ENM-502 HOMEWORK ASSIGNMENT 2

Dhruv Gupta - dgupta99@seas.upenn.edu

March 21 2022

1 Introduction

The objective of this analysis is to study a non-linear boundary-value problem defined on the unit-square domain $D = (0 \leq x \leq 1) \cup (0 \leq y \leq 1)$

$$R(u) = \nabla^2 u + \lambda u(1+u) = 0 \tag{1}$$

where $u(x, y)$ is 0 at all boundaries and solutions have to be found for $0 \leq \lambda \leq 1$. We tackle this problem after having previously solved a linear BVP using finite difference method, which is applied here as well, on a 30x30 grid. The non-linear problem is solved using an iterative solver (Newton's method) resulting in a system of linear equations of the form

$$\left.\frac{\partial \bar{\bar{R}}}{\partial \bar{u}}\right|_{u=u_k} .\delta \bar{u_k} = -R(\bar{u_k}) \tag{2}$$

followed by

$$u_{k+1} = u_k + \delta u_k \tag{3}$$

... where x_k denotes the solution at the k^{th} iteration.
Newton's method (or any iterative solver) relies on the initial guess, which is discussed in this report. This particular system is also important in the context of using analytic and arc-length continuation to find all solutions along different non-trivial solution branches and around limit points. This is studied in detail as well. This form of a parameterized non-linear problem is important in fields such as structural analysis or problems where it's necessary to study the solution as it changes with tuning of parameters.

2 Problem Setup and Formulation

2.1 Discretization

We discretize our domain into a uniform 30x30 grid

$$u(x_i, y_j) \quad s.t \quad i \in \{0, 1...30\}, j \in \{0, 1...30\} \tag{4}$$

$$h_x = h_y = \frac{1}{30} \tag{5}$$

and apply centered difference approximation to the non-linear problem shown in equation (1)

$$R(u_{i,j}) = \tag{6}$$
$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2} + \lambda u_{i,j}(1 + u_{i,j}) \tag{7}$$
$$= 0 \tag{8}$$

As in the previous homework we combine the two indices into one

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

and the problem now consists of 900 unknowns. This isn't in the form of $Au = b$ yet and we tackle that in the section on Newton's method.

2.2 Newton's Method

The crux of this problem is its non-linearity. This restricts us from easily writing down a system of equations $Au = b$. To approximate a solution, we use the Taylor series expansion of the residual

$$R(u) = R(u_0) + \left.\frac{\partial R}{\partial u}\right|_{u_0} (u - u_0) + O(h^2) \tag{9}$$

where we use the substitutions

$$u = u_{k+1}$$
$$u_0 = u_k$$

... and ignore the higher-order terms to give us our final system of linear equations (2) in the form of $Au = b$, where A is the Jacobian at any given iteration, u is δx_k, and b is $-R(u_k)$. Now, in finite difference form for the l^{th} row (and k^{th} iteration) of the Jacobian we have

$$\frac{\partial R}{\partial u_k^l} = -\frac{4}{h^2} + \lambda(1 + 2u_k^l) \tag{10}$$
$$\frac{\partial R}{\partial u_k^{l\pm1}} = \frac{1}{h^2} \tag{11}$$
$$\frac{\partial R}{\partial u_k^{l\pm(N_x+1)}} = \frac{1}{h^2} \tag{12}$$

... with all other elements being zero. This gives us a banded matrix as seen in figure 1. The boundary point equations are described as

$$1.\delta u_k^l = -u_k^l \tag{13}$$

We use Newton's method for its quick and wide domain of convergence. The entire method is expected to converge in less than 20 iterations for any of the solution branches. While the matrices for this problem are sparse, we use MATLAB's backslash in order to solve $Au = b$ wherever required.

