BVP-ODE

For the problem in exercise #5 of the document Tridiagonal Matrix-Examples.PDF

$$\frac{d^2u}{dx^2} = 1, \qquad u'(0) = 0.5, \qquad u(1) = 2$$

Chossing initially $\Delta x = 0.2$ numerically solve the BVP-ODE using

(1) Shooting Method with Euler Method
(2) Shooting Method with RK-4. Use the following notation to help grading:

Auxiliary Variables	Slopes
A	A1, A2, A3, A4
B	B1, B2, B3, B4

(3) The Finite Difference-Tridiagonal Matrix Method

SHOOTING METHOD WITH either Euler or RK-4

Reduce the original 2nd order ODE to a set of TWO 1st order ODEs. As the ODE is of second order, we need two auxiliary variables which will yield TWO 1st order ODEs.

A = u
B = A' = u'
B'=A''=u''

Using the above definitions and the original equation yield a system of 1st order ODEs:

A'=B compare to the standard format $\frac{dA}{dx} = f_1(x, A, B)$
B'=1 compare to the standard format $\frac{dB}{dx} = f_2(x, A, B)$

The two boundary conditions become

BC	IC's
u'(0)=0.5	B(0)=0.5
u(1)=2	A(1)=2

If you analyze carefully, the transformation A(1)=2 is not an IC and can't become one. But it is used to require that the A solution ends at A(1)=2

Then the ODE set will be:

ODEs	Initial Condition
A'=B	A(0)=GUESS
B'=1	B(0)=0.5

SHOOTING METHOD WITH RK-4

Note: The accuracy of the guessed value of A(0)=GUESS for the first equation is going to be evaluated with the calculated value of A(x=1) vs the real value A(x=1)=2. We assume values of A(x=0) such as the value A(x=1) yields a value equal to 2 (or below and above 2 which allows interpolation).

(1) The code below was executed with RK-4 (code below) with the initial guess of A(x=0)=0 , which produced A(x=1)=1.0. This is a low value compared with the real A(x=1)=2.0.
(2) Then the code was executed with A(x=0)=2, which produced A(x=1)=3.0. This is a high value compared with the real A(x=1)=2.0.
(3) An easy linear interpolation yields A(x=0)=1, then the code was executed and produced A(x=1)=2.0. This is exactly the real value. NO further interpolation is needed.

The obtained results are summarized below:

	GUESS	Computed Values	Quality of Guess
	A(0)=0	A(1)=1.0	LOW
INTERPOLATION	A(0)=1	A(1)=2.0	REAL
	A(0)=2	A(1)=3.0	HIGH

```
% RK4.m
% Runge-Kutta 4th Order Method (file: RK4sysODE_ex3.m)
% Solves a system of ODE using Runge-Kutta 4th order method
% four different values of h are tested
% Variables:
% h=[step size],x=[a,b] domain, x=independent variable
% A, B =[auxiliary variables] A1,A2,A3,A4 =[slopes]
% B1, B2, B3, B4 = [slopes]

clc, clear all, close all

fid=fopen('RKoutput3.txt','w'); % write results on external file

colorM={'cx','b*','g+','ro'};  % different markers for the plot
                               % first letter is the color
                               % 2nd letter is the marker shape
h=0.2;     % initial step size

for jj=1:1:4   % 4 iterations, halving h each time

    a=0; b=1;          % solution range
```

