SOLVING SYSTEMS OF NONLINEAR COUPLED PDE'S USING NEWTON'S METHOD

LIZ URIBE AND JEANNIE WALLDREN

ABSTRACT. We are seeking solutions $u \in \mathbb{R}^n$ and $v \in \mathbb{R}^n$ to the system of coupled superlinear partial difference equations $-Lu+su+u^3+\beta v^2 u = 0$ and $-Lv+sv+v^3+\beta u^2 v = 0$, where L is the matrix corresponding to the Laplacian operator on a graph with Neumann boundary conditions.

1. INTRODUCTION.

intro This paper considers *nonlinear coupled partial difference equations* (PdE's) with Neumann boundary conditions. In particular, we use bifurcation branch following algorithms required for finding solutions $u \in \mathbb{R}^n$ and $v \in \mathbb{R}^n$ to the discrete nonlinear system

pde (1)
$$\begin{aligned} -Lu + su + u^3 + \beta v^2 u = 0 \\ -Lv + sv + v^3 + \beta u^2 v = 0 \end{aligned} ,$$

which bifurcate from branches connected to the trivial branch $u = v = \sqrt{\frac{-s}{1+\beta}} w$, where w is a vector of ones. We began our research with a primary goal of finding nontrivial solutions at $\beta = -1$, where there is an asymptote for this branch.

Our research involved writing a MATLAB program, see for Section 10 an edited version, to solve this system. We began by considering an uncoupled scalar system of Equation 1 where $\beta = 0$. This particular problem had already been solved by [1] so it was a good place to start so that we could verify that our code was outputing accurate results. After our scalar code consistently calculated correct results, we began to expand the program to handle the coupled Equation 1. To verify some results of our coupled equation, we used Mathematica to calculate solutions, as seen in Section 6.

Our methods are similar to those described by John M. Neuberger, James W. Swift, and Nándor Sieben in their preprint article titled *Automated Bifurcation Analysis for Nonlinear Elliptic Partial Difference Equations on Graphs*, 2007. Where they explain two modified translations of the Gradient Newton-Galerkin Algorithm (GNGA, as defined in [2]) in order to completely automate bifurcation branch following. As a result, we can find most solutions to the discrete nonlinear system.

2000 *Mathematics Subject Classification.* 20C35, 35P10, 65N25.

Key words and phrases. Bifurcation, PdE, GNGA, Neumann, Coupled.

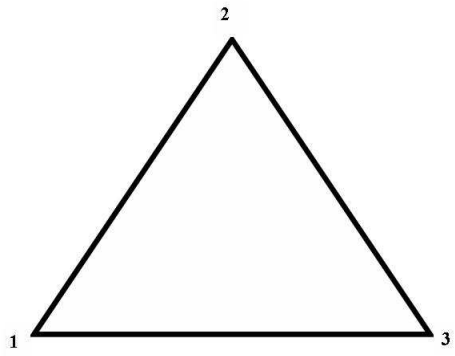

FIGURE 1. C_3 Graph C3

$$L = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

FIGURE 2. Graph Laplacian for C_3 graph_laplacian

2. Finding the Graph Laplacian with Neumann Boundary Conditions

Laplacian The graph Laplacian, L, of the PdE is used in solving discrete problems. It approximates a continuous function, where the Laplacian operator, Δ, is used for a partial differential equation.

For any given graph, we can build a graph Laplacian matrix, L for a simple connected graph. L will be a matrix of dimension nxn where n is the number of vertices of the given graph. First, in each of the diagonal entries, $L_{i,i}$, input the degree of the corresponding vertex, i. For all other entries, $L_{i,j}$, if i is adjacent to j, input a value of -1. Otherwise, if they are not adjacent, enter 0. Since the sum of all entries in a row will be 0, we know that for any constant eigenvector of the Laplacian, ψ, $L\psi = 0$. Thus, using this method, L will always have Neumann boundary conditions.

For example, consider the cyclic graph with 3 vertices, C_3 in Figure 1. Since each vertex has degree 2, we place a 2 in all of the entries along the diagonal of L, as seen in Figure 2. Notice that vertex one is adjacent to vertex two, so we place a -1 in $L_{1,2}$ and in $L_{2,1}$. This method is continued until all adjacent vertices are given a value of -1. Any remaining unfilled entries in the matrix are given the value of zero, in our example the graph also happens to be complete, and therefore the correpsonding graph Laplacian has no zeros.

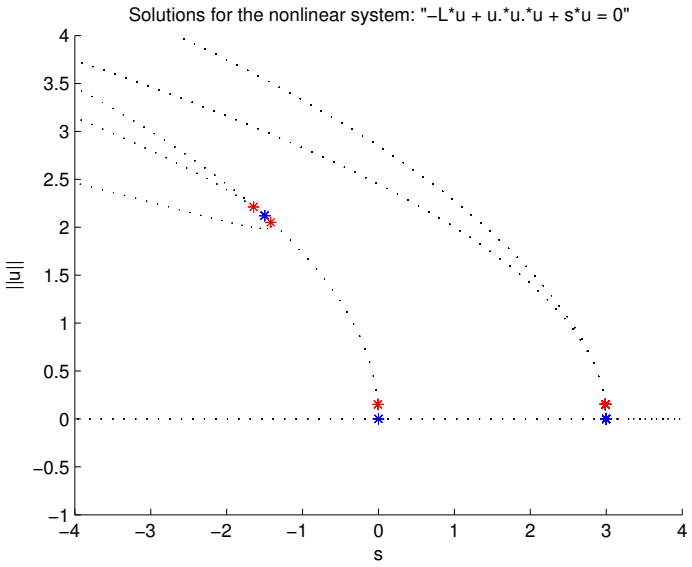

FIGURE 3. Bifurcation Diagram. Depicts solutions of the scalar system, using $f_s(u) = su + u^3$. Stars are bifurcation or cylinder points. fig:scalar_plot

3. Defining the Systems of Interest

scalar_system

3.1. **Defining the Scalar System.** We begin our research by reproducing results of an uncoulpled scalar system. Consider

$$N(u) = -Lu + f_s(u)$$

where L is the graph Laplacian, u is a vector in $\mathbb{R}^n$, and $f_s(u)$ is superlinear, i.e. a nonlinear function to some power greater than 1. As previously mentioned in Section 1, we began by trying to solve $N(u) = 0$ on C_3 with as a s variable. In all of our simulations $f_s(u) = su + u^3$, where $u^3 := \begin{bmatrix} u_1^3 \\ \vdots \\ u_n^3 \end{bmatrix}$ and $s \in \mathbb{R}$ for both scalar and coupled systems.

Our solutions using this scalar system on the C_3 graph result in the bifurcation diagram seen in Figure 3. Definitions of bifurcation and cylinder points will be given in Sections 5.3 and 5.5.