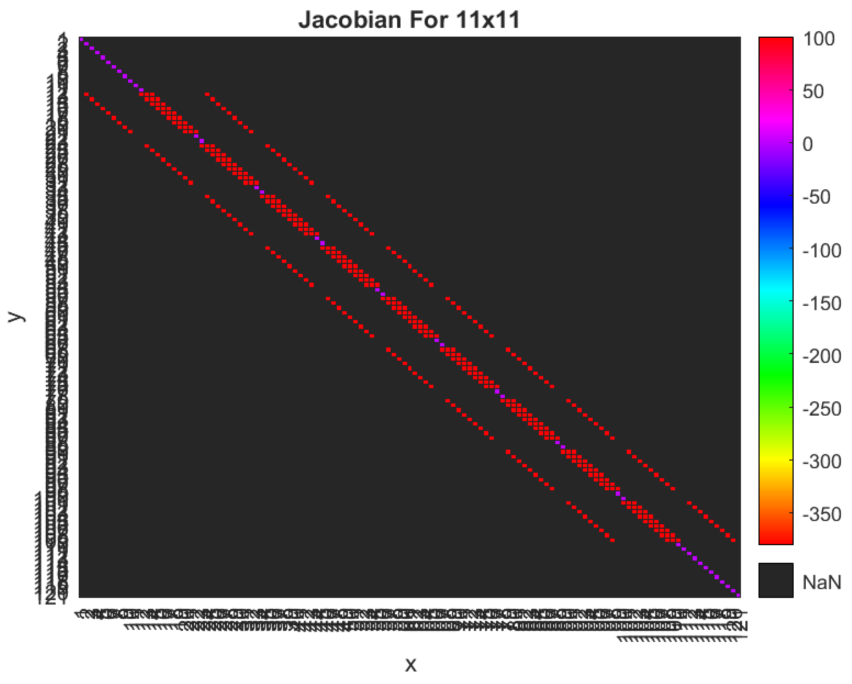

Figure 1: Sample Jacobian for 11x11 grid (All Zeros were set to NaN for illustration purposes)

2.3 First Non-Trivial Solutions

2.3.1 Initial Guess

As discussed earlier, it is difficult to obtain non-trivial solutions to this problem using a random guess in Newton's method. It is likely that we would converge to a flat solution. Hence, to find a good initial guess that would guide us to a non-trivial solution, we look at the problem in the limit $||u|| <<< 1$

$$\nabla^2 u + \lambda u = 0 \tag{14}$$

... since $(1+u) \to 1$. Solutions to this eigenvalue problem are of the form

$$\begin{aligned} \hat{u}(x,y) &= A_{mn} \sin(m\pi x)\sin(n\pi y) \\ m^2 + n^2 &= \frac{\lambda}{\pi^2} \quad (m, n = 1, 2, 3...) \end{aligned} \tag{15}$$

... which implies solutions for this approximation exist at $\lambda = 2\pi^2, 5\pi^2$ in our region of interest. Thus, a good initial guess for Newton's method (at a λ that is close to either of the eigenvalues) would be of the form in (15).

2.3.2 Analytic Continuation

We established earlier that we can find a non-trivial solution with an educated initial guess from the eigenvalue approximation. In order to find solutions outside the approximation we need to employ continuation techniques. Analytic continuation is the first of such methods that allows us to find an initial guess for a changed parameter $(\lambda_0 + \delta\lambda)$, given the solution at λ_0. The initial guess for the second solution is found using a first-order approximation at the first solution point

$$u_2^0 = u_1 + \left.\frac{\partial u}{\partial \lambda}\right|_{\lambda_0} (\lambda_1 - \lambda_0) \tag{16}$$

and the Taylor-series approximation of $R(u, \lambda)$

$$R(u_1, \lambda_1) = R(u_0, \lambda_0) + \left.\frac{\partial R}{\partial u}\right|_{(u_0,\lambda_0)} (\delta u) + \left.\frac{\partial R}{\partial \lambda}\right|_{(u_0,\lambda_0)} (\delta\lambda) \tag{17}$$

Using (17) we obtain

$$J_{(u_0,\lambda_0)} \cdot \frac{\partial u}{\partial \lambda} = -\left.\frac{\partial R}{\partial \lambda}\right|_{(u_0,\lambda_0)} \tag{18}$$

The Jacobian found from the first solution is also reused here in analytic continuation. Thus, using equation (16) we can find an initial guess for the second non-trivial solution at $\lambda_1 = \lambda_0 + \delta\lambda$, where we pick an appropriate λ (0.01 in my code), such that the solution can still roughly follow the eigenvalue approximation, otherwise Newton's method could converge to a trivial solution.

2.4 Arc-Length Continuation

In order to find solutions all along the branch we can not rely entirely on analytical continuation as it becomes a poorer approximation for a non-linear solution curve. We use arc-length continuation in this case. Starting with two solutions close to each other, we use the arc-length as a parameter to continue along the solution branch. One advantage of this method is that it can travel through bifurcation points, such as the one shown in Figure 3, where the non-trivial solution branch intersects the trivial solution branch at $\lambda = 2\pi^2$. In that situation the original Jacobian that is used earlier in analytic continuation becomes singular and the problem does not have a unique solution. Arc-length continuation modifies the problem to give us a non-singular matrix and thereby a unique solution.
Arc-length is defined as

