Ilia Kheirkhah

ENM 5020

3/14/2023

Homework 2

Newtons Method with Analytic and Arc-Length Continuation for Solving Nonlinear Differential Equations

Contents

Introduction

So far, we have become very familiar with solving systems of linear equations. These are equations of the form $Ax = b$. When a problem is written in this form, we have many tools to find solutions. Although this is a very powerful tool, it is not sufficient to solve many problems we will face as they take the form of nonlinear systems of equations. These problems take the form $R(x) = 0$. Examples of nonlinear problems are solutions to quadratic, cubic, or other high order polynomials, problems involving logarithms or exponentials, etc. This essentially includes any problem that is not a linear problem.

Sometimes, we can solve these problems analytically. For example, recall the quadratic equation which can be used to solve any quadratic equation. However, in many cases, we will not be able to find an analytic solution. As such, we instead resort to iterative approaches. With iterative approaches, undergo the following process. We first take a guess that we believe to be close to the real solution. Call this prediction x^0. We then run this through some function $\phi(x)$. This give us x^1. We repeat this process until our change in consecutive results is below some tolerance. More generally, we can say that the process can be written as:

$$x^{k+1} = \phi(x^k) \qquad \textbf{(1)}$$

The major step in developing a good iterative method is picking a function $\phi(x)$ which, $\phi(x^*) = x^*$ where $R(x^*) = 0$. In this homework, we will explore Newton's Method for finding $\phi(x)$. We will then apply this method to finding solutions to the discretized form of the following differential equation.

$$\nabla^2 u + \lambda u(1 + u) = 0 \qquad \textbf{(2)}$$

On the domain

$$D = (0 \leq x \leq 1) \cup (0 \leq y \leq 1) \qquad \textbf{(3)}$$

With

$$u = 0 \text{ on all boundaries} \qquad \textbf{(4)}$$

Methods

In this section, we will explore in more detail our approach to discretizing our differential equation, our iterative method, and how we generate initial guesses. We will first review Finite Difference Method. Then we will explore how we will develop Newton's Method. At this point, we will begin our discussion on how our initial guesses will be generated. First, we will discuss the eigenvalue problem that our differential equation can be simplified to which allows us to find our first solution. From there, we can talk about Analytic Continuation, which allows us to obtain a decent initial guess based on the information from our first solution. Finally, we will explore Arc-Length Continuation and its influence on Newton's Method which will allow us to fully explore our differential equation in a way not possible with only Analytic Continuation or a simpler Newton's Method.

Discretization

Recall our differential equation.

$$\nabla^2 u + \lambda u(1+u) = \frac{d^2u}{dx^2} + \frac{d^2u}{dy^2} + \lambda u(1+u) = 0 \tag{5}$$

Solving this problem analytically would be a difficult problem. As a result, we wish to numerically solve this problem. To do this, we first need to leave the continuous domain and enter a discretized domain. To do this, we use Finite Difference Methods to divide the domain into a 30x30 grid. This gives us a total of 900 nodes which we need to solve for to find a complete solution. Finite Difference Method applies Taylor Series expansion to find discrete, approximate formulas for derivatives. This process was explored in detail in the previous assignment. As such, some details will be left out for the sake of conciseness. However, the following will highlight the important features of finite difference method.

First, let us define how we will number our nodes. Since we have a 30x30 grid, we need a way to translate this 2-D structure into a 900x1 vector. To do this, we will define the index in our vector l in relation to i and j, which are the x and y indices of our 30x30 grid respectively. If we let $N = 30$ be the number of nodes in one direction, we can say:

$$l = (j-1)N + i$$

Now that we have this numbering scheme, we can demonstrate how Finite Difference Methods apply to our system. Reviewing our Taylor Series, one can show that:

$$\frac{d^2u_l}{dx^2} = \frac{u_{l-1} - 2u_l + u_{l+1}}{h^2} \tag{6}$$

$$\frac{d^2u_l}{dy^2} = \frac{u_{l-N} - 2u_l + u_{l+N}}{h^2} \tag{7}$$

Recall that h is defined our the distance between two nodes.

$$h = \frac{1}{N-1} \tag{8}$$

Notice, our finite difference equations for derivatives cannot be evaluated at the boundaries as one of these indices would be out of our domain. This is where our boundary conditions come in to fill this void. Applying these equations to our differential equation, we can see our problem becomes finding solutions to the following equation for all nodes.

$$\frac{u_{l-1} - 2u_l + u_{l+1}}{h^2} + \frac{u_{l-N} - 2u_l + u_{l+N}}{h^2} + \lambda u_l(1+u_l) = 0 \tag{9}$$

This is a discretized version of our differential equation. This is much easier to solve than our continuous problem. However, the solution is still not trivial as we have a nonlinear system due to the u_l^2 that comes out of the third term. This is where iterative methods, and specifically Newton's Methods, come to help us solve this problem.

