8.1 The Euler or Tangent Line Method

Note Title 3/26/2020

1.

By columns 3 & 4, it is assumed $h = 0.01$ and $h = 0.001$.

Note $f_n(t_n, y_n) = 1 - t_n + 4y_n$

$h = 0.01$: $y_{n+1} = y_n + (0.01)(1 - t_n + 4y_n)$

$y_0 = 1$,

$t_0 = 0$: $y_1 = 1 + (0.01)[1 - 0 + 4(1)] = 1 + 0.05 = 1.05$

$t_1 = 0.01$: $y_2 = 1.05 + 0.01[1 - 0.01 + 4(1.05)] = 1.1019$

$t_2 = 0.02$: $y_3 = 1.1019 + 0.01[1 - 0.02 + 4(1.1019)] = 1.15578$

$t_{10} = 0.10$: $y_{10} = \ldots$

Table 8.1.1 only reports values for $t_{10} = 0.10$,

$t_{20} = 0.20, \ldots$ etc. (200 calculations for $h = 0.01$,

2,000 computations for $h = 0.001$).

Using MATLAB to do these computations for both

$h = 0.01$ and $h = 0.001$, showing values for only $t = 0.1, 0.2, \ldots$

```
clear;
%Change TableTimes[], StepSizes[], y(1) the initial value,
%       slope=dy/dt, solution, T.Properties.VariableNames
%times to display the y values in the table
TableTimes = [0 0.1 0.2 0.3 0.4 .5 1.0 1.5 2.0];
NumRows = length(TableTimes);
%incremental time values for computing y and y'
StepSizes = [0.01 0.001];
NumSteps = length(StepSizes);
%initialize table array: columns for h + time + exact
V = zeros(NumRows, NumSteps + 2);
V(1,1) = TableTimes(1);  %may not always specify initial value at t=0
for i = 1:NumRows  %populate exact values for solution in last Col
    t = TableTimes(i);
    V(i,NumSteps+2) = -3/16 + (1/4)*t + (19/16)*exp(4*t);
end
Col = 1;   % initialize column index to array V for below loop
for h = StepSizes  %Compute values for each time given step size h
    Col = Col +1;
    Row = 1;   %start at row 1 for each column
    t=TableTimes(1):h:TableTimes(end);
    % slope = dy/dt = f(t,y)
    slope = @(t,y) 1 - t + 4*y;
    % pre-allocate memory for y, f
    y = zeros(length(t),1);
    f = zeros(length(t),1);
    y(1) = 1;   %really y0, but MATLAB not zero based
    f(1) = slope(t(1), y(1));
    V(Row,Col) = y(1);
    for i = 2:length(t)
        y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
        f(i) = slope(t(i), y(i));
        %display table only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,1) = t(i);
            V(Row,Col) = y(i);
        end
    end
end
format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h_01', 'h_001', 'Exact'}
```

T = 9×4 table

	t	h_01	h_001	Exact
1	0	1	1	1
2	0.1000	1.5953	1.6076	1.6090
3	0.2000	2.4645	2.5011	2.5053
4	0.3000	3.7390	3.8207	3.8301
5	0.4000	5.6137	5.7755	5.7942
6	0.5000	8.3767	8.6771	8.7120
7	1.0000	60.0371	64.3826	64.8978
8	1.5000	426.4082	473.5598	479.2592
9	2.0000	3.0293e+03	3.4842e+03	3.5402e+03

2.

Entries for columns 3 & 4 are for $h = 0.01$, $h = 0.001$

There is slight modification of MATLAB code shown in #1. This is the "backward" method.

From $y_{n+1} = y_n + h\left[1 - t_{n+1} + 4y_{n+1}\right]$,

$$y_{n+1} = \frac{y_n + h(1 - t_{n+1})}{1 - 4h} \quad [1]$$

For $h = 0.01$, or $h = 0.001$, $1 - 4h \neq 0$.

Note that y_{n+1} and t_{n+1} have the same index.