$$(\delta s)^2 = ||\delta u||_2^2 + (\delta\lambda)^2 \tag{19}$$

with the continuation performed as

$$\begin{aligned} \mathbf{u_2^0} &= \mathbf{u_1} + (\delta s)\left(\frac{\partial \mathbf{u}}{\partial s}\right)_1 \\ \lambda_2^0 &= \lambda_1 + (\delta s)\left(\frac{\partial \lambda}{\partial s}\right)_1 \end{aligned} \tag{20}$$

to find the initial guess for the next solution using two previous solutions. The δs is set to 1 for my program. A more efficient program could adjust this quantity based on the gradient of the solution branch in order to speed-up or slow-down whenever necessary.
The arc-length 'residual' is defined as

$$\eta(s, \lambda, \mathbf{u})|_1 = |s_1 - s_0|^2 - ||\mathbf{u}_1 - \mathbf{u}_0||^2 - |\lambda(s_1) - \lambda(s_0)|^2 \tag{21}$$

which is used to define the modified problem as

$$\hat{\mathbf{J}}|_1 \begin{pmatrix} \frac{\partial \mathbf{u}}{\partial s} \\ \frac{\partial \lambda}{\partial s} \end{pmatrix} = -\left.\left(\frac{\partial \hat{\mathbf{R}}}{\partial s}\right)\right|_1 \tag{22}$$

where $\hat{\mathbf{R}} = \begin{pmatrix} \mathbf{R} \\ \eta \end{pmatrix}$ and

$$\hat{\mathbf{J}}|_1 = \begin{pmatrix} \frac{\partial R}{\partial \mathbf{u}}|_1 & \frac{\partial R}{\partial \lambda}|_1 \\ \\ \frac{\partial \eta}{\partial \mathbf{u}}|_1 & \frac{\partial \eta}{\partial \lambda}|_1 \end{pmatrix}$$
$$\left(\frac{\partial \hat{\mathbf{R}}}{\partial s}\right)\Bigg|_1 = \begin{pmatrix} 0 \\ \frac{\partial \eta}{\partial s} \end{pmatrix}_1 \tag{23}$$

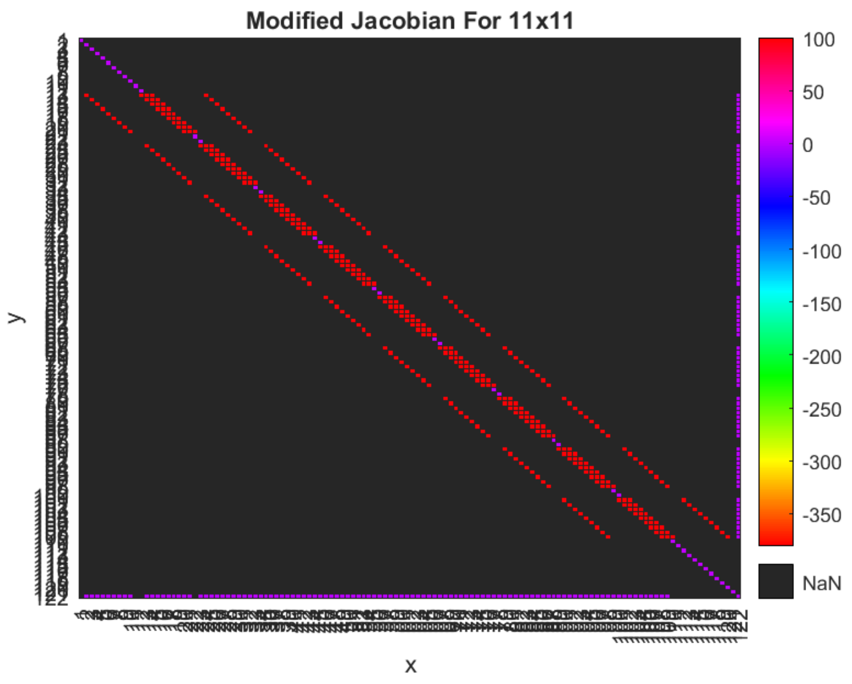

Figure 2: Modified Jacobian For 11x11 grid With Non-Zero Row and Column at the Edges (All Zeros were set to NaN for illustration purposes)

Due to the addition of the extra row and column in the Jacobian in equation (23), and as seen in figure 2, the matrix is now non-singular, which allows us to perform Newton's method on this system of linear equations in order to converge to the right $(\mathbf{u}, \lambda)$ on the solution branch. The full-newton method is

$$(\hat{\mathbf{J}}^{\mathbf{k}})\Big|_{s2} \begin{pmatrix} \delta \mathbf{u}^k \\ \\ \delta \lambda^k \end{pmatrix}\Bigg|_{s2} = -(\hat{\mathbf{R}}^{\mathbf{k}})\Big|_{s2}$$
$$\lambda^{k+1} = \lambda^k + \delta\lambda^k$$
$$\mathbf{u}^{k+1} = \mathbf{u}^k + \delta\mathbf{u}^k \tag{24}$$