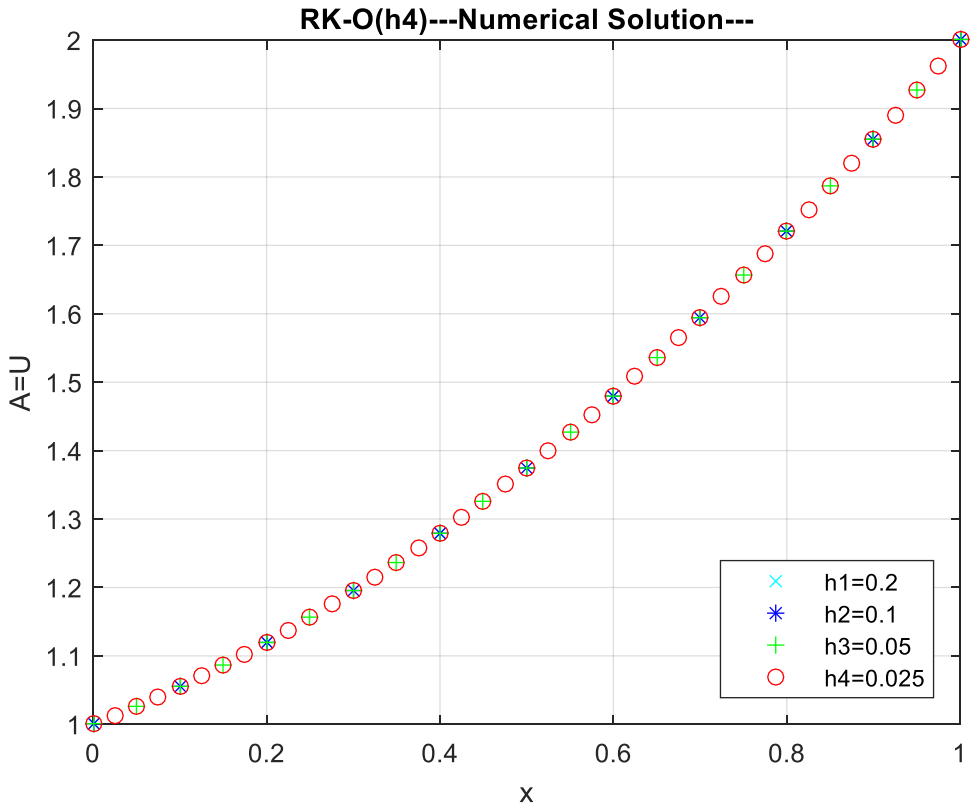
```
    x = [a:h:b];      % Calculates up to t(n)
    n = numel(x);     % number of x-elements, number of discrete pnts.

    A = zeros(1,n);   % Memory preallocation [to speed code)
    B = zeros(1,n);

    A(1) = 1.0;       % initial condition GUESS
    B(1) = 0.5;       % initial condition

    F1 =@(B) B;         % change the function as you desire
    F2 =1;

    % table title
    fprintf(fid,'%7s %7s %7s %7s \n','i','x(i)','A(i)','B(i)');

    for ii=1:1:(n-1)                % calculation loop
        A1 = F1(B(ii));
        B1 = F2;

        A2 = F1(B(ii)+0.5*h*B1);
        B2 = F2;

        A3 = F1(B(ii)+0.5*h*B2);
        B3 = F2;

        A4 = F1(B(ii)+h*B3);
        B4 = F2;

        A(ii+1) = A(ii) + (1/6)*h*(A1+2*A2+2*A3+A4);  % main equation
        B(ii+1) = B(ii) + (1/6)*h*(B1+2*B2+2*B3+B4);

        fprintf(fid,'%7d %7.2f %7.3f %7.3f \n',ii, x(ii),A(ii),B(ii));

    end

    fprintf(fid,'%7d %7.2f %7.3f %7.3f \n', n, x(n), A(n),B(n));

    plot(x,A,colorM{jj});  % Plot statements
    hold on           % allows multiple plots in a single graph

    h=(1/2)*h  % half the step for each new iteration
end

title('RK-O(h4)---Numerical Solution---');
ylabel('A=U');    xlabel('x');
grid on
lgd=legend('h1=0.2','h2=0.1','h3=0.05','h4=0.025');
lgd.Location='southeast';

hold off
fclose(fid);
```

OUTPUT. Only the first results with h=0.2, and the guesses A(x=1)=1 are shown. The others results, with smaller h's are not shown. Method is so accurate than all results with different h's overlap.

i	x(i)	A(i)	B(i)
1	0.00	1.000	0.500
2	0.20	1.120	0.700
3	0.40	1.280	0.900
4	0.60	1.480	1.100
5	0.80	1.720	1.300
6	1.00	2.000	1.500

h=0.2

NOTE That with the guessed value of A(x=0)=1 the obtained result with the Shooting Method is in fact A(x=1)=2.0 (which corresponds to A(6)=2.0 in matlab code).



