Math 337 Homework 08

Andy Reagan

March 16, 2014

1. Use the Gerschgorin Circles Theorem and the fact that the eigenvalues of real symmetric matrices are real to obtain the best estimate for the location of the eigenvalues of the following tri-diagonal matrix:

$$A = \begin{pmatrix} a & -1 & 0 & . & . & . & 0 \\ -1 & a & -1 & 0 & . & . & 0 \\ 0 & -1 & a & -1 & 0 & . & 0 \\ & & & . & . & . & \\ 0 & . & . & 0 & -1 & a & -1 \\ 0 & . & . & . & 0 & -1 & a \end{pmatrix},$$

where a is a real number. In particular, what is the minimum distance between an eigenvalue of this matrix and zero?

Solution: By the Gerschgorin Circles Theorem, all of the eigenvalues are found in circles centered at a. The radius of these circles is 1, corresponding to the first and last rows, and 2 otherwise. Since the eigenvalues are real, we know that they fall in the interval $[a-2, a+2]$ on the real line.

In particular, the distance between the eigenvalues of this matrix and 0 is at least $a-2$ if $a > 2$ and $a+2$ is $a < 2$. If $|a| \leq 2$, the minimum distance is 0 (i.e., the eigenvalues can well be 0).

2. Consider a linear BVP

$$y'' + 2(2-x)y' = 2(2-x), y(0) = -1, y(6) = 5.$$

Discretize it using scheme (8.4) with $h = 1$.

(i) Verify that you obtain a linear system

$$\begin{pmatrix} -2 & 2 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 3 & -2 & -1 \\ 0 & 0 & 0 & 4 & -2 \end{pmatrix} \begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ -2 \\ -4 \\ 4 \end{pmatrix}.$$

(ii) Solve is using MATLAB. What do you obtain?

(iii) The result you have obtain in part (ii) occurs because one of the conditions of Theorem 8.3 is violated. What is the condition?

Solution:

(i) For this problem, equation (8.4) becomes

$$Y_0 = -1; \tag{1}$$
$$(1 + h(2 - x_n))Y_{n+1} - 2Y_n + (1 - h(2 - x_n))Y_{n-1} = 2h^2(2 - x_n), \quad 1 \leq n \leq 5 \tag{2}$$
$$Y_6 = 5. \tag{3}$$

Setting $h = 1$ in (4), this become

$$(3 - x_n)Y_{n+1} - 2Y_n + (-1 + x_n)Y_{n-1} = 4 - 2x_n; \tag{4}$$

We have the following 5 discrete equations for $Y_{1\ldots5}$, where Y_0 and Y_6 are the supplied BC.

$$\begin{aligned}
(3 - x_1)Y_2 - 2Y_1 + (-1 + x_1)Y_0 &= 4 - 2x_1; \\
(3 - x_2)Y_3 - 2Y_2 + (-1 + x_2)Y_1 &= 4 - 2x_2; \\
(3 - x_3)Y_4 - 2Y_3 + (-1 + x_3)Y_2 &= 4 - 2x_3; \\
(3 - x_4)Y_5 - 2Y_4 + (-1 + x_4)Y_3 &= 4 - 2x_4; \\
(3 - x_5)Y_6 - 2Y_5 + (-1 + x_5)Y_4 &= 4 - 2x_5.
\end{aligned}$$

Plugging in $x_i = i$ and $Y_0 = -1, Y_6 = 5$, and moving constants to the RHS, we are left

$$\begin{aligned}
2Y_2 - 2Y_1 &= 2; \\
Y_3 - 2Y_2 + Y_1 &= 0; \\
-2Y_3 + 2Y_2 &= -2; \\
-Y_5 - 2Y_4 + 3Y_3 &= -4; \\
-2Y_5 + 4Y_4 &= 4.
\end{aligned}$$

Intuitively obvious to the casual observer, this agrees with the desired linear system.

(ii) MATLAB obtain a vector of all NaN's, the matrix being singular to working precision.

```
% HW08 Problem 02

% set A,r from our problem
A = [-2,2,0,0,0;
    1,-2,1,0,0;
    0,2,-2,0,0;
    0,0,3,-2,-1;
    0,0,0,4,2];
r = [2;0;-2;-4;4];

% solve
x = A\r;
disp(x);
```

(iii) The requirement of Theorem 8.3 that $h\mathcal{P} \leq 2$ is violated. Here $P(x)$ achieves a magnitude of 8, at $x = 6$, and we had chose $h = 1$. Therefore, our $h\mathcal{P} = 8$.

3. (a) Give an operation count for finding L and U for a tridiagonal matrix, as per Eq. (8.21).
   (b) Give operation counts for solving the systems in (8.17), as per (8.22) and (8.23).
   (c) Given the total operations count for the Thomas algorithm.

Solution:

(a) We compute $M - 1$ new α values, and $M - 1$ new β values. Each new α requires one operation, and each β requires 2. Therefore, we use $3(M - 1)$ operations.

(b) Solving for $\vec{z}$ requires exactly $2(M - 1)$ operations. The solving for $\vec{y}$ requires exactly $1 + 3(M - 1)$ operations. In total, this is $1 + 5(M - 1)$ operations.

(c) The total operation count is (adding the two previous counts) $1 + 8(M - 1)$.

4. Use the `thomas.m` function, posted under "Codes for examples and selected homework problems," to solve a tridiagonal system $A\vec{y} = \vec{r}$ where A has '2' on the main diagonal and '-1' on the two subdiagonals. Take $\vec{r} = [1, -1, 1, -1, \ldots]^T$ and $M = 1000$ and 5000. Now solve the same system using Matlab's solver. Here, you need to investigate *two* cases: One, when A is constructed as a regular (i.e., full) matrix and two, when it is constructed as a sparse matrix.

   Compare the computational times required to solve this system for your code and for the MATLAB's solver, in those two cases. In particular, comment on *how the computational times scale with* M in each of the three cases considered.

Solution: I compare with the following code, and plot how the computational times scale with M (using more values of M). I find that both the `thomas.m` code and MATLAB's sparse solver scale linearly with M, with MATLAB's solver being faster.

