp1,"a[iii(%d,%d)+M2]=%.12f;", j,k,a[iii(j,k)+M2]);
else fprintf(fp1,"a[iii(%d,%d)]=0.00000;",j,k);
fprintf(fp1,"\n");
}


fprintf(fp1,"\n\n");
fflush(fp1);
return 0;
}

// Compute eigenvalues and eigenvectors.
void do_evals()
{
long j,info,k,m=2*M2;

double  cphess[2*M2][2*M2];
long lwork=8*M2;
double work[8*M2];

// Need to make a copy of the hessian so that the
// original is not destroyed.
for(j=0;j<2*M2;j++)
for(k=0;k<2*M2;k++)
cphess[j][k]=hess[j][k];
long one1=1;
dsyev_("V","U",&m, *cphess, &m, evals, work, &lwork, &info);

if (info!=0)
{
printf("\n\n!!!!!!!Error Computing Eigenvalues!!!!!!!\n\n");
printf("Info=%d",info);
exit(0);
}



morse=0;

for(j=0;j<2*M2;j++)
if (evals[j]<morsethresh) morse++;

FILE *efuncs;

// This file will contain the eigenfunctions of the Hessian
if ((efuncs= fopen("efuncs.txt","w")) == NULL)
{
printf("Error opening write file efuncs.txt. \n");
exit(0);
}

int i;

// This prints the first 6 eigenvectors of the Hessian
// In practice, you don't need any more than 6
// eigenvectors to bifurcate off of a main branch.
for(j=0; j<6; j++)
{
for(i=0;i<2;i++)
{
for(k=0;k<M2;k++)
{
fprintf(efuncs,"a[%d]+=%.9f; ",k+(M2*i),cphess[j][k+(M2*i)]);
}
fprintf(efuncs,"\n");
}
}

fclose(efuncs);
```

```
}

void print_evals(int num)
{
int j;

printf("\n1st %d Evals:\n",num);

for(j=0;j<num;j++)
{
printf("%6.2f ",evals[j]);

}
printf("\n");

printf("Morse index:%d\n",morse);

return;
}

void fprint_evals(int num)
{
int j;

fprintf(fp1,"\n1st %d Evals:\n",num);

for(j=0;j<num;j++)
{

fprintf(fp1,"%6.2f ",evals[j]);
}

fprintf(fp1,"\n");

fprintf(fp1,"Morse index:%d\n",morse);

return;
}
```

```
// This routine used for diagnostics only.
void print_hess(int inc)
{
int j,k;

printf("\nhess:\n");
for (j=0; j<M2; j+=inc)
{
for(k=0; k<M2; k+=inc)
printf("%7.3f ",hess[j][k]);
printf("\n");
}
}

void print_uv(int inc)
{

int x,y,i;

printf("\nu:\n");

for(x=0;x<=N;x+=inc)
{
for(y=0;y<=N;y+=inc)
{
i=(N+1)*x+y;
printf("%7.2f ",u[i]);
}
printf("\n");
}

printf("\nv:\n");

for(x=0;x<=N;x+=inc)
{
for(y=0;y<=N;y+=inc)
{
```

```
i=(N+1)*x+y;
printf("%7.2f ",v[i]);
}
printf("\n");
}
}

void fprint_uv(int inc)
{
//Must be a divisor of N
int x,y,i;
FILE *ugraph;
FILE *vgraph;

ugraph = fopen("ugraph.txt","w");
vgraph = fopen("vgraph.txt","w");

for(x=0;x<=N;x+=inc)
{
for(y=0;y<=N;y+=inc)
{
i=(N+1)*x+y;

fprintf(ugraph,"%.12f ",u[i]);
}

fprintf(ugraph,"\n");
}
fclose(ugraph);


for(x=0;x<=N;x+=inc)
{
for(y=0;y<=N;y+=inc)
{
i=(N+1)*x+y;

fprintf(vgraph,"%.12f ",v[i]);
```

```
}

fprintf(vgraph,"\n");
}

fclose(vgraph);
}


//Remember to set_a() after scanning u and v.
// Read in uv data from the files named uname and vname,
// global constants. This is one way to set initial
// values for u and v. Here the actual function is read
// in, the set_a() computes the coefficients.
void fscan_uv(int inc)
{
//Must be a divisor of N
int x,y,i;
float a;
FILE *ugraph;
FILE *vgraph;

if ((ugraph = fopen(&uname[0],"r"))==NULL)
{
printf("\nError openning ugraph fscan");
exit(10);
}

if ((vgraph = fopen(&vname[0],"r"))==NULL)
{
printf("\nError openning vgraph fscan");
exit(10);
}

for(x=0;x<=N;x+=inc)
{
for(y=0;y<=N;y+=inc)
{
```

```
i=(N+1)*x+y;

fscanf(ugraph,"%f",&a);
u[i]=(double) a;