The solution for the U problem is the **A-problem** solution, only. In the above graph markers overlap, which means that reduction of error is negligible and the reduction of h was not needed. The largest value of h=0.2 is OK.

SHOOTING METHOD WITH EULER

This code was actually produce by simplifying the RK-4. It is important for the simplicity of the code but it's less accurate:

```
% EulerSystem.m
% Solves a system of ODE using Euler method
% Four different values of h are tested
% Variables:
% h=[step size],x=[a,b] domain
% A,B =[auxiliary variables] A1=[slopes]
% B1 [slopes]

clc, clear all, close all

fid=fopen('Eulerout.txt','w'); % write results on external file

colorM={'cx','b*','g+','ro'};  % different markers for the plot
                                                    % first letter is the color
                                                    % 2nd letter is the marker shape
h=0.25; % initial step size

for jj=1:1:4   % 4 iterations, halving h each time

   a=0; b=1;                % solution range
   x = [a:h:b];             % Calculates up to t(n)
   n = numel(x);            % number of x-elements, number of discrete points

   A = zeros(1,n);          % Memory preallocation [to speed code)
   B = zeros(1,n);

   A(1) = 1.0;              % initial condition GUESS
   B(1) = 0.5;      % initial condition

   F1 =@(B) B;        % change the function as you desire
   F2 =1;

   % table title
   fprintf(fid,'%7s %7s %7s %7s \n','i','x(i)','A(i)','B(i)');
```

```
   for ii=1:1:(n-1)              % calculation loop
      A1 = F1(B(ii));
      B1 = F2;

      A(ii+1) = A(ii) + h*A1;  % main equation
      B(ii+1) = B(ii) + h*B1;

      fprintf(fid,'%7d %7.2f %7.3f %7.3f \n',ii, x(ii), A(ii),B(ii));

   end

   fprintf(fid,'%7d %7.2f %7.3f %7.3f \n',n, x(n), A(n),B(n));

   plot(x,A,colorM{jj});   % Plot statements
   hold on                 % allows multiple plots in a single graph

   h=(1/2)*h  % half the step for each new iteration
end

title('Euler Method-O(h)---Numerical Solution---');
ylabel('A=U');    xlabel('x');
grid on
lgd=legend('h1=0.25','h2=0.125','h3=0.0675','h4=0.03375');
lgd.Location='southeast';

hold off
fclose(fid);
```

FINITE DIFFERENCE METHOD-TRIDIAGONAL MATRIX

Example 5

$$d^2u/dx^2 = 1\ , \quad u'(0) = 0.5\ , u(1) = 2$$

The analytic solution is $u(x) = x^2/2 + x/2 + 1$. Choosing $\Delta x = 0.2$ and following the same first step for Example 1, we obtain

$$\begin{aligned} u_0 - 2\,u_1 + u_2 &= (\Delta x)^2 \\ u_1 - 2\,u_2 + u_3 &= (\Delta x)^2 \\ u_2 - 2\,u_3 + u_4 &= (\Delta x)^2 \\ u_3 - 2\,u_4 + u_5 &= (\Delta x)^2 \end{aligned} \quad ,$$

The second boundary condition is just $u_5 = 2$. To incorporate the first boundary condition, at $x = 0$ we approximate the first derivative by $u'(x_i) \approx (u_{i+1} - u_i)/\Delta x$ (the *forward difference* scheme), which leads to

$$u_1 - u_0 = 0.5\ \Delta x \quad ==> \quad u_0 = u_1 - 0.5\ \Delta x \ .$$

Then, the discrete system becomes

$$\begin{aligned} u_1 - 0.5\ \Delta x - 2\,u_1 + u_2 &= (\Delta x)^2 \\ u_1 - 2\,u_2 + u_3 &= (\Delta x)^2 \\ u_2 - 2\,u_3 + u_4 &= (\Delta x)^2 \\ u_3 - 2\,u_4 + 2 &= (\Delta x)^2 \end{aligned} \quad . \qquad (6)$$

With a slight rearrangement and using $(\Delta x)^2 = 0.04$, Eq. (6) becomes

(Ex. 5 continued)

$$\begin{pmatrix} -1 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 0.14 \\ 0.04 \\ 0.04 \\ -1.96 \end{pmatrix} \ ,$$

which can be readily solved.
(**Exercise**: Finish the solution of the above matrix equation and compare the numerical solution with analytic solution.)