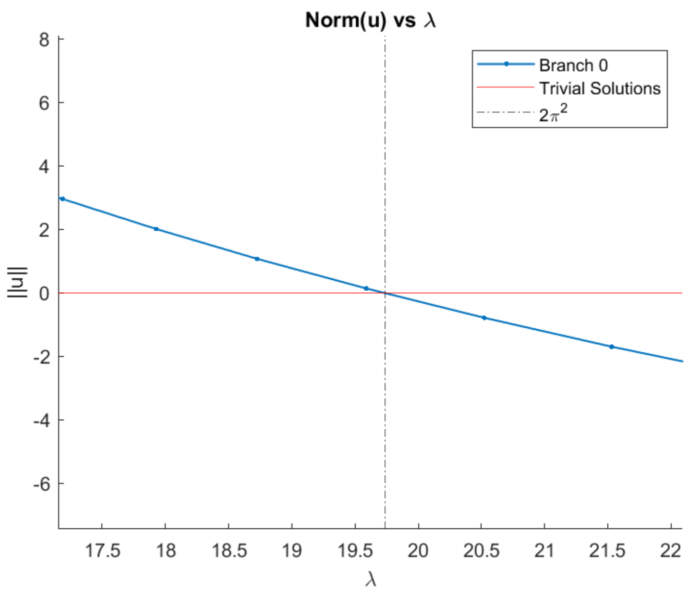

Figure 3: Transcritical Bifurcation Point

3 Results and Discussion

3.1 Plot of norm(u) versus lambda

The modified norm for the solutions is plotted. The convention used is

1. Positive Norm - If the solution is a single hill. In the case of a hill and a valley, if the hill comes before the valley.
2. Negative Norm - If the solution is a single valley. In the case of a hill and a valley, if the valley comes before the hill.

In figure 4 the branch 0 represents solutions that have either a hill or valley only. Branch 0 was obtained by first finding the solution branches on either side of $\lambda = 2\pi^2$ and then using the first two solutions close to 0 as our starting point for arc-length continuation till $\lambda \cong 60$.
Branches 2 and 3 are based upon initial guesses of $(A, m, n) = (\pm 2, 2, 1)$ from equation (15), thus having a hill and valley. The solution branch 0 diverges as it reaches $\lambda = 0$ and is evident from the high density of solutions in that region.