Notice the branch curving out to the left of the trivial branch from $\beta = 0$ in Figure 3. This is a significant set of solutions because it can be easily written in closed form. Suppose u is constant, i.e. all of its components are equal. Then,

$$-Lu = 0 \text{ since } u \text{ is constant},$$
$$su_i + u_i^3 = 0,$$
$$u_i^3 = -su_i,$$

$$u_i^2 = -s.$$

Therefore, $u_i = \sqrt{-s}$ for $i \in \{1, 2, ..., n\}$.

coupled_system

3.2. **Defining the Coupled System.** The bulk of our research was spent studying a coupled system of equations that can be defined as follows: Let

$$N(U) = -LU + \nabla W(U)$$

where $U := \begin{bmatrix} u \\ v \end{bmatrix}$, and $LU := \begin{bmatrix} Lu \\ Lv \end{bmatrix}$ and N(U) equals Equation 1. $W(U)$ will be defined and looked at in more detail in Section 4 To find solutions to this system we set $N(U) = 0$. For this coupled system, $u \in \mathbb{R}^n$ and $v \in \mathbb{R}^n$ are vectors, $s \in \mathbb{R}$ is fixed, and $\beta \in \mathbb{R}$ is a variable, where $v^2 u := \begin{bmatrix} v_1^2 u_1 \\ \vdots \\ v_n^2 u_n \end{bmatrix}$. We found a branch of trivial solutions for the coupled system where u and v are constant and $u_i = v_i$ using the solution branch found in Subsection 3.1 where u was constant such that $u_i = \sqrt{-s}$ for $i \in \{1, 2, ..., n\}$. Assume u and v are constant. We will consider the first half of the system, $-Lu + su + u^3 + \beta v^2 u = 0$, and the same will be true for the second part by symmetry.

$$-Lu = 0 \text{ since } u \text{ is constant},$$
$$su + u^3 + \beta v^2 u = 0,$$
$$su + u^3 + \beta u^3 = 0 \text{ since } u = v,$$
$$su + (1 + \beta)u^3 = 0,$$
$$u^2 = \frac{-s}{(1 + \beta)},$$

Therefore, $u_i = v_i = \sqrt{\frac{-s}{1+\beta}}$ for $i \in \{1, 2, ..., n\}$.

4. Demonstrating the Equivilance of the Hessian and the Negative Jacobian

Jacobian

In [1] the aforementioned authors use a Hessian matrix in their algorithms for branch following. In their article, the Hessian is the Jacobian of a gradient of the functional $J(U)$, where $J(U) := -LU \cdot U - W(U)$, and $W(U)$ is the anti-derivative of $\nabla W(U)$. A Jacobian is a matrix of partial differential equations of one or more functions. For example, given functions with parameters x and y, $f(x, y)$ and $g(x, y)$, the Jacobian is written as $Jac \begin{bmatrix} f(x,y) \\ g(x,y) \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{bmatrix}$. In our research, we used the negative Jacobian, denoted $-Jac(N(U))$, in place of the Hessian, denoted $\nabla^2 J(U)$. We will show that this is legitimate.

For our coupled system we used $Jac(N(U))$. To build this we thought of $N(U)$ as a matrix of equations $N(u,v)$ and $N(v,u)$, as seen in Equation 1. So our Jacobian looks like the following:

$$\begin{bmatrix} \frac{\partial N(u,v)}{\partial u} & \frac{\partial N(u,v)}{\partial v} \\ \frac{\partial N(v,u)}{\partial u} & \frac{\partial N(v,u)}{\partial v} \end{bmatrix} = \begin{bmatrix} -L+3u^2+s+\beta v^2 & \text{diag}(2\beta vu) \\ diag(2\beta vu) & -L+3v^2+s+\beta u^2 \end{bmatrix}.$$

The following is a simple proof of why $\nabla^2 J(U)$ and $-Jac(N(U))$ are interchangeable. Recall,

$$N(U) = -LU + \nabla W(U) = 0$$

where, $W : \mathbb{R}^n \to \mathbb{R}$.

$$\nabla W(U) = \begin{bmatrix} su+u^3+\beta v^2 u \\ sv+v^3+\beta u^2 v \end{bmatrix} = \begin{bmatrix} \frac{\partial W}{\partial u} \\ \frac{\partial W}{\partial v} \end{bmatrix}$$

$$W(U) = \begin{bmatrix} \frac{1}{2}su^2+\frac{1}{4}u^4+\frac{\beta}{2}v^2u^2+h(v) \\ \frac{1}{2}sv^2+\frac{1}{4}v^4+\frac{\beta}{2}v^2u^2+h(u) \end{bmatrix}$$

$$h(v) = \frac{1}{2}sv^2+\frac{1}{4}v^4 \text{ and } h(u) = \frac{1}{2}su^2+\frac{1}{4}u^4$$

$$W(u,v) = \frac{s}{2}(u^2+v^2)+\frac{1}{4}(u^4+v^4)+\frac{\beta}{2}v^2u^2$$

The functional, $J(U) = \frac{1}{2}LU \cdot U - W(U)$

$$\begin{aligned} J'(U)(V) &= LU \cdot V - \nabla W(U) \cdot V \\ &= -(-LU + \nabla W(U)) \cdot V \\ &= -N(U) \cdot V \\ &= \nabla J(U) \cdot V \text{ by definition of directional derivative.} \end{aligned}$$

Thus, $\nabla J(U) = -N(U)$.
Therefore, $\nabla^2 J(U) = -Jac(N(U))$.

5. Applying Newton's Method and its Variations

Newton

5.1. **Newton's Method.** Newton's method is used to find solutions to functions set equal to zero. Its scalar form is

$$a_{k+1} = a_k - \frac{f(a_k)}{f'(a_k)} \text{ for } k \in \mathbb{N}.$$

It takes in an initial guess, a_0, and calculates a new value, a_1. This is one iteration. The next iteration sets $a_2 = a_1 - \frac{f(a_1)}{f'(a_1)}$. This process is repeated until convergence, i.e. until $\frac{f(a_k)}{f'(a_k)}$ is close enough to 0. For a good initial guess, this method is extremely acurate, however if a poor initial guess is made, it may not converge or may converge to an unwanted value.