For a non-sparse matrix, MATLAB's solver scales superlinearly (perhaps quadraticaly) with M, and becomes useless quickly.

```
% HW08 Problem 4

clear all;

Mvec=[100,1000,5000,10000,50000];
for i=1:length(Mvec)
    M = Mvec(i);
    a = -ones(M-1,1);
    b = 2.*ones(M,1);
    c = -ones(M-1,1);
    r = ones(M,1);
    r(2:2:M) = -ones(floor(M/2),1);

    tic;
    y = thomas(a,b,c,r);
    tom_t(i) = toc;



    % too slow for the bigger problem
    if i<5
```

```
        A = diag(a,-1) + diag(b) + diag(c,1);
        % check this is correct
        % A(1:10,1:10)
        tic;
        y = A\r;
        mat_t(i) = toc;
    end

    % make sparse
    a = -ones(M,1);
    c = -ones(M,1);
    A = spdiags(a,-1,M,M) + spdiags(b,0,M,M) + spdiags(c,1,M,M);
    % verify sparse storage
    % A(1:10,1:10)

    tic;
    y = A\r;
    mat_t_sp(i) = toc;
end


figure;
plot(Mvec,tom_t)
hold on;
plot(Mvec,mat_t_sp,'r')
legend('thomas algorithm','matlab sparse solver')
xlabel('M','FontSize',20)
ylabel('time','FontSize',20)

figure;
plot(Mvec(1:end-1),mat_t)
xlabel('M','FontSize',20)
ylabel('time','FontSize',20)
```

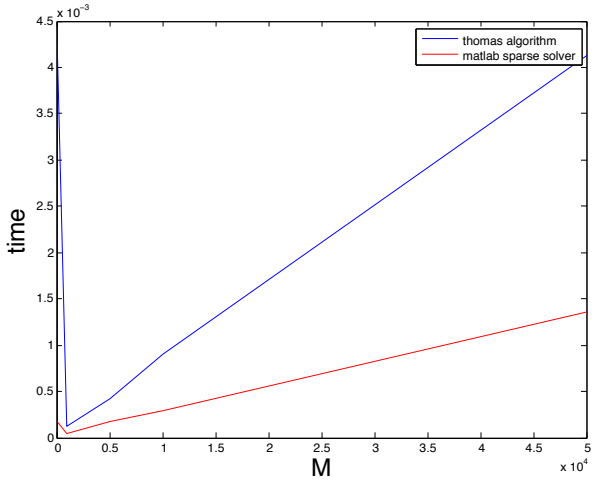

Figure 1: Scaling of computational time for thomas.m and MATLAB's built in solver, with a sparse matrix, versus M.

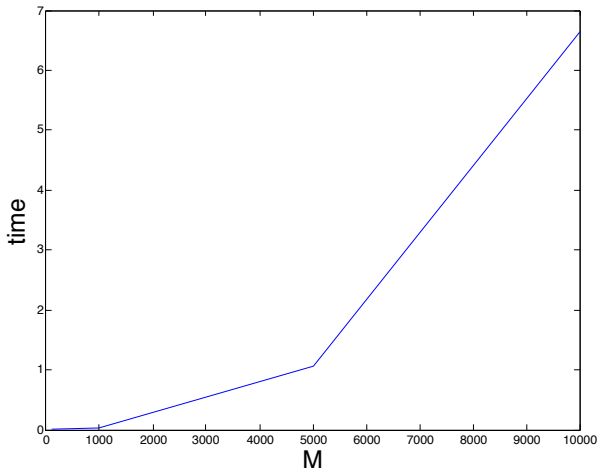

Figure 2: Scaling of computational time for MATLAB's built in solver, with a non-sparse matrix, versus M. It is impractical on a personal computer to extend M much further than a 10,000 by 10,000 matrix, which takes 7 second to solve.

5. Redo Problem 4 of HW07 using the discretized BVP (8.4) with $h = 0.09$ (ste size $h = 0.1$ will not "fit" into the interval $[0, 1.62]$). Compare the result with that found in HW07. Which method, shooting or finite-difference discretization, is preferable for solving BVPs like this one?

Bonus (a) Plot the error of your numerical solution. Explain the result.

Bonus (b) Repeat the problem with $h = 0.01$ and plot the error. Explain why it is greater than that for $h = 0.09$.

Solution: The problem is

$$y'' = 30^2(y - 1 + 2x), \quad y(0) = 1, \quad y(1.62) = -2.24.$$

Discretizing using (8.4) is

$$\begin{aligned} &Y_0 = 1; \\ &Y_{n+1} - Y_n(2 - 30^2h^2) + Y_{n-1} = 30^2h^2(2x_n - 1), \quad 1 \le n \le N - 1; \\ &Y_N = -2.24 \end{aligned}$$

I build this matrix in MATLAB and solve it using the built in solver.

Clearly, the finite-difference discretization is preferable to (sinlge) shooting for BVP like this one.

```
% HW08 Problem 5
%
%

h = 0.09;
x = 0:h:1.62;
y0 = 1;
yf = -2.24;
% set r
r = (30^2*h^2.*(2.*x(2:end-1)-1))';
% account for BC
r(1) = r(1) - y0;
r(end) = r(end) - yf;
% build A
A = diag((2-30^2*h^2)*ones(length(x)-2,1))+diag(ones(length(x)-3,1),-1)...
    +diag(ones(length(x)-3,1),1);

% check sizes
size(A)
size(r)

% solve
y = A\r;

% plot
tmpfigh = gcf;
clf;
figshape(600,600);
set(gcf,'Color','none');
set(gcf,'InvertHardCopy', 'off');
set(gcf,'DefaultAxesFontname','helvetica');
set(gcf,'DefaultLineColor','r');
```

```
set(gcf,'DefaultAxesColor','none');
set(gcf,'DefaultLineMarkerSize',5);
set(gcf,'DefaultLineMarkerEdgeColor','k');
set(gcf,'DefaultLineMarkerFaceColor','g');
set(gcf,'DefaultAxesLineWidth',0.5);
set(gcf,'PaperPositionMode','auto');

plot(x',[y0;y;yf],'LineWidth',2);
set(gca, 'fontsize',18)
xlabel('x','FontSize',20)
ylabel('y','FontSize',20)

psprintcpdf_keeppostscript(sprintf('andy_hw08_prb05_%02g',1));

% print('-depsc2','-zbuffer','-r200',sprintf('andy_hw08_prb05_%02g.eps',1))
% system(sprintf('epstopdf andy_hw08_prb05_%02g.eps',1));
```

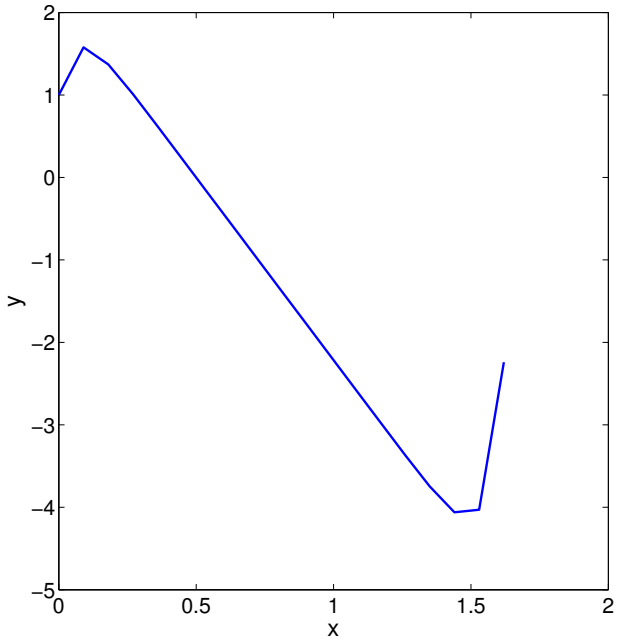

Figure 3: Solution the HW07 problem 4, using the finite difference scheme given by Eq (8.4).