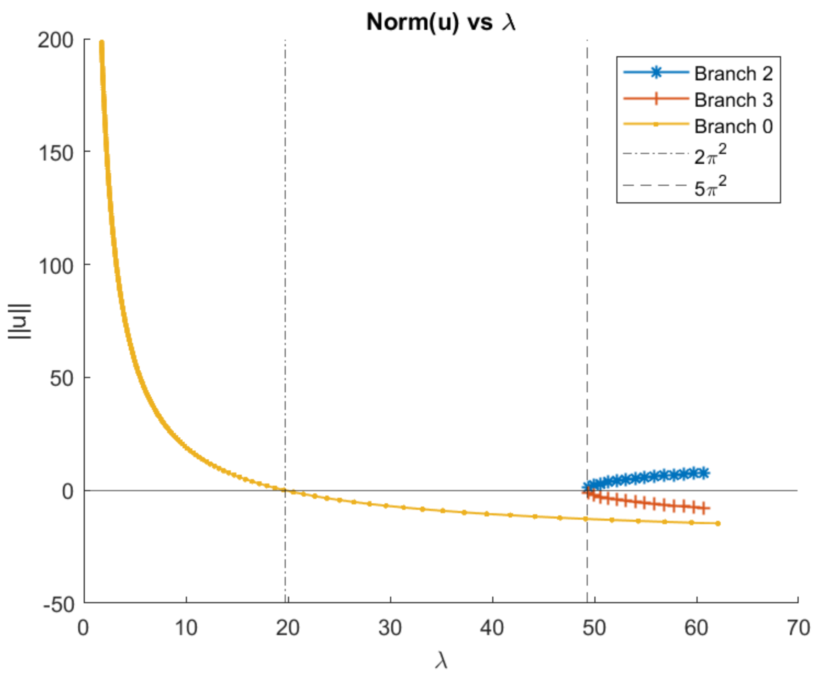

Figure 4: Norm Plot For Branch 0 and 2,3

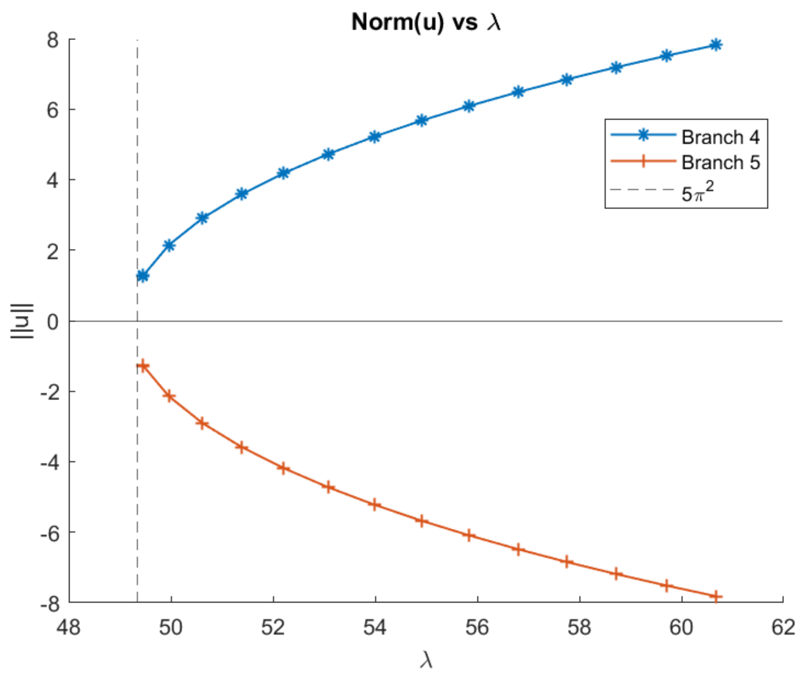

Figure 5: Norm Plot For Branch 4,5

Figure 5 shows the branches 4 and 5, which overlap with 2 and 3 but are shown separately here. These branches are different in that the hill and valley is oscillating along the x-axis instead. The initial guesses for these branches were $(A, m, n) = (\pm 2, 2, 1)$.

3.2 Contour plots

1. Branch 0 - Figures 6 through 8 represent the contour plots for the first branch of solutions, showing only a hill or valley. As was seen in the norm plots, the solution turns from a hill to a valley at the $\lambda = 2\pi^2$ mark. Figure 7 shows the plot for a solution close to the inversion point, where the amplitude of the solution is significantly smaller than the rest ($||u|| <<< 1$). Figure 8 shows a complete valley at the end of the branch.

2. Branch 2,3 - Branches 2 and 3 are characterised by the hill-valley solution that is oscillating in the y-axis. In figure 9, the positive norm solution is defined on the left, and the other plot represents the inverted solution (akin to $-A$).