For our purposes, we used the form

$$U_{k+1} = U_k - Jac(N(U_k))^{-1}N(U_k) \text{ for } U_k = \begin{bmatrix} u_k \\ v_k \end{bmatrix}.$$

Since the Jacobian is not always invertible, in our MATLAB program we used the psuedo inverse command.

tGNGA 5.2. **Tangent Method (tGNGA).** The Tangent Gradient Newton Galerkin Algorithm (tGNGA) was developed in [1]. It is a variation of Newton's method where, given two known solutions on the same branch, (U_{k-1}, β_{k-1}) and (U_k, β_k), we can find a subsequent solution on the same branch. To obtain a good initial guess, we find a direction vector, d, by using the known solutions such that $d = \frac{(u_{k-1}-u_k, \beta_{k-1}-\beta_k)}{||(u_{k-1}-u_k, \beta_{k-1}-\beta_k)||}$. We then find a value for $\chi = \begin{bmatrix} \chi_U \\ \chi_\beta \end{bmatrix}$, which acts like the term $Jac(N(U_k))^{-1}N(U_k)$ from Section 5.1. We find χ by solving the system

$$\begin{bmatrix} -Jac(N(U)) & -\frac{\partial N}{\partial \beta} \\ d_U & d_\beta \end{bmatrix} \begin{bmatrix} \chi_U \\ \chi_\beta \end{bmatrix} = \begin{bmatrix} -N(U) \\ 0 \end{bmatrix}.$$

Once these values are found, we use the following, updating χ with each iteration, until χ approachs 0.

$$\begin{bmatrix} U_{k+1} \\ \beta_{k+1} \end{bmatrix} = \begin{bmatrix} U_k \\ \beta_k \end{bmatrix} - \begin{bmatrix} \chi_U \\ \chi_\beta \end{bmatrix}$$

The tangent method is used with Newton's method to follow solutions branches in our bifurcation diagrams.

MI 5.3. **Morse Index and Bifurcation Points.** As we follow our solution branches with the tangent method, we keep a record of the Morse index, $MI(U)$, of the last two solutions. Suppose U is a solution to $N(U) = 0$ and is non-degenerate, i.e. with no zero eigenvalues of the negative Jacobian. Then $MI(U)$ is the number of negative eigenvalues of the negative Jacobian. For two consecutive solutions, our code compares the stored Morse indices of each point. If they are not equal, there may exist a bifurcation point between those two solutions. A bifurcation point indicates the start of a new branch of solutions. At these points there will be at least one zero eigenvalue of the negative Jacobian. The difference of the Morse indices of the two points will indicate how many eigenvalues are zero.

secant 5.4. **Secant Method.** When a change of Morse index along a branch is detected, the secant method is used to find the bifurcation point. It is similar to Newton's method in that we repeat steps of the algorithm until convergence, but the secant is not a variation of Newton's method. We begin by looking at the eigenvalues of the negative Jacobian of each of the solutions. These eigenvalues are arranged in ascending order and assigned an index. Since the Morse index changed, we know that the number of negative eigenvalues differ by at least one. For example, suppose that β_k has MI of 3 and β_{k-1} has MI of 1 as shown in Table 1. These values will be the first two used by the secant method.

	Signs	of	Eigenvalue		
β_1	-	-	-	+	...
β_0	-	+	+	+	...
index	1	2	3	4	...

eigenvalues TABLE 1. Example of how to choose an index for the secant method.

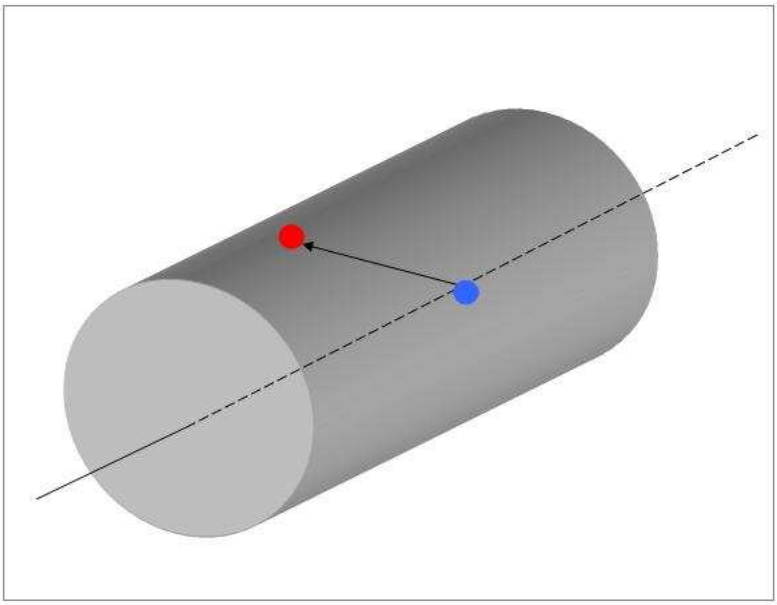

FIGURE 4. Representation of theoretical cylinder. The point in the center is a bifurcation point. The other point is the initial guess. cylinder

Notice that there is a change of sign at index 2. In this example, we would then store that index and the corresponding eigenvalues, denoted b_0 and b_1. This comparison is only done for the first iteration. These eigenvalues and β's are used in the secant method in the following way:

$$\beta_{k+1} = \beta_k - b_k(\frac{\beta_k - \beta_{k-1}}{b_k - b_{k-1}})$$

In subsequent iterations, b_k and b_{k-1} are updated, however the index remains the same. The secant method converges when β_{k+1} is close enough to β_k.

5.5. **Cylinder Method (cGNGA).** The Cylinder Gradient Newton Galerkin Algorithm, (cGNGA), was also devolped in [1]

Again, it is a variation of Newton's method. Once the secant method converges to a bifurcation point, the cylinder method attempts to find a

point on a new branch, which we call a cylinder point. There are several steps in the implementation of cGNGA. It involves theoretically building a cylinder of radius δ centered at the bifurcation point, U_{bif}, and using a point on the cylinder, U_0, as the initial guess, see Figure 4. We find a random point on the cylinder by first examining the critical eigenspace, E, of the bifurcation point where, E = the eigenvectors corresponding to any zero eigenvalues in the negative Jacobian. We construct a unit vector, d, by taking a random linear combination of the vectors in E and normalizing. Then calculate $U_0 = U_{bif} - \delta d$ and $\beta_0 = \beta_{bif}$ as the initial values to begin cGNGA. The algorithm is given below:

$\begin{bmatrix} U_{k+1} \\ \beta_{k+1} \end{bmatrix} = \begin{bmatrix} U_k \\ \beta_k \end{bmatrix} - \begin{bmatrix} \chi_U \\ \chi_\beta \end{bmatrix}$ Where $\begin{bmatrix} \chi_U \\ \chi_\beta \end{bmatrix}$ is updated each iteration by solving:

$$\begin{bmatrix} -Jac(N(U)) & -\frac{\partial N}{\partial \beta} \\ P_E U & 0 \end{bmatrix} \begin{bmatrix} \chi_U \\ \chi_\beta \end{bmatrix} = \begin{bmatrix} -N(U) \\ \kappa \end{bmatrix},$$

where $\kappa = \frac{1}{2}(||P_E U||^2 - \delta^2)$.