So the MATLAB code uses

$y(i) = impSol(t(i), y(i-1))$ where

impSol refers to the formula in [1]

T = 9×4 table

	t	h_01	h_001	Exact
1	0	1	1	1
2	0.1000	1.6237	1.6105	1.6090
3	0.2000	2.5491	2.5096	2.5053
4	0.3000	3.9286	3.8396	3.8301
5	0.4000	5.9908	5.8131	5.7942
6	0.5000	9.0801	8.7473	8.7120
7	1.0000	70.4524	65.4200	64.8978
8	1.5000	542.1243	485.0582	479.2592
9	2.0000	4.1727e+03	3.5974e+03	3.5402e+03

code on next page

```
clear;
%Change TableTimes[], StepSizes[], y(1) the initial value,
%       slope=dy/dt, solution, T.Properties.VariableNames
%times to display the y values in the table
TableTimes = [0 0.1 0.2 0.3 0.4 .5 1.0 1.5 2.0];
NumRows = length(TableTimes);
%incremental time values for computing y and y'
StepSizes = [0.01 0.001];
NumSteps = length(StepSizes);
%initialize table array: columns for h + time + exact
V = zeros(NumRows, NumSteps + 2);
V(1,1) = TableTimes(1);  %may not always specify initial value at t=0
for i = 1:NumRows  %populate exact values for solution in last Col
    t = TableTimes(i);
    V(i,NumSteps+2) = -3/16 + (1/4)*t + (19/16)*exp(4*t);
end
Col = 1;   % initialize column index to array V for below loop
for h = StepSizes  %Compute values for each time given step size h
    Col = Col +1;
    Row = 1;   %start at row 1 for each column
    t=TableTimes(1):h:TableTimes(end);

    % from implicit solution, get y_next
    recip = 1/(1 - 4*h); % decrease number of divisions
    impSol = @(t,y) (y + h*(1 - t))*recip;
    % pre-allocate memory for y
    y = zeros(length(t),1);
    y(1) = 1;   %really y0, but MATLAB not zero based

    V(Row,Col) = y(1);
    for i = 2:length(t)
        y(i) = impSol(t(i), y(i-1));
        %display table only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,1) = t(i);
            V(Row,Col) = y(i);
        end
    end
end
format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h_01', 'h_001', 'Exact'}
```

3.

Use MATLAB, and slightly modify code in #1 and #2 above, but display just one table.

Forward formula: $y_{n+1} = y_n + h(5t_n - 3\sqrt{y_n})$

Backward formula: $y_{n+1} = y_n + h(5t_{n+1} - 3\sqrt{y_{n+1}})$

$\therefore y_{n+1} + 3h\sqrt{y_{n+1}} = y_n + 5ht_{n+1}$

Starting with $y(0)=2$, the right-hand side is always positive. $\therefore$ Let $z = y_n + 5ht_{n+1}$ and let $y_{n+1} = x^2$.

$\therefore x^2 + 3hx - z = 0$, take $x = \dfrac{-3h + \sqrt{9h^2+4z}}{2}$

Then $y_{n+1} = x^2$, noting that $\sqrt{9h^2+4z} > 3h$, so x above is always positive.

T = 5×5 table

	t	h_05	h_025	bk_05	bk_025
1	0	2.0000	2.0000	2.0000	2.0000
2	0.1000	1.5998	1.6112	1.6434	1.6330
3	0.2000	1.2929	1.3136	1.3716	1.3530
4	0.3000	1.0724	1.1001	1.1776	1.1527
5	0.4000	0.9302	0.9626	1.0533	1.0241

code on next page

```
clear;
%Change TableTimes[], StepSizes[], y(1) the initial value,
%       slope=dy/dt, T.Properties.VariableNames

% times to display the y values in the table
TableTimes = [0 0.1 0.2 0.3 0.4];
NumRows = length(TableTimes);
%incremental time values (=h) for computing y and y'
StepSizes = [0.05, 0.025, 0.05, 0.025];
NumSteps = length(StepSizes);
%initialize table array: columns for times + h values
V = zeros(NumRows, 1 + NumSteps);
V(1,1) = TableTimes(1);  %may not always specify initial value at t=0

Col = 1;   % initialize column index to array V for below loop
for h = StepSizes  %Compute values for each time given step size h
    Col = Col +1;
    Row = 1;   %start at row 1 for each column
    t=TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y
    y = zeros(length(t),1);
    y(1) = 2;   % really y0, but MATLAB not zero based

    if Col < 4  % columns 2, 3 for standard Euler method
        % slope = dy/dt = f(t,y)
        slope = @(t,y) 5*t - 3*sqrt(y);
        % pre-allocate memory for f
        f = zeros(length(t),1);
        f(1) = slope(t(1), y(1));
        V(Row,Col) = y(1);
        for i = 2:length(t)
            y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
            f(i) = slope(t(i), y(i));
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    else        % columns 4, 5 for backward method
        V(Row,Col) = y(1);
        for i = 2:length(t)
            z = y(i-1) + 5*h*t(i);
            x = (-3*h + sqrt(9*h^2 + 4*z))/2;
            y(i) = x^2;
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    end
end
format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h_05', 'h_025', 'bk_05', 'bk_025'}
```