3. Branch 4,5 - Similar to branches 2 and 3, except we see an oscillation in the x-axis instead in figure 10.

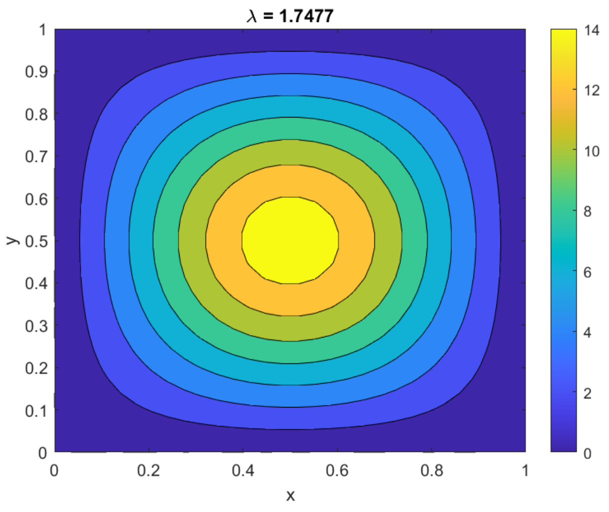

Figure 6: Branch 0 Plot A

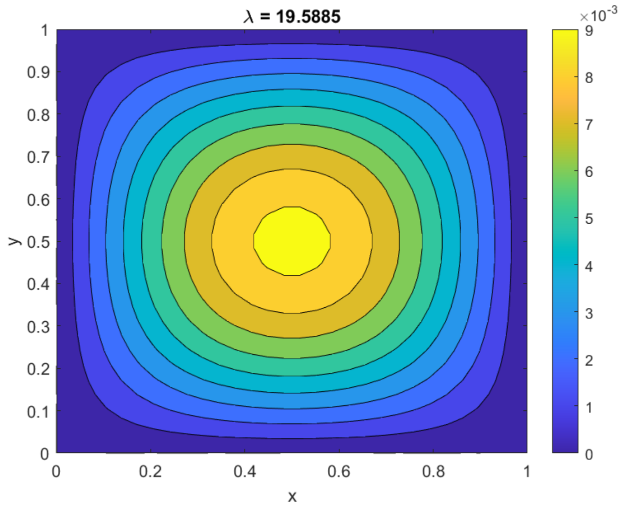

Figure 7: Branch 0 Plot B

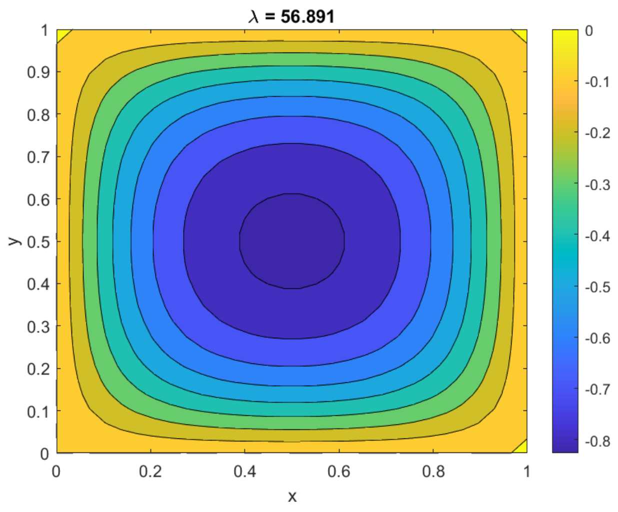

Figure 8: Branch 0 Plot C

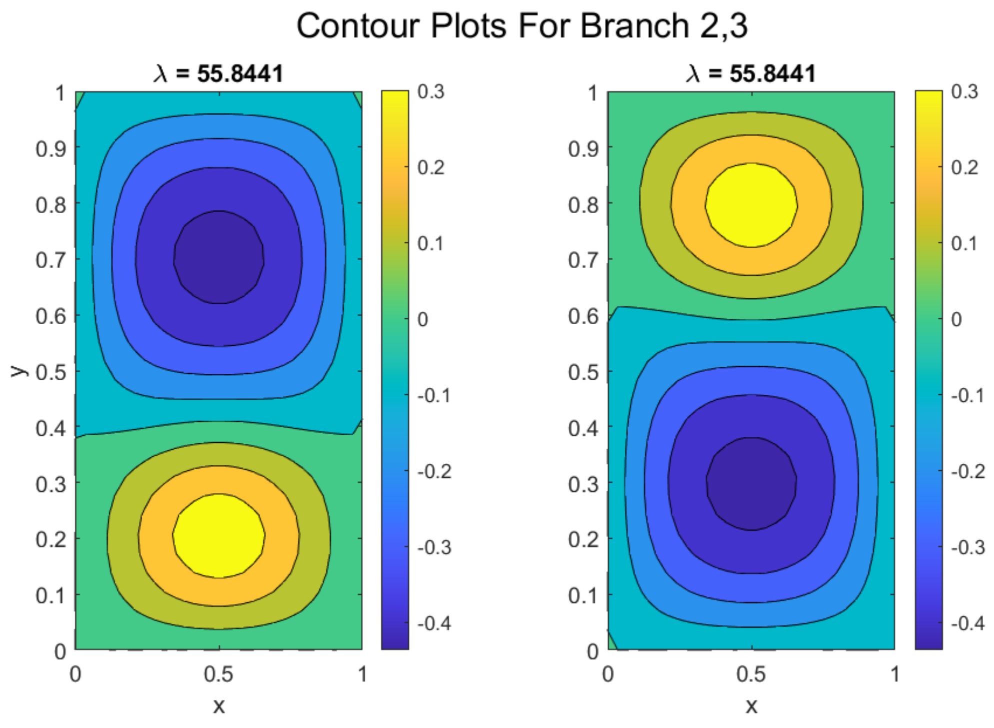

Figure 9: Branch 2,3 Plots

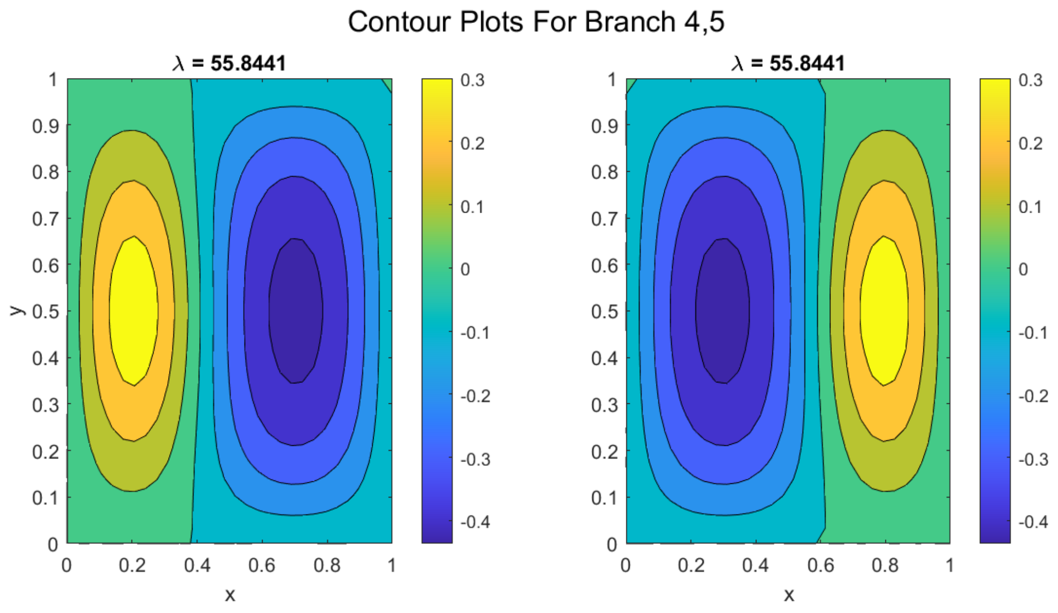

Figure 10: Branch 4,5 Plots

4 Conclusions

Compared to the previous assignment where we solve a linear differential equation, the non-linear problem is much more difficult to solve. A wrong initial guess in the iterative solver sends us to the trivial branch. Newton's method however works well when we give it an educated initial guess. While we use Newton's method for the first two solutions of all branches, the modified Newton was also used to find solutions along the rest of the branch along with arc-length continuation. It's convenient to be able to modify the Newton module to accommodate the modified Jacobian from arc-length continuation. Arc-length continuation works well in finding all solutions in branch 0, but it fails to wrap around the saddle-node bifurcation point that would connect branches 2 and 3 or 4 and 5. Trying to find solutions in that region results in errors in the program. A more robust implementation of arc-length continuation will need to find a way around this.

A Code

A.1 main.m

```
nx = 29;
ny = nx;
sz = (nx+1)^2;
x_ = 0:(1/nx):1;
y_ = 0:(1/nx):1;
m = 1; %m=2 for branch 2
n = 1; %n=2 for branch 3
lam0 = (m^2 + n^2)*pi^2+1e-3;