6. Method for Verifying Bifurcation Points

mathematica

When we began finding bifurcation points on the trivial branches, we wanted to convince ourselves that these were the only points that had zero eigenvalues of their negative Jacobians. We used Mathematica to find the zeros of the determinant of the negative Jacobian. Recall

$$Jac(N(U)) = \begin{bmatrix} -L + 3u^2 - 1 + \beta v^2 & \text{diag}(2\beta vu) \\ diag(2\beta vu) & -L + 3v^2 - 1 + \beta u^2 \end{bmatrix}$$

for $s = -1$. Then for $u = v = \frac{1}{\sqrt{1+\beta}}$, we know $3u^2 - 1 + \beta v^2 = \frac{2}{1+\beta}$. So, our negative Jacobian is written as

$$-Jac(N(U)) = \begin{bmatrix} L - \text{diag}(\frac{2}{1+\beta}) & \text{diag}(\frac{-2\beta}{1+\beta}) \\ \text{diag}(\frac{-2\beta}{1+\beta}) & L - \text{diag}(\frac{2}{1+\beta}) \end{bmatrix}$$

By writing the negative Jacobian in this way, the determinant is a function of β. Zeros of this function should correspond to bifurcation points. For example, consider the C_3 graph, we will factor out $\frac{1}{\sqrt{1+\beta}}$ from the negative Jacobian, and set it equal to the following instead:

$$-Jac(N(U)) = \begin{bmatrix} 2\beta & -1-\beta & -1-\beta & -2\beta & 0 & 0 \\ -1-\beta & 2\beta & -1-\beta & 0 & -2\beta & 0 \\ -1-\beta & -1-\beta & 2\beta & 0 & 0 & -2\beta \\ -2\beta & 0 & 0 & 2\beta & -1-\beta & -1-\beta \\ 0 & -2\beta & 0 & -1-\beta & 2\beta & -1-\beta \\ 0 & 0 & -2\beta & -1-\beta & -1-\beta & 2\beta \end{bmatrix}.$$

$$\begin{aligned} det(-Jac(N(U))) &= 4 + 48\beta + 180\beta^2 + 192\beta^3 - 84\beta^4 - 240\beta^5 - 100\beta^6 \\ &= (\beta+1)^3(\beta+0.2)^2(\beta-1). \end{aligned}$$

If we set this equal to zero and solve for β, we find that the bifurcation points should appear at the following values of β : -1, -0.2, and 1. In Figure 5, we find bifurcation points at $\beta = -0.2$ and 1, but at -1 there is an asymptote for that branch. This may be because we factored $\frac{1}{\sqrt{1+\beta}}$ out of the Jacobian before we took the determinant.

7. Review of Numerical Solutions to the PdE on the C_3 Graph

bif_diags

In this section we will review results from our coupled MATLAB programs. In each of the examples we set $s = -1$. We follow three trivial branches of solutions to search for bifurcations, as seen in Figure 5. The first is the curved branch which has an asymptote at $\beta = -1$. This was found as described in Section 3.2 where u and v are constant and $u_i = v_i = \frac{1}{\sqrt{1+\beta}}$. The second trivial branch is simply the $u = v = 0$ branch. It is the horizontal branch on the bottom of the diagram. The third trivial branch is also a horizontal line at $||U|| = \sqrt{3}$ where $u = \sqrt{-s} = 1$, $v = 0$ or vice versa. This figure also shows bifurcation branches off of the first trivial branch at $\beta = -0.2$ that cross $\beta = -1$. Recall that one of our initial goals of this research was to find solutions to this system at $\beta = -1$ because our main trivial branch has an asymptote at that point. We found these solutions for that vaule of β off of the bifurcation branches:

$$U = \begin{bmatrix} 0.4226 \\ 0.6667 \\ 1.5774 \\ 0.4226 \\ 0.6667 \\ 1.5774 \end{bmatrix} \text{ and } U = \begin{bmatrix} 0.4520 \\ 0.4520 \\ 1.6422 \\ 0.4975 \\ 1.3555 \\ 1.3555 \end{bmatrix}.$$

Also notice in this Figure that there is a branch at $\beta = 4$. We have been unable to duplicate this branch from this perspective. However when we change the y-axis in other experiments we have found other bifurcation branches at that point. As we analyzed this diagram, we came to the conclusion that the apparent intersection of the first and third trivial branches was not possible as there is no vector that has the properties of both of these branches.

Since we knew this intersection was false we chose to replot the diagram from multiple perspectives, keeping β as the x-axis and adjusting the manner in which we plotted our solutions on the y-axis. First we looked at $||u|| + ||v||$ as seen in Figure 6. Here it is obvious that the first and third trivial branches do not intersect at $\beta = 1$, but they still appear to intersect. Figure 7 is a diagram of the same system of solutions with $||u|| - ||v||$ as a y-axis. Here we see that the center horizontal line is an overlap of our first and second trivial branches. In this Figure at first glance, it appears that we may have found a fourth trivial branch at $-\sqrt{3}$, but quickly realized that in the first couple of diagrams the third trivial branch was actaully two overplapping branches. Recall that the third trivial branch was when $u = 1$ and $v = 0$ or vice versa, when we plotted $||U||$ or $||u|| + ||v||$ these two sets of solution have the same value. But when we plot $||u|| - ||v||$ these two sets of solutions have different values, so we get two branches. Also in Figure 7, notice the two

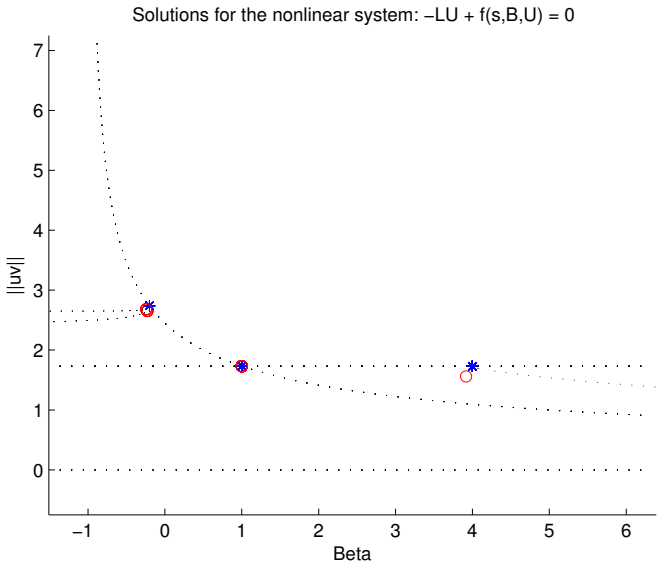

FIGURE 5. Example of coupled equation on C_3 plotting $(\beta, ||U||)$. C3_Unorm

floating bifurcation points. We believe that although these may be actual solutions to the system, since they are not on a branch, they are probably a result of bad initial guesses, which as we said is where Newton's method breaks down. In our attempt to avoid plotting any erroneous points on our diagrams, we modified our code to throw out any solutions that were not close enough to the previous solutions.

