ENM 502 : NUMERICAL METHODS AND MODELLING

ASSIGNMENT #3

SUBMITTED ON 1ST APRIL 2014 BY :

RAMALINGAM KAILASHAM

kailr@seas.upenn.edu

Project Statement and Introduction

We are required to solve the following non-linear boundary value problem defined on the unit square domain D = (0≤x≤1) U (0≤y≤1)

$$(\nabla.\nabla)\, u + \lambda u(1+u) = \varepsilon \sin(\pi x) \tag{1}$$

With u(x,y) = 0 on all boundaries (∂D)

For solving any non-linear equation, we need an intelligent initial guess. For this particular problem, we can solve the linearized approximation, i.e

$$(\nabla.\nabla)\mathrm{u}' + \lambda \mathrm{u}' = 0 \tag{2}$$

Where u' is used to show that this is not the same u as in Equation 1. It does **not** represent the derivative of u. This approximation is only valid when ||u|| is small. We have made this assumption implicitly.

The above eigen-value problem can be solved using Finite Fourier Transform to obtain

$$u' = A \sin(m\pi x) \sin(n\pi y) \tag{3}$$

$$\lambda_{mn} = \pi^2 (m^2 + n^2) \tag{4}$$

Where m and n are integers and A is an arbitrary constant.

As the problem statement wants us to analyse solution branches in the range 0≤λ≤60, we will stick to the solutions of u' that correspond to m=1,n=1, m=1, n=2 and m=2, n=1. These three branches shall henceforth be referred to as (1,1), (1,2) and (2,1) respectively.

The value of the arbitrary constant A has been chosen as 1. This choice was found to give the right kind of solution and solution-norm behavior.

Solution methods used

The given unit square domain was discretized into a 31 x 31 grid using centred-difference approximation. The set of non-linear equations was solved using the Full Newton's method. Analytic continuation was used to track the behavior of solution norms with respect to the parameter λ and ε. Arc-length continuation was also used for the same purpose and to see if better results were obtained.

Outline of Full Newton's method

This method involves solving

$$\mathbf{J}\,\delta u^k = -R^k \tag{5}$$

And using δx^k to update

$$u^{k+1} = u^k + \delta u^k \tag{6}$$

where R(u) = 0 is the residual vector and

$$J_{ij} = \frac{\partial Ri}{\partial uj}$$

These two steps are carried out till $||\,\delta u^k||$ is less than a suggested tolerance value. In this project, a tolerance value of 10^{-5} is used.

The Newton's method for analytic continuation, accepts an initial guess at a given value of λ and returns the converged solution at the **same value** of λ.

The Newton's method for arc-length continuation, accepts an initial guess for both the solution and λ at a given length along the curve and converges to give the actual solution and λ value at that point.

Outline of Analytic continuation

In this case, R(u,λ) = 0 is the residual vector.

$\frac{\partial u}{\partial \lambda} = \boldsymbol{J}$ where J is the same as that obtained from the Newton's method.

The following equation

$$J\frac{\partial u}{\partial \lambda} = -\frac{\partial R}{\partial \lambda} \tag{7}$$

Is solved to obtain $\frac{\partial u}{\partial \lambda}$

The initial guess for the next point is generated using

$$U_2^{(0)} = \underline{u}_1 + \frac{\partial u}{\partial \lambda}(\lambda_2 - \lambda_1) \tag{8}$$

The step size in λ is appropriately hosen so that convergence is smooth. This method is also called “incremental loading”, a reference to the fact that it first originated in structural analysis where gravity is used as a parameter and is slowly tuned up.

Outline of Arc-length continuation(ALC)

The detailed procedure for implementing arc-length continuation was given in the Supplementary Notes section of the assignment. Only the main equations are discussed here.

To begin ALC, two solutions u_0 and u_1 are needed at two values λ_1 and λ_2 such that

$$(ds)^2 = (d\lambda)^2 + ||du||^2 \tag{9}$$

To find the initial guess for u and λ at an unknown point “2”, we use

$$u_2 = u_1 + \left(\frac{\partial u}{\partial s}\right)(\delta s) \tag{10}$$

$$\lambda_2 = \lambda_1 + \left(\frac{\partial \lambda}{\partial s}\right)(\delta s) \tag{11}$$

$\left(\frac{\partial u}{\partial s}\right)$ and $\left(\frac{\partial \lambda}{\partial s}\right)$ are found by solving the equation

$$\left(\hat{\underline{\underline{J}}}\right)\Big|_{SR}\left(\begin{array}{c}\frac{\partial \underline{u}}{\partial s}\\ \\ \frac{\partial \lambda}{\partial s}\end{array}\right)\Bigg|_{SR} = -\left(\frac{\partial \hat{\underline{R}}}{\partial s}\right)\Big|_{SR} \tag{12}$$

Once the initial guesses have been found, they are fed into the Newton’s method code and run till convergence is reached in both u and λ.

Thus the Newton’s method for the analytic and ALC routines will be slightly different.

The Jacobian obtained in the Analytic continuation cases is a banded one with a penta-diagonal structure.

The Jacobian obtained in the ALC case has an arrow structure.

Sign Conventions and Procedure (Analytic Continuation)

Figures 1 and 2 show the plot of $||u||_2$ vs λ for the three different solution branches at ϵ = 0. These plots were generated using Analytic Continuation.

Additional plots showing the variation of $||u||_2$ with λ for various ϵ values from 0 to 1 have been attached as a part of Appendix B.

The || . || operator always returns a positive value. For the sake of clarity, norms corresponding to certain solutions have been declared as positive and those for the rest have been declared as negative. The sign convention is explained below.

At low values of λ, say below 5, the ||u|| values are of the order of 100s. To capture the behavior accurately, such high ||u|| points have been removed from the graph.

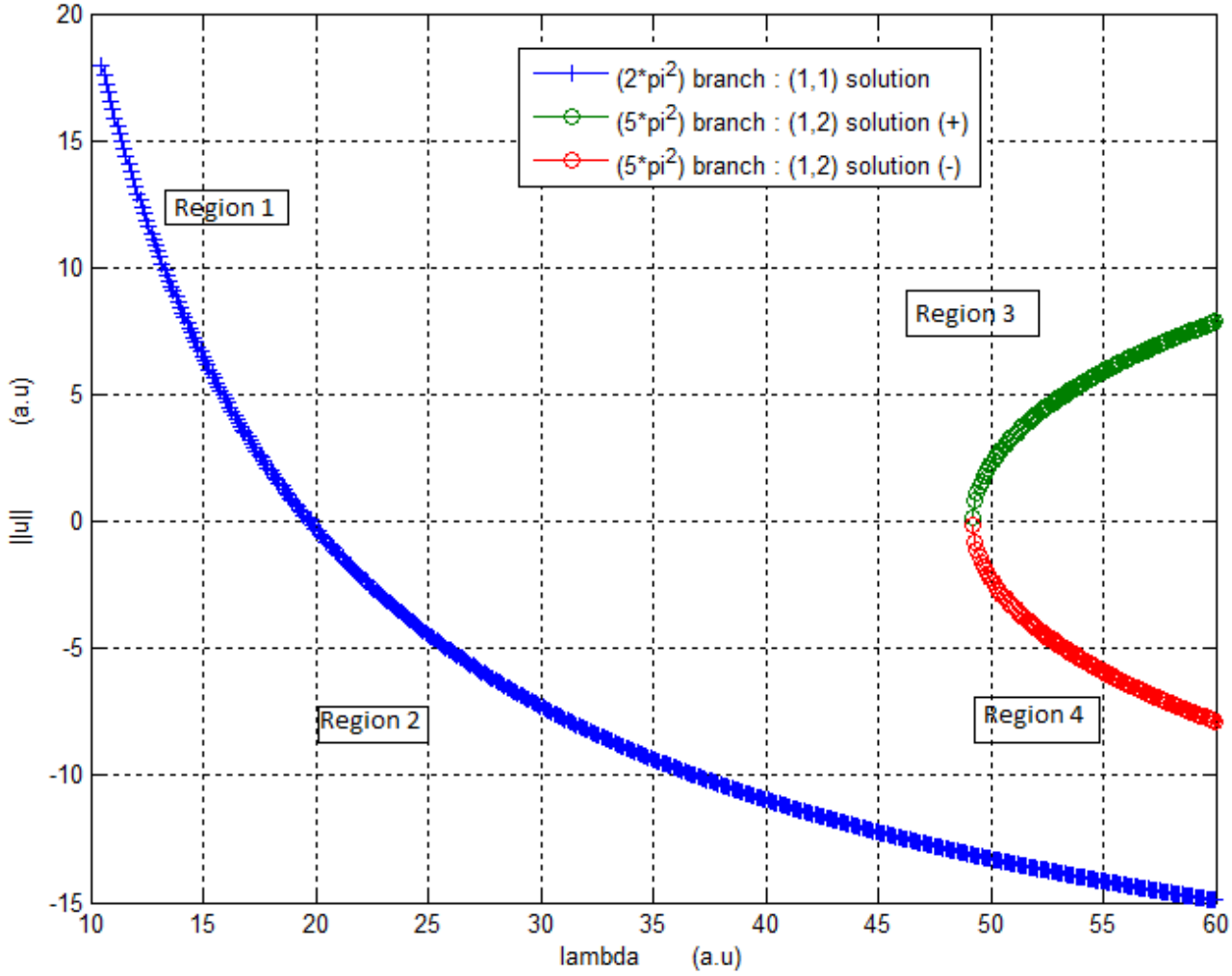

Figure 1 : Plot showing the variation of ||u|| with λ for ϵ = 0. Generated using analytic continuation.