Newton’s Method

As with many of the methods we use in this class, Newton’s Method is a clever application of Taylor series to solve equations of the form $R(x) = 0$. To apply Newton’s Method, you first need to provide an initial guess, x_0. If we then do a Taylor Series expansion of $R(x)$ about x_0, we can say:

$$R(x) \approx R(x_0) + \left(\frac{dR}{dx}\right)_{x_0} (x - x_0) + O(dx^2) \tag{10}$$

If we want to find the solution x^*, we can plug this into the Taylor Series expansion to get an approximate solution.

$$0 = R(x_0) + \left(\frac{dR}{dx}\right)_{x_0} (x^* - x_0) + O(dx^2) \tag{11}$$

If we rearrange this, we can show:

$$\left(\frac{dR}{dx}\right)_{x_0} \delta x = -R(x_0) + O(dx^2) \tag{12}$$

In theory, if we could show that $O(dx^2)$ is actually zero, solving this equation would give us x^* direction. This would be the case for a linear system of equations. However, for a nonlinear system, these higher order terms are nonzero. Therefore, although applying this will generally get us closer to our solution, it will not get us exactly to our solution. Equation 12 is Newton’s Method. From your initial prediction, you calculate $-R(x_0)$ and $\left(\frac{dR}{dx}\right)_{x_0}$. Then, you solve this equation for δx and then you update your prediction by saying $x_1 = x_0 + \delta x$. You repeat this problem until δx falls below some tolerance. Currently, Equation 12 is written for specifically x_0. We can write this more generally in the below equation.

$$\left(\frac{dR}{dx}\right)_{x^k} \delta x = -R(x^k) \tag{13}$$

$$x^{k+1} = x^k + \delta x \tag{14}$$

For our problem, our x variable is actually u. Our R variable represents our differential equation function. Both of these are 900x1 vectors. This means that the derivative of R with respect to x, which is actually the derivative of R with respect to u a derivative of a vector taken with respect to another vector. This is the Jacobian of R, $J(x^k)$ which is a 900x900 matrix. This means that the above problem is a linear system of equations in the form $Ax = b$ which can be solved using the methods we are already familiar with. We then repeat this process until our answer has converged.

Up until now, we have been talking about converging to some tolerance in a very abstract way. To put a number to this concept, we will define convergence as when $\delta x \leq 10^{-6}$. This is a fairly common convergence criteria and allows for all the solutions we get to have error significantly less than one percent.

Initial Guess

To start our functions, we need to develop an initial guess. To do this, consider the eigenvalue problem:

$$\nabla^2 u + \lambda u = 0 \tag{15}$$

This is very similar to our problem but is missing the λu^2 term. However, for very solutions that have a small magnitude, this u^2 term is very small compared to the other terms. That means that, by solving this problem, we can get good initial guesses for our real problem. We know from PDE studies that the above differential equation has solutions of the form:

$$u = A\sin(n\pi x)\sin(m\pi y) \tag{16}$$

For this solution, we can then say our eigenvalues are defined as follows:

$$\lambda = (n^2 + m^2)\pi^2 \tag{17}$$

By finding solutions for this problem, we can then stray slightly from our exact eigenvalue to get a new solution for nonlinear differential equation by applying Newton's Method. For this assignment, I found that $A = -0.1$ and $\lambda_{guess} = \lambda + 0.3$ converged on nontrivial solutions for the λ values we were using. As such, this is what I went for as my initial guess for the 3 solution branches that we explored.

Analytic Continuation

At this point, we have developed a way to get a solution at one point when the solution is fairly small in magnitude. However, we would like to now expand our reach to λ values that do not necessarily result in small u values. Although our initial guess technique discussed before could, possibly, converge, we would fundamentally be guessing around to find the proper A value to get any nontrivial results. We would like to avoid guessing as much as possible, so this is where continuation schemes come to help find us an initial guess for a more rigorous method.

The idea with analytic continuation is, once again, to apply Taylor Series expansion to estimate where we expect the result to end up. However, instead of moving in the solution dimension, we will introduce a parameter which we would like to vary. In this problem, we want to vary λ as our main parameter. We have a solution for some λ_1 and want to find an initial guess for some other λ_2.

To introduce some syntax, we would still like to find the solution to $R(u) = 0$. However, we would like to introduce a parameter in this function, λ. We will write this in our function as $R(u;\lambda) = 0$. Doing a multidimensional Taylor Expansion, we can then say:

$$R(u_2;\lambda_2) = R(u_1;\lambda_1) + \frac{\partial R}{\partial u}\delta u + \frac{\partial R}{\partial \lambda}\delta\lambda \tag{18}$$

We know that $R(u_1;\lambda_1)$ is a converged solution and therefore is sufficiently close to 0 which we can call it zero. In addition, we would ideally land on a location where $R(u_2;\lambda_2)$ would be very close to a solution, as that is the entire idea of a continuation scheme. As such, we can say that this term will also be zero. This means, we can write Equation 18 as:

$$\frac{\partial R}{\partial u}\left(\frac{\delta u}{\delta \lambda}\right) = -\frac{\partial R}{\partial \lambda} \tag{19}$$

Notice that $\frac{\partial R}{\partial u}$ is once again our Jacobian matrix and that this is a linear system of equations that we can solve to get $\frac{\delta u}{\delta \lambda}$. Once we have this value, we can pick any reasonable value to move in the λ direction and we can get a reasonable prediction as to what u_2 should be. Of course, $\delta\lambda$ should be relatively small to ensure that our Taylor Series approximation does not result in too much error which will prevent our solution from converging. In this homework, I have chosen $\delta\lambda = 0.1$ which worked well for this assignment.