6. Solve the BVP

$$(1+x)^2 y'' = 2y - 4, \qquad y(0) = 0, \qquad y(1) + 2y'(1) = 2$$

using the second-order accurate discretization (8.4) (for $n = 1, \ldots, N-1$) of this BVP. Use Method 1 of Sec. 8.4 modified in such a way that it can handle the mixed type BC at the *right* end point of the interval.

Confirm that you numerical solution has the second order of accurarcy by comparing it at different h with the exact solution $y_{\text{exact}} = 2x/(1+x)$. For this, do the following:

(i) Run your code with $h = 0.05$ and $h = 0.025$;

(ii) Plot the error as a function of x;

(iii) Confirm that the maximum error scales as $O(h^2)$.

Solution: To account for the mixed BC,

$$A_1 y(b) + A_2 y'(b) = \beta,$$

we introduce the point Y_{N+1} and approximate $y'(b)$ with a second order approximation

$$y'(b) = \frac{Y_{N+1} - Y_{N-1}}{2h} + O(h^2).$$

The mixed BC is therefore discretized as

$$A_1 Y_N + A_2 \frac{Y_{N+1} - Y_{N-1}}{2h} = \beta. \tag{5}$$

The ODE discretized at x_N is

$$(1 + P_N) Y_{N+1} - \left(2 - h^2 Q_N\right) Y_N + \left(1 - \frac{h}{2} P_N\right) Y_{N-1} = h^2 R_N \tag{6}$$

Solving for Y_{N+1} in (**??**) into (**??**) we have

$$-\left(2 - h^2 Q_N + 2h \frac{A_1}{A_2}\left(1 + \frac{h}{2} P_N\right)\right) Y_N + 2 Y_{N-1} = h^2 R_N - \left(1 + \frac{h}{2} P_N\right) \frac{2h\beta}{A_2} \tag{7}$$

where the first two terms on the LHS and the last term on the RHS make these matrix diagonal entries different. I solve this in MATLAB, and plot the error versus h. We find that the maximum error increases fourfold when h is doubled, confirming that the method is $O(h^2)$ accurate.

```
% HW08 Problem 6
%
%

% FOR MAKING SECOND PLOT
% tmpsym = {'o','s','v','o','s','v'};
% tmpcol = {'g','b','r','k','c','m'};
% tmpfigh = gcf;
% clf;
% figshape(600,600);
% set(gcf,'Color','none');
% set(gcf,'InvertHardCopy', 'off');
% set(gcf,'DefaultAxesFontname','helvetica');
% set(gcf,'DefaultLineColor','r');
% set(gcf,'DefaultAxesColor','none');
% set(gcf,'DefaultLineMarkerSize',5);
% set(gcf,'DefaultLineMarkerEdgeColor','k');
% set(gcf,'DefaultLineMarkerFaceColor','g');
% set(gcf,'DefaultAxesLineWidth',0.5);
% set(gcf,'PaperPositionMode','auto');

hvec = 10^-4:10^-3:10^-1; % FIRST PLOT
% hvec = [.05,.05/2]; % SECOND PLOT
for i=1:length(hvec)
    h = hvec(i);
    x = (0:h:1)';
    y0 = 0;
    yf = 2;
    % next two for mixed BC
```

```
    a1 = 1;
    a2 = 2;
    % set r, N of them
    r = -4*h^2./((1+x(2:end)).^2);
    % account for BC
    r(1) = r(1) - y0;
    r(end) = r(end)-2*h*yf/a2;
    % build A
    A = diag(-2-h^2*2./((1+x(2:end)).^2))... % main diagonal
        +diag(ones(length(x)-2,1),-1)... % sub
        +diag(ones(length(x)-2,1),1); % super
    % account for mixed BC
    A(end,end-1:end) = A(end,end-1:end)+[1,-2*h*a1/a2];
    % disp(A);

    % check sizes
    % size(A)
    % size(r)

    % solve
    yNum = [y0;A\r];
    % figure;
    yExact = 2.*x./(1+x);
    % FOR MAKING SECOND PLOT
%     plot(x,abs(yExact-yNum),'LineWidth',2,'Color',tmpcol{i});
%     hold on;
%     set(gca, 'fontsize',18);

    maxError = max(abs(yExact-yNum));
    errorVec(i) = maxError;
end

% FOR MAKING SECOND PLOT
% xlabel('x','FontSize',20)
% ylabel('error','FontSize',20)
% tmp1h = legend({'h = 0.05','h = 0.025'});
% set(tmp1h,'FontSize',17);
% legend boxoff
% psprintcpdf_keeppostscript(sprintf('andy_hw08_prb06_%02g',2));

errorVec = errorVec';

% FIST PLOT
% plot error versus h
tmpfigh = gcf;
clf;
figshape(600,600);
set(gcf,'Color','none');
set(gcf,'InvertHardCopy', 'off');
set(gcf,'DefaultAxesFontname','helvetica');
set(gcf,'DefaultLineColor','r');
set(gcf,'DefaultAxesColor','none');
set(gcf,'DefaultLineMarkerSize',5);
set(gcf,'DefaultLineMarkerEdgeColor','k');
set(gcf,'DefaultLineMarkerFaceColor','g');
set(gcf,'DefaultAxesLineWidth',0.5);
set(gcf,'PaperPositionMode','auto');

plot(hvec,errorVec,'LineWidth',2);
set(gca, 'fontsize',18)
xlabel('h','FontSize',20)
ylabel('max␣error','FontSize',20)
psprintcpdf_keeppostscript(sprintf('andy_hw08_prb06_%02g_tom',1));
```