amn =-2; %-2 for valley first, hill second; +2 for hill first valley second
asoln = @(x,y) amn*sin(m*pi*x).*sin(n*pi*y); %initial guess
[X,Y] = ndgrid(x_,y_);
u0init = reshape(asoln(X,Y),sz,1);
miter = 20;
tol = 1e-5;


%Assuming now we have the correct initial guess. We use analytic cont. to
%find first two solutions
[u0,iter,J0] = newton(nx,u0init,miter,lam0,tol);
du1 = J0\-drlam(u0,nx);
dlam = 1e-2;
u1init = u0 + dlam*du1;
lam1 = lam0 + dlam;
[u1,iter,J1] = newton(nx,u1init,miter,lam1,tol); %SECOND SOLUTION

usoln = [u0 u1];
lams = [lam0 lam1];
%now we have first two solutions using analytic solution.

%move on to using ALC to find initial guess for next one
s0 = 0;
```

```
s1 = sqrt((lam1-lam0)^2 + (norm(u1-u0))^2);
Jnew = [J1 drlam(u1,nx); dnu(u1,u0,lam1,lam0)'];
bnew = [zeros(900,1) ; 2*(s1-s0)];
dalc = Jnew\-bnew;
ds = 1;
uinit = u1 + ds*dalc(1:sz);
laminit = lam1 + ds*dalc(end);
lamk = lam1;
uk = u1;
sk = s1;

count = 1; %to stop if it's too slow
while (lamk<60 && count<200)
    [uk1,lamk1,iter,Jk1] = fullnewton(nx,uinit,u0,miter,laminit,lam0,tol);
    usoln = [usoln uk1];
    lams = [lams lamk1];
    %Arc Length for next lambda. We have k1 and k solution.
    sk1 = sqrt((lamk1-lam0)^2 + norm(uk1-u0)^2);
    bnew = [zeros(900,1) ; 2*(sk1-s0)];
    dalc = Jk1\-bnew; %the Jnew is just Jk1 now since we use the augmented one for the fullnewton
    %initial guesses for next lambda
    uinit = uk1 + ds*dalc(1:sz);
    laminit = lamk1 + ds*dalc(end);
    uk = uk1;
    sk = sk1;
    lamk = lamk1;
    count = count+1;
end
unorm = tests(usoln,0);
plot(lams,unorm,'-o')
```