}


}
fclose(ugraph);


for(x=0;x<=N;x+=inc)
{
for(y=0;y<=N;y+=inc)
{
i=(N+1)*x+y;

fscanf(vgraph,"%f",&a);
v[i]=(double) a;
}


}

fclose(vgraph);
}

// a backup routine to backup to old u and v graphs.
void backup_uv()
{
int i,x,y,inc=1;

float a;
double uu[(N+1)*(N+1)],vv[(N+1)*(N+1)];

FILE *ugraph;
FILE *vgraph;
```

```
if ((ugraph = fopen("ugraph.txt","r"))==NULL)
{
printf("\nbackup_uv:Error openning ugraph\n");
return;
}

if ((vgraph = fopen("vgraph.txt","r"))==NULL)
{
printf("Error openning vgraph.txt");
return;
}

for(x=0;x<=N;x+=inc)
{
for(y=0;y<=N;y+=inc)
{
i=(N+1)*x+y;

fscanf(ugraph,"%f",&a);
uu[i]=(double) a;

}


}
fclose(ugraph);


for(x=0;x<=N;x+=inc)
{
for(y=0;y<=N;y+=inc)
{
i=(N+1)*x+y;

fscanf(vgraph,"%f",&a);
vv[i]=(double) a;
}
```

```
}

fclose(vgraph);

if ((ugraph = fopen("ugraph.bak","w"))==NULL)
{
printf("Error openning ugraph");
exit(10);
}

if ((vgraph = fopen("vgraph.bak","w"))==NULL)
{
printf("Error openning vgraph");
exit(10);
}



for(x=0;x<=N;x+=inc)
{
for(y=0;y<=N;y+=inc)
{
i=(N+1)*x+y;

fprintf(ugraph,"%.12f ",uu[i]);
}

fprintf(ugraph,"\n");
}
fclose(ugraph);


for(x=0;x<=N;x+=inc)
{
for(y=0;y<=N;y+=inc)
{
```

```
i=(N+1)*x+y;

fprintf(vgraph,"%.12f ",vv[i]);
}

fprintf(vgraph,"\n");
}

fclose(vgraph);
}


// Compute U from the Fourier coefficients
void set_uv()
{
int x,y,i,k;

double n=N;


for(x=0; x<N+1; x++)
for(y=0;y<N+1; y++)
{
i=(N+1)*x+y; // This acheives the effect of
u[i]=0; // having a 2 dimensional matrix
v[i]=0; // in a one dimesional array.
for(k=0;k<M2;k++)
{
u[i]+=a[k]*psi(lami1[k],lami2[k],x/n,y/n);

v[i]+=a[k+M2]*psi(lami1[k],lami2[k],x/n,y/n);

}
}
}

// Pring the gradient to screen.
// Mainly used for diagnostics.
```

```
void print_grad()
{
int i;

printf("\nGrad:\nu: v:\n");

for(i=0;i<M2;i++)
printf("%7.2f %7.2f\n",grad[i],grad[i+M2]);
newline(1);
}

//print num blank lines to screen
void newline(int num)
{
int i;
for(i=0;i<num;i++) printf("\n");
}

void print_report(int iter, double norm)
{
printf("\nIteration: %d Norm: %.6f\n",iter, norm);
printf("c=%.3f J(u)=%f\n",c,J());
printf("Epsilon=%f\n",epsilon);
print_evals(9);
//print_uv(4);

//disp_coeff(3);
printf("\n---------------------------------------\n");

return;
}

void fprint_report(int iter, double norm)
{
fprintf(fp1,"\nIteration: %d Norm: %.6f\n",iter, norm);
fprintf(fp1,"c=%.3f J(u)=%f\n",c,J());
fprintf(fp1,"Epsilon=%f\n",epsilon);
fprint_evals(9);
```

```
//fprint_uv(4);
fdisp_coeff(6);
fprintf(fp1,"\n--------------------------------------\n");

return;
}

// Compute J
double J()
{
int i;
double sum1=0,sum2=0;

for(i=0; i<M2; i++)
sum1+=(a[i]*a[i] + a[i+M2]*a[i+M2])
*lam[i];

sum1/=2;