Note that for both methods, $t_n = t_0 + nh$

and $t_0 = 0$. $\therefore t_n = nh$, $n = 0, 1, 2, 3, \ldots$

$\therefore$ Forward method could be written as

$$y_{n+1} = y_n + h(5nh - 3\sqrt{y_n})$$

A simple MATLAB code is:

```
clear
h = 0.05;
yOld = 2;  % y0
LastTime = 0.4;
nStop = LastTime/h;
for n = 0:nStop
    tOld = n*h;
    yNew = yOld + h*(5*tOld - 3*sqrt(yOld));
    % only display at t = 0.1, 0.2, 0.3, 0.4
    if mod(n+1,2) == 0
        tNew = (n+1)*h;
        disp([tNew, yNew])
    end
    yOld = yNew;
end
```

```
    0.1000    1.5998

    0.2000    1.2929

    0.3000    1.0724

    0.4000    0.9302
```

This gives the answer for just $h = 0.05$

The code is slightly altered for each h value.

The more extensive code displays everything in one simple table.

4.

Forward formula: $y_{n+1} = y_n + h(2y_n - 3t_n)$

Backward formula: $y_{n+1} = y_n + h(2y_{n+1} - 3t_{n+1})$

$= y_n + 2hy_{n+1} - 3ht_{n+1}$

$\therefore y_{n+1} = \frac{y_n - 3ht_{n+1}}{1-2h}$

The results are:

T = 5×5 table

	t	h_05	h_025	bk_05	bk_025
1	0	1.0000	1.0000	1.0000	1.0000
2	0.1000	1.2025	1.2039	1.2086	1.2069
3	0.2000	1.4160	1.4194	1.4310	1.4268
4	0.3000	1.6429	1.6490	1.6704	1.6627
5	0.4000	1.8859	1.8957	1.9308	1.9180

MATLAB code is on the next page. The new modification from #3 is setting $y(0) = 1$, and setting the new forward formula for $f(t_n, y_n)$ called slope = @(t,y) 2y-3t $\therefore$ slope$(t_n, y_n) = f(t_n, y_n)$

For the backward method, impSol = @(t,y) (y-3ht)/(1-2h)

$\therefore y_{n+1} =$ impSol(t_{n+1}, y_n) in the code.

```
clear;
%Change TableTimes[], StepSizes[], y(1) the initial value,
%       slope=dy/dt, T.Properties.VariableNames

% times to display the y values in the table
TableTimes = [0 0.1 0.2 0.3 0.4];
NumRows = length(TableTimes);
%incremental time values (=h) for computing y and y'
StepSizes = [0.05, 0.025, 0.05, 0.025];
NumSteps = length(StepSizes);
%initialize table array: columns for times + h values
V = zeros(NumRows, 1 + NumSteps);
V(1,1) = TableTimes(1);  %may not always specify initial value at t=0

Col = 1;   % initialize column index to array V for below loop
for h = StepSizes  %Compute values for each time given step size h
    Col = Col +1;
    Row = 1;   %start at row 1 for each column
    t=TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y
    y = zeros(length(t),1);
    y(1) = 1;   % really y0, but MATLAB not zero based

    if Col < 4  % columns 2, 3 for standard Euler method
        % slope = dy/dt = f(t,y)
        slope = @(t,y) 2*y - 3*t;
        % pre-allocate memory for f
        f = zeros(length(t),1);
        f(1) = slope(t(1), y(1));
        V(Row,Col) = y(1);
        for i = 2:length(t)
            y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
            f(i) = slope(t(i), y(i));
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    else        % columns 4, 5 for backward method
        V(Row,Col) = y(1);
        % from implicit solution, get y_next
        recip = 1/(1 - 2*h); % decrease number of divisions
        impSol = @(t,y) (y - 3*h*t)*recip;
        for i = 2:length(t)
            y(i) = impSol(t(i), y(i-1));
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    end
end
format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h_05', 'h_025', 'bk_05', 'bk_025'}
```

5.