Figures 8 and 9 are 3-dimensional depictions of bifurcation diagrams of the solutions to the system on the C_3 graph. They represent different rotations of the same diagram. It is obvious from these two Figures that none of the trivial branches intersect at all. However, there may be a branch connecting three of them at $\beta = 1$. We believe this because of the arc of cylinder points, but we are not certain of this since the tangent method failed to follow the arc.

8. Review of Numerical Solutions to the PdE on the Higher Dimensional Graphs

weird

8.1. K_2, C_4, and Petersen Graphs. Our first attempt at adapting our code to work in higher dimesions was using the cyclic graph of 4 vertices, C_4. This generated a bifurcation diagram that baffled us for some time. The first diagram seemed to bifurcate at nearly every point along the curved trivial branch. When we looked at the numerical results, we saw that every point along this branch was degenerate. This prompted us to use the method described in Section 6 to see where we should have found bifuraction points, but it turned out that the determinate of the negative Jacobian for this

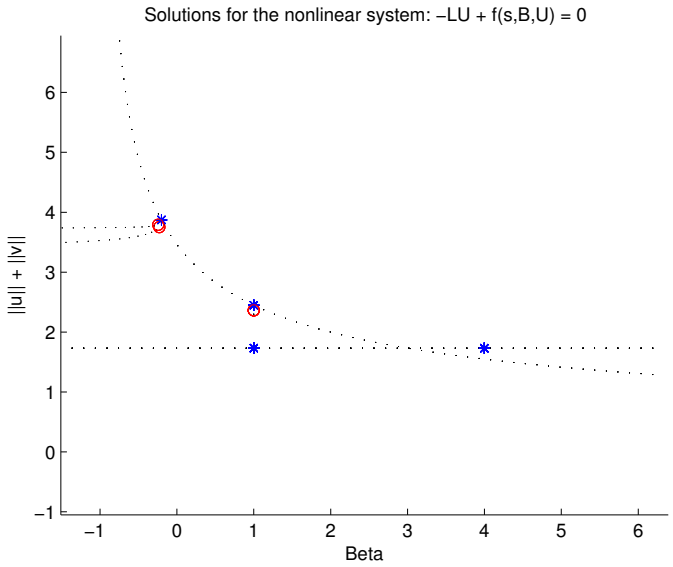

FIGURE 6. Example of coupled equation on C_3 plotting $(\beta, ||u|| + ||v||)$. C3_plus

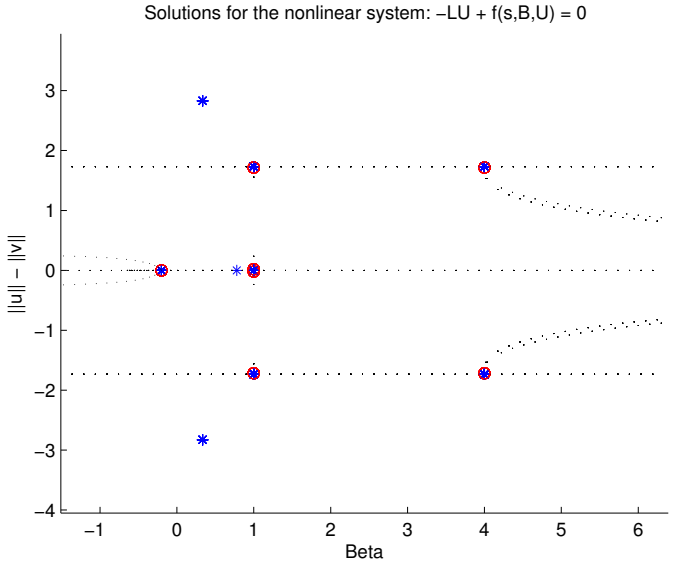

FIGURE 7. Example of coupled equation on C_3 plotting $(\beta, ||u|| - ||v||)$. C3_minus

graph was zero for all values of β along that branch. We attempted to re-analyze our code and steps to find out why this graph in particular had this peculiar property. We also noticed the same phenomenon with the complete

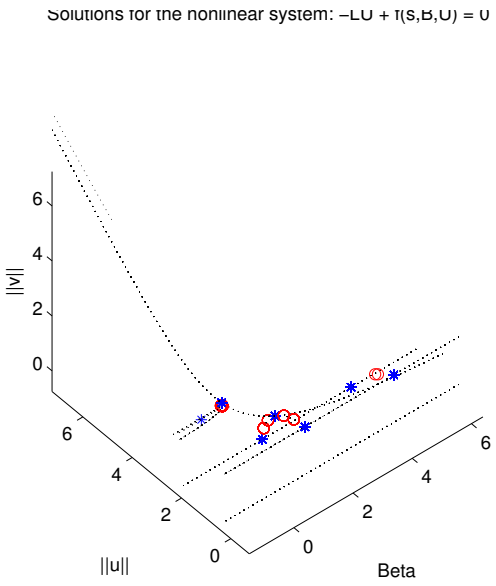

FIGURE 8. Example of coupled equation on C_3 plotting $(\beta, ||u||, ||v||)$. Original point of view C3_3D_1

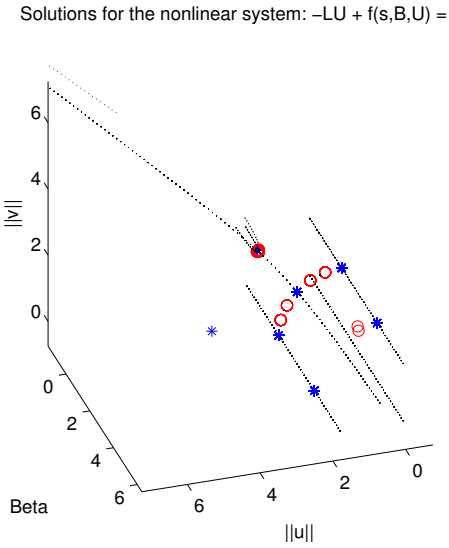

FIGURE 9. Example of coupled equation on C_3 plotting $(\beta, ||u||, ||v||)$. Rotated point of view. C3_3D_2

graph of two vertices, K_2 and the Petersen graph. We were presenting our findings to a group of colleagues when one of them pointed out that this

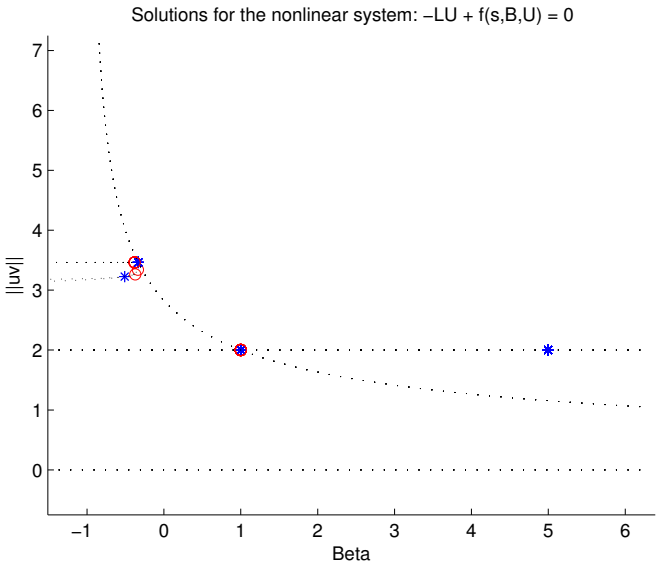

FIGURE 10. Example of coupled equation on K_4 plotting $(\beta, ||U||)$ K4

was happening because we were fixing $s = -1$ which is a bifurcation point on the trivial branch of the scalar solution of the graph.