A.2 newton.m

```
%Simple Newton
function [xf,iter,A] = newton(nx,xinit,miter,lam,tol)
ny = nx;
s = (nx+1)^2;
[A,b] = jacobian(nx,ny,lam,xinit);
dx = A\-b;
xk = xinit;
iter = 0;
while (norm(dx)>tol)
    xk1 = xk + dx;
    [A,b] = jacobian(nx,ny,lam,xk1);
    dx = A\-b;
    xk = xk1;
    iter = iter+1;
    if(iter>miter)
```

```
        disp('Max Iterations Reached')
        return;
    end
end
xf = xk;
end
```

A.3 fullnewton.m

```
function [uf,lamf,iter,Jnew] = fullnewton(nx,uinit,u0,miter,laminit,lam0,tol)
ny = nx;
sz = (nx+1)^2;
s0 = 0;
[A,b] = jacobian(nx,ny,laminit,uinit);
Jnew = [A drlam(uinit,nx); dnu(uinit,u0,laminit,lam0)']; %modified jacobian
sinit = sqrt((laminit-lam0)^2 + norm(uinit-u0)^2);
bnew = [b;(sinit-s0)^2 - norm(uinit-u0)^2 - (laminit-lam0)^2];
diff = Jnew\-bnew;

uk = uinit;
lamk = laminit;
iter = 0;
while (norm(diff(1:sz))>tol && abs(diff(end))>tol)
    %convergence with the previous vals.
    uk1 = uk + diff(1:sz);
    lamk1 = lamk + diff(end);
    %Next iteration
    [A,b] = jacobian(nx,ny,lamk1,uk1);
    Jnew = [A drlam(uk1,nx); dnu(uk1,u0,lamk1,lam0)'];
    sk1 = sqrt((lamk1-lam0)^2 + norm(uk1-u0)^2);
    bnew = [b;(sk1-s0)^2 - norm(uk1-u0)^2 - (lamk1-lam0)^2];
    diff = Jnew\-bnew;

    uk = uk1;
    lamk = lamk1;
    iter = iter+1;
    uf = uk;
    lamf = lamk;
    if(iter>miter)
        disp('Max Iterations Reached')
        return;
    end
end
uf = uk;
lamf = lamk;
end
```

A.4 jacobian.m

```
function [A,b] = jacobian(nx,ny,lam,uk)
h = 1.0/nx;
s=(nx+1)^2;
%Size - (nx+1)*(ny+1) x (nx+1)*(ny+1)
x=(1/h^2)*ones(s-1,1); %left and right of diagonal
y=ones(s,1); %diagonal
p=(1/h^2)*ones(s-(nx+1),1);
A=diag(p,-(nx+1))+diag(x,-1)+diag(y)+diag(x,1)+diag(p,(nx+1));
b = uk; %To account for boundary

%Replacing boundary values
A(1:(nx+1),:) = eye((nx+1),s);
A(end-nx:end,:) = rot90(eye((nx+1),s),2);
for i = (nx+2):(ny*(nx+1))
    if (mod((i-1),(nx+1))==0 || mod(i,(nx+1))==0)
        A(i,:) = zeros(1,s);
        A(i,i) = 1;
    else
        b(i) = rfunc(uk,h,nx,lam,i);
        A(i,i) = -(4/h^2)+lam*(1+2*uk(i));
    end
end
```

A.5 dnu.m

```
function [b] = dnu(u1,u0,lam1,lam0)
b = -2*(u1-u0);
b(end+1) = -2*(lam1-lam0);
end
```

A.6 drlam.m

```
function [b] = drlam(u,nx)
b = zeros((nx+1)^2,1);
ny = nx;
for i = (nx+2):(ny*(nx+1))
    if (mod((i-1),(nx+1))~=0 && mod(i,(nx+1))~=0)
        b(i) = u(i)*(1+u(i));
    end
end
end
```

A.7 rfunc.m

```
function val = rfunc(uk,h,nx,lam,i)
val = (uk(i-nx-1) + uk(i-1) + (lam*(h^2)*(1+uk(i))-4)*uk(i) + uk(i+1) + uk(i+nx+1))/(h^2);
```

A.8 tests.m

```
%%USED TO PLOT THE NORM PLOTS
%%ACCOUNTING FOR CONVENTIONS USED
function [unorm] = tests(usoln,sgn)
sz = size(usoln);
unorm = zeros(sz(2),1);
for i=1:sz(2)
    if sgn == 0
        unorm(i) = norm(usoln(:,i));
    else
        unorm(i) = sign(usoln(500,i))*norm(usoln(:,i));
end
end
```