Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER Computers and Mathematics with Applications 51 (2006) 1021–1046
An International Journal
computers & mathematics
with applications
www.elsevier.com/locate/camwa

Interpolation for Nonlinear BVP in Circular Membrane with Known Upper and Lower Solutions

S. K. SEN
Department of Mathematical Sciences
Florida Institute of Technology
150 West University Boulevard
Melbourne, FL 32901-6975, U.S.A.
sksen@fit.edu

H. AGARWAL
Energy Research Center
Lehigh University
117 ATLSS Drive
Bethlehem, PA 18018, U.S.A.
haa6@lehigh.edu

T. SAMANTA
Department of Mathematical Sciences
Florida Institute of Technology
150 West University Boulevard
Melbourne, FL 32901-6975, U.S.A.
tsamanta@fit.edu

(Received and accepted September 2005)

Abstract—A successive nonextrapolatory linear interpolation is described to solve a singular two-point boundary value problem arising in circular membrane theory. The problem is associated with a second-order nonlinear ordinary differential equation for which the upper and lower bounds of the solution is analytically established/known. The importance and the scope of these bounds in solving the problem is stressed. Also depicted graphically are the lower and upper solutions as well as the true and iterated solutions. In addition, discussed are the reasons why linear interpolation, and not nonlinear interpolation or bisection which are possible procedures, has been employed. © 2006 Elsevier Ltd. All rights reserved.

Keywords—Circular membrane, Linear interpolation, Nonlinear ordinary differential equation, Two-point boundary value problem.

0898-1221/06/$ - see front matter
doi:10.1016/j.camwa.2005.09.004

1. INTRODUCTION

THE PROBLEMS. The equation for a circular membrane under normal uniform pressure may be written in a simplified form as [1–4],

$$D^2y + \left(\frac{3}{x}\right) Dy + \frac{k}{y^2} = 0, \qquad 0 < x < 1, \tag{1}$$

where $D \equiv \frac{d}{dx}$, $D^2 \equiv \frac{d^2}{dx^2}$, $k > 0$ is a constant, x is the radial coordinate and $y(x)$ the radial stress. At the center (for symmetry), i.e., at $x = 0$,

$$Dy(0) = 0. \tag{2}$$

At the edge, i.e., at $x = 1$, we have the condition,

$$y(1) = \lambda > 0, \tag{3.1}$$

where λ is a constant or the condition,

$$Dy(1) + (1 - v)y(1) = 0, \qquad 1 - v > 0. \tag{3.2}$$

If we now introduce the change of variable, in equation (1), $x = 1/t$, then we get

$$D^2y - \left(\frac{1}{t}\right) Dy + \left(\frac{k}{t^4}\right)\left(\frac{1}{y^2}\right) = 0, \qquad 1 < t < \infty, \tag{4}$$

as an infinite interval boundary value problem, where $y(1) = \lambda > 0$ and the value of $Dy(1)$ is such that $y(t)$ remains bounded. Or, more precisely, $Dy(\infty) = 0$.
We focus on the numerical solution of the following singular nonlinear boundary value problems using known analytically derived lower and upper solutions [1], where $v_1 = 1 - v$ and $v_3 = 3 - v$.

Problem P1 : ODE (1) with BCs (2), (3.1), and $y(x) \in [\lambda,\ (k/(8\lambda^2))(1 - x^2) + \lambda]$.
Problem P2 : ODE (1) with BCS (2), (3.2), and

$$y(x) \in \left[\left(\frac{kv_1^2}{v_3^2}\right)^{1/3} \frac{((v_3/v_1) - x^2)}{2},\ \left(\frac{kv_1^2}{32}\right)^{1/3} \left(\left(\frac{v_3}{v_1}\right) - x^2\right)\right].$$

Problem P3: ODE (4) with BCs (3.1), $Dy(\infty) = 0$, and $y(t) \in [\lambda,\ \lambda + ((k/(8\lambda^2))(1 - (1/t^2))]$.

Problems P1 and P3 are same except that P3 is derived by the nonlinear transformation $x = 1/t$ from P1. The pros and cons of such a transformation (i.e., to which extent the transformation helps) in the realm of numerical computation/graphical representation of solution are discussed to enable one to select one of the Problems P1 and P3 for the purpose of computation and consequent interpretation of the solution. It may be seen that the foregoing lower and upper solutions within the square brackets are not essential to solve the aforesaid three problems. The knowledge of the reasonably narrow bounds imposed by the lower and upper solutions, however, is useful, particularly in sensitive problems, i.e., the problems where the function $y(x)$ or $y(t)$ are violently fluctuating [5].

THE PROCEDURE. The numerical solution procedures for a general two-point boundary value problem (BVP) associated with ordinary differential equations (ODEs)—linear or nonlinear, coupled or not—have already been discussed [6,7]. In the procedure for nonlinear ODEs, we have chosen arbitrarily numerical values for the dependent variable and its derivatives so as to convert the BVP to the initial value problem (IVP). The nonlinear problem is then solved iteratively

using the linear interpolation/mean-value theorem. This interpolation does not compulsorily insist that extrapolation should be excluded. Such a procedure may fail to solve sensitive/singular problems although, for well-behaved problems, the procedure is good enough. The nonlinear singular ODE under study is such an example where the foregoing procedure is most often likely to fail if we do not have reasonably a good idea (narrow bounds) about the numerical values of the variable and/or its derivative(s). One may obtain such an idea/bound through the physical problem/laboratory experiments and/or through not-too-wide mathematically derived bounds. In the successive nonextrapolatory linear interpolation procedure (SNLIP) presented here, we have made use of the lower and upper bounds of the solution of the ODE already derived [1]. Interestingly we, through our numerical experiments, discovered that such bounds are indeed useful for ODEs similar to the one to be solved. In fact, if we do not make use of the bounds, we might end up searching vast real region without hitting at the true initial value(s) of the given BVP. This is so particularly when the singular nonlinear ODE is sensitive.

In Section 2, we specify the lower and upper solutions for the problems and provide the MATLAB programs for the reader to check readily the lower and upper solutions just by copying, pasting, and executing the MATLAB programs.. The SNLIP is described to solve the problems in Section 3. MATLAB programs for SNLIP along with numerical examples and the concerned graphical presentation of lower, upper, iterated solutions, and true solution are presented in Section 4. The graphs readily demonstrate how the iterated and true solutions crawl through the path bounded by upper and lower solutions. Section 5 includes conclusions that specifically focus on the importance and use of linear interpolation rather than nonlinear interpolation and bisection.

2. LOWER AND UPPER SOLUTIONS

The lower solution $\alpha(x)$ and upper solution $\beta(x)$, derived mathematically in [1] for Problem P2 are, observing that $v_1 = 1 - v$, $v_3 = 3 - v$,

Lower solution:

$$\alpha(x) = 0.5\left[K\left(\frac{v_1^2}{v_3^2}\right)\right]^{1/3}\left[\left(\frac{v_3}{v_1}\right) - x^2\right], \tag{5}$$

Upper solution:

$$\beta(x) = \left[M\left(\frac{v_1^2}{32}\right)\right]^{1/3}\left[\left(\frac{v_3}{v_1}\right) - x^2\right], \tag{6}$$

where $0 < K \leqslant q(x) \leqslant M$ for $0 \leqslant x \leqslant 1$, $q(x)$ being a continuous function.

We may choose, for instance, $K = k = 1$, $M = k = 1$ when we allow $q(x) = 1$ (constant). Or, we may choose $K = k = 0.1$, $M = 1$ for $q(x) = x$. The specific bounds defined in Problems P1, P2, and P3 in Section 1 conform to the former (restricted) choice, i.e., the choice where $K = k$, $M = k$. We may allow x to vary from 0_+ (say, $x = 0.0001$ since the ODE is singular and the point of singularity is at $x = 0$) to 1_- (say, $x = 0.9991$ since the ODE is defined for $x < 1$) with a step-size $h = 0.0333$ (say). For numerous possible values of these parameters, the linear interpolation procedure presented here is applicable.

The lower and upper solutions using the MATLAB program [5] for Problem P2 with the former choice may be given as below.

```
function [ ] = lbubsol_1( )
n = 0; K = 1; M = 1; v = 0.1 : 0.2 : 0.9; x = 0 : 0.1 : 1;
fprintf('\n The solution bounds when K = 1 and M = 1 \n')
for i = 1:5
  fprintf('\n   v      x      lowerbound   upperbound \n')
  for j = 1:11
    xx = x(j); vv = v(i); v1 = 1-vv; v3 = 3-vv; Z = (v3/v1) - xx^2;
```

```
    alpha = 0.5*((K*v1^2)/v3^2)^(1/3)*(Z); lbsol = alpha;
    beta = ((M*v1^2)/32)^(1/3)*(Z); ubsol = beta;
    fprintf('%7.4f %7.4f %7.4f %7.4f \n', vv, xx, lbsol, ubsol)
    n = n + 1;
  end
end
```

and issuing the MATLAB command

```
>> lbubsol_1
```

in the command window would be as given partially (to conserve space) below.

The solution bounds when $K = 1$ and $M = 1$ (for $q(x) = 1$).

v	x	lowerbound	upperbound	v	x	lowerbound	upperbound
0.1000	0.0000	0.7385	0.9461	0.3000	0.0000	0.7841	0.9578
0.1000	0.1000	0.7362	0.9432	0.3000	0.1000	0.7821	0.9553
0.1000	0.2000	0.7293	0.9343	0.3000	0.2000	0.7760	0.9479
.				.			
.				.			
0.1000	0.8000	0.5918	0.7582	0.3000	0.8000	0.6540	0.7989
0.1000	0.9000	0.5529	0.7083	0.3000	0.9000	0.6195	0.7567
0.1000	1.0000	0.5093	0.6525	0.3000	1.0000	0.5808	0.7095
v	x	lowerbound	upperbound				
0.5000	0.0000	0.8550	0.9921	0.7000	0.8000	0.9036	0.9919
0.5000	0.1000	0.8533	0.9901	0.7000	0.9000	0.8817	0.9679
0.5000	0.2000	0.8481	0.9842	0.7000	1.0000	0.8573	0.9410
.							
.				v	x	lowerbound	upperbound
0.5000	0.8000	0.7455	0.8651	0.9000	0.0000	1.3795	1.4251
0.5000	0.9000	0.7165	0.8314	0.9000	0.1000	1.3788	1.4244
0.5000	1.0000	0.6840	0.7937	0.9000	0.2000	1.3768	1.4224
v	x	lowerbound	upperbound	.			
0.7000	0.0000	0.9859	1.0822	.			
0.7000	0.1000	0.9846	1.0808	0.9000	0.8000	1.3374	1.3816
0.7000	0.2000	0.9808	1.0765	0.9000	0.9000	1.3263	1.3701
.				0.9000	1.0000	1.3138	1.3572
.				>>			

Replacing $K = 1$ by $K = 0.1$ in the second and third lines of the foregoing MATLAB program and retaining the rest of the program and the command identical, we obtain, for Problem P2, the following.