K4 8.2. **Bifurcation Diagram Solutions on the K_4 Graph.** When we couldn't get a good graph for C_4, we tried the complete graph of 4 vertices to test our code in another eight dimensional space. Figure 8.2 shows the resulting bifurcation diagram. It is similar to the bifurcation diagram of C_3, which is also a complete graph. The most obvious difference between the diagrams is that the bifurcation points at $\beta = -0.2$ and 4 on Figure 5 shift to $\beta = -\frac{1}{3}$ and 5 on Figure 8.2 respectively.

C5 8.3. **Bifurcation Diagram Solutions on the C_5 Graph.** We included Figures 11 and 12 to demonstrate how different initial random guesses affect the outcome of our diagrams. Since our code generates different random guesses when it calls the cylinder method, it may not find the same solution branches each time the program is run. An interesting thing to notice in Figure 12 is that there appears to be a bifurcation branch from approximatly $\beta = -0.288$ that starts to approach -1 and curves back up to rejoin the original branch or follow it closely along the asymptote.

K5 8.4. **Bifurcation Diagram Solutions on the K_5 Graph.** The bifurcation points on the trivial branchs of K_5, seen in Figure 13, shift in the same way that the bifurcation points in Figure 8.2 shift with respect to Figure 5. Also, by observation, bifurcation plots of complete graphs are cleaner than those of cyclic graphs. That is, they do not have as many bifurcation points or branches.

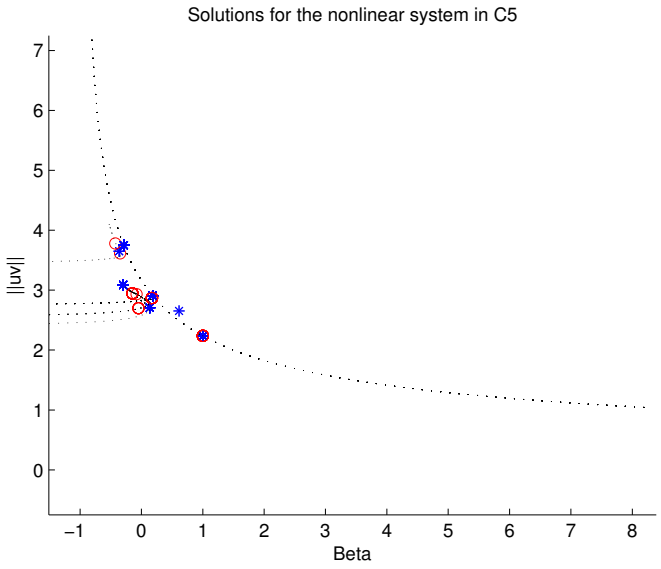

FIGURE 11. Example of coupled equation on C_5 plotting $(\beta, ||U||$ C5_1

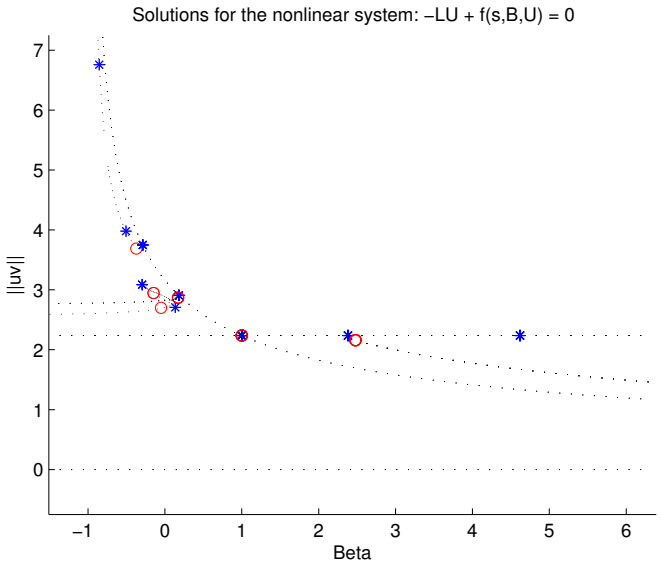

FIGURE 12. Second example of coupled equation on C_5 plotting $(\beta, ||U||)$ C5_2

C6 8.5. **Bifurcation Diagram Solutions on the C_6 Graph.** Figures 14 and 15 represent bifurcation diagrams of the C_6 graph. We originally plotted Figure 14 but chose to look at it from the perspective of Figure 15 when

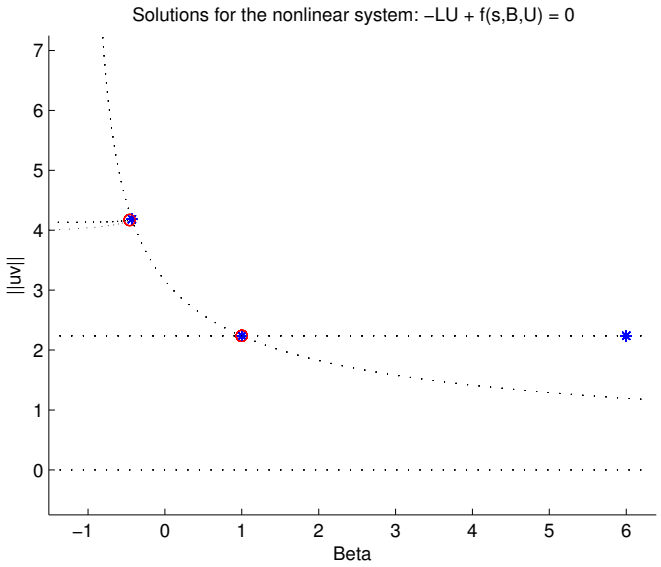

FIGURE 13. Example of coupled equation on K_5 plotting $(\beta, ||U||)$ K5

we noticed a dark bifurcation branch. We had suspected that this branch was a result of our code getting caught in an infinite loop. As it appears in Figure 15, there seems to be a branch that loops back to itself from that bifurcation point. We also see many more secondary bifurcattions than when we plotted $||U||$.