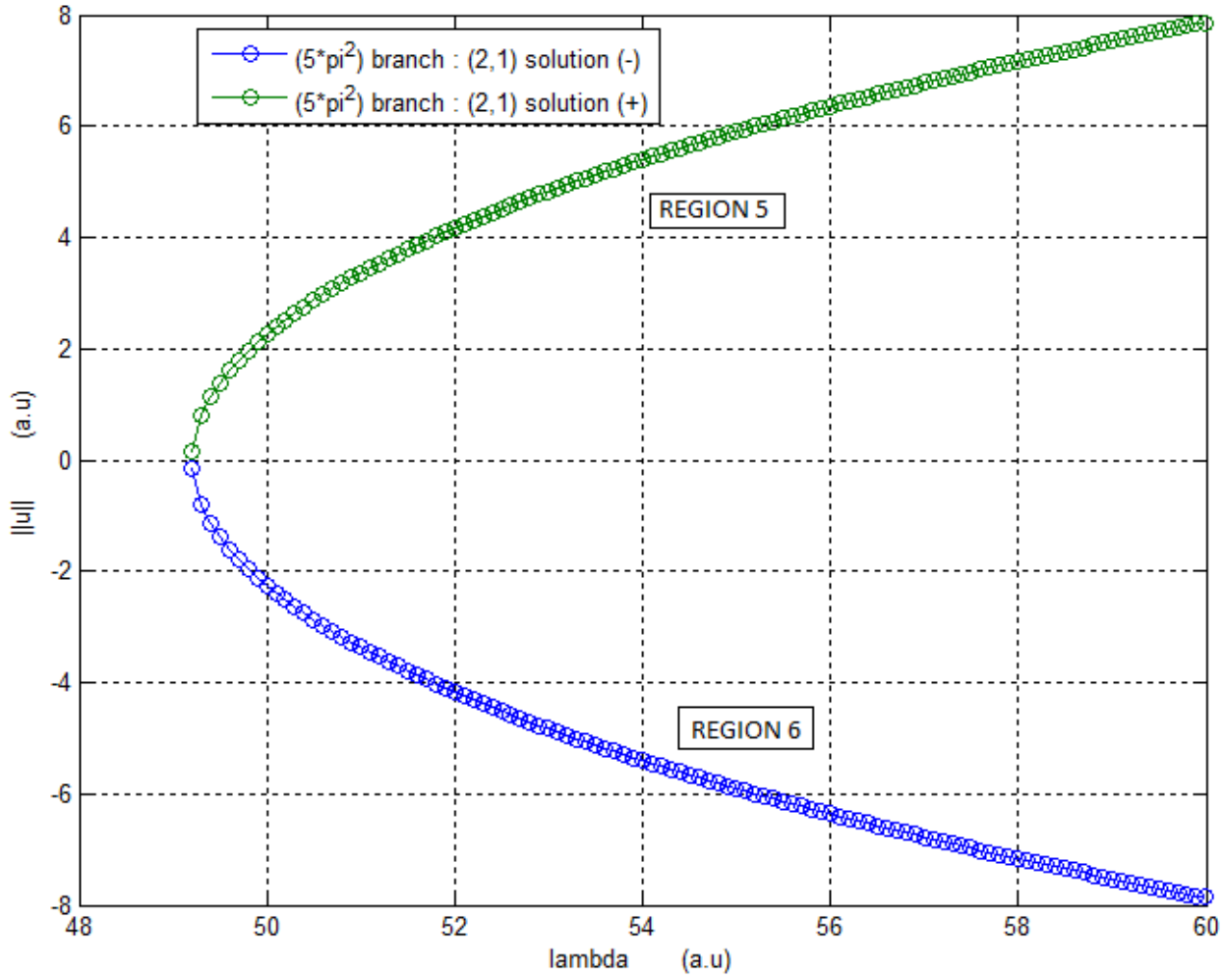

Figure 2 : Plot showing the variation of ||u|| with λ for the (2,1) solution branch for ϵ = 0. Generated using analytic continuation.

Region 1 (0≤λ≤2π^2) : (1,1) Solution branch

The values in this region were generated by starting at a λ value of ($2\pi^2$ – 0.15) and decreasing λ in steps of 0.1 till 0.3892. However, all the data points have not been plotted to preserve the scale of the graph. Value of A was taken as 1. (See Equation 3).

The solutions in this region look like this and such “hill” type solutions have been assigned a positive norm:

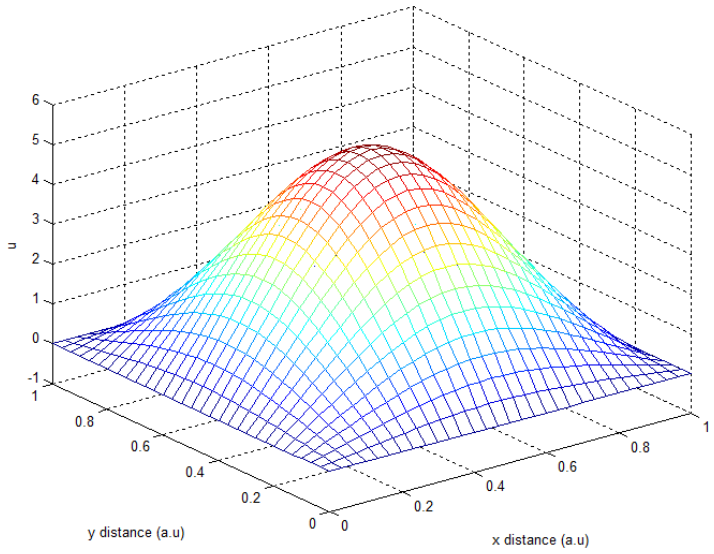

Figure A1 : General shape of solutions in Region 1 of the graph

Region 2 ($2\pi^2 \leq \lambda \leq 60$) : (1,1) Solution branch

Following the run for region 1, λ is at 0.3892. The values in this region were generated by starting at this value and incrementing λ in steps of 0.1 till 60. Value of A was taken as 1 (See Equation 3).

The solutions in this region 2 look like this and such “bowl” type solutions have been assigned a negative norm. :

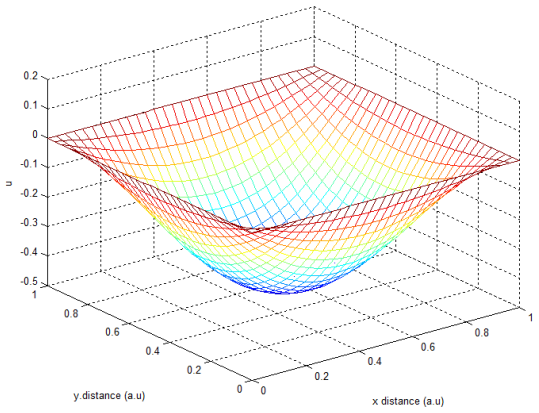

Figure A2 : General shape of solutions in Region 2 of the graph

Thus we see that the solution changes from hill type to bowl type on passing through the eigen value $2\pi^2$. At $2\pi^2$, the only solution obtained is the trivial one, i.e u=0 at all points.

Region 3 ($5\pi^2 \leq \lambda \leq 60$) : (1,2) Solution branch

To obtain this branch, value of A was taken as 1. The values in this region were generated by starting at a λ value of ($5\pi^2 - 0.15$) and incrementing λ in steps of 0.1 till 60.[It is important to note that due to rounding, the value of $5\pi^2$ as generated by MATLAB is higher than the actual value of $5\pi^2$.]. The solutions in this region look as shown below and were assigned a positive norm.

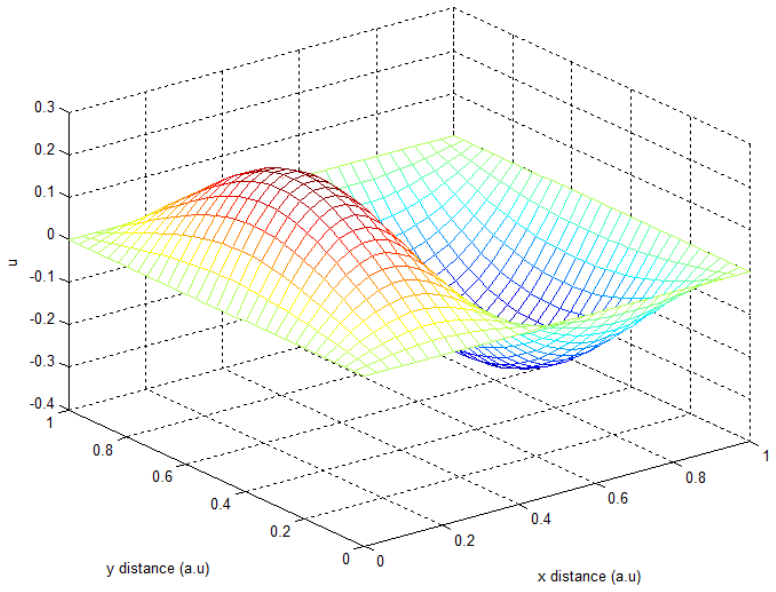

Figure A3 : General shape of solutions in Region 3 of the graph

Region 4 ($5\pi^2 \leq \lambda \leq 60$) : (1,2) Solution branch

To obtain this branch, value of A was taken as -1. The values in this region were generated by starting at a λ value of ($5\pi^2 - 0.15$) and incrementing λ in steps of 0.1 till 60. The solutions in this region look as shown below and were assigned a negative norm.

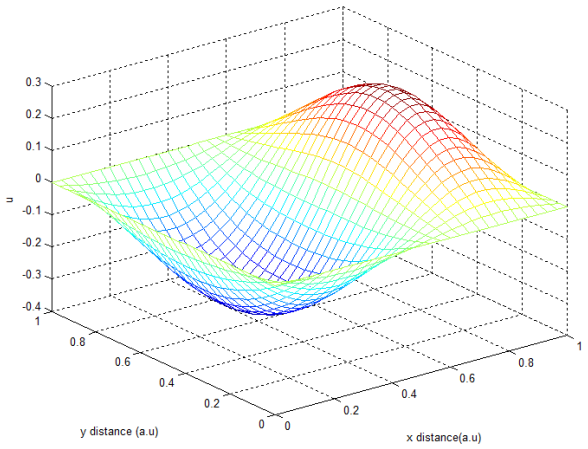

Figure A4 : General shape of solutions in Region 4 of the graph

Region 5 ($5\pi^2 \leq \lambda \leq 60$) : (2,1) Solution branch

To obtain this branch, value of A was taken as 1. The values in this region were generated by starting at a λ value of ($5\pi^2 - 0.15$) and incrementing λ in steps of 0.1 till 60. The solutions in this region look as shown below and were assigned a positive norm.