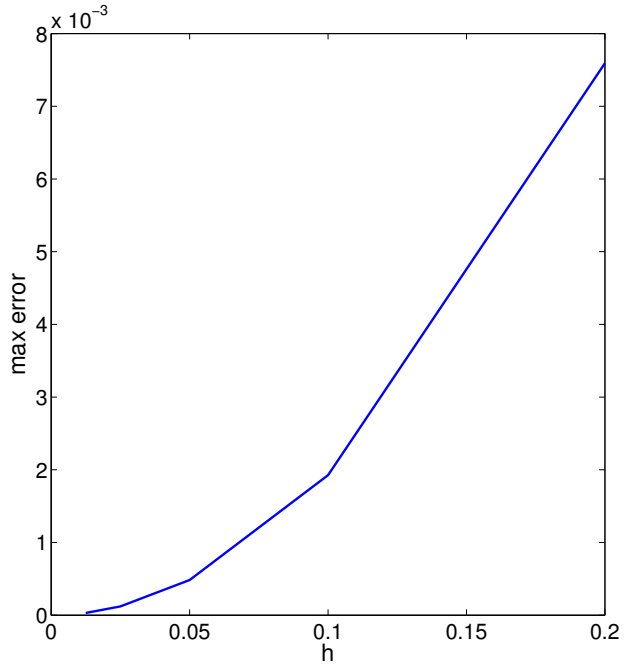

Figure 4: Error in the solution to the BVP with mixed BC at $x = b$, using Method 1 of Sec 8.4, versus h. We confirm that the method is second order.

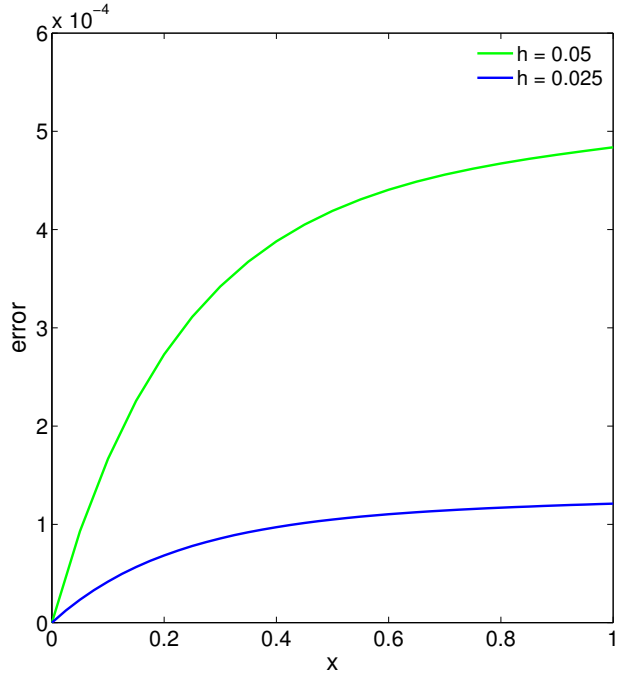

Figure 5: Error in the solution to the BVP with mixed BC at $x = b$, using Method 1 of Sec 8.4, versus x for two values of h.

7. Show that if condition (8.42) and the two conditions stated one line below it hold, then the coefficient matrix in Method 2 based on Eq. (8.39) is SDD.

   Bonus part: Equations (8.36) and (8.39) each lead to a second-order accurate method. Therefore, solutions obtained by those methods must differ by $O(h^3)$. Show *analytically* that this is indeed the case.

   Solution: Condition (8.42) is that

$$A_1 A_2 \leq 0$$

   meaning that they are of opposite parity. The other conditions ($Q \leq 0, h\mathcal{P} \leq 2$) will also be used. Consider that the only change in the coefficient matrix from using Method 2, from the form of the original triadiagonal matrix (8.6) is the first row.

   To retain SDD of the coefficient matrix, we therefore require that the entry in row 1, column 1 is greater than the entry in row 1, column 2. Those entries are the coefficients on $Y_{0,1}$, which appear in equation (8.39). Taking the hint, I begin by multiplying (8.39) by h/A_2. This yeilds:

$$hY_0\frac{A_1}{A_2} + Y_1 - Y_0 - \frac{h^2}{2}R_0 + \frac{h}{2}P_0Y_1 - \frac{h}{2}P_0Y_0 + \frac{h^2}{2}Q_0Y_0 = \frac{\alpha h}{A_2}$$

   The terms in Y_0 and Y_1 are thus

$$\begin{aligned} Y_0: &\quad h\frac{A_1}{A_2} - 1 - \frac{h}{2}P_0 + \frac{h^2}{2}Q_0 \\ Y_1: &\quad 1 + \frac{h}{2}P_0 \end{aligned}$$

   Now to show that the maginude of the Y_0 terms is greater than the magnitude of the Y_1 terms, we have (note we drop the inequality since every term is positive by Condition 8.42 and the other conditions)

$$\begin{aligned} \left|h\frac{A_1}{A_2} - 1 - \frac{h}{2}P_0 + \frac{h^2}{2}Q_0\right| &= \left|\left(1 + \frac{h}{2}P_0\right) - h\frac{A_1}{A_2} - \frac{h^2}{2}Q_0\right| \\ &= \left(1 + \frac{h}{2}P_0\right) - h\frac{A_1}{A_2} - \frac{h^2}{2}Q_0 \\ &> 1 + \frac{h}{2}P_0 \\ &= \left|1 + \frac{h}{2}P_0\right|, \end{aligned}$$

   which is the magnitude of the coefficient on Y_1, as desired.

8. Solve the BVP stated in Problem 5 of HW 7 using Picard iterations.

Solution: Plot and solution code follow.