Once we have applied Analytic continuation, we have a good initial guess at a new λ value and can apply Newton's Method to get a second solution we will call u_1. We will call the solution we used to find this initial guess u_0. It seems that Analytic continuation allows us to find initial guesses for any λ value given enough time. However, Analytic continuation is not flawless. If one considers a graph of $||u||_2$ against λ, locations where the tangent line of the curve is vertical or nearly vertical corresponds to solutions where the Jacobian is singular or nearly singular. As we know from our previous studies, singular matrices result in unsolvable systems of equations and nearly singular matrices result in high error. As such, analytic continuation becomes extremely unreliable at these locations. It also happens that our differential equation has some of these locations, so we need another tool to allow us to deal with these difficult domains. This is where Arc-Length Continuation allows us to fully explore our domain.

Arc-Length Continuation

With Analytic Continuation, we would step along the λ direction. However, this meant that there were some locations where our Jacobian could be singular. Another major issue is that, when there are multiple solutions for a given λ, finding both values is difficult. Arc-Length continuation deals with this problem by, instead of moving in the direction of the slope, it moves in the direction of increasing arc length. This allows us to follow the curve and increase or decrease λ as necessary to go to our next point. Let us review how this is done mathematically.

Let us define the Arc Length as s. When we take steps to change s, we do so in steps of δs. Once we have taken this step, we want to know what approximate λ and u values we should guess to reach a good result. Notice that we are not keeping λ constant and changing u, we are changing both λ and u. This will require us to change our Newton's Method to account for this. We will discuss this in the next section, but for now take the process of converging for granted.

Recall that up to this point, we have two converged solutions u_0 and u_1 at λ_0 and λ_1. We want to find initial guesses u_2, λ_2 of the following form:

$$\lambda_2 = \lambda_1 + \delta s\left(\frac{d\lambda}{ds}\right) \tag{20}$$

$$u_2 = u_1 + \delta s\left(\frac{du}{ds}\right) \tag{21}$$

The problem here is finding the derivative we need. With another Taylor Series about solution 1, we can show that the following linear system can solve our problem.

$$\hat{J}\begin{pmatrix}\frac{du}{ds}\\ \frac{d\lambda}{ds}\end{pmatrix} = -\left(\frac{d\hat{R}}{ds}\right) \tag{22}$$

Notice that the $\hat{J}$ and $\hat{R}$ variables we have here are not the regular Jacobian and Residual, but instead the Augmented Jacobian and Augmented Residual.

$$\hat{R} = \begin{pmatrix}R\\ \eta\end{pmatrix} \tag{23}$$

$$\eta = |s_1 - s_0|^2 - \left||u(s_1) - u(s_0)|\right|^2 - |\lambda(s_1) - \lambda(s_0)|^2 \tag{24}$$

$$\hat{J} = \begin{pmatrix} J & \frac{\partial R}{\partial \lambda}\\ \frac{\partial \eta}{\partial u} & \frac{\partial \eta}{\partial \lambda}\end{pmatrix} \tag{25}$$

Notice that, in Equation 24, s is being used as a sort of index. s_0 is the arclength at solution 0 and s_1 is the arclength at solution 1. As such, by passing them in for the index of u and λ it is the same as calling u and λ as the associated solution. Although it is not particularly important to know, in detail which point needs to be our reference zero arclength point, it is important to know what direction is increasing arclength and what direction is decreasing arclength as it will influence how we can interpret the results.

Once we solve the linear system described in Equation 22, our all we need to apply equations 20 and 21 to get our initial guess for point 2. Once we have this, we need to apply Newton's Method to converge to a solution. However, we have changed our R vector to $\hat{R}$ this means that instead of using the regular Jacobian and Residual, Newton's Method, which in this case we will call the Augmented Newton's Method, will make use of the Augmented Residual and Jacobian. That is to say the iteration algorithm we will use is:

$$\lambda^{k+1} = \lambda^k + \delta\lambda \tag{26}$$

$$u^{k+1} = u^k + \delta u \tag{27}$$

$$\hat{J}\begin{pmatrix}\delta u\\ \delta\lambda\end{pmatrix} = -\hat{R} \tag{28}$$

Results

Now that we have all the tools we need, we can now find an initial guess for any integer values of n, m and use arc length continuation all the way up to our bounding λ values. Since we are limited to $\lambda \leq 60$, we will explore the branches were $n = 1, m = 1$ and $n = 1, m = 2$ and $n = 2, m = 1$. This gives us 3 different solution branches. Since we need to explore both the forward and backward directions along the curve, this results in us executing the same code 6 times. Along each run, we will also keep track of the converged values for u at each λ and save these to an array we return. From here, we can generate Figure 1. This shows us how the magnitude of u changes as a function of λ in addition to the various solution branches we can encounter.