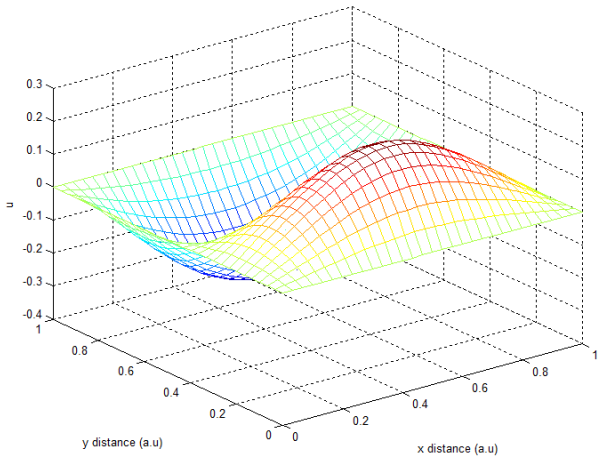

Figure A5 : General shape of solutions in Region 5 of the graph

Region 6 ($5\pi^2 \leq \lambda \leq 60$) : (1,2) Solution branch

To obtain this branch, value of A was taken as -1. The values in this region were generated by starting at a λ value of ($5\pi^2 - 0.15$) and incrementing λ in steps of 0.1 till 60. The solutions in this region look as shown below and were assigned a negative norm.

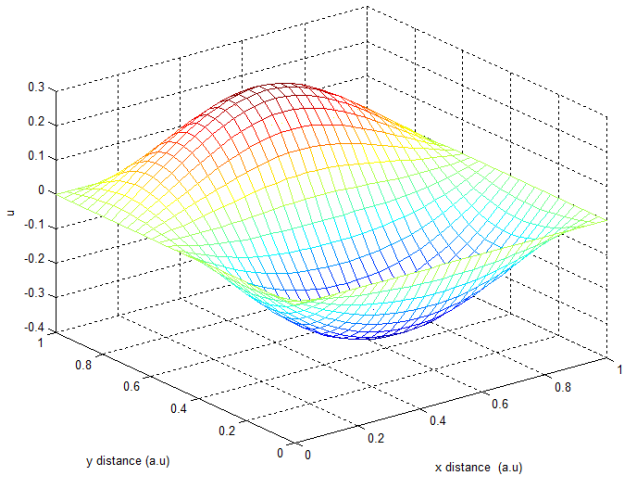

Figure A6 : General shape of solutions in Region 6 of the graph

Even though the solutions to the (2,1) and (1,2) branches look very different, they have the same norms. Hence they would overlap each other if plotted on the same graph and that is why the graphs in Figures 1 and 2 have been plotted separately.

Both the (1,2) and (2,1) branches start with one orientation and flip over on crossing the zero plane. At $5\pi^2$, the only solution is the trivial one, i.e u = 0 at all points.

The representative contour plots corresponding to solutions in each of these six regions have been attached in Appendix B.

Sign Conventions and Procedure (Arc Length Continuation)

Figures 3 and 4 show the plot of $||u||_2$ vs λ for the three different solution branches at ϵ = 1. These plots were generated using Arc Length Continuation.

The sign conventions used for plotting these graphs are the same as that for the Analytic case. A main difference is that the (1,2) and (2,1) solution branches were generated without any need for restarts.

At low values of λ, say below 5, the ||u|| values are of the order of 100s. To capture the behavior accurately, such high ||u|| points have been removed from the graph.

More importantly, at these values, the change in ||u|| is very large for an infinitesimally small change in λ. This contradicts the assumption made while writing Equation 9. The ALC code takes increasingly longer times to converge at low λ values.

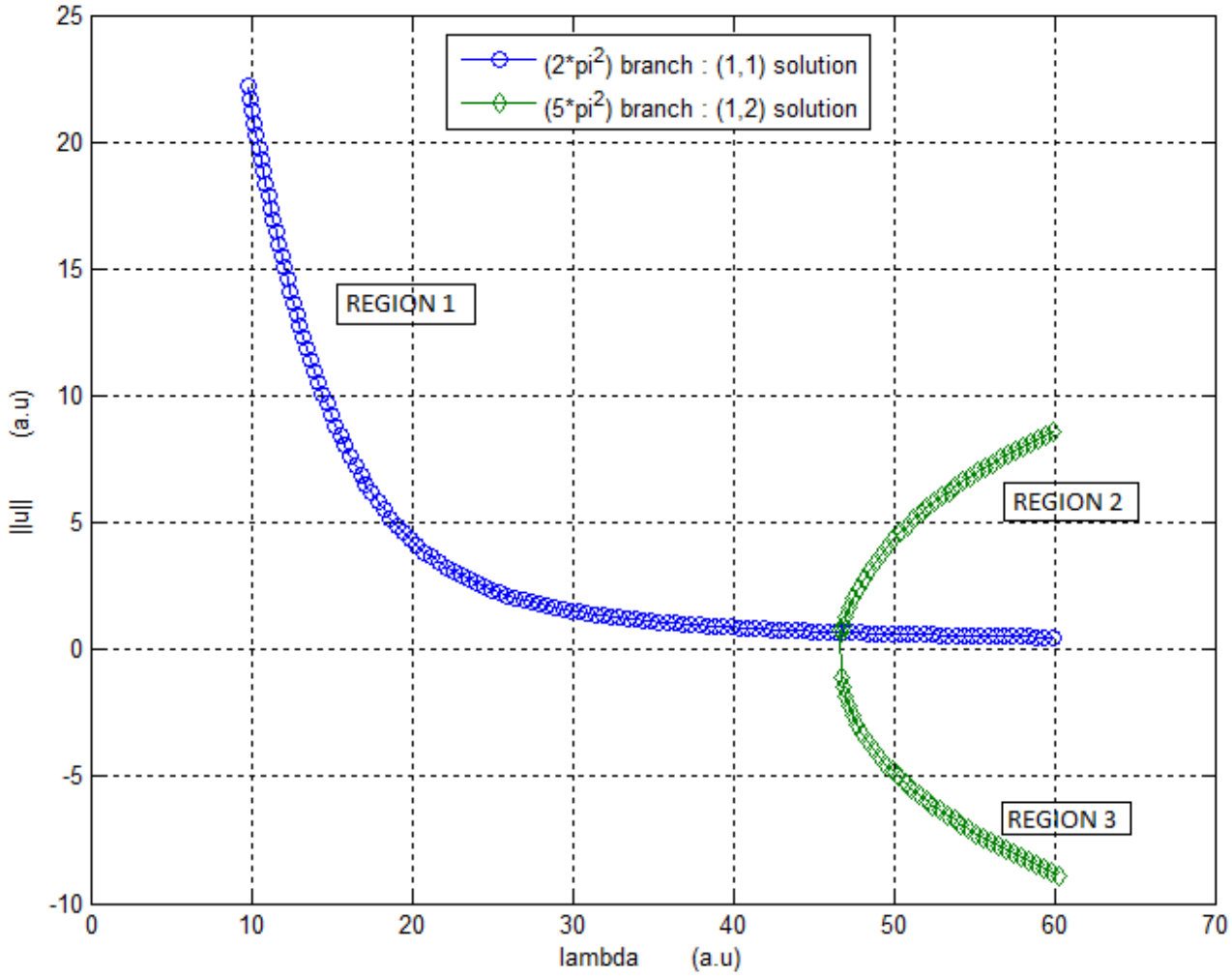

Figure 3 : Plot showing the variation of ||u|| with λ for ϵ = 1. Generated using ALC.

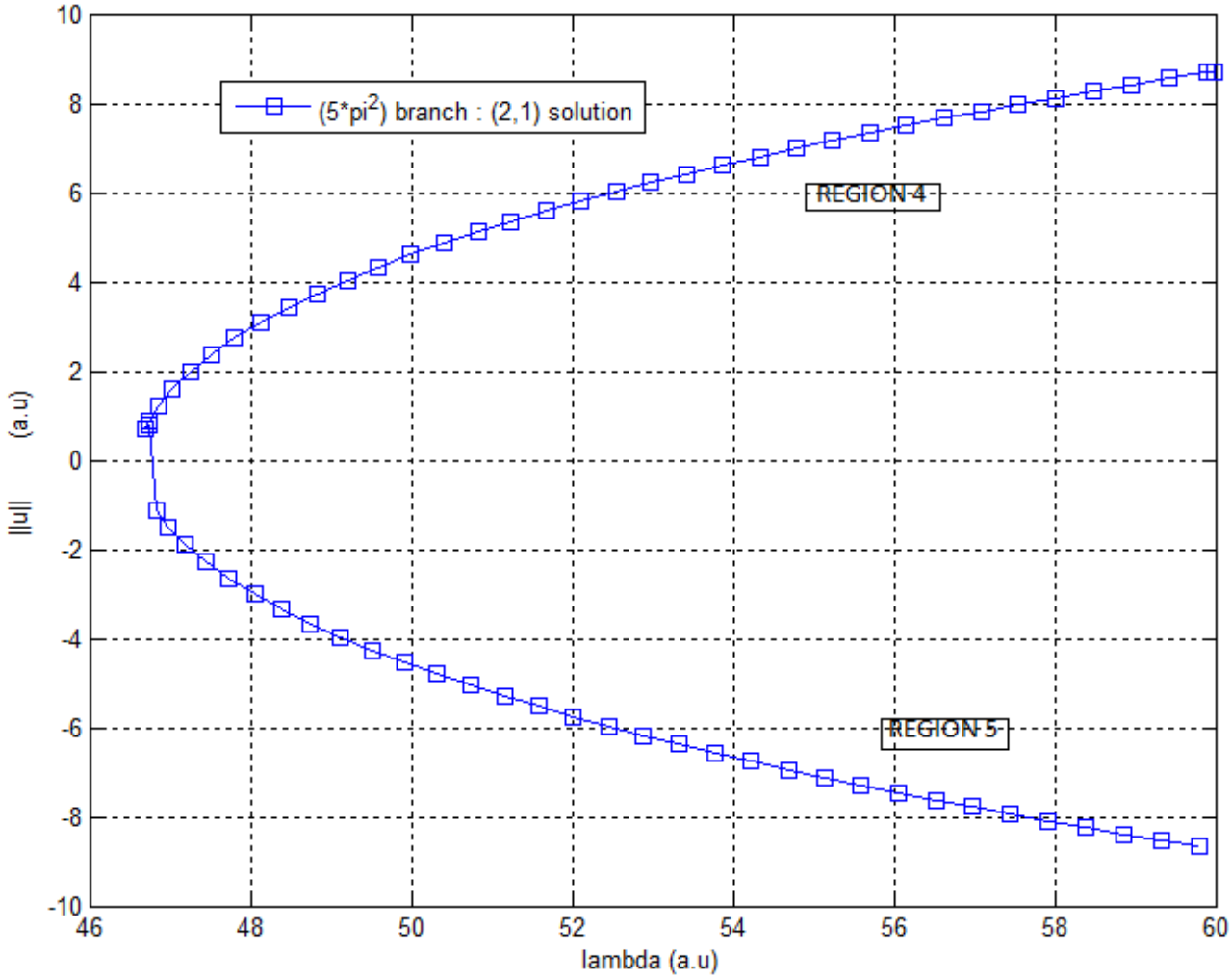

Figure 2 : Plot showing the variation of ||u|| with λ for the (2,1) solution branch for ϵ = 1. Generated using ALC.