The number of iterations does not appear to depend on h, nor does the convergence. For both values of h, the shallow solution converges while the deep solution diverges. The convergence of the shallow solution is achieved in 11 iterations for each h.

```
% HW08 Problem 08
%
% solve the nonlinear BVP using picard iterations

% IC, BC
y0 = 1; yf = 1;
hvec = [0.1,0.2];
ysoln = {'shallow','deep'};
tol = 10^(-6);
errorcell = cell(4,1);
for i=1:2
    h = hvec(i);
    tvec = (0:h:2)';
    % set both y solutions
    yvec = {tvec.^0,[1-16.*(0:h:1)';-31+16.*(1+h:h:2)']};
    N = length(tvec)-2;
    A = (spdiags(ones(N,1),-1,N,N)+spdiags(-2.*ones(N,1),0,N,N)+spdiags(ones(N,1),1,N,
        N));

    for j=1:2
        err = 1;
        itercount = 0;
        errvec = [];
        % set Y initially
        y = yvec{j};
        y = y(2:end-1);
        figure;
        while err > tol
            itercount=itercount+1;
            ynew = A\andy_hw08_prb08_r(tvec(2:end-1),y,h,y0,yf);
            disp(ynew);
            err = sqrt(sum((ynew-y).^2));
            disp(err);
            errvec = [errvec err];
            y = ynew;
            plot(tvec,[y0;y;yf])
            %pause(0.5);
            hold on;
        end
        fprintf('took␣%g␣iterations␣for␣y␣%g␣with␣h␣=␣%g\n',itercount,j,h);
        errorcell{i+2*(j-1)} = errvec;
        soln = [y0;y;yf];
        % MAKE INTERMEDIATE PLOTS
%         figure;
%         plot(tvec,soln);
%         xlabel('x','FontSize',20);
%         ylabel('y','FontSize',20);
%         title(sprintf('h = %g, %s y',h,ysoln{j}));
%         set(gcf, 'units', 'inches', 'position', [1 1 10 10])
%         set(gcf,'PaperPositionMode','auto')
%         print('-depsc2','-zbuffer','-r200',sprintf('andy_hw08_prb08_%02g.eps',j+2*(i
    -1)))
%         system(sprintf('epstopdf andy_hw08_prb08_%02g.eps; \\rm andy_hw08_prb08_%02g
    .eps',j+2*(i-1),j+2*(i-1)));
%
```

```
%         figure;
%         plot(1:length(errvec),log10(errvec));
%         xlabel('iteration','FontSize',20);
%         ylabel('log10(error)','FontSize',20);
%         title(sprintf('h = %g, %s y',h,ysoln{j}));
%         set(gcf, 'units', 'inches', 'position', [1 1 10 10])
%         set(gcf,'PaperPositionMode','auto')
%         print('-depsc2','-zbuffer','-r200',sprintf('andy_hw08_prb08_%02g_err.eps',j
    +2*(i-1)))
%         system(sprintf('epstopdf andy_hw08_prb08_%02g_err.eps; \\rm andy_hw08_prb08_
    %02g_err.eps',j+2*(i-1),j+2*(i-1)));
    end
end

%% make nice plots
figure;
% plot error versus iterations
tmpfigh = gcf;
clf;
figshape(600,600);
set(gcf,'Color','none');
set(gcf,'InvertHardCopy', 'off');
set(gcf,'DefaultAxesFontname','helvetica');
set(gcf,'DefaultLineColor','r');
set(gcf,'DefaultAxesColor','none');
set(gcf,'DefaultLineMarkerSize',5);
set(gcf,'DefaultLineMarkerEdgeColor','k');
set(gcf,'DefaultLineMarkerFaceColor','g');
set(gcf,'DefaultAxesLineWidth',0.5);
set(gcf,'PaperPositionMode','auto');

plot(log10(errorcell{1}),'LineWidth',2,'Color','b');
hold on;
plot(log10(errorcell{2}),'LineWidth',2,'Color','g');
tmp1h = legend({'h␣=␣0.1','h␣=␣0.2'});
set(tmp1h,'FontSize',17);
legend boxoff
set(gca, 'fontsize',18)
xlabel('iteration','FontSize',20)
ylabel('log10(error)','FontSize',20)
psprintcpdf_keeppostscript(sprintf('andy_hw08_prb08_shallowerror'));

%% make nice plots
figure;
% plot error versus iterations
tmpfigh = gcf;
clf;
figshape(600,600);
set(gcf,'Color','none');
set(gcf,'InvertHardCopy', 'off');
set(gcf,'DefaultAxesFontname','helvetica');
set(gcf,'DefaultLineColor','r');
set(gcf,'DefaultAxesColor','none');
set(gcf,'DefaultLineMarkerSize',5);
set(gcf,'DefaultLineMarkerEdgeColor','k');
set(gcf,'DefaultLineMarkerFaceColor','g');
set(gcf,'DefaultAxesLineWidth',0.5);
set(gcf,'PaperPositionMode','auto');

plot(log10(errorcell{3}),'LineWidth',2,'Color','b');
hold on;
plot(log10(errorcell{4}),'LineWidth',2,'Color','g');
tmp1h = legend({'h␣=␣0.1','h␣=␣0.2'});
set(tmp1h,'FontSize',17);
legend boxoff
```

```
set(gca, 'fontsize',18)
xlabel('iteration','FontSize',20)
ylabel('log10(error)','FontSize',20)
psprintcpdf_keeppostscript(sprintf('andy_hw08_prb08_deeperror'));

% close all
```

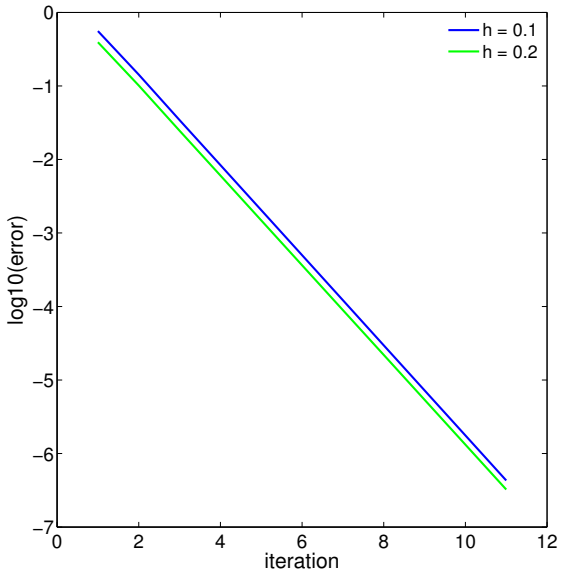

Figure 6: Logarithm of the error versus iteration number for the Picard iterations, starting close to the shallow solution. Since this looks like a straight line, we observe that the Picard method converges linearly.

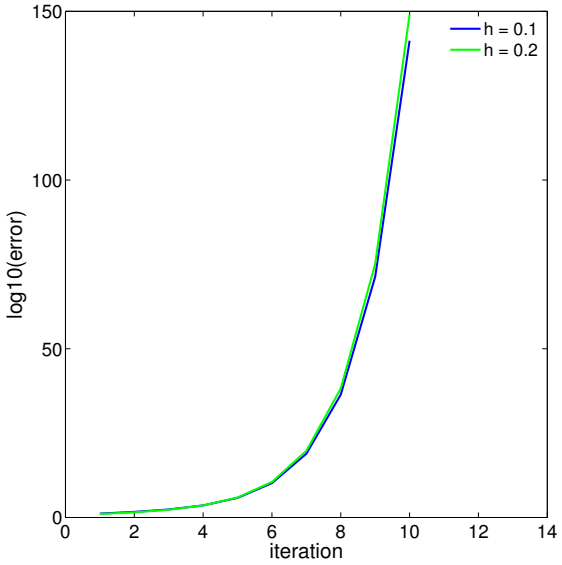

Figure 7: Logarithm of the error versus iteration number for the Picard iterations, starting close to the deep solution. The iterations do not converge, and the error runs away.