The solution bounds when $K = 0.1$ and $M = 1$ (if $q(x) = x$).

v	x	lowerbound	upperbound	v	x	lowerbound	upperbound
0.1000	0.0000	0.3428	0.9461				
0.1000	0.1000	0.3417	0.9432	0.5000	0.7000	0.3580	0.8949
0.1000	0.2000	0.3385	0.9343	0.5000	0.8000	0.3461	0.8651
.				0.5000	0.9000	0.3326	0.8314
.				0.5000	1.0000	0.3175	0.7937
0.1000	0.8000	0.2747	0.7582				
0.1000	0.9000	0.2566	0.7083	v	x	lowerbound	upperbound
0.1000	1.0000	0.2364	0.6525	0.7000	0.0000	0.4576	1.0822
				0.7000	0.1000	0.4570	1.0808
v	x	lowerbound	upperbound	0.7000	0.2000	0.4552	1.0765
0.3000	0.0000	0.3640	0.9578	.			
0.3000	0.1000	0.3630	0.9553	.			
0.3000	0.2000	0.3602	0.9479	0.7000	0.8000	0.4194	0.9919

.				0.7000	0.9000	0.4093	0.9679
.				0.7000	1.0000	0.3979	0.9410
0.3000	0.8000	0.3036	0.7989	v	x	lowerbound	upperbound
0.3000	0.9000	0.2875	0.7567	0.9000	0.0000	0.6403	1.4251
0.3000	1.0000	0.2696	0.7095	0.9000	0.1000	0.6400	1.4244
v	x	lowerbound	upperbound	0.9000	0.2000	0.6391	1.4224
0.5000	0.0000	0.3969	0.9921	.			
0.5000	0.1000	0.3961	0.9901	.			
0.5000	0.2000	0.3937	0.9842	0.9000	0.8000	0.6208	1.3816
.				0.9000	0.9000	0.6156	1.3701
.				0.9000	1.0000	0.6098	1.3572
0.5000	0.6000	0.3683	0.9207	>>			

The purpose to depict the lower and upper solutions is to focus on the fact that the lower-upper solution bounds are reasonably small which is desirable and fruitfully usable for two-point boundary value problems.

Evidently, the lower and upper solutions in equations (5) and (6) are usable for Problem P2, *viz.*, that with ODE (1) with BCs (2) and (3.2) and not for Problems P1 and P3.

3. SUCCESSIVE NONEXTRAPOLATORY LINEAR INTERPOLATION PROCEDURE (SNLIP)

The nonlinear two-point BVP associated with the second order ODE (1) having BCs (2) and (3.1) is solved as a system of two first-order ODEs iteratively as initial value problems (IVPs). The iteration uses the lower and upper solution bounds within which the solution is known mathematically to be present and performs successive linear extrapolation-free interpolations (SNLIs) to convert the BVP to an equivalent IVP. That is the IVP whose solution is the same as that of the BVP. We assume that the mathematically derived lower and upper solution bounds bracket a reasonably narrow (small) interval. Such an assumption is not essential. In case the bounds bracket a wide (large) interval, the chosen number of subintervals in each of which successive linear interpolations are employed will be larger needing more computing time. The narrow interval is divided into subintervals of equal size. In each subinterval, SNLIs are used. At least (possibly only one) one subinterval will bracket the required unknown initial condition which is accurately computed by SNLIs.

The ODE (1) with BCs (2) and (3.1), for notational convenience, can be written as, permitting the symbol $\approx$ to denote "representing"[1]

$$D^2y + ADy + By = C, \tag{7}$$

$$BCs: \text{ at } x = a = 0.0001\ (\approx 0_+),\ Dy = 0 \text{ while at } x = b = 0.9991(\approx 1_-),\ v_1 y + Dy = 0$$

where $v_1 = 1 - v$, $A = 3/x$, $B = K/y^3$, $C = 0$ are, in general, functions of x and y. Suppose $v = 0.5$ ($K = 0.1$, $M = 1$). Let at $x = a = 0.0001$, $y = y_a = \theta_1$ (some arbitrary numerical value chosen here as the lower solution $y = \theta_1 = 0.3969$). Also, at $x = a = 0.0001$, $y = y_a = \theta_2$ (another arbitrary numerical value chosen here as the upper solution $y = \theta_2 = 0.9921$.

There are two ways by which we can scan the whole region (interval), *viz.*, upper solution − lower solution $= \theta_2 - \theta_1$. One way is to consider the whole interval $\theta_2 - \theta_1$ as one subinterval while the other way is to treat/divide the whole interval $\theta_2 - \theta_1$ as a sequence of several adjacent subintervals. The former way would work fine as long as the function $y(x)$ is not violently

[1]Although the symbol $\approx$ denotes "is essentially equal to" [8,9], we use this symbol to denote "representing". This usage does not have the significant potential to mislead a reader since a symbol 1_- represents anyone of infinite possible values sufficiently close (in the given context) to 1 and strictly less than 1.

fluctuating in the whole interval and the specified boundary condition is satisfied at only one point of the interval. The later way, on the other hand, is more general and can take care of reasonably violent fluctuating functions; also, if the specified boundary condition is satisfied at more than one point of the interval, the second way with appropriate choice of number of subintervals would provide all the possible solutions of the two-point BVP associated with the ODEs. Although in the circular membrane equation problem solved here, the former way which may be considered as a particular case of the later way is good enough, we present here the more general second way, *viz.*, the multiple subinterval way in which the whole interval is scanned subinterval by subinterval much more meticulously, and obviates the chance of missing a possible solution of the BVP. We provide a MATLAB program for the multiple subinterval case for the reader to work out/check solutions for various possible parametric values for the circular membrane problem.

For the sake of explaining the SNLI procedure, we consider, without introducing any confusion, the whole interval $\theta_2 - \theta_1$ as one subinterval. Let

$$v_1 y_b + D y_b = F_1 \text{ at } x = b = 1_- \text{ with } y_a = \theta_1, \tag{8}$$

$$v_1 y_b + D y_b = F_2 \text{ at } x = b = 1_- \text{ with } y_a = \theta_2, \tag{9}$$

be of opposite signs[2]. Then, we compute the refined value of θ using the linear interpolation/mean-value theorem as

$$\theta_r = \theta_1 - [(\theta_1 - \theta_2)F_1/(F_1 - F_2)]. \tag{10}$$

Take $y = y_a = \theta_r$ and find

$$v_1 y_b + D y_b = F_r, \qquad \text{at } x = b = 1_-, \qquad \text{with } y_a = \theta_r, \tag{11}$$

and note its sign. We then take F_1, θ_1 from equation (8) or F_2, θ_2 from equation (9) and F_r, θ_r from equation (11) such that the F are of opposite signs. We continue this procedure iteratively until we get the desired accuracy subject to the precision of the computer/program used [10]. Having known the sufficiently accurate θ, the BVP becomes/gets converted to an IVP which is automatically solved in the iterative process. In fact, any IVP associated with a system of ODEs, linear or nonlinear, coupled or noncoupled can always be solved by any step-by-step procedure noniteratively.

4. MATLAB PROGRAM FOR SNLIP AND NUMERICAL EXAMPLES

Consider the Problem P1: $y'' + k/y^2 + (3/x)y' = 0$, $0 < x < 1$ with boundary conditions $y'(0) = 0$, $y(1) = \lambda > 0$(λ is chosen as 0.4), where $\lambda \leqslant y(x) \leqslant (k/8\lambda^2)\,(1 - x^2) + \lambda$. Observe that Problem P1 is free from parameter v but depends on λ. The MATLAB program for this problem, when $k = 0.1$ is as follows.

```
%File name to be used for execution of Problem P1 is lbubsol_xkp1problem1
%since we have kept the following FIRST FILE lbubsol_xkp1problem1 under this
%name; the FIRST (main) FILE uses the SECOND file bvplbubsolbased1problem1
%which in turn uses the THIRD FILE ydash

function [ ] = lbubsol_xkp1problem1( )
k = 0.1; lmda = 0..4;
% k needs to be numerically changed everywhere in this and the third files
```

[2]Although it is not essential that the F_1 and F_2 at $x = b$ would be of opposite signs, i.e., an extrapolatory interpolation could also work if the solution of the BVP is not violently fluctuating, it would be useful/necessary to insist on the opposite signs to ensure bracketing a solution for a rapidly fluctuating function $y(x)$.

```
%while lmda needs to be changed in this and the second files when required.
x = .0001 : .0333 : .9995;
fprintf('\n   v      x      lowerbound   upperbound \n')
%The foregoing k = 0.1 should be changed as and when we change k.
  fprintf('\n x lowerbound upperbound \n')
  for j = 1:31
    xx = x(j);
    alpha=lmda; beta = (k/(8*lmda^2))*(1-x(j)^2)+lmda;
    lbsol = alpha; lbsoln(j)=lbsol; ubsol = beta; ubsoln(j)=ubsol;
    fprintf('%7.4f %7.4f %7.4f \n', xx, lbsol, ubsol)
  end
  fprintf('\n Lower soln at x=0+ Upper soln at x=0+ \n')
  fprintf('%7.4f %7.4f \n', lbsoln(1), ubsoln(1)),

  [Y]=bvplbubsolbased1problem1(lbsoln(1), ubsoln(1));
fprintf('\n Col 1: x, Cols 2, 3, .., last but one: successive iterated sols y(x), Col
last: true sol y(x) \n')
  Y1=[x' Y],
    s=size(Y); Yfcindex = s(:, 2); true_sol = Y(:, Yfcindex);

  plot(x', lbsoln, 'r+', x', ubsoln, 'bx', x', true_sol, 'ko', x', Y);
  xlabel('x'); ylabel('Solution y(x) for ODE D^2y + k/y + 3Dy/x=0');
  title('Convergence of solns y(x) between lower and upper solns');
  legend('Lower soln','Upper soln', 'True soln', 'Iterated solns');

%SECOND FILE bvplbubsolbased1problem1
%subprogram bvplbubsolbased1problem1(lowersol, uppersol)
function[Y]=bvplbubsolbased1problem1(lowersol, uppersol);

xspan=.0001:.0333:.9991; k=1; lmda =0.4;
% k here is an integer variable, has nothing to do with the same
%symbol k used inside the subsubprogram ydash.
%Value of lmda MUST be numerically changed in the foregoing
%line as well as main and other subprogram (wherever it occurs).

xspansize=size(xspan); nxstep=xspansize(1, 2);
interval_of_sol=uppersol-lowersol;
if interval_of_sol<=3,
  %In the given context, if the interval of upper and lower solutions
  %is too large then such an interval may not be of much use. In the
  %current membrane problem, the upper bound, viz., 3 is good enough.

  step=interval_of_sol/1; i=1;
  %The foregoing interval of solution y(x) is divided into 1
  %subinterval(s). Depending on the specified context, this number of
  %subintervals may be made smaller (not less than 1) or larger.

  for y0m=lowersol:step:uppersol,
    Dy0=0; y0=[y0m, Dy0];
    %Here y at x = .0001 is taken as y0m somewhat arbitrarily and
    %Dy at x =.0001 is 0 (already given).
```

```
    [x, y]=ode15s('ydash', xspan, y0); %soln=[x, y]
    %To observe the solution, remove % in the foregoing line.

    xsize=size(x); nx=xsize(1,1);
    if nxstep==nx,
      y0p(i)= y0m; y1p(i) = y(nx, 1); Dy1p(i) = y(nx, 2);

      F(i)=y1p(i)-lmda;

      i=i+1;
    end;
  end;
  count=i-1;
  %Check F(i) for sign change and then apply interpolation or bisection.
  %Because of nonlinearity, extrapolation is not advisable as it may fail
  %in some applications.

  for i = 1: (count - 1);
    while F(i)*F(i+1) < 0
      y0c = y0p(i) - ((y0p(i) - y0p(i + 1))/(F(i)-F(i + 1)))*F(i);
      y0=[y0c, Dy0]; %new initial condition
      [x, y] = ode15s('ydash', xspan, y0); soln = [x, y];

      Y(:, k)=soln(:, 2); k=k+1;
      %Columns of Y provide the iterative solns. y(x), where
      %the last column of Y should be the true soln. y(x).

      Fr=y(nx, 1)-lmda;

      if abs(Fr) <=0.5*10^-4
        required_sol = soln;
        fprintf('\n Fr denotes accuracy of final soln \n')
        Fr, break,
      end;

      if Fr*F(i) < 0
        F(i + 1) =Fr;
        y0p(i + 1) = y0c;
      else F(i) = Fr; y0p(i) = y0c;
      end;
    end;
  end;
end;
true_sol=required_sol(:,2);

%THIRD FILE ydash
%subsubprogram ydash
function ydash=f(x, y);
%Keep this function subprogram in a DIFFERENT file named "ydash".
%k is taken as 0.1. For any other value of k, say k = 1, replace 0.1 by
```