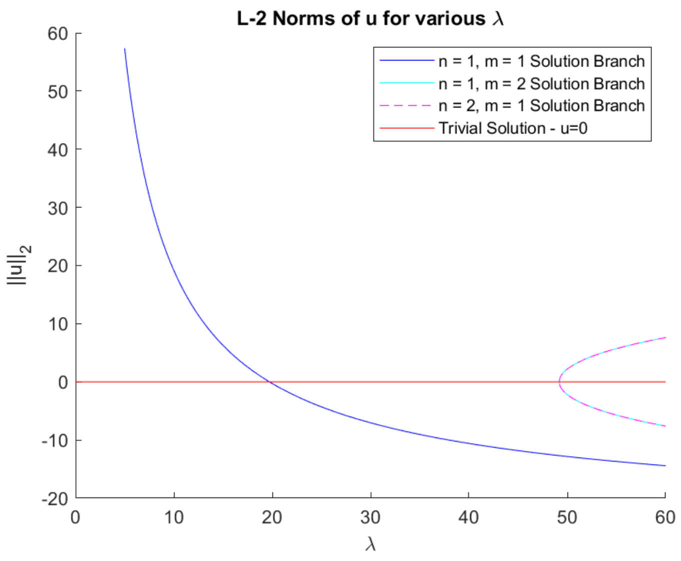

Figure 1: $\lambda - ||u||_2$ *Graph for each Solution Branch*

Notice here that although norms are normally always positive, we have negative norms on our graph. This is a result of us defining valley solutions as having negative norms while peak solutions have positive norms. Examples of these can be seen in Figures 2a-f which depict the various final solutions at the bounding λ values.

Although we were told to explore $0 \leq \lambda \leq 60$, we cannot actually reach all the way to $\lambda = 0$ as our magnitude tends towards infinity. This means that the graph becomes more steep which means that our arc length continuation takes longer and longer to run through the same distance in λ. As such, I chose to end at $\lambda = 5$ as it shows the trend without sacrificing too much time.

In addition, notice that the solution branches for $n = 1, m = 2$ and $n = 2, m = 1$ overlap with one another. This is a result of these two solution branches not being too distinct in shape. They are simply 90 degree rotations of one another, so we expect the norms to be identical. Now let us take a look at the contour plots

If we consider the looks of each graph as compared to the behavior we expect from Figure 1, We can see our first point of interest when comparing Figures 2a and 2b. Although 2a is significantly closer to $2\pi^2$, the magnitude is significantly higher than Figure 2b. This is matched by the slowing graph of the valley type solutions associated with this branch at higher λ values.

Here, it is important we discuss our convention for what we consider a positive and negative norm for these more interesting results. In Figures 2c through 2f, the valley portion of the graph is larger than the peak portion. As such, we cannot simply use that as our signifier. We instead define positive norms as those when the valley is closer to the origin. The idea behind this is that the graph is “increasing” going further away from the origin, so the norm is positive. However, this is clearly arbitrary but picked for the sake of communication.

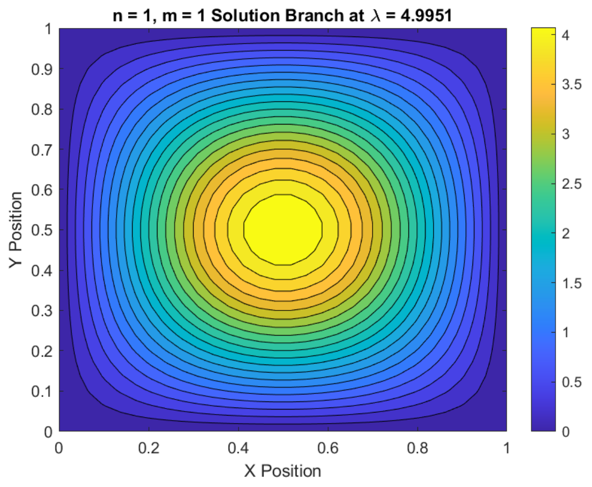

Figure 2a: n=1, m=1 Solution Branch at $\lambda \approx 5$. Peak Type Solution, Positive Norm

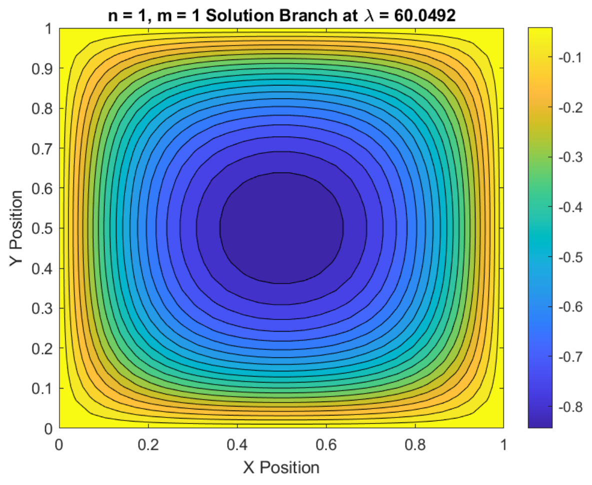

Figure 2b: n=1, m=1 solution branch at $\lambda \approx 60$. Valley Type Solution, Negative Norm