Region 1 (0≤λ≤60) : (1,1) Solution branch

This branch was generated by first calculating the value of u at λ_0 = 8.3892 and λ_1 = 8.3992 using Analytic continuation to generate initial guesses to be fed into the Newton's method of ALC. The value of s is set to 0 at λ_0. The step size in "s" is 0.5. All the solutions in this range are of the "hill" type and hence have a positive norm. This branch never passes through 0. The forcing function ensures that the trivial solution never satisfies Equation 1.

Regions 2 and 3 were generated in one run without any restarts, unlike the analytic case.

Region 2 ($5\pi^2 \leq \lambda \leq 60$) : (1,2) Solution branch

This region of the (1,2) branch was generated by first calculating the value of u at λ_0 = 59.9980 and λ_1 = 59.8980 using Analytic continuation to generate initial guesses to be fed into the Newton's method of ALC. The value of s is set to 0 at λ_0. The step size in "s" is -0.5

The general shape of solutions in this region is of the same type as Region 3 in the previous case and have a positive norm. Though ||u|| approaches 0, it never passes through the zero plane because of the forcing function.

Region 3 ($5\pi^2 \leq \lambda \leq 60$) : (1,2) Solution branch

As no restarts are required, this is simply a continuation of the curve from Region 2.ALC ensures that the curve jumps over from Region 2 to Region 3 automatically.

The general shape of solutions in this region is of the same type as Region 4 in the previous case and are assigned a negative norm. Though ||u|| approaches 0, it never passes through the zero plane because of the forcing function.

Region 4 ($5\pi^2 \leq \lambda \leq 60$) : (2,1) Solution branch

This region of the (2,1) solution branch was generated in exactly the same way as Region 2 of the (1,2) branch. It has been plotted separately so that the graphs don't overlap.

Region 5 ($5\pi^2 \leq \lambda \leq 60$) : (2,1) Solution branch

As no restarts are required, this is simply a continuation of the curve from Region 4.ALC ensures that the curve jumps over from Region 4 to Region 5 automatically.

The representative contour plots corresponding to solutions in each of these five regions have been attached in Appendix C.

A break in the curve is observed while passing from Region 2 to Region 3 (and also Region 4 to Region 5). A smaller step size in "s" does not solve the problem. The contour plots of the solution look as expected, though.

Though the general shape of solutions in both the ϵ=0 and ϵ=1 case are the same, there are subtle differences introduced by the forcing function. This is explained in the next section.

Effect of forcing function on overall solution structure

By comparing Figures 1 and 3, we see that the forcing function causes ||u|| to increase.

For the same value of λ, the ||u|| for the case with no forcing function is always lesser than the ||u|| with a forcing function present.

We also see that for any value of ϵ other than 0, the solution branches do not pass though the zero plane.

Given below is a hand-drawn sketch for the solution structure when $\epsilon << 1$:

APPENDIX A

List of variables used in the codes and their meanings.

Variable in code	Mathematical form
R_u	$R(u)$
J_u	$\frac{\partial R}{\partial u}$
J_lambda	$\frac{\partial R}{\partial \lambda}$
J_epsilon	$\frac{\partial R}{\partial \epsilon}$
del_u_calc	δu
delx_delp	$\frac{\partial u}{\partial \lambda}$
delx_delq	$\frac{\partial u}{\partial \epsilon}$
J_aug	$\hat{\underline{\underline{J}}}\Big\|_{SR} = \begin{pmatrix} \frac{\partial \underline{R}}{\partial \underline{u}}\Big\|_{SR} & \frac{\partial \underline{R}}{\partial \lambda}\Big\|_{SR} \\ \frac{\partial \eta}{\partial \underline{u}}\Big\|_{SR} & \frac{\partial \eta}{\partial \lambda}\Big\|_{SR} \end{pmatrix}$
R_aug	$\hat{\underline{R}} = \begin{pmatrix} R \\ \eta \end{pmatrix}$
J_s	$\frac{\partial \eta}{\partial s}$
eta_u	$\frac{\partial \eta}{\partial u}$
eta_lambda	$\frac{\partial \eta}{\partial \lambda}$

analytic.m

```
%******************* INPUT PARAMETERS ********************************** %
% q    : value of epsilon at which analytic continuation is to be carried %
%        out                                                               %
% mode : Will perform analytic continuation in epsilon if this parameter  %
%        is entered as 'epsilon'.                                          %
%                                                                          %
%****************** OUTPUT VALUES ***********************************%
%
% sol_la      :  Returns the family of solutions that have been stepped up
%                in lambda at initial value of epsilon (usually 0).       %
% sol_ep_temp : Contains the family of solutions at a given epsilon value %
%               for one particular lambda value.                           %
% sol_ep      : Contains the family of solution over the entire range of   %
%                lambda for eosilon value not equal to zero.               %
% norm_la     : contains the 2-norm of each solution in sol_la            %
% norm_ep     : contains the 2-norm of each vector in sol_ep              %
%                                                                          %
%**************************************************************************%

function [sol_la, sol_ep_temp, sol_ep, norm_la,norm_ep] = analytic(q,mode)
%q refers to epsilon value
count=0;
sol_ep_temp=zeros(11,961);
sol_ep = zeros(11,961);
norm_ep = zeros(1,10);
norm_la = zeros(1,10);
f_actual = zeros(961,1);
u_actual = zeros(961,1);
h=1/30;
for i=1:31
    for j=1:31
        count = ((j-1)*31 + i);
        x_i = (i-1)*h;  %setting up the vector with actual solutions!!
                        %solving the given equation subject to the boundary
conditions that
                        %u(x,y) = 0 at all the boundaries, we get u(x,y) =
                        %sin(pi*m*x)sin(pi*n*y).
        y_j = (j-1)*h;
        f_actual(count)= sin(pi*x_i);% to generate the RHS of Au = f
        u_temp(i,j) =1 * (sin(pi*1*x_i)*sin(pi*1*y_j));
        u_actual(count)= u_temp(i,j);
    end
end

var_count = 0;
lambda_o = (pi^2)*(1^2 + 1^2) -0.15;
order=[1:961];
u_initial = u_actual;

for lambda = lambda_o:-0.1:10
```

```
    var_count = var_count + 1;

    [sol_la(var_count,:), J_P, P, J_lambda,J_epsilon] =
newton_new(0.00001,30,lambda,q,u_actual,f_actual);
    delx_delp = Substitute((J_P),order,961,(P*J_lambda));
    u_actual = sol_la(var_count,:)' + (delx_delp)*(-0.1);

end
u_actual = sol_la(var_count,:)';
lambda_o = 10.0892;
var_count=0;

for lambda = lambda_o:0.1:60

    var_count = var_count + 1;

    [sol_la(var_count,:), J_P, P, J_lambda,J_epsilon] =
newton_new(0.00001,30,lambda,q,u_actual,f_actual);
    delx_delp = Substitute((J_P),order,961,(P*J_lambda));
    norm_la(var_count) = 1*norm(sol_la(var_count,:));

    u_actual = sol_la(var_count,:)' + (delx_delp)*(0.1);

end

if(strcmp(mode,'epsilon'))
%implementing analytic continuation in epsilon
var_count=0;
u_actual = sol_la(1,:)';


for epsilon =q:0.1:1
    var_count = var_count+1;

    [sol_ep_temp(var_count,:), J_P, P, J_lambda,J_epsilon] =
newton_new(0.00001,30,lambda_o,epsilon,u_actual,f_actual);
    delx_delq = Substitute((J_P),order,961,(P*J_epsilon));
    %disp(size(delx_delp));
    %disp(size(sol(var_count,:)'));
    u_actual = sol_ep_temp(var_count,:)' + (delx_delq)*(0.1);
end


u_actual = sol_ep_temp(11,:)';
q=1.0;
var_count=0;
lambda_o = 10.0892;

for lambda = lambda_o:0.1:60

    var_count = var_count + 1;
```

```
    [sol_ep(var_count,:), J_P, P, J_lambda,J_epsilon] =
newton_new(0.00001,30,lambda,q,u_actual,f_actual);
    delx_delp = Substitute((J_P),order,961,(P*J_lambda));
    norm_ep(var_count) = norm(sol_ep(var_count,:));
    u_actual = sol_ep(var_count,:)' + (delx_delp)*(0.1);

end
end
end
```

newton_new.m

```
%************************* INPUT PARAMETERS ****************************%
%
% tol        : Tolerance value. Set as 10e-5
% no         : No. of points on the unit grid along x or y direction.
%              Set as 30.
% p          : value of lambda
% q          : value of epsilon
% sol_branch : initial guess
%f_actual    : forcing function
%
%************************ OUTPUT VALUES ********************************%
%
% sol1      : converged solution
% J_P       : LU decomposed Jacobian
% P         : Permutation matrix obtained during LU decomposition.
% J_lambda  : as explained on the first page of Appendix A.
% J_epsilon : as explained on the first page of Appendix A.
%
%***********************************************************************%


function [sol1, J_P,P,J_lambda,J_epsilon] = newton_new(tol, no, p,
q,sol_branch,f_actual)
%p refers to lambda
%q refers to epsilon
h=1/no;
syms x1;
syms x2;
m = no + 1;
N = m^2;
x_i=0;
y_j=0;
u_temp = zeros(m);
branch_count=0;
r=0;
s=0;
count = 0;
order=[1:N];
u_actual = sol_branch;

norm_u=1;
while(norm_u>tol)
```

```
[Jac_u,Resid_u, J_lambda,J_epsilon] =
Jac_Res_new(u_actual,f_actual,h,p,q,m,tol);
[L,U,P] = lu(Jac_u);
del_u_calc = Substitute(lu(P*Jac_u),order,N,(P*Resid_u));
norm_u = norm(del_u_calc,2);
u_t = u_actual;
u_t = u_t + del_u_calc;
u_actual = u_t;
end
sol1 = u_actual;   %returns the converged solution to the analytic routine
J_P = lu(P*Jac_u); %returns the LU decomposed Jacobian to the analytic routine
end
```