%1 in the line just below this line.
ydash=[y(2); -0.1/y(1)^2 - 3*y(2)/x];
>> lbubsol_xkp1problem1

The solution bounds when $k = 0.1$.

x	lowerbound	upperbound
0.0001	0.4000	0.4781
0.0334	0.4000	0.4780
0.0667	0.4000	0.4778
0.1000	0.4000	0.4773
0.1333	0.4000	0.4767
0.1666	0.4000	0.4760
0.1999	0.4000	0.4750
0.2332	0.4000	0.4739
0.2665	0.4000	0.4726
0.2998	0.4000	0.4711
0.3331	0.4000	0.4695
0.3664	0.4000	0.4676
0.3997	0.4000	0.4656
0.4330	0.4000	0.4635
0.4663	0.4000	0.4611
0.4996	0.4000	0.4586
0.5329	0.4000	0.4559
0.5662	0.4000	0.4531
0.5995	0.4000	0.4500
0.6328	0.4000	0.4468
0.6661	0.4000	0.4435
0.6994	0.4000	0.4399
0.7327	0.4000	0.4362
0.7660	0.4000	0.4323
0.7993	0.4000	0.4282
0.8326	0.4000	0.4240
0.8659	0.4000	0.4195
0.8992	0.4000	0.4150
0.9325	0.4000	0.4102
0.9658	0.4000	0.4053
0.9991	0.4000	0.4001

Lower soln at $x = 0 + 0.4000$
Upper soln at $x = 0 + 0.4781$

Fr denotes accuracy of final soln

$Fr = 2.0794\text{e} - 005$

Col 1: x, Cols 2, 3, .., last but one: successive iterated sols $y(x)$, Col last: true sol $y(x)$
$Y1 =$

0.0001	0.4654	0.4639	0.4637	0.5662	0.4463	0.4447	0.4445
0.0334	0.4653	0.4638	0.4637	0.5995	0.4439	0.4423	0.4421
0.0667	0.4651	0.4636	0.4635	0.6328	0.4414	0.4398	0.4396
0.1000	0.4648	0.4633	0.4631	0.6661	0.4387	0.4370	0.4369
0.1333	0.4643	0.4628	0.4627	0.6994	0.4359	0.4342	0.4340
0.1666	0.4637	0.4622	0.4621	0.7327	0.4328	0.4311	0.4310
0.1999	0.4630	0.4615	0.4614	0.7660	0.4297	0.4279	0.4278
0.2332	0.4622	0.4607	0.4605	0.7993	0.4263	0.4245	0.4244
0.2665	0.4612	0.4597	0.4596	0.8326	0.4228	0.4210	0.4208
0.2998	0.4601	0.4586	0.4584	0.8659	0.4190	0.4172	0.4170
0.3331	0.4589	0.4573	0.4572	0.8992	0.4151	0.4133	0.4131
0.3664	0.4575	0.4560	0.4558	0.9325	0.4110	0.4091	0.4090
0.3997	0.4560	0.4544	0.4543	0.9658	0.4067	0.4048	0.4046
0.4330	0.4543	0.4528	0.4526	0.9991	0.4022	0.4002	0.4000
0.4663	0.4525	0.4510	0.4508	>>			
0.4996	0.4506	0.4490	0.4489				
0.5329	0.4485	0.4469	0.4468				

From Figure 3, for $k = 0.2$, $\lambda = 0.4$ in Problem P1, the iterated solution crosses the upper bound of the solution. The procedure SNLIP has no problem to converge and obtain the true solution. The true solution, however, remains well within the lower-upper solution bounds.

Like the program for Problem P3, the program for Problem P1 which is the same through the nonlinear transformation $x = 1/t$, is unable to meet the integration tolerance requirement for $\lambda = 0.4$ and for $k = 0.3, 0.4, 0.8$ as it should be.

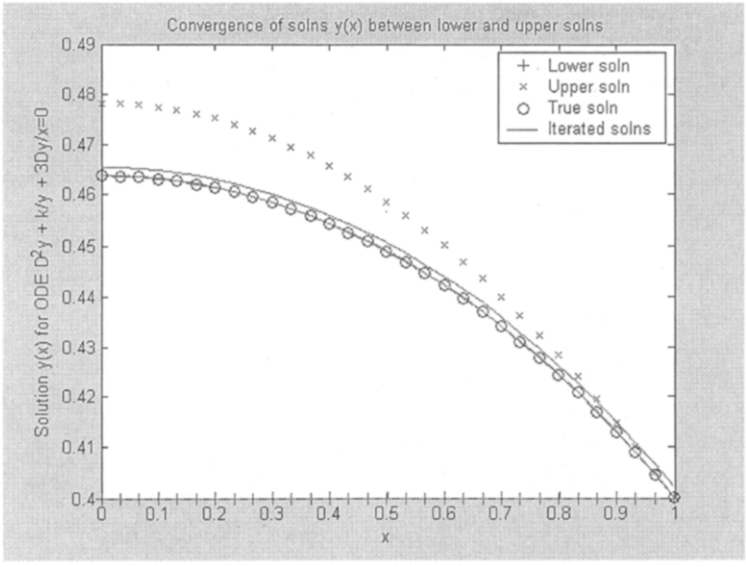

Figure 1. $k = 0.1$, $\lambda = 0.4$ for Problem P1.

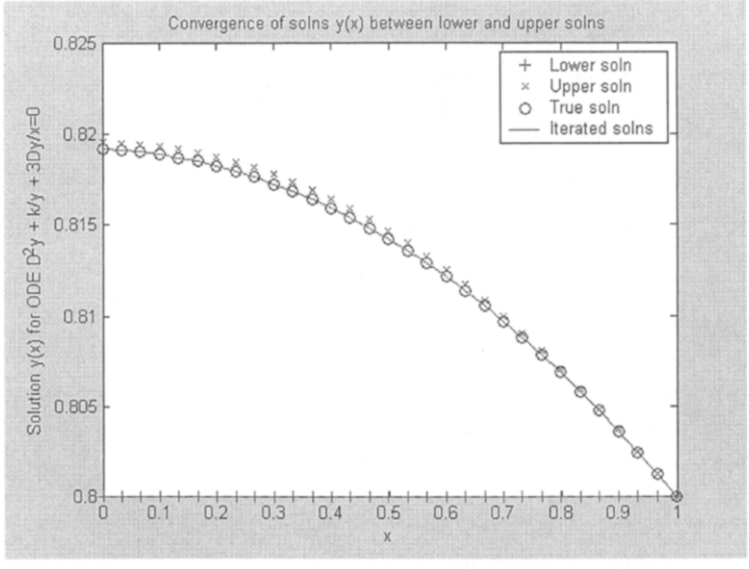

Figure 2. $k = 0.1$, $\lambda = 0.8$ for Problem P1. (Does not depend on v. The convergence is very fast.)

Consider the Problem P2: $y'' + k/y^2 + (3/x)y' = 0$, $0 < x < 1$ with boundary conditions $y'(0) = 0$, $y'(1) + (1 - v)y(1) = 0$, where $v1 = 1 - v > 0$, $v3 = 3 - v$, and the bounds for the solution $y(x)$ are $0.5[kv1^2/v3^2]^{1/3}[v3/v1 - x^2] \leqslant y(x) \leqslant [kv1^2/32]^{1/3}[v3/v1 - x^2]$. The MATLAB program for this problem, when $v = 0.5$, $k = 0.1$ is as follows.

```
%File name to be used for execution of Problem P2 is
%lbubsol_xvp5kp1problem2
%since we have kept the following FIRST FILE lbubsol_xvp5kp1problem2 under
%this name;
%the FIRST FILE uses the SECOND file bvplbubsolbased1problem2
%which in turn uses the THIRD FILE ydash
```

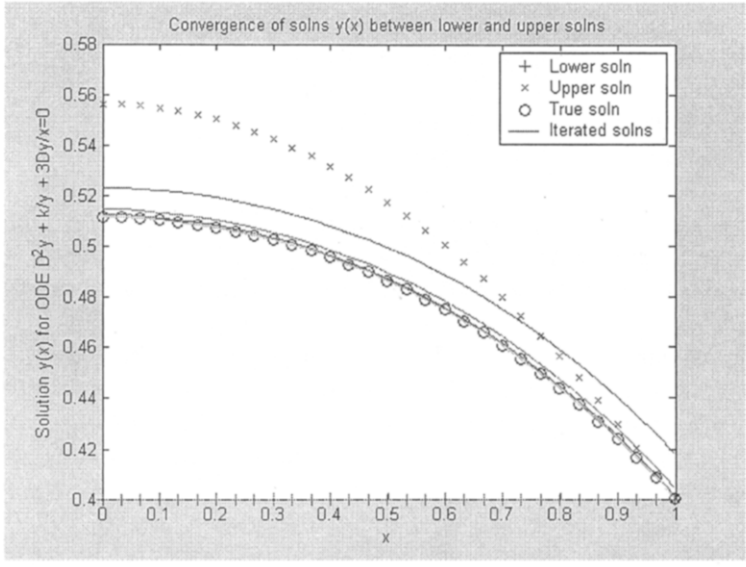

Figure 3. $k = 0.2$, $\lambda = 0.4$ for Problem P1. An iterated solution here crosses the upper solution line and then crosses back into the lower-upper solution bounds and finally produces the required true solution.