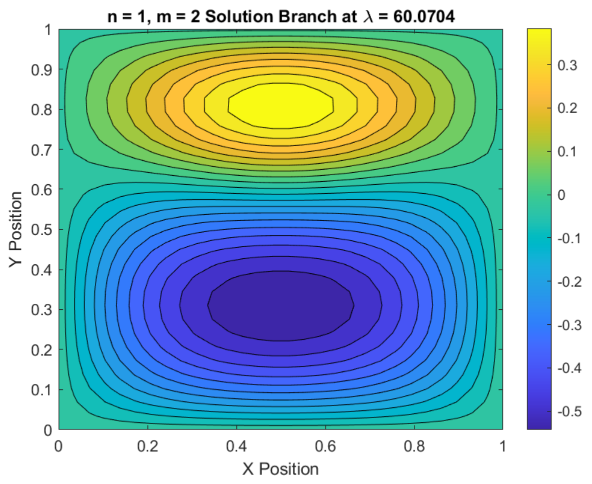

Figure 2c: n=1, m=2 Solution Branch at $\lambda \approx 60$. Positive Norm Solution (Valley Closer to Origin)

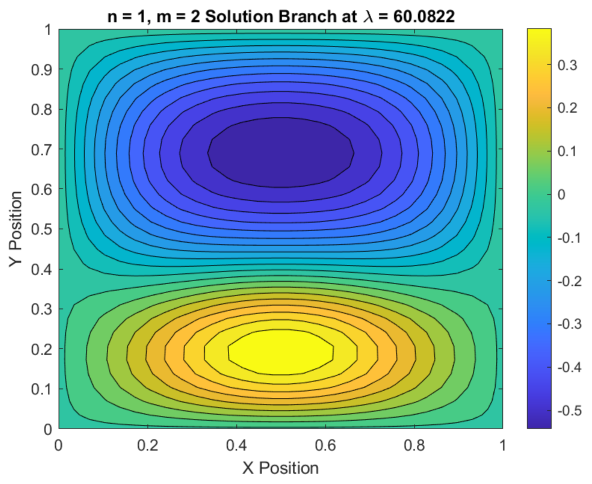

Figure 2d: n=1, m=2 Solution Branch at $\lambda \approx 60$. Negative Norm Solution (Valley Farther from Origin)

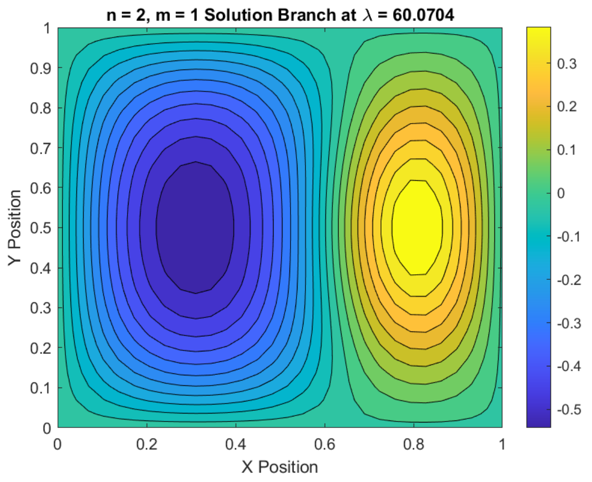

Figure 2e: n=2, m=1 Solution Branch at $\lambda \approx 60$. Positive Norm Solution (Valley Closer to Origin)

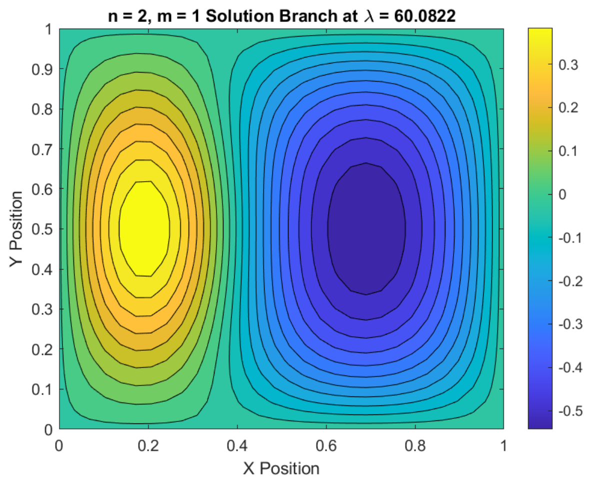

Figure 2f: n=2, m=1 Solution Branch at $\lambda \approx 60$. Negative Norm Solution (Valley Farther from Origin)

Now let us look at the second and third solution branches. As we discussed, the solutions from each solution branch are a 90 degree rotation between each one. This confirms why our norm graphs were overlapping for these branches. However, it is interesting to note that, in all of these solutions, the valley is noticeably bigger than the peak. I believe this is a result of us picking positive values for λ. If we were to explore similar branches at negative values of λ, I believe we would see the peak being larger than the valley. However, this would require significant alterations to our current analysis since our eigenvalue problem approach for finding an initial guess would not work. The eigenvalue problem only has positive eigenvalues. This means that we cannot use it to find negative values. This is barring us using imaginary numbers for n and m, which would result in a negative eigenvalue. I believe this points us to the solution for negative λ not being a product of sine functions, but a product of hyperbolic sine functions or, potentially a product of a hyperbolic sine with a regular sine for some specific eigenvalues. However, these are all theorizations. A more detailed exploration is warranted to understand this problem on a deeper level.