Jac_Res_new.m

```
%************************* INPUT PARAMETERS ***************************%
%
% u_input : value of u based on which Jacobian and R(u) is to be
%           constructed
% f_input : forcing function
% h       : grid resolution
% p       : lambda
% q       : epsilon
% m       : (n+1) i.e number of points for finite difference grid +1
%
%************************** OUTPUT VALUES *****************************
%
% J_u       : as explained on the first page of Appendix A.
% R_u       : as explained on the first page of Appendix A.
% J_lambda  : as explained on the first page of Appendix A.
% J_epsilon : as explained on the first page of Appendix A.
%
%**********************************************************************

function[J_u,R_u,J_lambda,J_epsilon] = Jac_Res_new(u_input, f_input, h, p,q,m)

u_input = u_input';
f_input = f_input';
N = m^2;
J_u = zeros(N);
J_lambda = zeros(N,1);
J_epsilon = zeros(N,1);
norm_u = 1;
del_u_calc = zeros(N,1);
R_u = zeros(N,1);

count=0;


for i=1:m
    for j=1:m
        count = ((j-1)*m + i);

        % setting up co-efficients for the nodes at the edges
```

```
          if(i==1 && j==1) %point (1,1)
             J_u(count,count) = 1;
             J_lambda(count) = 0;
             J_epsilon(count)=0;
             R_u(count) = 0;

         elseif (i==1 && j==m) %point(1,m)
            J_u(count,count) = 1;
            J_lambda(count) = 0;
            J_epsilon(count)=0;
            R_u(count) = 0;
         elseif (i==m && j==1) %point(m,1)
             J_u(count,count) = 1;
             J_lambda(count) = 0;
             J_epsilon(count)=0;
             R_u(count) = 0;
          elseif (i==m && j==m) %point(m,m)
             J_u(count,count) = 1;
             J_lambda(count) = 0;
             J_epsilon(count)=0;
             R_u(count) = 0;
         % end of setting up co-efficients for nodes

         %setting up coefficients for points ON the boundary line
         %using central difference

         elseif (i==1 && i~=j && j~=m)
            J_u(count,count) = 1;
            J_lambda(count) = 0;
            J_epsilon(count)=0;
            R_u(count) = 0;
         elseif (i==m && i~=j && j~=1)
             J_u(count,count) = 1;
             J_lambda(count) = 0;
             J_epsilon(count)=0;
             R_u(count) = 0;
         elseif(j==1 && i~=j && i~=m)
             J_u(count,count) = 1;
             J_lambda(count) = 0;
             J_epsilon(count)=0;
             R_u(count) = 0;
         elseif(j==m && i~=j && i~=1)
            J_u(count,count) = 1;
            J_lambda(count) = 0;
            R_u(count) = 0;
            J_epsilon(count)=0;
          else
             R_u(count) = u_input(count)*((-4/h^2)+ p) + p*u_input(count)^2 +
(1/h^2) * (u_input(count-1) + u_input(count+1) + u_input(count+m) +
u_input(count-m)) - q*f_input(count);
             J_u(count,count) = -4/(h^2) + p + 2*p*u_input(count);
             J_u(count,(count-1)) = 1/(h^2);
             J_u(count,(count+1)) = 1/(h^2);
             J_u(count,(count-m)) = 1/(h^2);
             J_u(count,(count+m)) = 1/(h^2);
```

```
            J_lambda(count) = u_input(count) + (u_input(count))^2;
            J_epsilon(count) = -f_input(count);
        end
    end
end
R_u = -R_u;
J_lambda = -J_lambda;
J_epsilon = -J_epsilon;
end
```

arclength.m

```
%********************** INPUT PARAMTERS ****************************** %
%
% q        : epsilon value at which ALC is to be peformed
% lambda_0 : initial value of lambda at some point.
% lambda_1 : initial value of lambda at apoint close to lambda_0
% sol_ep   : solution branch generated by analytic continuation.
%             Used to generate the first initial guess
%
%********************** OUTPUT VALUES ******************************** %
%
% sol_arc        : solution returned by arc-length continuation
% lambda_values  : array containing lambda values at which a converged
%                  solution has been has found.
% norm_la_alc    : stores the 2-norm of each solution in sol_arc
%
%*************************************************************************%

function[sol_arc,lambda_values,norm_la_alc] =
arclength(q,lambda_0,lambda_1,sol_ep)

count=0;

f_actual = zeros(961,1);
u_actual = zeros(961,1);
h=1/30;
for i=1:31
    for j=1:31
        count = ((j-1)*31 + i);
        x_i = (i-1)*h;  %setting up the vector with actual solutions!!
                        %solving the given equation subject to the boundary
conditions that
                        %u(x,y) = 0 at all the boundaries, we get u(x,y) =
                        %sin(pi*m*x)sin(pi*n*y).
        y_j = (j-1)*h;
        f_actual(count)= sin(pi*x_i);% to generate the RHS of Au = f
    end
end

sol0 = sol_ep(109,:)';
sol1 = sol_ep(108,:)';
L=0;
U=0;
P=0;
```

```
order = [1:962];
var_count = 0;
norm_la_alc(1) = norm(sol0);
norm_la_alc(2) = norm(sol1);
lambda_values(1) = lambda_0;
lambda_values(2) = lambda_1;

%Using arc-length continuation ONCE to get a good initial guess u_2 and
%lambda_2

s = sqrt( (lambda_0-lambda_1)^2 + (norm(sol0-sol1))^2 ); %Equation 9
[J_aug,R_aug,J_s] = Jac_Res_ALC(sol0,sol1,f_actual,h,lambda_0,lambda_1,q,31,-
s);


sol_temp = J_aug\J_s;
delu_dels = sol_temp(1:961);
del_la_dels = sol_temp(962);
sol0 = sol1;
sol1 = (sol1 + delu_dels*(-0.5));
lambda_0 = lambda_1;
lambda_1 = lambda_1 + (del_la_dels*(-0.5));

%Beginning of loop for Arc length continuation

while(var_count<80)

var_count = var_count + 1;

[sol_arc(var_count,:),sol_lambda,J_aug,J_s,P] = newton_ALC(sol0,sol1,-
0.5,lambda_0,lambda_1,q,f_actual,0.0001,30);

sol_temp = Substitute((J_aug),order,962,(P*J_s));

delu_dels = sol_temp(1:961);
del_la_dels = sol_temp(962);

sol0 = sol_arc(var_count,:)';
lambda_0 = sol_lambda;
sol1 = (sol1 + delu_dels*(-0.5));
lambda_1 = lambda_1 + (del_la_dels*(-0.5));

lambda_values(2+var_count) = lambda_0;
norm_la_alc(2+var_count) = norm(sol_arc(var_count,:));

end

end
```

newton_ALC.m

```
%************************ INPUT PARAMETERS ***************************** %
%
% sol_branch0 : converged solution(say u0) at an initial lambda value.
% sol_branch1 : guess for a solution(say u1) at another lambda value.
% s           : step size in s.
% p0          : lambda value at sol_branch0
% p1          : initial gues for a lambda corresponding to sol_branch1
% q           : epsilon
% f_actual    : forcing function
% tol         : tolerance. Set at 10e-5
% no          : number of points on the unit grid in either x or y
%               direction. Set as 30 here.
%
%********************* OUTPUT VALUES **********************************%
%
% sol1        : converged value of the solution at given s.
% sol_lambda  : converged value of lambda at iven s.
% J_aug       : as explained on the first page of Appendix A.
% J_s         : as explained on the first page of Appendix A.
% P           : Permutation matrix at the end of the LU decomposition of
%               J_aug.
%
%*************************************************************************%
function [sol1, sol_lambda, J_aug,J_s, P] =
newton_ALC(sol_branch0,sol_branch1,s,p0,p1, q, f_actual,tol, no)
%p refers to lambda
%q refers to epsilon
h=1/no;
m = no + 1;
N = m^2;
count = 0;
order=[1:(N+1)];
u_actual0 = sol_branch0;
u_actual1 = sol_branch1;
norm_u=1;
norm_la = 1;
while(norm_u>tol && norm_la>tol) % to ensure convergence in both lambda AND u

[J_aug,R_aug, J_s] = Jac_Res_ALC(u_actual0,u_actual1,f_actual,h,p0,p1,q,m,s);
[L,U,P] = lu(J_aug);
sol = Substitute(lu(P*J_aug),order,(N+1),(P*R_aug));
del_u = sol(1:961);
del_la = sol(962);
norm_u = norm(del_u,2);
norm_la = norm(del_la);
u_t = u_actual1;
u_t = u_t + del_u;
u_actual1 = u_t;
```

```
p1 = p1 + del_la;
end
sol1 = u_actual1;
sol_lambda = p1;
J_aug = lu(P*J_aug); %returns the LU decomposed Jacobian to the ALC method
end
```