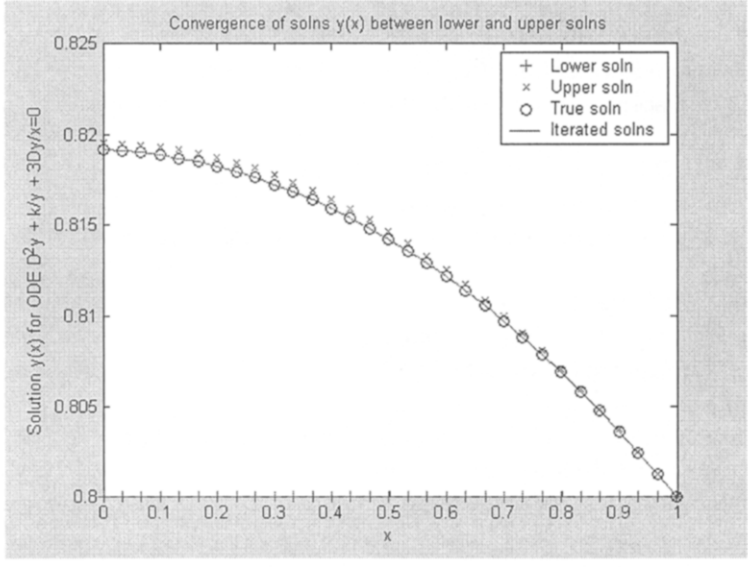

Figure 4. $k = 0.1$ $\lambda = 0.8$ for Problem P1. Just one iteration was required to get the true solution. The true solution is almost the upper solution.

```
%FIRST FILE lbubsol_xvp5kp1problem2
%main program lbubsol_xvp5kp1problem2
function [ ] = lbubsol_xvp5kp1problem2( )
k=.1; v=.5;
%The values of k and v above should be appropriately written if they are
%not0.1 and 0.5, respectively.
x = .0001 : .0333 : .9995;
fprintf('\n The solution bounds when k = 0.1 \n')
%Change the value of k in the above print statement it it is not 0.1
  fprintf('\n   v       x       lowerbound     upperbound \n')
```

```
    for j = 1:31
      xx = x(j); vv = v; v1 = 1-vv; v3 = 3-vv;
      Z = (v3/v1) - xx^2;
      alpha = 0.5*((k*v1^2)/v3^2)^(1/3)*(Z);
      beta = ((k*v1^2)/32)^(1/3)*(Z);

      lbsol = alpha; lbsoln(j)=lbsol;
      ubsol = beta;ubsoln(j)=ubsol;
      fprintf('%7.4f %7.4f %7.4f %7.4f \n', vv,xx, lbsol, ubsol)
        end
    fprintf('\n Lower soln at x=0+ Upper soln at x=0+ \n')
    fprintf('%7.4f %7.4f \n', lbsoln(1), ubsoln(1)),
    [Y]=bvplbubsolbased1problem2(.5, lbsoln(1), ubsoln(1));
      fprintf('\n Col 1: x, Cols 2, 3, .., last but one: successive iterated sols y(x),
Col last: true sol y(x) \n')

    Y1=[x' Y],
    %fprintf('%7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f \n', x', Y)
    s=size(Y); Yfcindex = s(:, 2); true_sol = Y(:, Yfcindex);

    plot(x', lbsoln, 'r+', x', ubsoln, 'bx', x', true_sol, 'ko', x', Y);
    xlabel('x'); ylabel('Solution y(x) for ODE D^2y + k/y + 3Dy/x=0');
    title('Convergence of solns y(x) between lower and upper solns');
    legend('Lower soln','Upper soln', 'True soln', 'Iterated solns');

  %SECOND FILE bvplbubsolbased1problem2
  %subprogram bvplbubsolbased1problem2
  function[Y]=bvplbubsolbased1problem2(v, lowersol, uppersol);

  xspan=.0001:.0333:.9991; k=1;
  xspansize=size(xspan); nxstep=xspansize(1, 2);
  interval_of_sol=uppersol-lowersol;
  if interval_of_sol<=3,
    %In the given context, if the interval of upper and lower solutions
    %is too large then such an interval may not be of much use. In the
    %current membrane problem, the upper bound, viz., 3 is good enough.

    step=interval_of_sol/1; i=1;
    %The foregoing interval of solution y(x) is divided into 1
    %subinterval(s). Depending on the specified context, this number of
    %subintervals may be made smaller or larger.

    for y0m=lowersol:step:uppersol,
      Dy0=0;
      y0=[y0m, Dy0];
        %Here y at x = .0001 is taken as y0m somewhat arbitrarily and
        %Dy at x =.0001 is 0 (already given).

        [x, y]=ode15s('ydash', xspan, y0); %soln=[x, y]
        %To observe the solution, remove % in the foregoing line.
```

```
      xsize=size(x); nx=xsize(1,1);
      if nxstep==nx,
        y0p(i)= y0m; y1p(i) = y(nx, 1); Dy1p(i) = y(nx, 2);
        F(i)=(1-v)*y1p(i)+Dy1p(i);
        %If one wants to see the successive values of F, replace the
        %foregoing ; by ,

        i=i+1;
      end;
    end;
    count=i-1;
    %Check F(i) for sign change and then apply interpolation or bisection.
    %Because of nonlinearity, extrapolation is not advisable as it may fail
    %in some applications.

    for i = 1: (count - 1);
      while F(i)*F(i+1) < 0
      y0c = y0p(i) - ((y0p(i) - y0p(i + 1))/(F(i)-F(i + 1)))*F(i);
      y0=[y0c, Dy0]; %new initial condition
      [x, y] = ode15s('ydash', xspan, y0); soln = [x, y];

      Y(:, k)=soln(:, 2); k=k+1;
      %Columns of Y provide the iterative solns. y(x), where
      %the last column of Y should be the true soln. y(x).

      Fr=(1 - v)*y(nx, 1) + y(nx, 2);
      if abs(Fr) <=0.5*10^-4
        v, required_sol = soln; Fr, break,
      end;

      if Fr*F(i) < 0
        F(i + 1) =Fr;
        y0p(i + 1) = y0c;
      else F(i) = Fr; y0p(i) = y0c;
      end;
    end;
  end;
end;
true_sol=required_sol(:,2);

%THIRD FILE ydash
%subsubprogram ydash
function ydash=f(x, y);
%Keep this function subprogram in a DIFFERENT file named "ydash".
%k is taken as 0.1. For any other value of k, say k = 1, replace 0.1 by
%1 in the line just below this line.
ydash=[y(2); -0.1/y(1)^2 - 3*y(2)/x];

>> lbubsol_xvp5kp1problem2
```

The solution bounds when $k = 0.1$.

v	x	lowerbound	upperbound	0.5000	0.4996	0.3770	0.4375
0.5000	0.0001	0.3969	0.4605	0.5000	0.5329	0.3743	0.4343
0.5000	0.0334	0.3968	0.4604	0.5000	0.5662	0.3714	0.4310
0.5000	0.0667	0.3965	0.4601	0.5000	0.5995	0.3683	0.4274
0.5000	0.1000	0.3961	0.4596	0.5000	0.6328	0.3651	0.4236
0.5000	0.1333	0.3954	0.4589	0.5000	0.6661	0.3616	0.4196
0.5000	0.1666	0.3946	0.4579	0.5000	0.6994	0.3580	0.4155
0.5000	0.1999	0.3937	0.4568	0.5000	0.7327	0.3542	0.4111
0.5000	0.2332	0.3925	0.4555	0.5000	0.7660	0.3503	0.4065
0.5000	0.2665	0.3912	0.4540	0.5000	0.7993	0.3461	0.4017
0.5000	0.2998	0.3897	0.4522	0.5000	0.8326	0.3418	0.3967
0.5000	0.3331	0.3880	0.4503	0.5000	0.8659	0.3373	0.3914
0.5000	0.3664	0.3862	0.4481	0.5000	0.8992	0.3327	0.3860
0.5000	0.3997	0.3842	0.4458	0.5000	0.9325	0.3278	0.3804
0.5000	0.4330	0.3820	0.4432	0.5000	0.9658	0.3228	0.3746
0.5000	0.4663	0.3796	0.4405	0.5000	0.9991	0.3176	0.3686

Lower soln at $x = 0 + 0.3969$

Upper soln at $x = 0 + 0.4605$

$v = 0.5000$

$Fr = 1.9572\text{e} - 005$

Col 1: x, Cols 2, 3, ... , last but one: successive iterated sols $y(x)$, Col last: true sol $y(x)$

Y1 =

0.0001	0.4335	0.4291	0.4283	0.4281	0.4281	0.5329	0.4139	0.4091	0.4084	0.4081	0.4081
0.0334	0.4334	0.4290	0.4283	0.4280	0.4280	0.5662	0.4113	0.4065	0.4057	0.4054	0.4054
0.0667	0.4332	0.4288	0.4280	0.4278	0.4278	0.5995	0.4085	0.4036	0.4028	0.4026	0.4025
0.1000	0.4328	0.4284	0.4277	0.4274	0.4274	0.6328	0.4056	0.4006	0.3998	0.3995	0.3995
0.1333	0.4322	0.4279	0.4271	0.4269	0.4268	0.6661	0.4024	0.3973	0.3965	0.3963	0.3962
0.1666	0.4316	0.4272	0.4264	0.4262	0.4262	0.6994	0.3990	0.3939	0.3931	0.3928	0.3928
0.1999	0.4307	0.4263	0.4256	0.4254	0.4253	0.7327	0.3955	0.3902	0.3894	0.3892	0.3891
0.2332	0.4298	0.4253	0.4246	0.4244	0.4243	0.7660	0.3917	0.3864	0.3856	0.3853	0.3852
0.2665	0.4286	0.4242	0.4235	0.4232	0.4232	0.7993	0.3877	0.3823	0.3815	0.3812	0.3811
0.2998	0.4274	0.4229	0.4221	0.4219	0.4219	0.8326	0.3835	0.3779	0.3771	0.3768	0.3768
0.3331	0.4259	0.4214	0.4207	0.4204	0.4204	0.8659	0.3791	0.3734	0.3725	0.3722	0.3722
0.3664	0.4243	0.4198	0.4190	0.4188	0.4188	0.8992	0.3744	0.3686	0.3677	0.3674	0.3673
0.3997	0.4226	0.4180	0.4172	0.4170	0.4170	0.9325	0.3695	0.3635	0.3626	0.3623	0.3622
0.4330	0.4207	0.4161	0.4153	0.4150	0.4150	0.9658	0.3642	0.3581	0.3572	0.3569	0.3568
0.4663	0.4186	0.4139	0.4132	0.4129	0.4129	0.9991	0.3588	0.3524	0.3515	0.3512	0.3511
0.4996	0.4163	0.4116	0.4108	0.4106	0.4105	>>					

For other data, *viz.*,

(i) $k = 0.99$, $v = 0.99$,

(ii) $k = 0.99$, $v = 0.8$,

(iii) $k = 0.99$, $v = 0.7$ in Problem P2, the program was unable to meet the four significant digit accuracy requirement and hence did not produce the result/graph.