MATLAB Program

```
% BVP-ODE
%

clc, clear, close
```

```
A=[-1, 1, 0, 0
    1,-2, 1, 0
    0, 1, -2, 1
    0, 0, 1, -2];

b=[0.14, 0.04, 0.04, -1.96]';

U=A\b;              % U is a column vector.

% Let's collect the whole solution in a new variable uu to include the BCs:
uu(1)=1.1; uu(2:1:5)=U'; uu(6)=2.0;

% Construct a Table of Results:
x=[0:0.2:1.0];
data=[x ; uu];
fprintf(' x   u \n');
fprintf('%.2f  %.2f \n',data);

% Plot
plot(x,uu,' or')
grid on
legend('u');
xlabel('X'); ylabel('U');
title('x vs u');
```

OUTPUT:

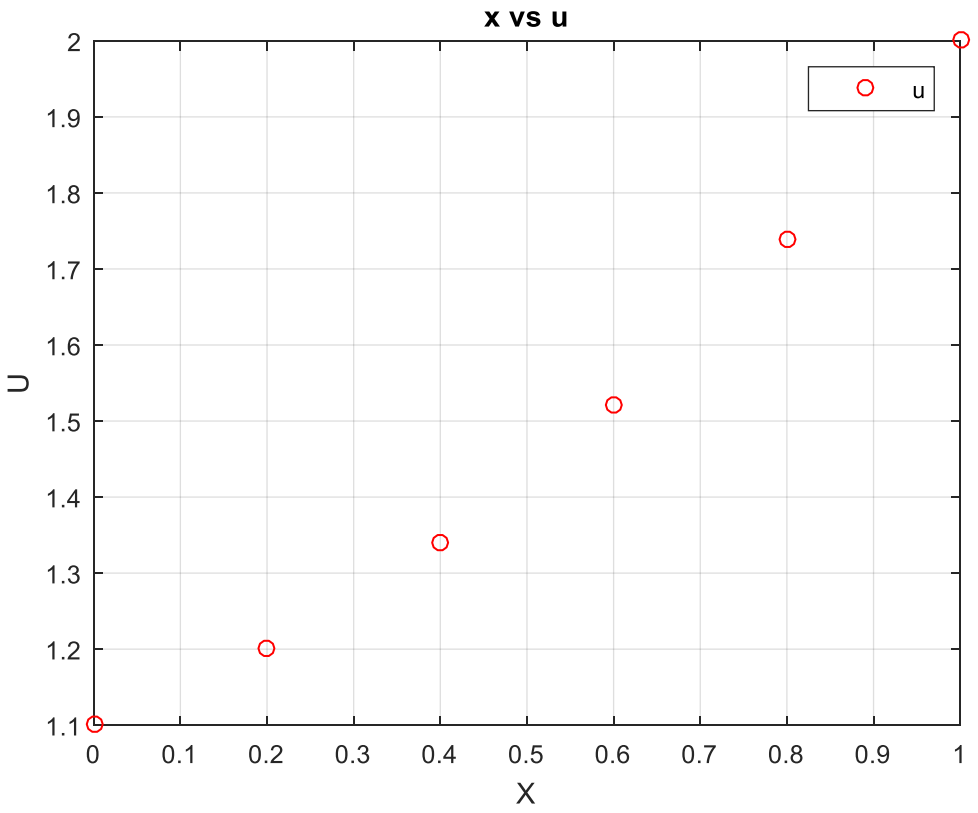
```
 x   u
0.00 1.10
0.20 1.20
0.40 1.34
0.60 1.52
0.80 1.74
1.00 2.00
```

PLEASE note that we computed u_o with:

$u_0 = u_1 - 0.5(\text{deltaX}) = 1.2 - (0.5)(0.2) = 1.1$.