Jac_Res_ALC.m

```
%************************ INPUT PARAMETERS ******************************%
%
% u_input0 : one value of solution.
% u_input1 : another value of the solution.
% f_input  : forcing function
% p0       : lambda value corresponding to u_input0
% p1       : lambda value corresponding to u_input1
% q        : epsilon value at which ALC is performed
% m        : (n+1) i.e number of points for finite difference grid +1
% s        : step size in s.
%
%************************ OUTPUT VALUES *********************************%
%
% J_aug : as explained on the first page of Appendix A.
% R_aug : as explained on the first page of Appendix A.
% J_s   : as explained on the first page of Appendix A.
%
%************************************************************************%


function[J_aug,R_aug,J_s] = Jac_Res_ALC(u_input0,u_input1, f_input, h,
p0,p1,q,m,s)

u_input0 = u_input0';
u_input1 = u_input1';
f_input = f_input';
N = m^2;
J_u = zeros(N);
J_lambda = zeros(N,1);
J_epsilon = zeros(N,1);
J_s = zeros(N+1,1);
R_u = zeros(N,1);
eta_u = zeros(1,N);

count=0;

for i=1:m
    for j=1:m
        count = ((j-1)*m + i);

        % setting up co-efficients for the nodes at the edges
         if(i==1 && j==1) %point (1,1)
            J_u(count,count) = 1;

            R_u(count) = 0;

        elseif (i==1 && j==m) %point(1,m)
```

```
            J_u(count,count) = 1;

            R_u(count) = 0;
        elseif (i==m && j==1) %point(m,1)
            J_u(count,count) = 1;

            R_u(count) = 0;
         elseif (i==m && j==m) %point(m,m)
            J_u(count,count) = 1;

            R_u(count) = 0;
        % end of setting up co-efficients for nodes

        %setting up coefficients for points ON the boundary line
        %using central difference.

        elseif (i==1 && i~=j && j~=m)
            J_u(count,count) = 1;

            R_u(count) = 0;
        elseif (i==m && i~=j && j~=1)
            J_u(count,count) = 1;

            R_u(count) = 0;
        elseif(j==1 && i~=j && i~=m)
            J_u(count,count) = 1;

            R_u(count) = 0;
        elseif(j==m && i~=j && i~=1)
            J_u(count,count) = 1;
            R_u(count) = 0;

         else
            R_u(count) = u_input1(count)*((-4/h^2)+ p1) + p1*u_input1(count)^2
+ (1/h^2) * (u_input1(count-1) + u_input1(count+1) + u_input1(count+m) +
u_input1(count-m)) - q*f_input(count);
            J_u(count,count) = -4/(h^2) + p1 + 2*p1*u_input1(count);
            J_u(count,(count-1)) = 1/(h^2);
            J_u(count,(count+1)) = 1/(h^2);
            J_u(count,(count-m)) = 1/(h^2);
            J_u(count,(count+m)) = 1/(h^2);

            J_lambda(count) = u_input1(count) + (u_input1(count))^2;
            eta_u(count) = -2*(u_input1(count) - u_input0(count));
        end
     end
end

eta = (s)^2 - (norm(u_input1-u_input0))^2 - (p1-p0)^2;
eta_lambda = -2*(p1-p0);
eta_u = cat(2,eta_u,eta_lambda);
J_u = cat(2,J_u,J_lambda);
J_aug = cat(1,J_u,eta_u);
R_aug = -1* cat(1,R_u,eta);
```

```
%%J_aug and J_s are needed for the arc-length continuation method
J_s(N+1) = 2*s;
J_s = -1* J_s;

end
```

Substitute.m

```
%%%%%%%%%%%%%%%%%%%%%% INPUT PARAMETERS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%A - The LU decomposed version of the A in Ax = b                              %
%O - Array of size [1 x n] that contains info about row interchanges           %
%n - dimension of A                                                           %
%b - RHS of the equation Ax = b                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%% OUTPUT PARAMETERS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%sol - A [1 x n] array that contains the solution x of Ax=b                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [sol] = Substitute (A,O,n,b)
x = zeros(1,n);
%This part of the code takes care of forward substitution
for i =2:n
    sum = b(O(i));
    for j = 1 : (i-1)
        sum = sum - A(O(i),j)*b(O(j));
    end
    b(O(i)) = sum;
end
%End of forward substitution routine
x(n) = b(O(n))/A(O(n),n);

%This part does the backward substitution
for i = (n-1) : -1 : 1
    sum = 0;
    for j = (i+1) : n
        sum = sum + A(O(i),j)*x(j);
    end
    x(i) = (b(O(i)) - sum)/A(O(i),i);
end
%End of backward substitution
sol = x;

end
```

APPENDIX B (Analytic Continuation Plots)

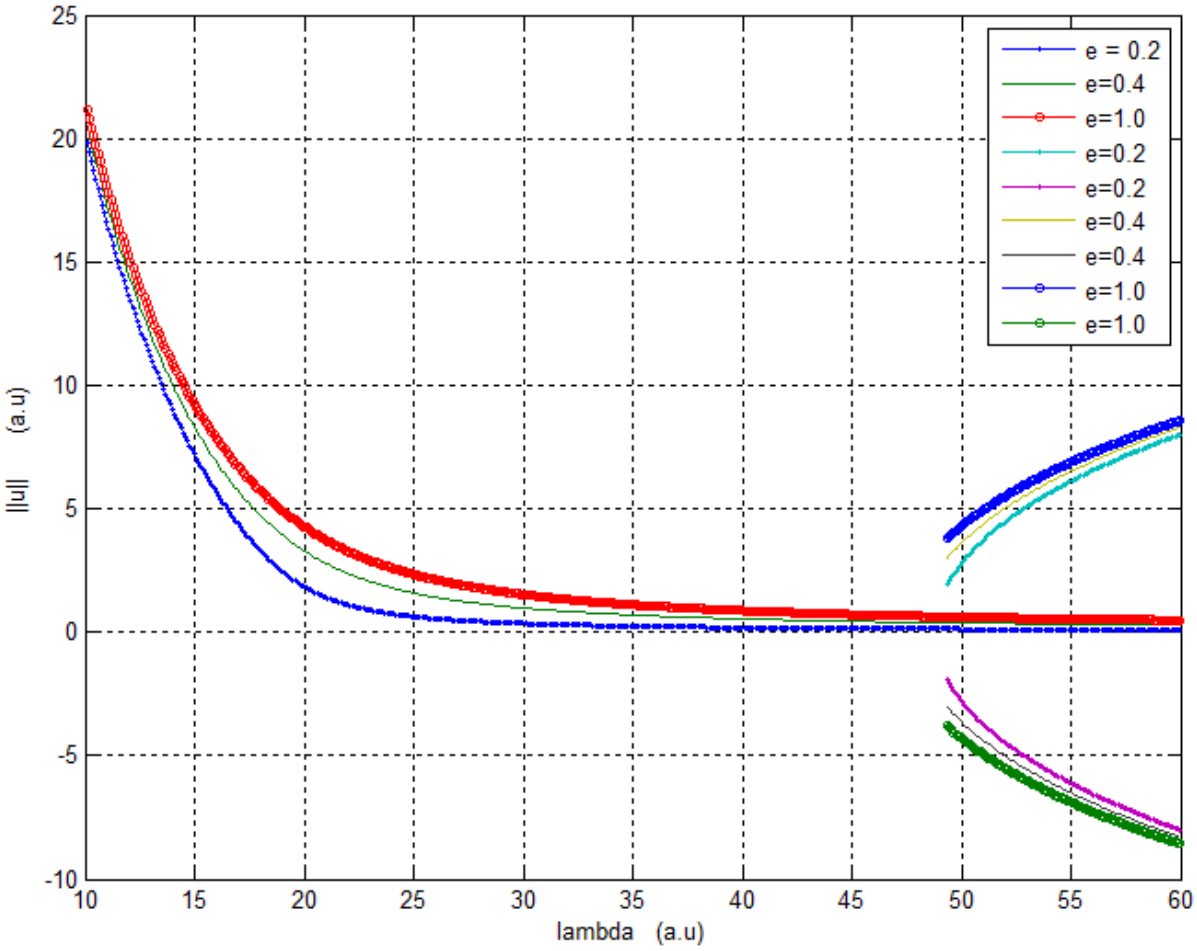

Figure B1 : Graph with plots of ||u|| vs λ for various values of ϵ. Generated using Analytic continuation