Further, in Problem P2, for larger v, the convergence is faster while for larger k convergence is slower. For example, $k = 0.99$, $v = 0.1$, the number of iterations in P2 is 7 which is considered relatively large.

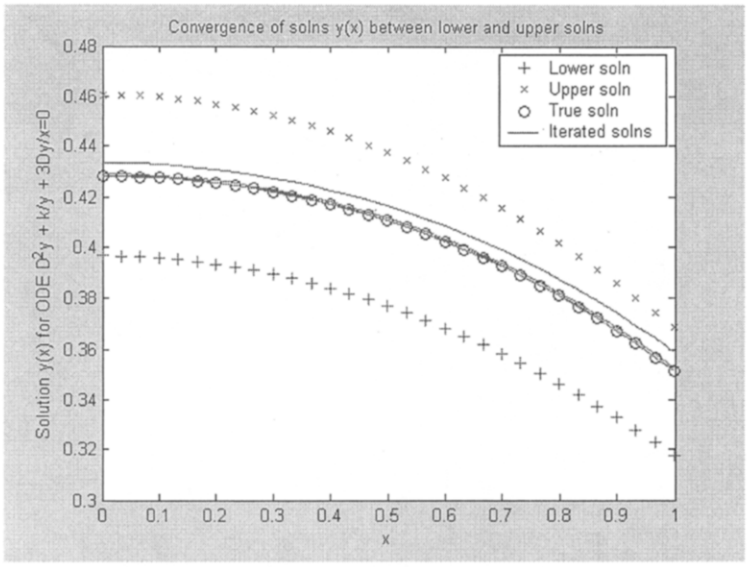

Figure 5. $k = 0.1$, $v = 0.5$ for Problem P2. The iterated and true solutions lie well within the lower-upper solution bound.

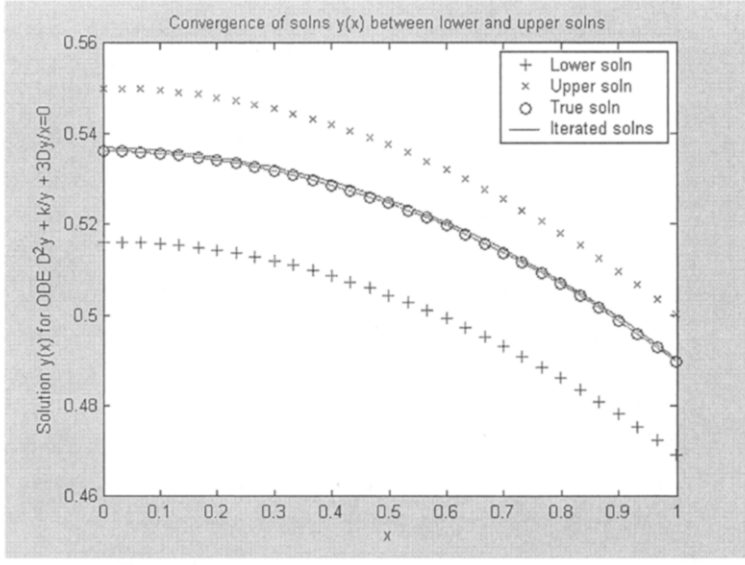

Figure 6. $v = 0.8$, $k = 0.1$ for Problem P2 As in Figure 5, here too the iterated solutions lie well within the lower-upper solution bounds needing less number of iterations.

Consider the Problem P3: $D^2y - (1/t)Dy + (k/t^4)(1/y^2) = 0$, $1 < t < \infty$ with boundary conditions $y(1) = \lambda > 0$, $Dy(\infty) = 0$, where $y(t) \in [\lambda, \lambda + ((k/(8\lambda^2))(1 - (1/t^2))]$. Observe that Problems P1 and P3 are independent of v. They depend on λ and k only. The MATLAB program for this problem, when $v = 0.5$, $k = 0.1$ is as follows.

```
% Problem P3 MATLAB Program
%FIRST FILE lbubsol_xkp1problem3
%main program lbubsol_xkp1problem3
function [ ] = lbubsol_xkp1problem3
lmda = 0.4; k=.1;
```

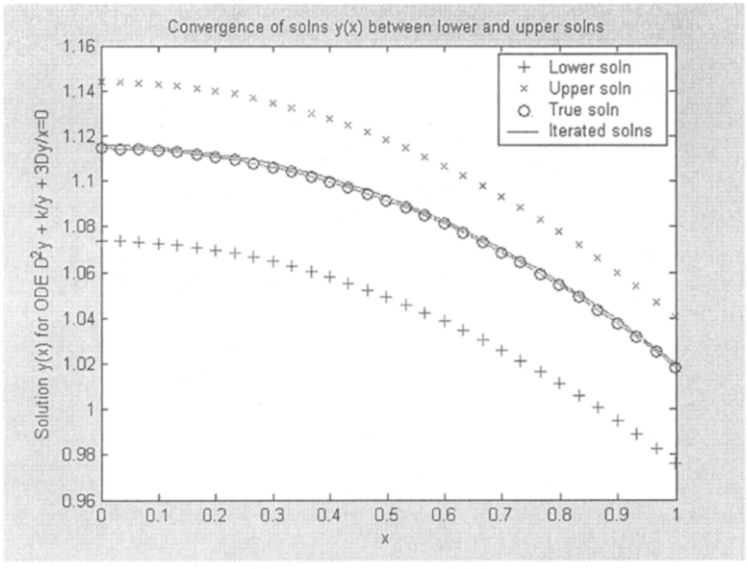

Figure 7. $v = 0.8$, $k = 0.9$ for Problem 2 As in Figures 5 and 6, the nature of solutions is the same; only the lower-upper solution bounds and the iterated and true solutions get shifted up.

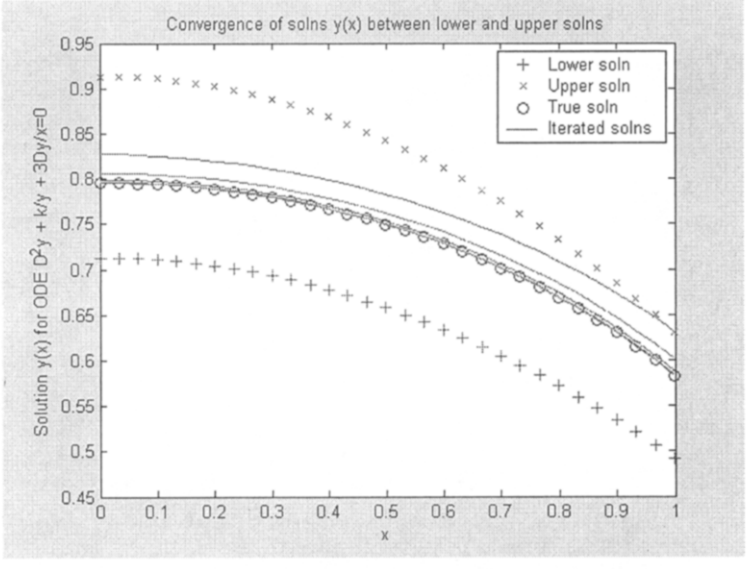

Figure 8. $k = 0.9$, $v = 0.1$ for Problem P2. While the true solution is well within the lower-upper solution bounds, an iterated solution just touches the upper solution.

```
%Change the values of lmda and k above as and when these are changed.
x = .0333 : .0333 : .9995; t=1./x;
fprintf('\n The solution bounds when k = 0.1 \n')
  fprintf('\n    t       lowerbound    upperbound \n')

  for j = 1:30
    tt = t(j); alpha=lmda; beta = (k/(8*lmda^2))*(1-(1/t(j)^2))+lmda;
    lbsol = alpha; lbsoln(j)=lbsol; ubsol = beta; ubsoln(j)=ubsol;
    fprintf('%7.4f %7.4f %7.4f \n', tt, lbsol, ubsol)
  end
```

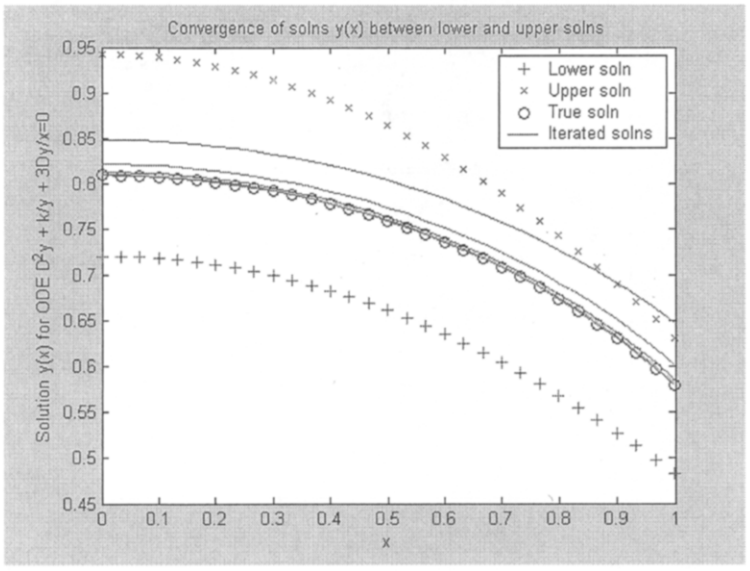

Figure 9. $k = 0.99$, $v = 0.1$ for Problem 2. Here the number of iterations is 7 which is relatively high; also an iterated solution did cross the upper solution bound. The SNLIP did not have any difficulty to get the required true solution which is well within the lower-upper solution bounds.