Conclusion

At this point, we have completed a detailed exploration of a small portion of the world of nonlinear equations. We applied Newton's Method to solve a Two-Dimensional Boundary Value problem of a discretized nonlinear equation. In the process, we also started to understand the behavior of the solution space through an analysis of how the solution norm behaves to various adjustments of our λ parameter in addition to what initial guesses we use. We used two continuation schemes – Analytic Continuation and Arc-Length Continuation – to allow us to move through the solution space somewhat freely to fully explore how each branch behaves.

Although we have done a fairly detailed exploration, there is still much to be explored. For example, one could ask the question of what happens when an eigenvalue occurs more than once when the n and m values are not reversed. For example, consider branch when $n = 5, m = 5$ and the branch when $n = 7, m = 1$. Both these branches have eigenvalues of $50\pi^2$ but we know that their graphs will not be rotations of one another due to their completely different n, m values. As such, it would be interesting to see the behavior of the graph at this location. At larger values, there will likely be more such occurrences that produce interesting results.

Next, one could explore the influence of introducing a third dimension to the problem. This would change our Jacobian Structure and require us to explore smaller grid points due to the N^3 number of points we would need to deal with. With $N = 30$, this would bring us to a 27,000 point simulation, far exceeding what MATLAB is capable of. This shows how quickly the difficulty of finding solutions increases when we introduce higher dimensions when we want real world results and the importance of well optimized code and progress in computing capabilities.

Finally, one should explore other nonlinear differential equations. In this problem, we could simplify our differential equation to an eigenvalue problem for small u to get an initial guess. However, problems such as the Navier-Stokes equation or Differential equations in Heat Transfer may not lend themselves to this same approach. The art of finding an initial guess purely from the continuous differential equation is a problem that must be tackled in a unique way for each situation. As such, exploration of different methods of developing initial guesses is likely the most fruitful next step in the study of nonlinear equations.

Regardless of what is to come, this assignment has shown us the power of Newton's Method when it comes to solving complex problems. Although it cannot give us an exact solution, we can pick the tolerance to which we need Newton's Method to converge to and it can, as long as we start within the radius of convergence of a solution, bring us arbitrarily close to the exact solution. This is a very powerful tool that we have developed.

Appendix

Main

```
clear; clc; close all;
% This is the main code that drives the operations necessary to find the
% contour plots and norms that we need for this assignment. It involves
% running through all three branches in both the forward and backward
% directions to get a full norm plot.

%inputs
N = 30;
lambdaMax = 60;
lambdaMin = 5;

% Run
normFig = figure;
hold on;
xlim([0 60]);
xlabel('\lambda');
ylabel('||u||_2');
title('L-2 Norms of u for various \lambda');
legend;
axNorms = gca; % Graph for Norms

% n = 1, m = 1 branch
[N11L,L11L] = runner(N,1,1,lambdaMin,lambdaMax,-1);
disp("Branch 1a Done");
exportgraphics(gcf,"11Contour1.png");
[N11R,L11R] = runner(N,1,1,lambdaMin,lambdaMax,1);
disp("Branch 1b Done");
exportgraphics(gcf,"11Contour2.png");
[~,ind]=min(N11L);
posNorms11 = N11L(ind+1:end);
posLambdas11 = L11L(ind+1:end);
negNorms11 = [N11L(1:ind); N11R];
negLambdas11 = [L11L(1:ind); L11R];

Norms11 = [flip(posNorms11); -negNorms11];
lambdas11 = [flip(posLambdas11); negLambdas11];
plot(axNorms,lambdas11,Norms11,'b-','DisplayName','n = 1, m = 1 Solution Branch');

 % n = 1, m = 2 branch
[N12L,L12L] = runner(N,7,1,lambdaMin,lambdaMax,-1);
disp("Branch 2a Done");
exportgraphics(gcf,"12Contour1.png");
[N12R,L12R] = runner(N,1,2,lambdaMin,lambdaMax,1);
disp("Branch 2b Done");
exportgraphics(gcf,"12Contour2.png");

[~,ind] = min(N12L);
posNorms12 = N12L(ind:end);
negNorms12 = [(N12L(1:ind-1)); N12R];

posLambdas12 = L12L(ind:end);
negLambdas12 = [(N12L(1:ind-1));N12R];

plot(axNorms,posLambdas12,posNorms12,'c-','DisplayName','n = 1, m = 2 Solution
Branch');
plot(axNorms,L12L,-N12L,'c-','HandleVisibility','off');

% n = 2, m = 1 branch
```

```
[N21L,L21L] = runner(N,2,1,lambdaMin,lambdaMax,-1);
disp("Branch 3a Done");
exportgraphics(gcf,"21Contour1.png");
[N21R,L21R] = runner(N,2,1,lambdaMin,lambdaMax,1);
disp("Branch 3b Done");
exportgraphics(gcf,"21Contour2.png");

[~,ind] = min(N21L);
posNorms21 = N21L(ind:end);
negNorms21 = [(N21L(1:ind-1)); N21R];

posLambdas21 = L21L(ind:end);
negLambdas21 = [(N21L(1:ind-1));N21R];

plot(axNorms,posLambdas21,posNorms21,'m--','DisplayName','n = 2, m = 1 Solution
Branch');
plot(axNorms,L21L,-N21L,'m--','HandleVisibility','off');

plot(axNorms,[0 60], [0 0],'r-','DisplayName','Trivial Solution - u=0');

exportgraphics(normFig,"NormPlot.png");
```