Region 1 ($0 \leq \lambda \leq 2\pi^2$) : (1,1) Solution branch

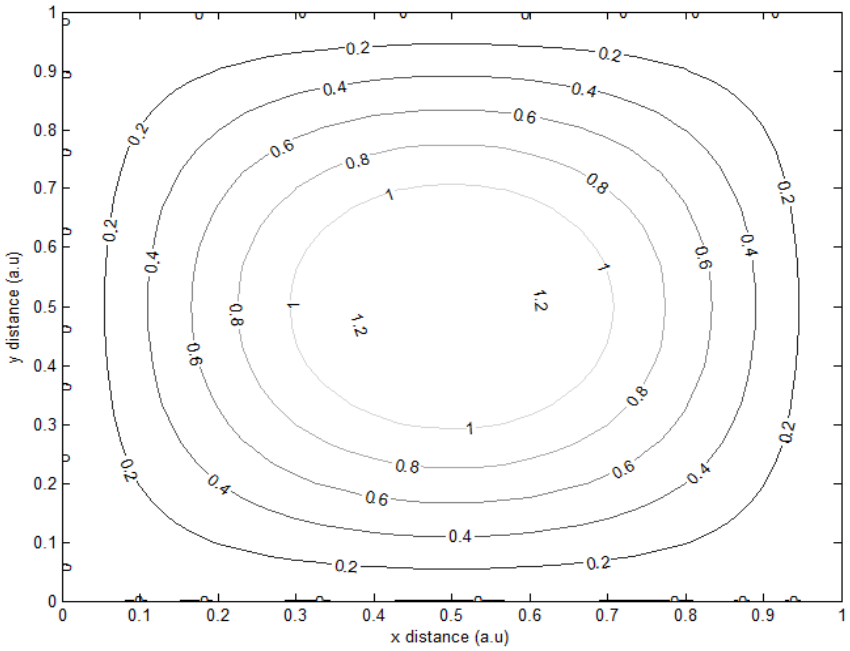

Figure B2 : Contour Plot showing solution structure in Region 1 of Figure 1

Region 2 ($2\pi^2 \leq \lambda \leq 60$) : (1,1) Solution branch

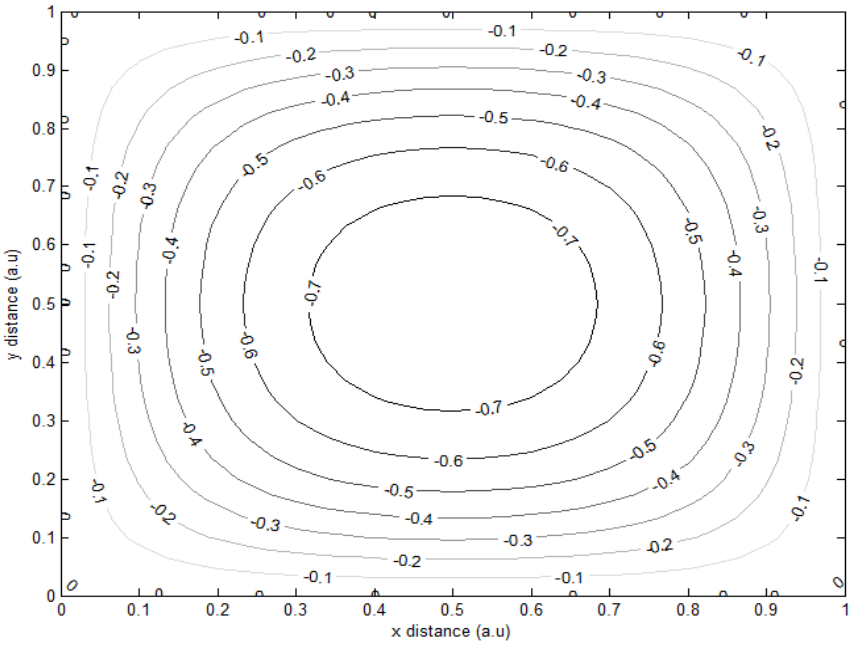

Figure B3 : Contour Plot showing solution structure in Region 2 of Figure 1

Region 3 ($5\pi^2 \leq \lambda \leq 60$) : (1,2) Solution branch

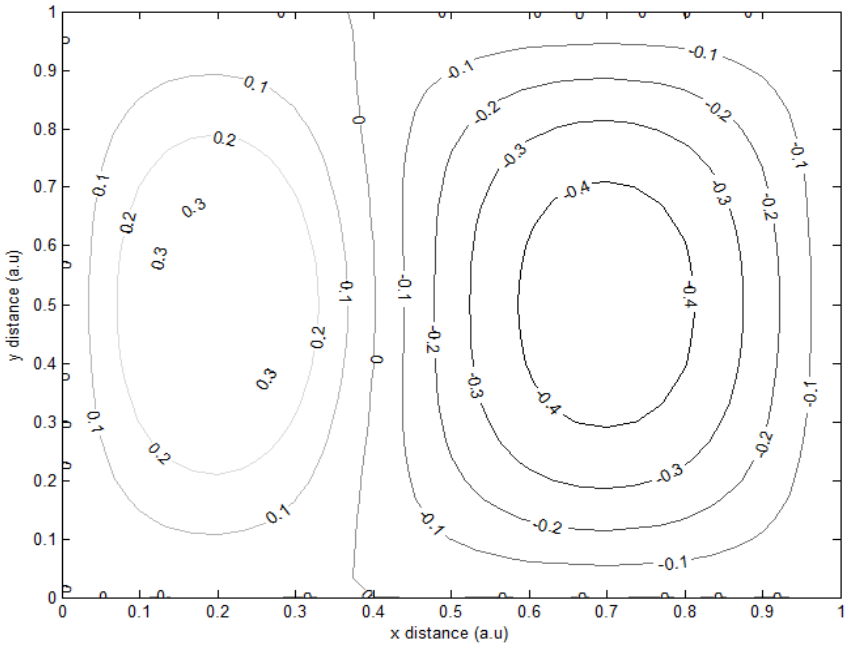

Figure B4 : Contour Plot showing solution structure in Region 3 of Figure 1

Region 4 ($5\pi^2 \leq \lambda \leq 60$) : (1,2) Solution branch

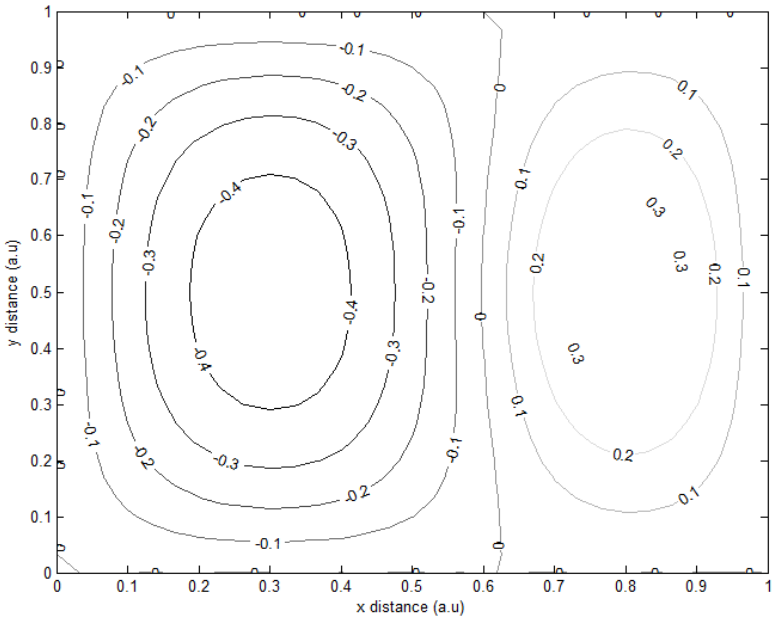

Figure B5 : Contour Plot showing solution structure in Region 4 of Figure 1

Region 5 ($5\pi^2 \leq \lambda \leq 60$) : (2,1) Solution branch

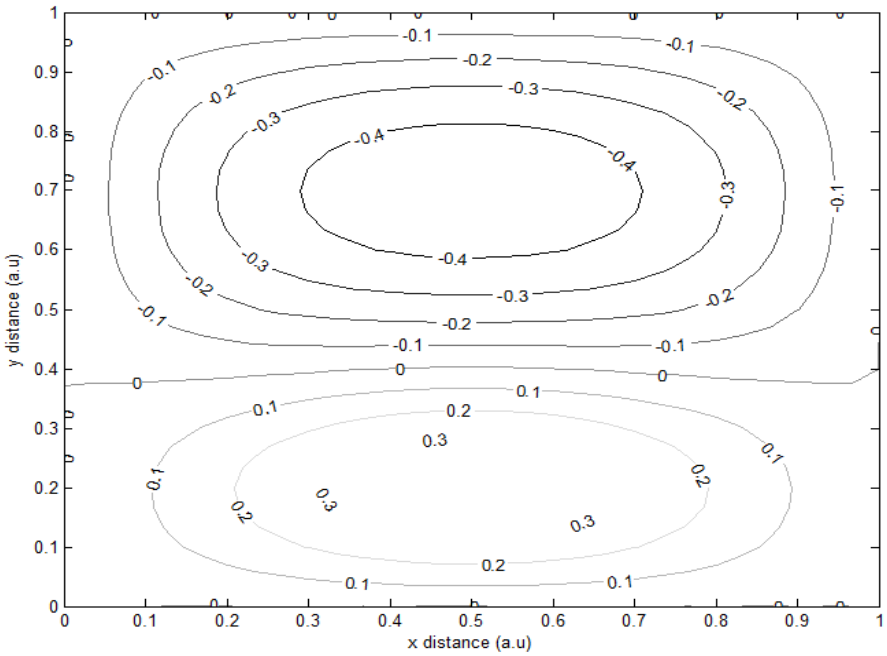

Figure B6 : Contour Plot showing solution structure in Region 5 of Figure 1

Region 6 ($5\pi^2 \leq \lambda \leq 60$) : (2,1) Solution branch

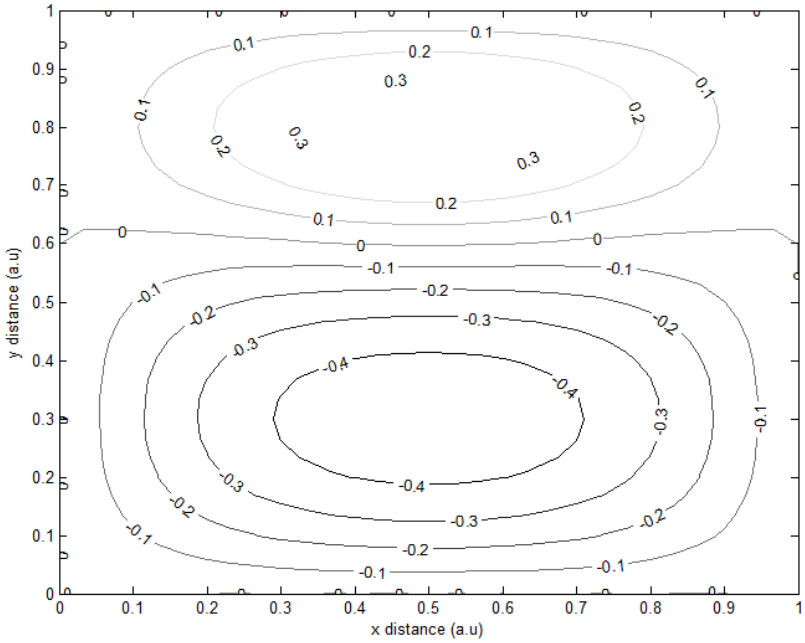

Figure B7 : Contour Plot showing solution structure in Region 6 of Figure 1

APPENDIX C (ALC plots)