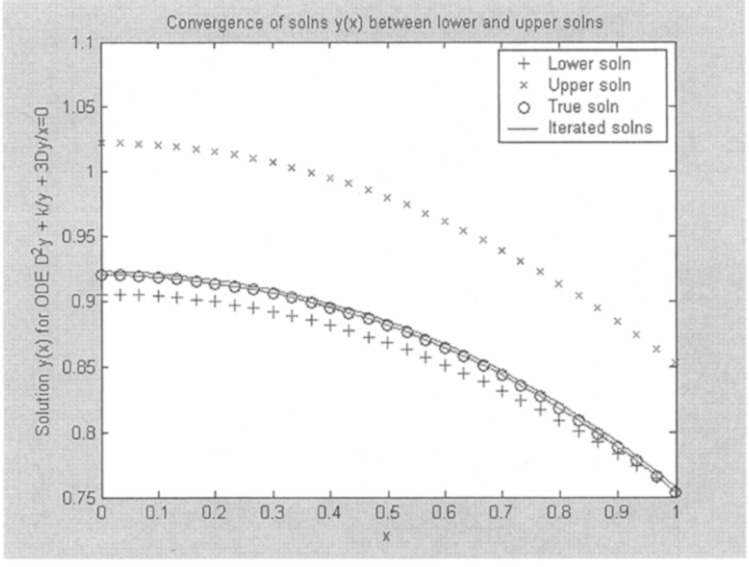

Figure 10. $k = 0.99$, $v = 0.6$ for Problem P2. Here the iterated and true solutions are close to the lower solution bound and the number of iterations needed to get four significant digit accuracy is just 2.

```
fprintf('\n Lower soln at t=inf Upper soln at t=inf \n')
fprintf('%7.4f %7.4f \n', lbsoln(1), ubsoln(1)),
[Y]=bvplbubsolbased1problem3(lbsoln(1), ubsoln(1));
fprintf('\n Col 1: t, Cols 2, 3, .., last but one: successive iterated sols y(t), Col last: true sol y(t) \n')
Y1=[t' Y],
s=size(Y); Yfcindex = s(:, 2); true_sol = Y(:, Yfcindex);
```

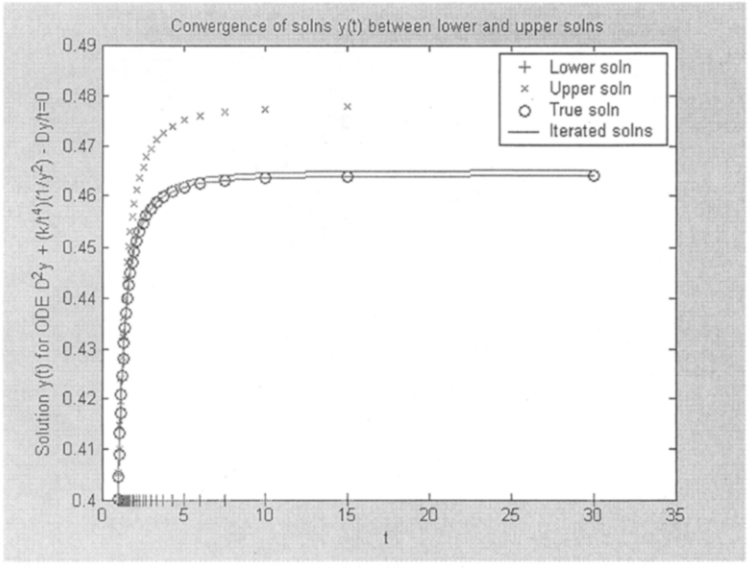

Figure 11. $k = 0.1$, $\lambda = 0.4$ for Problem P3. The intervals of the independent variable t $(= 1/x)$ are not equispaced since that would mean too many steps with very little variation in the solution and also computational error.

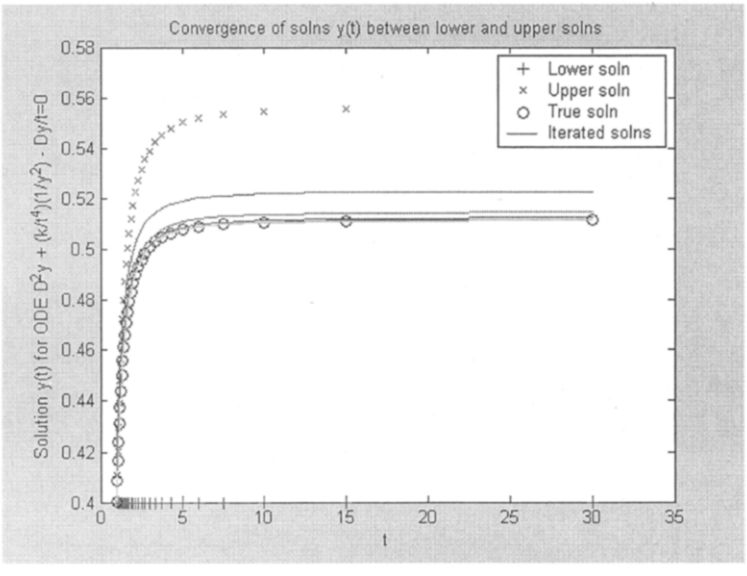

Figure 12. The foregoing graph is for $v = 0.5$, $k = 0.2$, $\lambda = 0.4$ for Problem P3.

```
plot(t', lbsoln, 'r+', t', ubsoln, 'bx', t', true_sol, 'ko', t', Y);
xlabel('t'); ylabel('Solution y(t) for ODE D^2y + (k/t^4)(1/y^2) - Dy/t=0');
title('Convergence of solns y(t) between lower and upper solns');
legend('Lower soln','Upper soln', 'True soln', 'Iterated solns');

%SECOND FILE bvplbubsolbased1problem3
%subprogram bvplbubsolbased1problem3(lowersol, uppersol)
function[Y]=bvplbubsolbased1problem3(lowersol, uppersol);

xspan=.0333:.0333:.9995; k=1; lmda =0.4;
%Change lmda above as and when this is changed.
tspan=1./xspan;
```

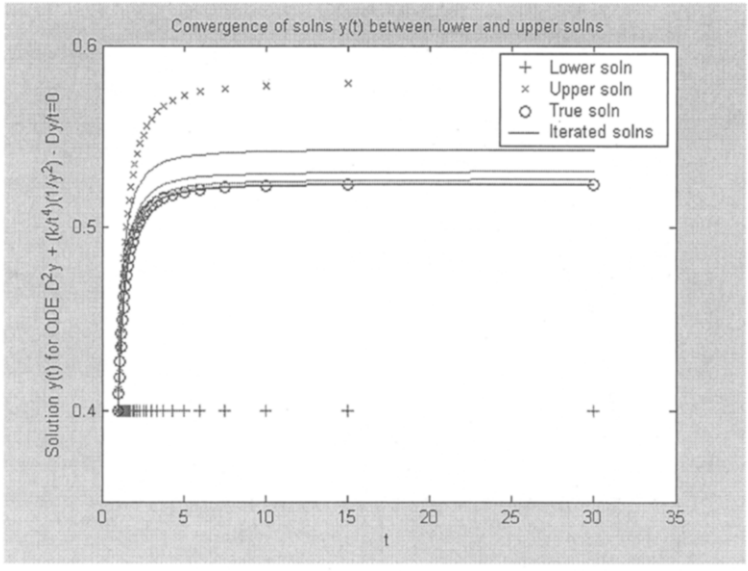

Figure 13. The foregoing figure is for $k = 0.23$, $\lambda = 0.4$ for Problem P3 The problem depends on λ and k and not on v.

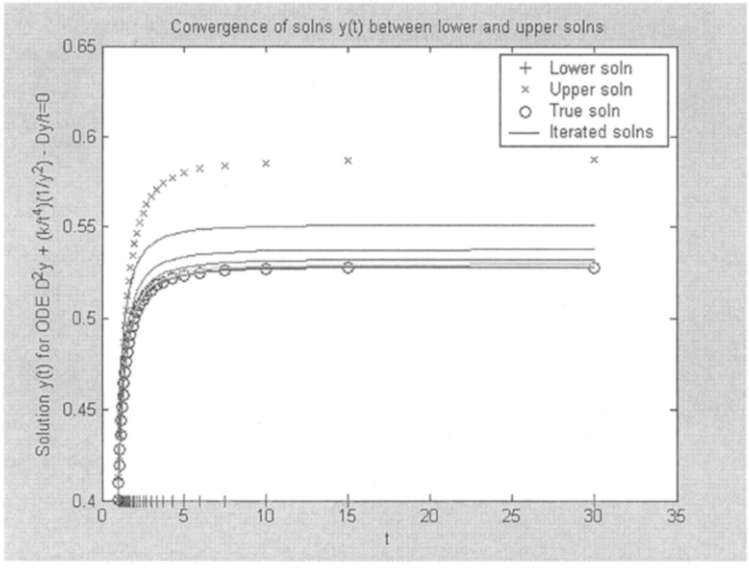

Figure 14. This figure is for $v = 0.5$, $k = 0.24$, $\lambda = 0.4$ for Problem P3.

```
% k here is an integer variable, has nothing to do with the same
%symbol k used inside the subsubprogram ydash.
%xspansize=size(xspan); nxstep=xspansize(1, 2);

tspansize=size(tspan); ntstep=tspansize(1, 2);

interval_of_sol=uppersol-lowersol;
if interval_of_sol<=3,
  %In the given context, if the interval of upper and lower solutions
  %is too large then such an interval may not be of much use. In the
  %current membrane problem, the upper bound, viz., 3 is good enough.
```

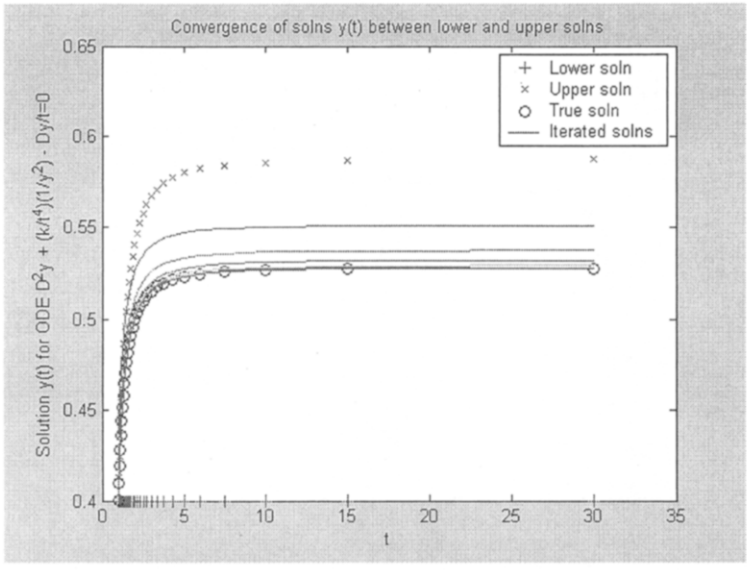

Figure 15. The foregoing figure is for $v = 0.8$, $k = 0.24$, $\lambda = 0.4$ for Problem P3. This problem does not depend on v. It depends on λ and k.