9. Repeat Problem 8 using the modified Picard's iterations.

Solution: Plot and solution code follow.

I present a summary of the results in a table, where under each value of h I report the number of iterations required. The "weird" behavior manifests as an iteration does not converge to the solution, or go off to infinity, but bounces between two (or more) different y. In the case of simple Picard, the solution bounces between two different y, and in modified Picard the behavior is more complicated. This conclusion was arrived at by plotting y at each iterate, as to see the behavior directly. In the code, setting "plot" to 1 in the Picard function shows these plots (it is suggested to lower the maximum iterate count in this case).

solution type	c	$h = 0.1$	$h = 0.2$
shallow	100	314	204
	20	77	74
	10	43	42
	1.5	10	10
	0.75	6	6
	0	11	11
	-0.5	25	24
	-0.9	327	416
	-1	–*	–*
	-1.5	–	–
deep	-2.9	–	–
	-4	70	57
	-3.9	335	162
	-3.75	–*	–*
	-3.62	–*	–*
	-3.6	–*	–*
	-3.55	–*	–*
	-3.5	–	–*
	-5.95	22	24
	-6.45	42	54
	-6.95	933	25
	-7.45	13	11
deeper	-4	–	–
	-3.9	–	–
	-5.95	24	26
	-6.45	46	59
	-6.95	–	–
	-7.45	–	–

Table 1: The iteration count for convergence of Modified Picard to the BVP in Problem 5 of HW7. The – denotes where the solution explodes (runs off the inifinity), and –* where the solution does not converge, but remains bounded.

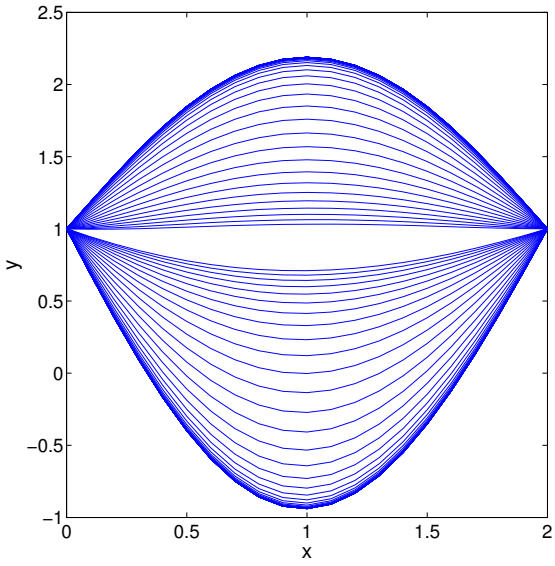

Figure 8: A plot of the iterations in a bounded orbit of length two. The iterations alternate between the top and bottom solutions.

```
% HW08 Problem 08
%
% solve the nonlinear BVP using picard iterations

% IC, BC
y0 = 1; yf = 1;
hvec = [0.1,0.2];
ysoln = {'shallow','deep','deeper'};

%% shallow solution
m = 1;
% cvec = [100,20,10,1.5,.75,0,-0.5,-0.9,-1,-1.5];
cvec = [-1];
for i=1:length(cvec)
    c = cvec(i);
    for j=1:2
        % set h
        h = hvec(j);
        % now set all the stuff that depends on h
        tvec = (0:h:2)';
        N = length(tvec)-2;
        % shallow, deep, deeper
        yvec = {tvec.^0,[1-16.*(0:h:1)';-31+16.*(1+h:h:2)'],...
            [1-20.*(0:h:1)';-39+20.*(1+h:h:2)']};
        % choose the y solution that we want
        y = yvec{m};
        % build A
        A = (spdiags(ones(N,1),-1,N,N)+spdiags(-2.*ones(N,1),0,N,N)+spdiags(ones(N,1)
            ,1,N,N));
        % test without modified method
        %k = andy_hw08_prb09_picard(@andy_hw08_prb08_r,A,tvec,y,1,10^(-6),h,'plot',1,'
            modified',0);
        % now run modified method (the default for the picard function)
        fprintf('%s␣y␣with␣h␣=␣%g,␣c␣=␣%g\n',ysoln{m},h,c);
        k = andy_hw08_prb09_picard(@andy_hw08_prb08_r,A,tvec,y,c,10^(-6),h,'plot',1);
        fprintf('took␣%g␣iterations␣for␣%s␣y␣with␣h␣=␣%g,␣c=%g\n',k,ysoln{m},h,c);
```

```
    end
end

% plot a figure showing the odd behavior
tmpfigh = gcf;
figshape(600,600);
set(gcf,'Color','none'); set(gcf,'InvertHardCopy', 'off');
set(gcf,'DefaultAxesFontname','helvetica');
set(gcf,'DefaultLineColor','r');
set(gcf,'DefaultAxesColor','none');
set(gcf,'DefaultLineMarkerSize',5);
set(gcf,'DefaultLineMarkerEdgeColor','k');
set(gcf,'DefaultLineMarkerFaceColor','g');
set(gcf,'DefaultAxesLineWidth',0.5);
set(gcf,'PaperPositionMode','auto');
set(gca, 'fontsize',18);
xlabel('x','FontSize',20);
ylabel('y','FontSize',20);
psprintcpdf_keeppostscript(sprintf('andy_hw08_prb09_illustrative'));

%% deep solution
m = 2;
cvec = [-2.9,-4,-3.9,-3.75,-3.62,-3.6,-3.55,-3.5,-5.95,-6.45,-6.95,-7.45];
for i=1:length(cvec)
    c = cvec(i);
    for j=1:2
        h = hvec(j);
        tvec = (0:h:2)';
        N = length(tvec)-2;
        yvec = {tvec.^0,[1-16.*(0:h:1)';-31+16.*(1+h:h:2)'],...
            [1-20.*(0:h:1)';-39+20.*(1+h:h:2)']};
        y = yvec{m};
        A = (spdiags(ones(N,1),-1,N,N)+spdiags(-2.*ones(N,1),0,N,N)+spdiags(ones(N,1)
            ,1,N,N));
        k = andy_hw08_prb09_picard(@andy_hw08_prb08_r,A,tvec,y,c,10^(-6),h,'plot',0);
       fprintf('took␣%g␣iterations␣for␣%s␣y␣with␣h␣=␣%g,␣c=%g\n',k,ysoln{m},h,c);
    end
end

%% deeper solution
m = 3;
cvec = [ -4,-3.9,-5.95,-6.45,-6.95,-7.45];
for i=1:length(cvec)
    c = cvec(i);
    for j=1:2
        h = hvec(j);
        tvec = (0:h:2)';
        N = length(tvec)-2;
        yvec = {tvec.^0,[1-16.*(0:h:1)';-31+16.*(1+h:h:2)'],...
            [1-20.*(0:h:1)';-39+20.*(1+h:h:2)']};
        y = yvec{m};
        A = (spdiags(ones(N,1),-1,N,N)+spdiags(-2.*ones(N,1),0,N,N)+spdiags(ones(N,1)
            ,1,N,N));
        k = andy_hw08_prb09_picard(@andy_hw08_prb08_r,A,tvec,y,c,10^(-6),h,'plot',0);
       fprintf('took␣%g␣iterations␣for␣%s␣y␣with␣h␣=␣%g,␣c=%g\n',k,ysoln{m},h,c);
    end
end
```