/* for(i=0;i<(N+1)*(N+1);i++)
sum2+=W(u[i],v[i]);
*/
sum2=simpsons(0,0,7);

return epsilon*sum1+sum2/epsilon;
}

// Compute J'
double do_grad()
{
int i;
double n= (double)N, dx=1/n, sum1,sum2,norm;

norm=0;
for(i=0; i<M2; i++)
{

sum1=simpsons(i,i,5);
```

```
sum2=simpsons(i,i,6);
grad[i]= epsilon*a[i]*lam[i] + sum1/epsilon;
grad[i+M2]=epsilon*a[i+M2]*lam[i] + sum2/epsilon;
norm+=grad[i]*grad[i]
+grad[i+M2]*grad[i+M2];
}
norm=sqrt(norm);
return norm;
}


// Find coefficients based on the current value of U=(u,v)
void set_a()
{
int k;

for(k=0; k<M2; k++)
{

a[k]=simpsons(k,k,8);
a[k+M2]=simpsons(k,k,9);


}


}

// Compute the Hessian
void do_hess()
{
int j,k;
double n= (double)N, dx=1/n, sum1,sum2,
sum3,sum4;

for(k=0;k<M2; k++)
{
for(j=0;j<M2;j++)
```

```
{
sum1=simpsons(j,k,1);
sum2=simpsons(j,k,2);
sum3=simpsons(j,k,3);
sum4=simpsons(j,k,4);

if (j==k)
{
hess[k][j]= epsilon*lam[j] + sum1/epsilon;
hess[k][j+M2]= sum2/epsilon;
hess[k+M2][j]= sum3/epsilon;
hess[k+M2][j+M2]= epsilon * lam[j] + sum4/epsilon;
}
else
{
hess[k][j]= sum1/epsilon;
hess[k][j+M2]= sum2/epsilon;
hess[k+M2][j]= sum3/epsilon;
hess[k+M2][j+M2]= sum4/epsilon;
}
}

}
return;
}

// Composite Simpson's rule.  func values:
// func=1-4 for hessian
// func=5-6 for gradiant
//  func=7 for J();
//  func=8-9 for setting a's from alread set U.
double simpsons(int j,int k,int func)
{

int x,y,i;
double J,j1=0,j2=0,j3=0,L=0,k1=0,k2=0,k3=0,Q=0,
n=N;
```

```
for(x=0;x<=N;x++)
{
k1=0; k2=0; k3=0;

for(y=1;y<N;y++)
{
i=(N+1)*x+y;
// See above comment for func value
if (func==1)
Q=psi(lami1[j],lami2[j],x/n,y/n)
* psi(lami1[k],lami2[k],x/n,y/n)
* fpu(u[i],v[i]);
else if (func==2)
Q=psi(lami1[j],lami2[j],x/n,y/n)
* psi(lami1[k],lami2[k],x/n,y/n)
* fpv(u[i],v[i]);
else if (func==3)
Q=psi(lami1[j],lami2[j],x/n,y/n)
* psi(lami1[k],lami2[k],x/n,y/n)
* gpu(u[i],v[i]);
else if (func==4)
Q=psi(lami1[j],lami2[j],x/n,y/n)
* psi(lami1[k],lami2[k],x/n,y/n)
* gpv(u[i],v[i]);
else if (func==5)
Q=psi(lami1[j],lami2[j],x/n,y/n)
 *f(u[i],v[i]);
else if (func==6)
Q=psi(lami1[j],lami2[j],x/n,y/n)
 *g(u[i],v[i]);
else if (func==7)
Q=W(u[i],v[i]);
else if (func==8)
Q=psi(lami1[j],lami2[j],x/n,y/n)
*u[i];
else if (func==9)
Q=psi(lami1[j],lami2[j],x/n,y/n)
*v[i];
if (y%2==0) k2=k2+Q;
else k3=k3+Q;
}

L=(k1+2*k2+4*k3)/(3*n);
if (x==0 || x==N)
j1+=L;
else if (x%2==0) j2+=L;
else j3+=L;
}

J=(j1+2*j2+4*j3)/(3*n);
return J;

}


void clear_a()
{
int i;

for(i=0; i<2*M2; i++)
{
a[i]=0;


}

}

// This is a ‘stand-alone’ function used outside of the
// main code. Can be used when computing the 4D sphere
// of efuncts.
int sig()
{
do_hess();
do_evals();
```