9. Conclusions and Further Research Possibilities

conclusion

In this section we will discuss furure possibilities for continued research using our findings.

There are several projects we consider to be appropriate continuations of our research using our code. One of our most puzzling findings was that we could not find cylinder points from bifurcation points that only had a single zero eigenvalue of the negative Jacobian. It is possible that adjusting the radius of the cylinder for these types of points may allow us to find a branch. To date, we have yet to fix this inconsistency. Another way to improve our program would be to automate it to make multiple random guesses in the cylinder method per run. As it is written, we only make a single random guess per run, and so we have to manually run the program several times to get a good bifurcation diagram with multiple bifurcation branches from a point. Also, our bifurcation diagrams can be make to look continuous. As it is written now, we plot each point individually as it is found in the tangent method and replace its information with the next point. A way to make the plot less disjointed would be to store all of the points that share a branch in an array and plot them at once. If we store this information in an output

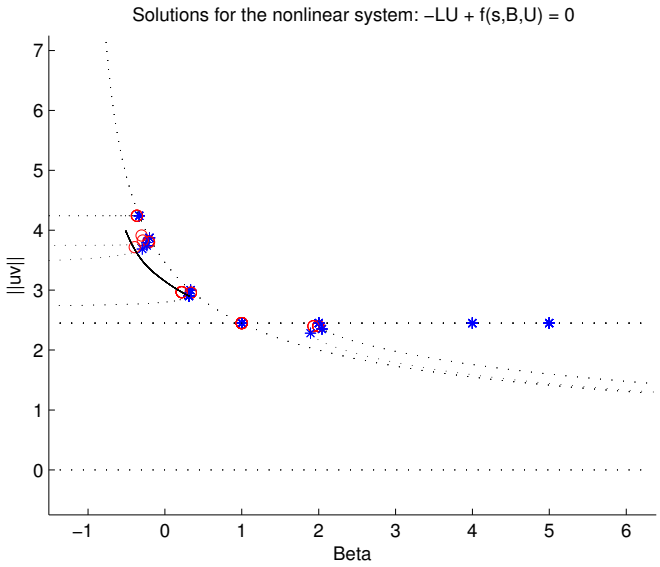

FIGURE 14. Example of coupled equation on K_5 plotting $(\beta, ||U||)$ C6_U

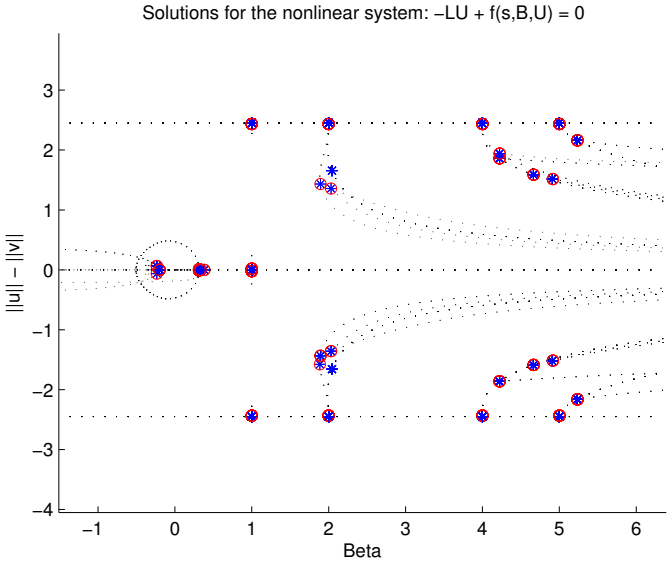

FIGURE 15. Example of coupled equation on K_5 plotting $(\beta, ||u|| - ||v||)$ C6_minus

file, we would also be able to replot the same data from several perspectives. Furthermore, we have yet to look at the same graphs with different values of s. Especially those whose trivial branches were degenerate at every point for

$s = -1$. This could be done in several ways. We could continue fixing s and varying β, looking at various values of s. We could vary s and fix β. Finally, we could vary both and create an animation that shows results as s changes or display the results in a 3-dimensional diagram where the branches would be surfaces. Lastly, it may prove interesting to look at these systems with other boundary conditions. We had begun this project with the intention of using Dirichlet boundary conditions. As of yet our code does not work with a matrix that mimics the Laplacian operator with these conditions.

While these are all interesting projects to us, it may be more worthwhile to consider modifying the suite of programs written by Drs. Neuberger, Sieben, and Swift to be able to solve the coupled system. A benefit of this would be that they have already debugged their code of some of the issues that our code still has. Their program already looks for multiple bifurcation branches with each run and the suite of programs they have developed is automated to create a graph Laplacian given any edge list. Also, they have included symmetry analysis which our program never considers.

As we wrap up our research, we find that in many cases the more questions we answer the more we have.

10. Appendix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% cpde()       (coupled pde final version)
%%  Created by Jeannie Walldren and Liz Uribe     May 31, 2007
%%  Last Edit:  July 9, 2007
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% main %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function main()
% Global variables defined
...
% Graph specifications
...
% Starts on u=v, uv = ones/zeros, uv = zeros/ones, and u=v=0 trivial branches
B1= B_max;
B2= B1 - speed;
uv1 = sqrt(1/(1+B_max))*ones(2*length(L),1);
uv2 = uv1;
uv1 = Newton(uv1, B1);
uv2 = Newton(uv2, B2);
queue = [uv1', B1, uv2', B2; %... uv = ones/zeros, uv = zeros/ones, and u=v=0 trivial branches omitted ];
dimension = size(queue);
row = dimension(1);
while (row > 0)
    queue = followbranch(queue);
    dimension = size(queue);
    row = dimension(1);
end
hold off;
%%followbranch(q)%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function q_new = followbranch(q)
global L;
global B_min;
global B_max;
global speed;
global uv_norm_max;
global uv_norm_min;
global tolerance;
dimen = size(q);
r = dimen(1);
uv_size = dimen(2)/2-1;
```