10. Repeat Problem 8 using the Newton-Raphson method. Answer all of the questios posed in Problem 8. As for the logarithm of the error versus the iteration count, you should obtain a parabola facing downward. Thus, the Newton-Raphson method converges (when it does so) *quadratically*. Is the type of convergence asymptomatically faster or slower than the linear convergence of Picard's methods?

Solution: Plot and solution code follow. I adapt the code from Problem 8, where the only differences are the construction of the matrix A at each iteration, the r vector, and that the result of the matrix solver is the error, which is subtracted for a new solution (it is not the new solution itself).

The number of iterations does not appear to depend on h, nor does the convergence. For both values of h, the shallow solution converges while the deep solution diverges. The convergence of the shallow solution is achieved in 11 iterations for each h.

```
% HW08 Problem 10
%
% solve the nonlinear BVP using Newton-Raphson

% IC, BC
y0 = 1; yf = 1;
hvec = [0.1,0.2];
ysoln = {'shallow','deep'};
tol = 10^(-6);
errorcell = cell(4,1);
for i=1:2
    h = hvec(i);
    tvec = (0:h:2)';
    % set both y solutions
    yvec = {tvec.^0,[1-16.*(0:h:1)';-31+16.*(1+h:h:2)']};
    N = length(tvec)-2;
    A = (spdiags(ones(N,1),-1,N,N)+spdiags(-2.*ones(N,1),0,N,N)+spdiags(ones(N,1),1,N,
        N));

    for j=1:2
        err = 1;
        itercount = 0;
        errvec = [];
        % set Y initially
        y = yvec{j};
        y = y(2:end-1);
        figure;
        while err > tol
            itercount=itercount+1;
            A = spdiags(-(2+2*h^2.*y./(2+tvec(2:end-1))),0,N,N)+spdiags(ones(N,1),-1,N
                ,N)+spdiags(ones(N,1),1,N,N);
            epsnew = A\andy_hw08_prb10_r(tvec(2:end-1),y,h,y0,yf);
            ynew = y-epsnew;
            disp(ynew);
            err = sqrt(sum((ynew-y).^2));
            disp(err);
            errvec = [errvec err];
            y = ynew;
            plot(tvec,[y0;y;yf])
            hold on;
            if itercount > 1000
                break
            end
        end
        fprintf('took␣%g␣iterations␣for␣y␣%g␣with␣h␣=␣%g\n',itercount,j,h);
```

```
        errorcell{i+2*(j-1)} = errvec;
        soln = [y0;y;yf];
    end
end

%% make nice plots
figure;
% plot error versus iterations
tmpfigh = gcf;
clf;
figshape(600,600);
set(gcf,'Color','none');
set(gcf,'InvertHardCopy', 'off');
set(gcf,'DefaultAxesFontname','helvetica');
set(gcf,'DefaultLineColor','r');
set(gcf,'DefaultAxesColor','none');
set(gcf,'DefaultLineMarkerSize',5);
set(gcf,'DefaultLineMarkerEdgeColor','k');
set(gcf,'DefaultLineMarkerFaceColor','g');
set(gcf,'DefaultAxesLineWidth',0.5);
set(gcf,'PaperPositionMode','auto');

plot(log10(errorcell{1}),'LineWidth',2,'Color','b');
hold on;
plot(log10(errorcell{2}),'LineWidth',2,'Color','g');
tmp1h = legend({'h␣=␣0.1','h␣=␣0.2'});
set(tmp1h,'FontSize',17);
legend boxoff
set(gca, 'fontsize',18)
xlabel('iteration','FontSize',20)
ylabel('log10(error)','FontSize',20)
psprintcpdf_keeppostscript(sprintf('andy_hw08_prb10_shallowerror'));

%% make nice plots
figure;
% plot error versus iterations
tmpfigh = gcf;
clf;
figshape(600,600);
set(gcf,'Color','none');
set(gcf,'InvertHardCopy', 'off');
set(gcf,'DefaultAxesFontname','helvetica');
set(gcf,'DefaultLineColor','r');
set(gcf,'DefaultAxesColor','none');
set(gcf,'DefaultLineMarkerSize',5);
set(gcf,'DefaultLineMarkerEdgeColor','k');
set(gcf,'DefaultLineMarkerFaceColor','g');
set(gcf,'DefaultAxesLineWidth',0.5);
set(gcf,'PaperPositionMode','auto');

plot(log10(errorcell{3}),'LineWidth',2,'Color','b');
hold on;
plot(log10(errorcell{4}),'LineWidth',2,'Color','g');
tmp1h = legend({'h␣=␣0.1','h␣=␣0.2'});
set(tmp1h,'FontSize',17);
legend boxoff
set(gca, 'fontsize',18)
xlabel('iteration','FontSize',20)
ylabel('log10(error)','FontSize',20)
psprintcpdf_keeppostscript(sprintf('andy_hw08_prb10_deeperror'));

% close all
```

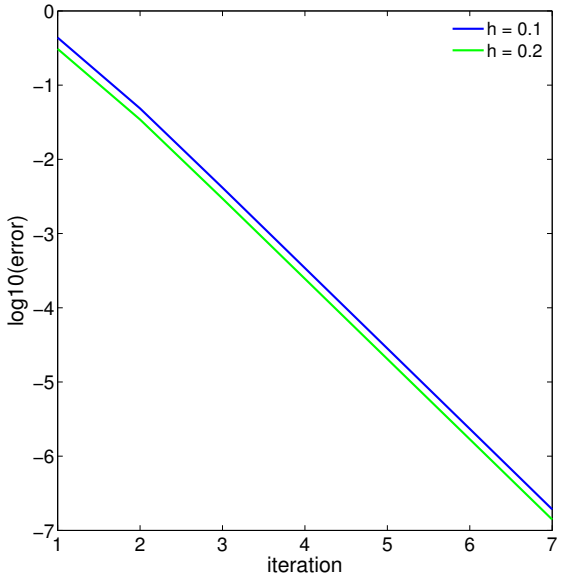

Figure 9: Logarithm of the error versus iteration number for the Picard iterations, starting close to the shallow solution. Since this looks like a straight line, we observe that the Picard method converges linearly.