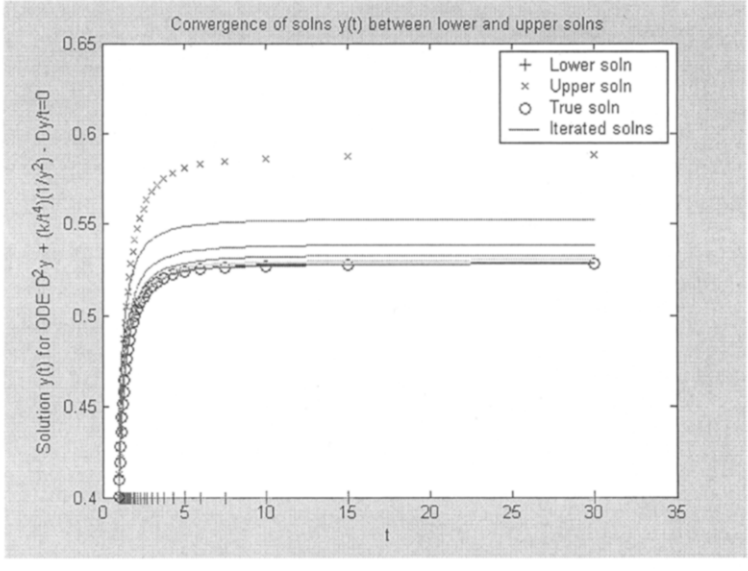

Figure 16. The foregoing figure is for $v = 0.8$, $k = 0.241$, $\lambda = 0.4$ for Problem P3. This problem does not depend on v. It depends on k and λ.

```
step=interval_of_sol/1; i=1;

  %The foregoing interval of solution y(x) is divided into 1
  %subinterval(s). Depending on the specified context, this number of
  %subintervals may be made smaller (not less than 1) or larger.

  for y0m=lowersol:step:uppersol,
    Dyinf=0;
    y0=[y0m, Dyinf];

    %Here y at x = .0001 is taken as y0m somewhat arbitrarily and
    %Dy at x =.0001 is 0 (already given).
```

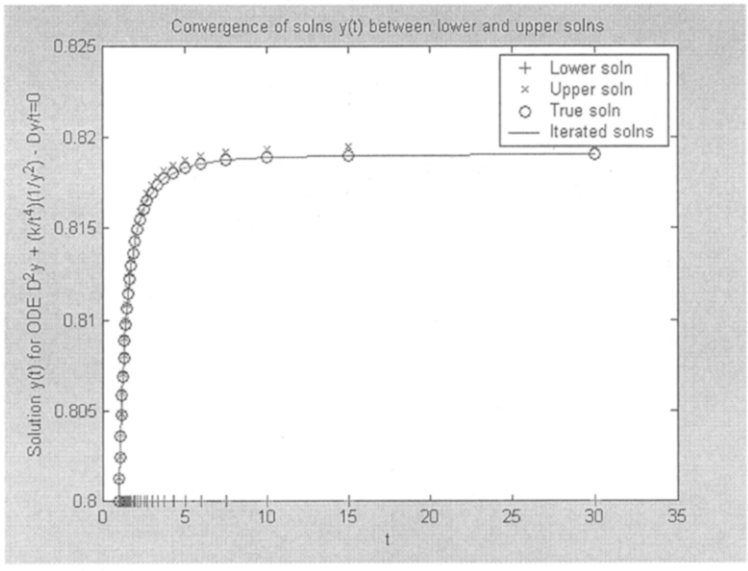

Figure 17. The foregoing figure is for $k = 0.1$ and $\lambda = 0.8$ for Problem P3.

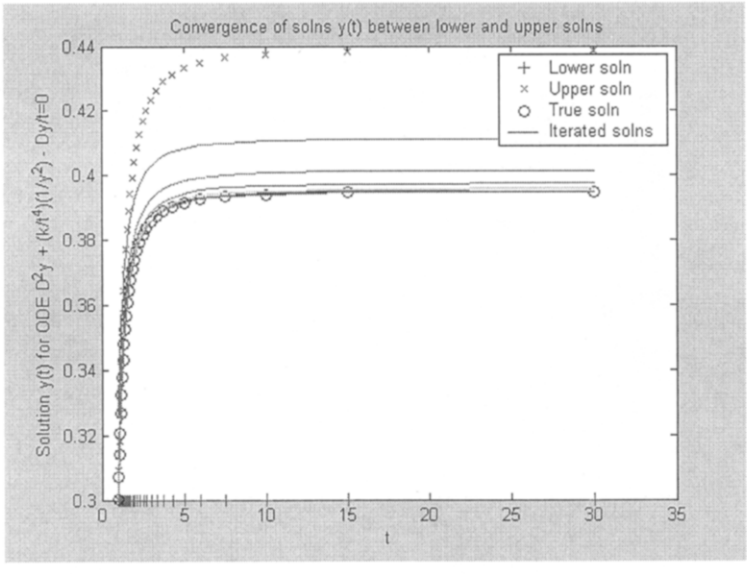

Figure 18. $\lambda = 0.3$, $k = 0.1$ for Problem P3 (It does not depend on v.)

```
  [t, y]=ode15s('ydashxe1bt', tspan, y0); soln=[t, y];
  tsize=size(t); nt=tsize(1,1);
    if ntstep==nt,

    y0p(i)= y0m; y1p(i) = y(nt, 1); Dy1p(i) = y(nt, 2);
      F(i)=y1p(i)-lmda;
      i=i+1;
    end;
end;
count=i-1;
%Check F(i) for sign change and then apply interpolation or bisection.
```

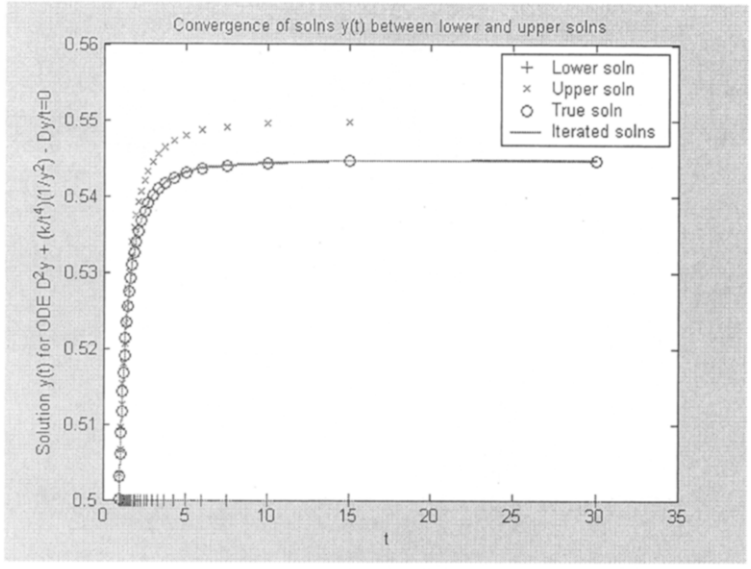

Figure 19. $\lambda = 0.5$, $k = 0.1$ for Problem P3.

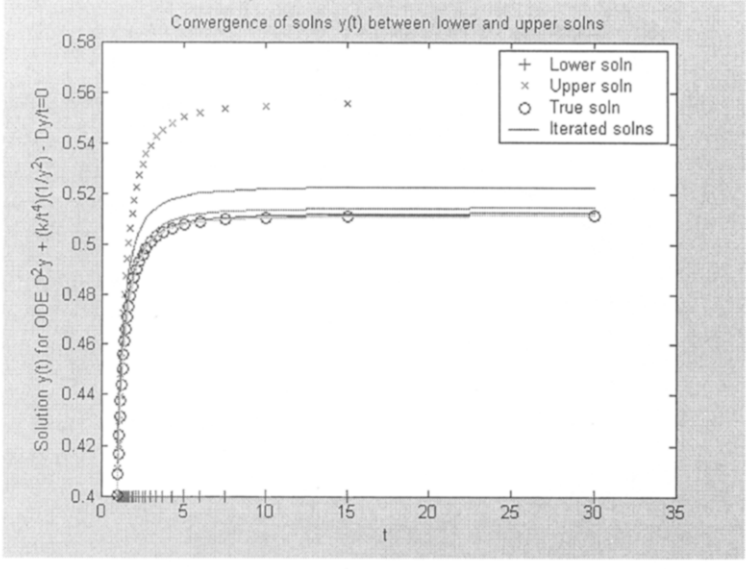

Figure 20. $k = 0.2$, $\lambda = 0.4$ for Problem P3.

```
%Because of nonlinearity, extrapolation is not advisable as it may fail
%in some applications.

for i = 1: (count - 1);
  while F(i)*F(i+1) < 0
    y0c = y0p(i) - ((y0p(i) - y0p(i + 1))/(F(i)-F(i + 1)))*F(i);

    y0=[y0c, Dyinf]; %new initial condition

    [t, y] = ode15s('ydashxe1bt', tspan, y0); soln = [t, y];

    Y(:, k)=soln(:, 2); k=k+1;
```

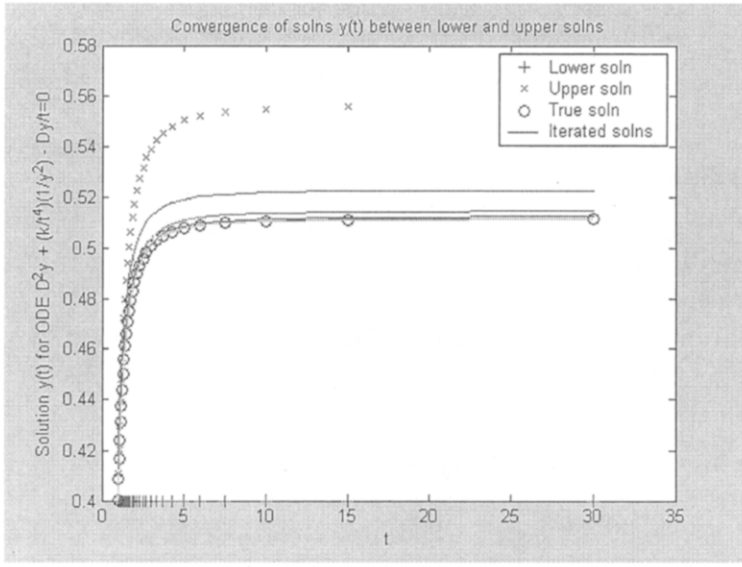

Figure 21. $k = 0.2$, $\lambda = 0.4$ for Problem P3.