```
return morse;
}

// This routine sets U to be the guessed MI2
// non-trivial solution.
void populate_u(double delta)
{
int x,y,i;
double n=N;
double a1,a2,b1,b2,c1,c2,val1,val2;

a1=-1.1821; a2= -2.04746; b1=-1.1821; b2= 2.04746;
c1=2.3642; c2= 0;

for(x=0;x<=N;x++)
for(y=0;y<=N;y++)
{
i=(N+1)*x+y;

val1=1-1/sqrt(3)*(x/n);
val2=1/sqrt(3)*(x/n) + (1-1/sqrt(3));

if ( (y/n > val1) && (y/n > val2) )
{
u[i]=delta*a1;
v[i]=delta*a2;
}

else
{
if (x/n < .5)
{
u[i]=delta*b1;
v[i]=delta*b2;
}
else
{
u[i]=delta*c1;
v[i]=delta*c2;
}
}
}

}

//Initial values are set here.
void init_values()
{
double a1,a2,b1,b2,c1,c2;

clear_a();

// These are critical points of W.
a1=-1.1821; a2= -2.04746;
b1=-1.1821; b2= 2.04746;
c1=2.3642; c2= 0;


// Coeffs are printed in this form to neum.txt
//u coeffs:
a[iii(0,0)]=-0.183342578126;a[iii(0,1)]=-0.000000000002;
a[iii(0,2)]=-0.000683534479;a[iii(0,3)]=0.000000000000;
a[iii(0,4)]=-0.000052314392;a[iii(0,5)]=0.000000000000;
a[iii(1,0)]=1.169971471586;a[iii(1,1)]=0.000000000009;
a[iii(1,2)]=-0.005064053567;a[iii(1,3)]=0.000000000004;
a[iii(1,4)]=-0.001947713348;a[iii(1,5)]=0.000000000002;
a[iii(2,0)]=0.293915169831;a[iii(2,1)]=0.000000000012;
a[iii(2,2)]=-0.005287883505;a[iii(2,3)]=0.000000000004;
a[iii(2,4)]=-0.001972313461;a[iii(2,5)]=0.000000000002;
a[iii(3,0)]=-0.017275064343;a[iii(3,1)]=0.000000000001;
a[iii(3,2)]=0.000231109072;a[iii(3,3)]=-0.000000000000;
a[iii(3,4)]=0.000166226789;a[iii(3,5)]=-0.000000000000;
a[iii(4,0)]=-0.016124417015;a[iii(4,1)]=-0.000000000001;
a[iii(4,2)]=0.000717245568;a[iii(4,3)]=-0.000000000001;
a[iii(4,4)]=0.000317233309;a[iii(4,5)]=0.00000;
a[iii(5,0)]=-0.007552779989;a[iii(5,1)]=-0.000000000001;
```

```
a[iii(5,2)]=0.000446084373;a[iii(5,3)]=-0.000000000000;
a[iii(5,4)]=0.00000;a[iii(5,5)]=0.00000;

//v coeffs:
a[iii(0,0)+M2]=0.317558660399;a[iii(0,1)+M2]=0.000000000004;
a[iii(0,2)+M2]=0.001183916446;a[iii(0,3)+M2]=-0.000000000000;
a[iii(0,4)+M2]=0.000090611184;a[iii(0,5)+M2]=-0.000000000000;
a[iii(1,0)+M2]=0.675483344113;a[iii(1,1)+M2]=0.000000000005;
a[iii(1,2)+M2]=-0.002923732691;a[iii(1,3)+M2]=0.000000000002;
a[iii(1,4)+M2]=-0.001124512826;a[iii(1,5)+M2]=0.000000000001;
a[iii(2,0)+M2]=-0.509076007207;a[iii(2,1)+M2]=-0.000000000022;
a[iii(2,2)+M2]=0.009158882895;a[iii(2,3)+M2]=-0.000000000007;
a[iii(2,4)+M2]=0.003416147123;a[iii(2,5)+M2]=-0.000000000003;
a[iii(3,0)+M2]=-0.009973763085;a[iii(3,1)+M2]=0.000000000000;
a[iii(3,2)+M2]=0.000133430886;a[iii(3,3)+M2]=-0.000000000000;
a[iii(3,4)+M2]=0.000095971082;a[iii(3,5)+M2]=-0.000000000000;
a[iii(4,0)+M2]=0.027928309523;a[iii(4,1)+M2]=0.000000000002;
a[iii(4,2)+M2]=-0.001242305766;a[iii(4,3)+M2]=0.000000000001;
a[iii(4,4)+M2]=-0.000549464210;a[iii(4,5)]=0.00000;
a[iii(5,0)+M2]=-0.004360599564;a[iii(5,1)+M2]=-0.000000000001;
a[iii(5,2)+M2]=0.000257546933;a[iii(5,3)+M2]=-0.000000000000;
a[iii(5,4)]=0.00000;a[iii(5,5)]=0.00000;

//fscan_uv(1);  // Scan in U from file
//set_a(); // Set coefficients.

}
```