Runner

```
% This function produces an initial guess for a given n,m and
% converges with Newton's method.
% After that it applies analytic continuation to find a second guess and
% applies newton's method again to converge. After this, we loop through
% applying arc length continuation and the augmented Newton's method to
% converge on solutions until we reach either the maximum or minimum lambda
% value we define. The direction variable allows us to shift ds from
% positive to negative, allowing us to go in the forward and backward
% directions for a given solution branch.

function [norms,lambdas] = runner(N,n,m,lambdaMin,lambdaMax,direction)
%Do not alter
h = 1/(N-1);
x = 0:h:1;
y = 0:h:1;
[x,y] = meshgrid(x,y);


% Make initial guess
A = -0.1;
guess = A*sin(n*pi*x).*sin(m*pi*y);

l0 = (n^2+m^2)*pi^2+.3;
tol = 1e-6;

guess0 = zeros(N^2,1);
for i = 1:N
    for j = 1:N
        l = (j-1)*N+i;
        guess0(l) = guess(i,j);
    end
end


% Converge for u0
u0 = Newtons(l0,guess0,N,tol);
```

```
s0 = 0; % Definte as starting point
% Analytic Continuation for u1_0
dl = .1;
guess1 = analyticContinuation(u0,l0,N,dl);
l1 = l0 + dl;

% Converge for u1
u1 = Newtons(l1,guess1,N,tol);

% Find s1
summ = 0;
for i = 1:length(u1)
    summ = summ + (u1(i)-u0(i)).^2;
end
summ = summ + dl.^2;
ds = sqrt(summ);
s1 = s0+ds;

% Arc Length Continuation
tol = 1e-6;
ds = direction*.1;
[guess2,l2,s2] = arclengthContinuation(N,s1,s0,u1,u0,l1,l0,ds);

% Augmented Newtons Method to Converge
% disp(1);
[u2,~]= augmentedNewtons(N,s2,s1,l2,l1,guess2,u1,tol);

clear summ;
% Repeat until l = 60
norms = zeros(60,1);
lambdas = zeros(60,1);
count = 1;
while l2 < lambdaMax && l2 >lambdaMin
    u0 = u1;
    l0 = l1;
    s0 = s1;

    u1 = u2;
    l1 = l2;
    s1 = s2;
    [guess2,l2,s2] = arclengthContinuation(N,s1,s0,u1,u0,l1,l0,ds);
     disp(l2);


    [u2,l2] = augmentedNewtons(N,s2,s1,l2,l1,guess2,u1,tol);
    norms(count) = sqrt(sum(u2.^2));
    lambdas(count) = l2;
    count = count+1;
end

figure;
contourf(x,y,vecToArray(u2,N),20);
colorbar;
xlabel("X Position");
ylabel("Y Position");
title("n = "+num2str(n) + ", m = "+num2str(m)+" Solution Branch at " + ...
    "\lambda = "+num2str(l2));
end
```

Generating Jacobian and Residual

```
function [R,J,drdl] = genJR(u,lambda,N)
%This function takes in our current u and lambda values and returns the
%R vector and the associated Jacobian for an NxN case of the differential
%equation provided in Assignment 2
%   This is purely a helper function that makes the code slightly more
%   readable. It is used to generate our R vector and the Jacobian which is
%   the derivative of this vector. For testing purposes, I allowed for
%   variations in the dimension of the matrix.

if(length(u) ~= N^2)
    R = -1;
    J = -1;
    return;
end

h = 1/(N-1);
% Generate
R = zeros(N^2,1);
J = zeros(N^2);
drdl = zeros(N^2,1);

for i = 1:N
    for j = 1:N
        l = (j-1)*N+i;
        drdl(l) = u(l)*(1+u(l));
        if(i== 1 || j == 1 || i == N || j == N)
            J(l,l) = 1;
            R(l) = u(l);
        else
            R(l) = (u(l-1)-2*u(l)+u(l+1))/h^2 +...
                (u(l-N)-2*u(l)+u(l+N))/h^2 + ...
                lambda*u(l)*(1+u(l));

            J(l,l)=-4/h^2 + lambda*(1+2*u(l));
            J(l,l-1) = 1/h^2;
            J(l,l+1) = 1/h^2;
            J(l,l+N) = 1/h^2;
            J(l,l-N) = 1/h^2;
        end
    end
end

end
```