Forward formula: $y_{n+1} = y_n + h(2t_n + e^{-t_n y_n})$

Backward formula: $y_{n+1} = y_n + h(2t_{n+1} + e^{-t_{n+1} y_{n+1}})$

There is no simple explicit formula for y_{n+1}.

∴ For each value of y_n, h, t_{n+1}, use MATLAB's vpasolve method to get y_{n+1}.

The results are:

T = 5×5 table

	t	h_05	h_025	bk_05	bk_025
1	0	1.0000	1.0000	1.0000	1.0000
2	0.1000	1.1024	1.1036	1.1072	1.1060
3	0.2000	1.2143	1.2166	1.2233	1.2211
4	0.3000	1.3348	1.3382	1.3480	1.3447
5	0.4000	1.4640	1.4683	1.4811	1.4769

The code is on the next page.

```
clear;
%Change TableTimes[], StepSizes[], y(1) the initial value,
%       slope=dy/dt, T.Properties.VariableNames

% times to display the y values in the table
TableTimes = [0 0.1 0.2 0.3 0.4];
NumRows = length(TableTimes);
%incremental time values (=h) for computing y and y'
StepSizes = [0.05, 0.025, 0.05, 0.025];
NumSteps = length(StepSizes);
%initialize table array: columns for times + h values
V = zeros(NumRows, 1 + NumSteps);
V(1,1) = TableTimes(1);  %may not always specify initial value at t=0

Col = 1;   % initialize column index to array V for below loop
for h = StepSizes  %Compute values for each time given step size h
    Col = Col +1;
    Row = 1;   %start at row 1 for each column
    t=TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y
    y = zeros(length(t),1);
    y(1) = 1;   % really y0, but MATLAB not zero based

    if Col < 4  % columns 2, 3 for standard Euler method
        % slope = dy/dt = f(t,y)
        slope = @(t,y) 2*t + exp(-t*y);
        % pre-allocate memory for f
        f = zeros(length(t),1);
        f(1) = slope(t(1), y(1));
        V(Row,Col) = y(1);
        for i = 2:length(t)
            y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
            f(i) = slope(t(i), y(i));  % store latest value
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    else        % columns 4, 5 for backward method
        V(Row,Col) = y(1);
        % get y_next using vpasolve
        syms x
        for i = 2:length(t)
            eqn = x == y(i-1) + h*(2*t(i) + exp(-t(i)*x));
            y(i) = vpasolve(eqn,x);
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    end
end
format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h_05', 'h_025', 'bk_05', 'bk_025'}
```

6.

Forward formula: $y_{n+1} = y_n + h\left[\frac{y_n^2 + 2t_n y_n}{3 + t_n^2}\right]$

Backward formula: $y_{n+1} = y_n + h\left[\frac{y_{n+1}^2 + 2t_{n+1} y_{n+1}}{3 + t_{n+1}^2}\right]$

$\therefore \left[\frac{h}{3 + t_{n+1}^2}\right] y_{n+1}^2 + \left[\frac{2h t_{n+1}}{3 + t_{n+1}^2} - 1\right] y_{n+1} + y_n = 0$

Get y_{n+1} by solving the quadratic equation.

Choose the root closest to y_n, since with small h, y_n and y_{n+1} should be fairly close in value.

The results are:

T = 5×5 table

	t	h_05	h_025	bk_05	bk_025
1	0	0.5000	0.5000	0.5000	0.5000
2	0.1000	0.5092	0.5097	0.5111	0.5106
3	0.2000	0.5222	0.5232	0.5262	0.5251
4	0.3000	0.5390	0.5406	0.5453	0.5437
5	0.4000	0.5599	0.5621	0.5688	0.5665