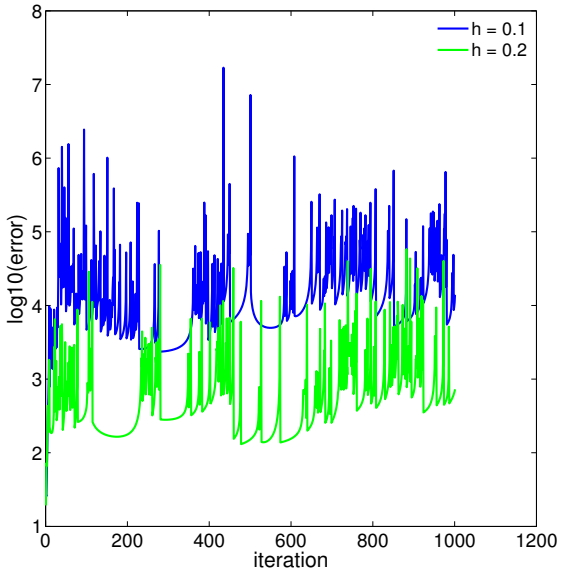

Figure 10: Logarithm of the error versus iteration number for the Picard iterations, starting close to the deep solution. The iterations do not converge, and the error runs away.

11. Compare your results in Problems 8-10 and draw conclusions. Namely, based on your experience with the previous three problems, list strengths and weaknesses for each of the three methods. Then explain under which circumstances you would and would not use, each of these methods.

Solution:

method	strengths	weaknesses
Picard	easy to code	cannot obtain deep solution linear convergence
modifid Picard	can obtain both solutions	linear convergence very sensitive to c
Newton-Raphson	fast (quad. conv.)	

Table 2: The strengths and weaknesses of the BVP iteration schemes tested.

Appendix 1: ODE Functions

```
% HW07 Problem 05
%
% solve the nonlinear BVP using the shooting method
% shoot iteratively
% use the secand method for solving

% IC, BC
y0 = 1; yf = 1;
h = 0.02;
tvec = 0:h:2;
tol = 10^(-3);

% initialize theta and f
thetacell = {[-2,-1],[2,1]};
for j=1:2
    err = 1;
    i=2;
    theta = thetacell{j};
    % disp(theta);
    shot1 = andy_ME(@andy_hw07_prb05_ODE,tvec,[y0;theta(1)],h,[]);
    shot2 = andy_ME(@andy_hw07_prb05_ODE,tvec,[y0;theta(2)],h,[]);
    f = [shot1(1,end),shot2(1,end)];
    while err > tol
        % generate new theta
        theta = [theta theta(end)-f(end)/((f(end)-f(end-1))/(theta(end)-theta(end-1)))];
        % solve the ODE
        shot = andy_ME(@andy_hw07_prb05_ODE,tvec,[y0;theta(end)],h,[]);
        % f is y(2) at that theta
        f = [f shot(1,end)-yf];
        % count the iterations
        i=i+1;
        % error is just f
        err = abs(f(end));
    end
    fprintf('took␣%g␣iterations␣for␣theta␣%g␣\n',i,j);
    % disp(theta);

    soln = shot;
    figure;
    plot(tvec,soln(1,:));
    xlabel('x','FontSize',20);
    ylabel('y','FontSize',20);
    set(gcf, 'units', 'inches', 'position', [1 1 10 10])
    set(gcf,'PaperPositionMode','auto')
    print('-depsc2','-zbuffer','-r200',sprintf('andy_hw07_prb05_%02g.eps',j))
    system(sprintf('epstopdf␣andy_hw07_prb05_%02g.eps;␣\\rm␣andy_hw07_prb05_%02g.eps',j,
        j));
end
```

Appendix 2: Numerical Methods

```
function r = andy_hw08_prb08_r(x,y,h,alpha,beta,varargin)

r = (h^2.*y.^2./(2+x));
% disp(r);
r(1) = r(1) - alpha; r(end) = r(end) - beta;
```

```
function [k,errvec] = andy_hw08_prb09_picard(rfunc,A,tvec,yguess,c,tol,h,varargin)
%
% implements the modified picard method
% made a function so that I can throw values of c at it
%
% return the iteration count

% parse input
pb = 0; vb = 0; m = 1;
for i=1:2:nargin-7
    switch varargin{i}
        case 'plot'
            pb = varargin{i+1};
        case 'verbose'
            vb = varargin{i+1};
        case 'modified'
            m = varargin{i+1};
    end
end

% don't need to solve for the BV
y = yguess(2:end-1);
% adjust A for the modified Picard
if m
    A = A-h^2*c.*spdiags(ones(length(A(:,1)),1),0,length(A(:,1)),length(A(:,1)));
end
k = 0;

if pb
    figure;
end

errvec = [];
err = 2*tol;

while err > tol
    k=k+1;
    if m
        ynew = A\(rfunc(tvec(2:end-1),y,h,yguess(1),yguess(end))-h^2*c.*y);
    else
        ynew = A\rfunc(tvec(2:end-1),y,h,yguess(1),yguess(end));
    end
    err = sqrt(sum((ynew-y).^2));
    if vb
        disp(ynew);
        disp(err);
    end
    errvec = [errvec err];
    y = ynew;
    if pb
        plot(tvec,[yguess(1);y;yguess(end)])
        hold on;
        pause(0.1);
    end
    if err > 10^5
```

```
        disp('solution exploded')
        break
    end
    if k > 10^3
        disp('iterations do NOT converge !!!!!!!!!!!!!!')
        break
    end
end

if err <= tol
    disp('converged!!')
end
```

```
function r = andy_hw08_prb10_r(x,y,h,alpha,beta,varargin)
% return the RHS r vector for Newton-Raphson
%
% make sure to pass x from x_1 to x_N-1
% and same with y

r = -2.*y-h^2.*y./(2+x);
r = r+[y(2:end);beta];
r = r+[alpha;y(1:end-1)];
```