THEN the full solution is:

i	0	1	2	3	4	5
x	0	0.2	0.4	0.6	0.8	1.0
u	1.1	1.2	1.34	1.52	1.74	2.0



COMPARING BOTH SOLUTIONS WITH THE EXACT SOLUTION

Where: u(EXACT)=X^2/2+X/2+1

i	x(i)	A(i)	u(i)	Exact
1	0	1.000	1.100	1.000
2	0.2	1.120	1.200	1.120
3	0.4	1.280	1.340	1.280
4	0.6	1.480	1.520	1.480
5	0.8	1.720	1.740	1.720
6	1	2.000	2.000	2.000

Column A(i) is the solution with the Shooting Method and column u(i) is the solution with FE-Tridiagonal Matrix Approach

The Shooting method is very accurate as it depends on RK-4. On the other hand, errors with the FE-Tridiaginal method may be due to using a forward difference approximation of error order h for the BC [Although the method itself is error order h^2].

Two ways to improve the results of the FD-TDM (1) use smaller h's and (2) use more accurate derivative representation for the first derivative in the BC.

IMPROVED FINITE DIFFERENCE METHOD

With the idea of improving accuracy, the derivative in the BC u'(0)=0.5 will be approximated with a forward difference formula with error order h^2. Use the magic tables to get the expression below. To see the effect of this change alone, the step size Δx will be kept constant

$$u_i' = \frac{\left(-\frac{3}{2}\right)u_i + 2u_{i+1} - \left(\frac{1}{2}\right)u_{i+2}}{\Delta x} = 0.5$$

i=0

$$\left(-\frac{3}{2}\right)u_0 + 2u_1 - \left(\frac{1}{2}\right)u_2 = 0.5\Delta x$$

which yields:

$$u_0 = \left(\frac{4}{3}\right)u_1 - \left(\frac{1}{3}\right)u_2 - \left(\frac{1}{3}\right)\Delta x$$

The first equation of the system for this problem

$$u_0 - 2u_1 + u_2 = (\Delta x)^2$$

Combine the last two equations:

$$\left(\frac{4}{3}\right)u_1 - \left(\frac{1}{3}\right)u_2 - \left(\frac{1}{3}\right)\Delta x - 2u_1 + u_2 = (\Delta x)^2$$

$$-\left(\frac{2}{3}\right)u_1 + \left(\frac{2}{3}\right)u_2 = (\Delta x)^2 + \left(\frac{1}{3}\right)\Delta x$$

The rest of the equations stay the same. The new first equation above and the "rest" and yield the following matrix:

$$\begin{pmatrix} -2/3 & 2/3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 0.1066 \\ 0.04 \\ 0.04 \\ -1.96 \end{pmatrix}$$

The solution for this system (see code below):

x u

0.00 0.97
0.20 1.12

0.40 1.28
0.60 1.48
0.80 1.72
1.00 2.00

which greatly improved the results. The code is shown below:

```
% BVP-ODE
% Finite Difference Method

clc, clear, close

A=[-2/3, 2/3, 0, 0
    1,-2, 1, 0
    0, 1, -2, 1
    0, 0, 1, -2];

b=[0.1066, 0.04, 0.04, -1.96]';

U=A\b;

h=0.2;

% Complete the solution by adding the BCs
uu(1)=(4/3)*U(1)-(1/3)*U(2)-(1/2)*h;
uu(2:1:5)=U'; uu(6)=2.0;

% Table of Results
x=[uu(1):h:uu(6)];
data=[x ; uu];
fprintf(' x   u \n');
fprintf('%.2f  %.2f \n',data);

% Plot
plot(x,uu,' or')
grid on
legend('u');
xlabel('x'); ylabel('U');
title('x vs u');
```

A comparison of all the solutions:

		Shooting	FD	FD	
		RK-4	Tridiagonal	Tridiagonal	Exact
i	x(i)	u(i)	u(i)	u(i)	Solution
1	0	1.000	1.100	0.970	1.000
2	0.2	1.120	1.200	1.120	1.120
3	0.4	1.280	1.340	1.280	1.280
4	0.6	1.480	1.520	1.480	1.480
5	0.8	1.720	1.740	1.720	1.720
6	1	2.000	2.000	2.000	2.000
			BC approx	BC approx	
			order h	order h2	

A comparative plot of all solution should follow (but doesn't as my coffee is gone)