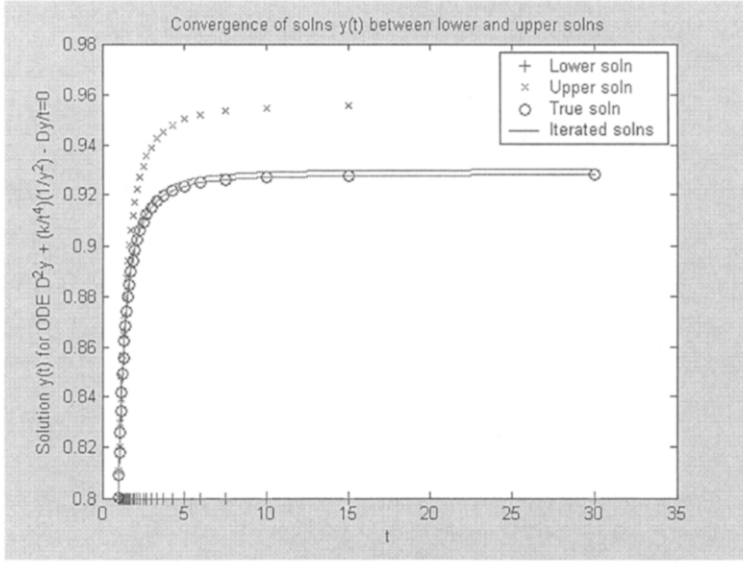

Figure 22. $\lambda = 0.8$, $k = 0.8$ for Problem P3.

```
%Columns of Y provide the iterative solns. y(x), where
%the last column of Y should be the true soln. y(x).

Fr=y(nt, 1)-lmda;

if abs(Fr) <=0.5*10^-4
  required_sol = soln;
  fprintf('\n Fr denotes accuracy of final soln \n')
  Fr, break,
end;

if Fr*F(i) < 0, F(i + 1) =Fr; y0p(i + 1) = y0c;
else F(i) = Fr; y0p(i) = y0c;
```

```
      end;
    end;
  end;
end;
true_sol=required_sol(:,2);

%THIRD FILE ydashxe1bt
%subsubprogram ydashxe1bt
function ydashxe1bt=f(t, y);
%Keep this function subprogram in a DIFFERENT file named "ydashxe1bt"
%k is taken as 0.1. For any other value of k, say k = 1, replace 0.1 by
%1 in the line just below this line.

ydashxe1bt=[y(2); -(0.1/t^4)*(1/y(1)^2)+y(2)/t];

>> lbubsol_xkp1problem3
```

The solution bounds when $k = 0.1$

t	lowerbound	upperbound	1.8769	0.4000	0.4559
30.0300	0.4000	0.4780	1.7665	0.4000	0.4531
15.0150	0.4000	0.4778	1.6683	0.4000	0.4501
10.0100	0.4000	0.4773	1.5805	0.4000	0.4469
7.5075	0.4000	0.4767	1.4300	0.4000	0.4399
6.0060	0.4000	0.4760	1.5015	0.4000	0.4435
5.0050	0.4000	0.4750	1.3650	0.4000	0.4362
4.2900	0.4000	0.4739	1.3057	0.4000	0.4323
3.7538	0.4000	0.4726	1.2513	0.4000	0.4282
3.3367	0.4000	0.4711	1.2012	0.4000	0.4240
3.0030	0.4000	0.4695	1.1550	0.4000	0.4196
2.7300	0.4000	0.4676	1.1122	0.4000	0.4150
2.5025	0.4000	0.4656	1.0725	0.4000	0.4102
2.3100	0.4000	0.4635	1.0355	0.4000	0.4053
2.1450	0.4000	0.4611	1.0010	0.4000	0.4002
2.0020	0.4000	0.4586			

Lower soln at $t = \inf$ 0.4000 Upper soln at $t = \inf$ 0.4780

Fr denotes accuracy of final soln

$Fr = 8.2556\text{e} - 006$

Col 1: t, Cols 2, 3, ... , last but one: successive iterated sols $y(t)$, Col last: true sol $y(t)$

$Y1 =$

30.0300	0.4651	0.4641	0.4640	1.7665	0.4459	0.4449	0.4448
15.0150	0.4649	0.4639	0.4638	1.6683	0.4435	0.4425	0.4424
10.0100	0.4646	0.4636	0.4635	1.5805	0.4410	0.4399	0.4398
7.5075	0.4641	0.4631	0.4630	1.5015	0.4383	0.4372	0.4371
6.0060	0.4635	0.4625	0.4624	1.4300	0.4354	0.4343	0.4342
5.0050	0.4628	0.4618	0.4617	1.3650	0.4324	0.4312	0.4312
4.2900	0.4619	0.4609	0.4609	1.3057	0.4291	0.4280	0.4279
3.7538	0.4609	0.4600	0.4599	1.2513	0.4258	0.4246	0.4245
3.3367	0.4598	0.4588	0.4588	1.2012	0.4222	0.4210	0.4209
3.0030	0.4586	0.4576	0.4575	1.1550	0.4184	0.4172	0.4171

2.7300	0.4572	0.4562	0.4561	1.1122	0.4145	0.4133	0.4132
2.5025	0.4557	0.4547	0.4546	1.0725	0.4103	0.4091	0.4090
2.3100	0.4540	0.4530	0.4529	1.0355	0.4060	0.4047	0.4046
2.1450	0.4522	0.4512	0.4511	1.0010	0.4014	0.4001	0.4000
2.0020	0.4503	0.4492	0.4492				
1.8769	0.4482	0.4471	0.4470				

>>

For $v = 0.5$ and $k = 0.25$ for Problem P3, the program is unable to meet integration tolerances. Clearly for these data, Problem P3 is ill-conditioned/sensitive and hence we should attempt to solve Problem P1 (corresponding to Problem P3) for the required results. It may and usually will so happen that both the problems are unable to meet the integration tolerances.

For $\lambda = 0.1, 0.2$ and $k = 0.1$, Problem P3, the program is unable to meet the integration tolerances. Hence, no solution/graph will be produced.

Let $k = 0.1$. The smaller the values of λ, the larger is the number of iterations for the convergence of the solution $y(x)$. For smaller values of λ, say $\lambda \leqslant 0.299$ for Problem P3, the program is unable to meet integration tolerances without reducing the step size. For larger values of λ, the program behaves much better in that the convergence is much faster and no failure takes place.

Let λ=0.4. For $k \geqslant 0.3$, the program does not meet the integration tolerances and hence, fails to produce results/graphs indicating that the problem posed is ill-conditioned.

If both λ and k are large, say both 0.8 then the problem is well-posed and the program works successfully. Here the number of iterations will be moderate, say, 2 or 3 for this problem, *viz.*, Problem P3. In fact, the SNLIP converges very fast. In most cases, it takes less than 10 iterations.

5. CONCLUSIONS

How Lower and Upper Solutions Help. The mathematically derived lower and upper solutions in [1] for the circular membrane problem are reasonably close, i.e., the interval "upper solution-lower solution" is reasonably small in this context. The knowledge of these bounds on actual solution allow us to search/scan only this small interval instead of the vast real line. This not only saves computing resources/time but also obviates the problem of possible "miss" of the solution.

However, a thorough knowledge of the actual physical membrane problem do provide the reader an idea of the reasonable value of the solution $y(x)$ at the initial point $x = 0_+$. Still reasonable mathematical solution bounds are much more desirable. In this case, the reader can save significant amount of time needed in mastering the concerned membrane theory with its physical implications. In addition, mathematical solution bounds are scientifically acceptable and precise while subjective knowledge of the circular or other form of membrane is not precise nor is it strictly scientifically acceptable. Further, although a "trial and error" approach may result in the subjective knowledge, failure in obtaining a solution is not completely ruled out.

If the BVP associated with the nonlinear ODEs is highly sensitive or has violent fluctuations, e.g., BVP with singular ODEs, a reasonably small interval between upper and lower bounds of the true solution is extremely desirable.

Generality/Restriction of MATLAB Program and its Ease of Modification. The MATLAB program for SNLIP is considerably general, very high-level, and easily readable without needing formal programming knowledge. This is quite unlike other lower level programming languages (compared to MATLAB) such as C, C^{++}, and Fortran. Appropriate comments embedded in the MATLAB program are helpful to the reader and permit easy modification for solving other BVPs associated with a system of ODEs, linear or nonlinear, coupled or noncoupled. The reader may also refer [5,7,8]. However, parameter passing from a subprogram to another subprogram in a MATLAB program, unlike programming languages such as Fortran, C, and C^{++}, does pose a

problem. Actual numerical values need to be used as parameters here—this reduces generalization or, in other words, imposes a restriction. For instance, The parameter k in the subprogram "ydash" needs explicit numerical value to be used instead of k.

WHY LINEAR INTERPOLATION AND NOT NONLINEAR ONE OR BISECTION. For a sensitive function $y(x)$, a nonlinear interpolation also becomes sensitive and may produce a value which may cross the region of convergence and hence, may fail. Bisection, on the other hand, may not fail but would take more iterations as it does not use the available knowledge of function values except, of course their signs. Linear interpolation, particularly a nonextrapolatory one, is much less sensitive and makes use of the already known function values to derive the benefit of nonlinear interpolation and that of bisection.

WHY FOUR SIGNIFICANT DIGITS IN THE INDEPENDENT VARIABLE x. No measuring device can produce an accuracy more than 0.005%. For any engineering application, four significant digit in the final result (to be used) is enough [5].

REFERENCES

1. R.P. Agarwal and D. O'Regan, Singular problems arising in circular membrane theory, *Dynamics of Continuous, Discrete and Impulsive Systems* **10**, 965–972, (2003).
2. A. Constantin, On an infinite interval boundary value problem, *Annali di Mathematica pura ed applicata* **176**, 379–394, (1999).
3. R.W. Dickey, The plane circular elastic surface under normal pressure, *Arch. Rat. Mech. Anl.* **26**, 219–236, (1967).
4. J.Y. Shin, A singular nonlinear boundary value problem in the nonlinear circular membrane under normal pressure, *J. Korean Math. Soc.* **32**, 761–773, (1995).
5. S.K. Sen and H. Agarwal, Mean value theorem for boundary value problems with given upper and lower solutions, *Computers Math. Applic.* **49** (9/10), 1499–1514, (2005).
6. E.V. Krishnamurthy and S.K. Sen, *Programming in* MATLAB, Affiliated East-West Press, New Delhi, (2003).
7. S.K. Sen and M. Chanda, A general scheme for solving ordinary differential equations under two-point boundary conditions, *J. Ind. Inst. Sci.* **54**, 139–145, (1972).
8. E.V. Krishnamurthy and S.K. Sen, *Numerical Algorithms: Computations in Science and Engineering*, Affiliated East-West Press, New Delhi, (2001).
9. D.E. Knuth, *The Art of Computer Programming 2*, Second Edition, Addison-Wesley, Reading, MA, (1981).
10. V. Lakshmikantham and S.K. Sen, *Computational Error and Complexity in Science and Engineering*, Elsevier, Amsterdam, (2005).