Analytic Continuation

```
% Applies analytic continuation to produce an initial guess some distance,
% dl, from our solution u.
function [u0,J,drdl] = analyticContinuation(u,lambda,N,dl)
[~,J,drdl] = genJR(u,lambda,N);
dudl = J\drdl;
du = dudl*dl;
u0 = u + du;
end
```

Newton’s Method

```
% This is an application of Newton's method to converge to a solution for a
% given initial guess u to some tolerance tol.
function [u,error,J] = Newtons(lambda,u,N,tol)

    error = 10*tol;
    e = 10 * tol;
    count = 1;
    [R,J] = genJR(u,lambda,N);
    while e > tol
        du = J\(-R);
        sum = 0;
        for i = 1:length(du)
            sum = sum + du(i).^2;
        end
        e = sqrt(sum);
        error(count) = e;
        count = count+1;
        u = u + du;
        [R,J] = genJR(u,lambda,N);
    end
end
```

Generating Augmented Jacobian and Augmented Residual

```
% This function is used mostly for ease of reading. It takes in the
% necessary information to generate our augmented Jacobian and augmented
% Residual.
function [R_hat,J_hat,dR_hat] = getAugmentedJRhat(N,ds,l1,l0,u1,u0)
J_hat = zeros(N^2+1);
R_hat = zeros(N^2+1,1);
dR_hat = zeros(N^2+1,1);

% Get old values
[R,J] = genJR(u1,l1,N);

% Fill in appropriately
J_hat(1:N^2,1:N^2) = J;
R_hat(1:N^2) = R;

% Fill dR_hat %%%%%%%%
dR_hat(end) = 2*(ds);

%Fill R_hat %%%%%%%%

sum = 0;
% Calculate 2-Norm of u1-u0
for i = 1:length(u1)
    sum = sum + (u1(i)-u0(i)).^2;
end

eta = (ds).^2-sum-(l1-l0).^2;
R_hat(end) =  eta;
% disp("eta="+num2str(eta));
% Fill J_hat %%%%%%%
detadu = zeros(1,N^2);
dRdl = zeros(N^2,1);

for i = 1:N
```

```
    for j = 1:N
        l = (j-1)*N+i;
        detadu (l) = -2*(u1(l)-u0(l));
        if(i== 1 || j == 1 || i == N || j == N)
            dRdl(l) = 0;
        else
            dRdl(l) = u1(l) * (1+u1(l));
        end
    end
end

detadl = -2*(l1-l0);
J_hat(1:N^2,end) = dRdl;
J_hat(end,1:N^2) = detadu;
J_hat(end,end) = detadl;
end
```

Arc Length Continuation

```
% This function applies Arc Length Continuation to produce an initial guess
% that allows us to follow the solution curves.
function [u2_0,l2,s2] = arclengthContinuation(N,s1,s0,u1,u0,l1,l0,ds)
    [~,Jhat,dRhat] = getAugmentedJRhat(N,(s1-s0),l1,l0,u1,u0);
    dRhat = -dRhat;
    dvec = Jhat\dRhat;
    duds = dvec(1:N^2);
    dlds = dvec(end);

    du = duds * ds;
    dl = dlds * ds;

    u2_0 = u1 + du;
    l2 = l1 + dl;
    s2 = s1+ds;
end
```

Augmented Newton’s Method

```
% This is an application of Newton's method that takes advantage of arc
% length continuation.
function [u,l,J,delta] = augmentedNewtons(N,s2,s1,l2,l1,u2,u1,tol)

    error(1) = 10*tol;
    error(2) = 100*tol;
    e = 10*tol;
    [Rhat,Jhat] = getAugmentedJRhat(N,(s2-s1),l2,l1,u2,u1);
    count = 1;
    while e > tol
        Rhat = - Rhat;
        delta = Jhat\Rhat;
        sums = 0;
        for i = 1:length(delta)
            sums = sums + delta(i).^2;
        end
        e = sqrt(sums);
        error(count) = e;
        %disp(e);
%           if(count > 5)
%               if(error(count) - error(count-1) < tol)
%                   u = delta;
```

```
%
%                   J = Jhat;
%                   return;
%               end
%           end
        du = delta(1:N^2);
        dl = delta(end);
%         e = sum(du.^2);
        u2 = u2 + du;
        l2 = l2 + dl;

        % find updated s2
        du = u2-u1;
        sums = 0;
        for i = 1:length(du)
            sums = sums + du(i).^2;
        end
        sums = (l2-l1).^2+sums;
        s2 = s1 + sqrt(sums);
        [Rhat,Jhat] = getAugmentedJRhat(N,(s2-s1),l2,l1,u2,u1);
        count = count+1;
    end
    u = u2;
    l = l2;
    J = Jhat;
end
```