Region 1 (0≤λ≤60) : (1,1) Solution branch

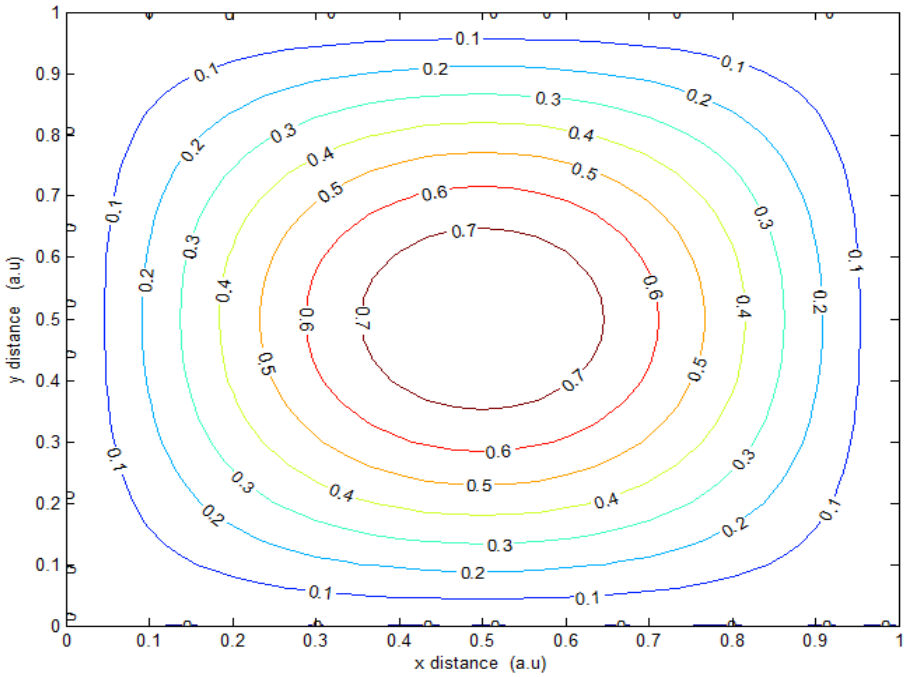

Figure C1 : Contour plot showing solution structure in Region 1 of Figure 3

Region 2 ($5\pi^2$≤λ≤60) : (1,2) Solution branch

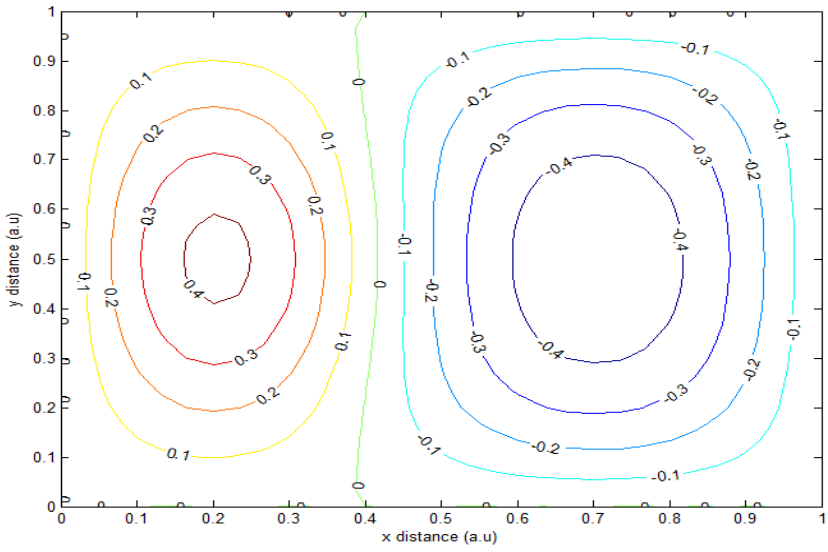

Figure C2 : Contour plot showing solution structure in Region 2 of Figure 3

Region 3 ($5\pi^2 \leq \lambda \leq 60$) : (1,2) Solution branch

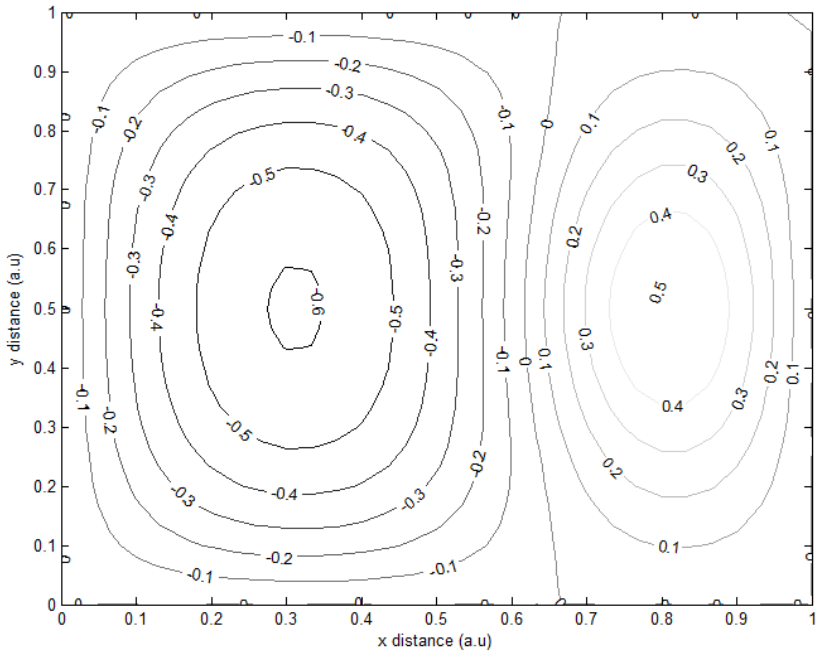

Figure C3 : Contour plot showing solution structure in Region 3 of Figure 3

Region 4 ($5\pi^2 \leq \lambda \leq 60$) : (2,1) Solution branch

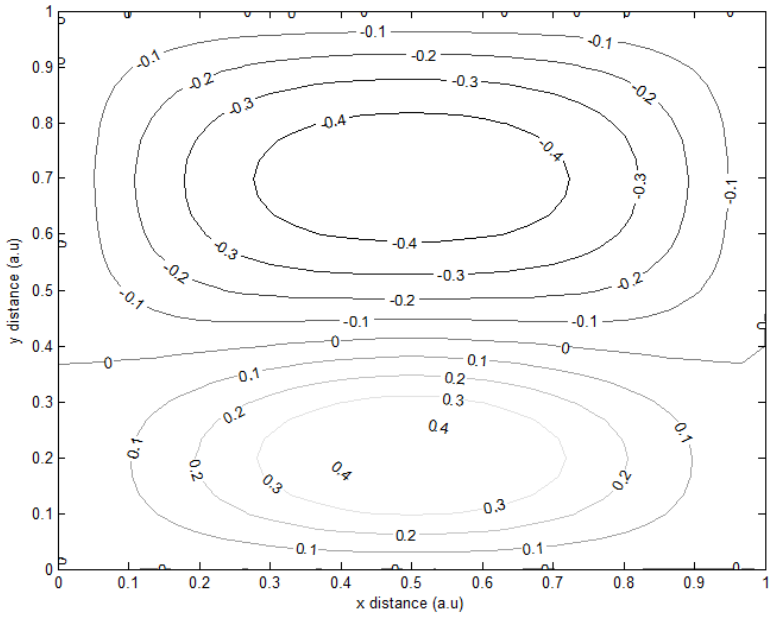

Figure C4 : Contour plot showing solution structure in Region 4 of Figure 3

Region 5 ($5\pi^2 \leq \lambda \leq 60$) : (2,1) Solution branch

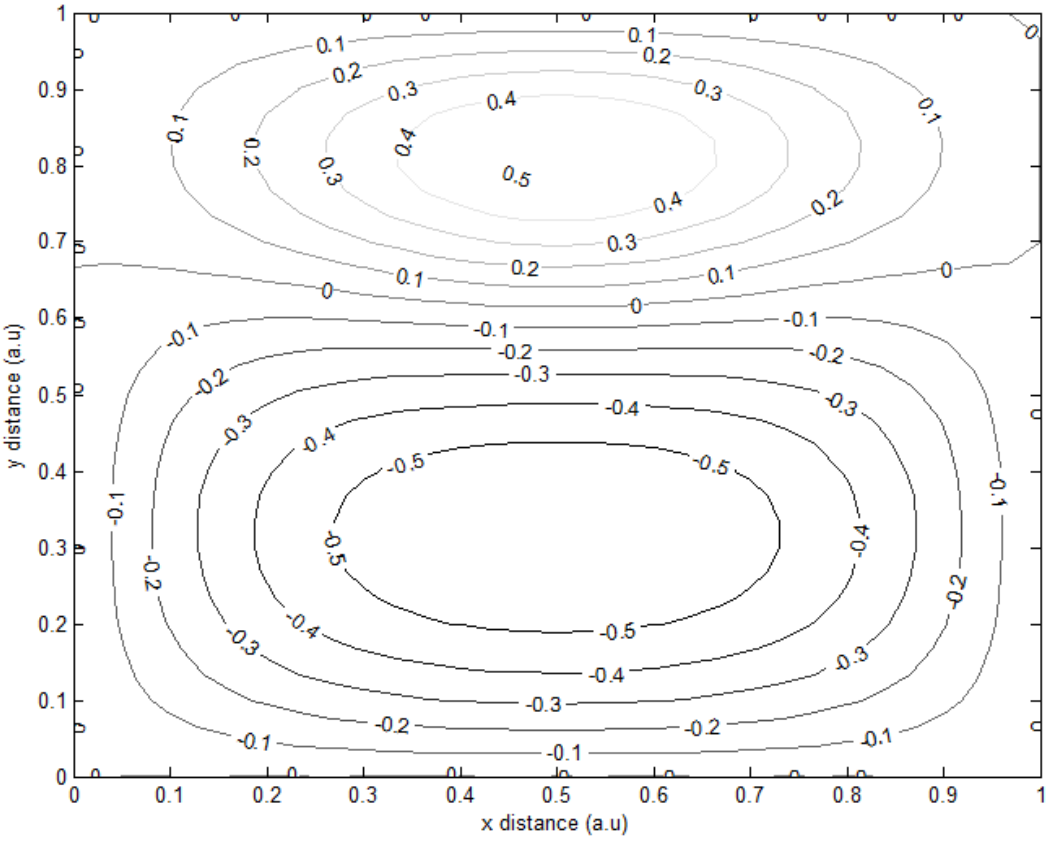

Figure C5 : Contour plot showing solution structure in Region 5 of Figure 3