```
uv1 = q(r,1:(uv_size))';
B1 = q(r,(uv_size + 1));
uv2 = q(r,(uv_size + 2):(2*uv_size + 1))';
B2 = q(r,(2*uv_size + 2));
q = q(1:r-1,:);
count = 0;
while (B1 > B_min) && (B1 <= B_max) && (norm(uv1) <= uv_norm_max)&& (norm(uv1) > uv_norm_min)
    [uv_tan, B_tan] = tangent(uv1, B1, uv2, B2);
    [m2 D2] = MI(uv2,B2);
    [m3 D3] = MI(uv_tan,B_tan);
    if (abs(norm(uv_tan) - norm(uv2)) < 2*speed) && (abs(B_tan - B2) < 2*speed)
        plot(B_tan, (norm(uv_tan)), 'k')
     end
    if abs(B2-B_tan)<10^-5
        break;
    end
    if m2 ~= m3
        [uv_bif, B_bif] = secant(uv2, B2, uv_tan, B_tan);
        if (abs(norm(uv_bif) - norm(uv_tan)) < speed) && (abs(B_tan - B_bif) < 2*speed)
            [morse Evals] = MI(uv_bif, B_bif)
            plot(B_bif, (norm(uv_bif)), 'b*')
    [E colsE]= critEspace(uv_bif,B_bif);
            [branch_uv, branch_B] = cylinder(uv_bif, B_bif)
            if  (abs(norm(uv_bif) - norm(branch_uv)) < 2*speed) && (abs(branch_B - B_bif) < 2*speed)
                plot(branch_B, (norm(branch_uv)), 'ro')
                q = [uv_bif', B_bif, branch_uv' branch_B; q];
            end
        end
    end
    B1 = B2;
    B2 = B_tan;
    uv1 = uv2;
    uv2 = uv_tan;
    count = count +1;
end
q_new = q;
dimen = size(q);
r = dimen(1);
%%critEspace(uv,B)  Finds critical eigenspace of a known bifurcation point%%
function [E colsE] = critEspace(uv,B)
global L;
h = -Jac(uv,B);
[evect eval] = eig(full(h));
E = [];
for i = [1:(2*length(L))]
    if (abs(eval(i,i)) <= 10^-5)
        E = [E, evect(:,i)];
    else
        continue
    end
end
k = size(E);
colsE = k(2);
%%PEuv(colsE, E, uv) Calculates the projection of the eigenspace E onto vector uv
function Proj = PEuv(colsE,E,uv)
global L;
P = zeros(2*length(L),1);
for i = [1:colsE]
    P = P + (E(:,i)'*uv)*E(:,i);
end
Proj = P;
%%cylinder(uv, B)%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [new_uv, new_B] = cylinder(uv, B)
global L;
global speed;
global max_iterations;
global tolerance;
[E colsE]= critEspace(uv,B);
low = -1;
high = 1;
c = low + (high-low) * rand(colsE,1);
c = c/(norm(c));
```

```
d = zeros((2*length(L)),1);
for i = [1:colsE]
    d = d + E(:,i)*c(i);
end
delta = 3*speed;
B_guess = B;
uv_guess = uv + delta*d;
count = 0;
while (count == 0 ||((norm(n)>= tolerance) && (count < max_iterations)))
    n = N(uv_guess,B_guess);
    h = -Jac(uv_guess,B_guess);
    P = PEuv(colsE, E, uv_guess);
    gradK = [P' 0];
    K = .5*(((norm(P))^2) - delta^2);
    u = uv_guess(1:(length(L)));
    v = uv_guess((length(L)+1):(2*length(L)));
    dNdB = [u.*(v.*v); v.*(u.*u)];
    H = [h -dNdB; gradK];
    Cchi = pinv(H)*[-n; K];
    uv_guess = uv_guess - Cchi(1:2*length(L));
    B_guess = B_guess - Cchi((2*length(L)+1));
    count = count + 1;
end
new_uv = uv_guess;
new_B = B_guess;
%%tangent(uv1, B1, uv2, B2)%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [new_uv, new_B] =  tangent(uv1, B1, uv2, B2)
global L;
global s;
global speed;
global max_iterations;
global tolerance;
d = [(uv2-uv1); B2-B1];
d = d/(norm(d));
uv = uv2 + speed*d(1:2*length(L));
B = B2 + speed*d((2*length(L)+1));
count = 0;
while (count == 0 ||((norm(n)>= tolerance) && (count < max_iterations)))
    count = count + 1;
    n = N(uv,B);
    h = -Jac(uv,B);
    u = uv(1:(length(L)));
    v = uv((length(L)+1):(2*length(L)));
    dNdB = [u.*(v.*v); v.*(u.*u)];
    H = [h -dNdB; d'];
    Tchi = pinv(H)*[-n' 0]';
    new = ([uv; B] -  Tchi);
    uv = new(1:2*length(L));
    B = new((2*length(L)+1));

 end
new_B = B;
new_uv = uv;
%% beta(uv1, B1, uv2, B2) This function is called by the secant function %%
%% to find initial beta (eigen) values and their index in the matrix.    %%
function [b1, b2, k] = beta(uv1, B1, uv2, B2)
global L;
global s;
D1 = eig(full(-Jac(uv1, B1)));
D2 = eig(full(-Jac(uv2, B2)));
for i = [1:(2*length(L))]
    if abs(D1(i)) < 10^-5
        D1(i) = 0;
    end
    if abs(D2(i)) < 10^-5
        D2(i) = 0;
    end
    if ((D1(i) < 0 && D2(i) >= 0)) || ((D1(i) >= 0 && D2(i) < 0))
        b1 = D1(i);
        b2 = D2(i);
        k = i;
        break;
```

```
    else
        b1 = D1(i);
        b2 = D2(i);
        k = i;
        continue;
    end
end
%% secant(uv1, B1, uv2, B2) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [uv, B] = secant(uv1, B1, uv2, B2)
global L;
global s;
global max_iterations;
global tolerance;
[b1, b2, k] = beta(uv1, B1, uv2, B2);
count = 0;
while (count == 0 ||((abs(B1 - B2) >= tolerance) && (count < max_iterations)))
    count = count + 1;
    B3 = B2 - b2*(B2 - B1)/(b2 - b1);
    B1 = B2;
    B2 = B3;
    uv1 = uv2;
    uv2 = Newton(uv2, B3);
    [m D] = MI(uv2, B3);
    b1 = b2;
    b2 = D(k);
    error = abs(B2 - B1);
end
B = B3;
uv = uv2;

%...MI(uv, B) omitted

%% Newton(uv, B) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function uv_out = Newton(uv, B)
global L;
global s;
global max_iterations;
global tolerance;
count = 0;
n = N(uv, B);
J = Jac(uv, B);
chi = (pinv(J))*n;
while (count == 0 ||(((norm(n)) >= tolerance) && (count < max_iterations)))
    uv = uv - chi;
    n = N(uv, B);
    J = Jac(uv, B);
    chi = (pinv(J))*n;
    count = count + 1;
end
uv_out = uv;

%...Jac(uv, B) and N(uv, B) omitted
```

References

NSS3 1. John M. Neuberger, Nándor Sieben, and James W. Swift, *Automated bifucation analysis for nonlinear elliptic partial difference equations on graphs*, Not yet published (2007).

NS 2. John M. Neuberger and James W. Swift, *Newton's method and Morse index for semilinear elliptic PDEs*, Internat. J. Bifur. Chaos Appl. Sci. Engrg. **11** (2001), no. 3, 801–820.

E-mail address: luribe@email.arizona.edu jeanniewalldren@yahoo.com

Department of Mathematics and Statistics, Northern Arizona University PO Box 5717, Flagstaff, AZ 86011-5717, USA