MATLAB code on next page.

```
clear;
%Change TableTimes[], StepSizes[], y(1) the initial value,
%       slope=dy/dt, T.Properties.VariableNames

% times to display the y values in the table
TableTimes = [0 0.1 0.2 0.3 0.4];
NumRows = length(TableTimes);
%incremental time values (=h) for computing y and y'
StepSizes = [0.05, 0.025, 0.05, 0.025];
NumSteps = length(StepSizes);
%initialize table array: columns for times + h values
V = zeros(NumRows, 1 + NumSteps);
V(1,1) = TableTimes(1);  %may not always specify initial value at t=0

Col = 1;   % initialize column index to array V for below loop
for h = StepSizes  %Compute values for each time given step size h
    Col = Col +1;
    Row = 1;   %start at row 1 for each column
    t=TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y
    y = zeros(length(t),1);
    y(1) = 0.5;   % really y0, but MATLAB not zero based

    if Col < 4  % columns 2, 3 for standard Euler method
        % slope = dy/dt = f(t,y)
        slope = @(t,y) (y^2 + 2*t*y)/(3 + t^2);
        % pre-allocate memory for f
        f = zeros(length(t),1);
        f(1) = slope(t(1), y(1));
        V(Row,Col) = y(1);
        for i = 2:length(t)
            y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
            f(i) = slope(t(i), y(i));  % store latest value
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    else        % columns 4, 5 for backward method
        V(Row,Col) = y(1);
        % get y_next using roots
        for i = 2:length(t)
            a = h/(3+t(i)^2);  % specify quadratic coefficients
            b = 2*h*t(i)/(3+t(i)^2) - 1;
            c = y(i-1);
            r = roots([a b c]);
            % choose root closest to y(i-1)
            if abs(r(1)-y(i-1)) < abs(r(2)-y(i-1))
                y(i) = r(1);
            else
                y(i) = r(2);
            end
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    end
end
format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h_05', 'h_025', 'bk_05', 'bk_025'}
```

7.

Forward formula: $y_{n+1} = y_n + h(t_n^2 - y_n^2)\sin(y_n)$

Backward formula: $y_{n+1} = y_n + h(t_{n+1}^2 - y_{n+1}^2)\sin(y_{n+1})$

Will need to use MATLAB vpasolve method to solve for y_{n+1}.

The results are:

T = 5×5 table

	t	h_05	h_025	bk_05	bk_025
1	0	-1.0000	-1.0000	-1.0000	-1.0000
2	0.1000	-0.9205	-0.9226	-0.9281	-0.9263
3	0.2000	-0.8575	-0.8609	-0.8701	-0.8672
4	0.3000	-0.8080	-0.8123	-0.8240	-0.8203
5	0.4000	-0.7700	-0.7750	-0.7887	-0.7843

MATLAB code is on the next page.

```
clear;
%Change TableTimes[], StepSizes[], y(1) the initial value,
%       slope=dy/dt, T.Properties.VariableNames

% times to display the y values in the table
TableTimes = [0 0.1 0.2 0.3 0.4];
NumRows = length(TableTimes);
%incremental time values (=h) for computing y and y'
StepSizes = [0.05, 0.025, 0.05, 0.025];
NumSteps = length(StepSizes);
%initialize table array: columns for times + h values
V = zeros(NumRows, 1 + NumSteps);
V(1,1) = TableTimes(1);  %may not always specify initial value at t=0

Col = 1;   % initialize column index to array V for below loop
for h = StepSizes  %Compute values for each time given step size h
    Col = Col +1;
    Row = 1;   %start at row 1 for each column
    t=TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y
    y = zeros(length(t),1);
    y(1) = -1;   % really y0, but MATLAB not zero based

    if Col < 4  % columns 2, 3 for standard Euler method
        % slope = dy/dt = f(t,y)
        slope = @(t,y) (t^2 - y^2)*sin(y);
        % pre-allocate memory for f
        f = zeros(length(t),1);
        f(1) = slope(t(1), y(1));
        V(Row,Col) = y(1);
        for i = 2:length(t)
            y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
            f(i) = slope(t(i), y(i));  % store latest value
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    else        % columns 4, 5 for backward method
        V(Row,Col) = y(1);
        % get y_next using solve
        syms x
        for i = 2:length(t)
            eqn = x == y(i-1) + h*(t(i)^2 - x^2)*sin(x);
            y(i) = vpasolve(eqn,x);
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    end
end
format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h_05', 'h_025', 'bk_05', 'bk_025'}
```

8.

Forward formula: $y_{n+1} = y_n + h(0.5 - t_n + 2y_n)$

Backward formula: $y_{n+1} = y_n + h(0.5 - t_{n+1} + 2y_{n+1})$

$$\therefore y_{n+1} = \frac{y_n + 0.5h - ht_{n+1}}{1 - 2h}$$

The results are:

T = 5×5 table

	t	h_025	h_0125	bk_025	bk_0125
1	0	1.0000	1.0000	1.0000	1.0000
2	0.5000	2.9033	2.9351	3.0395	3.0031
3	1.0000	7.5400	7.7096	8.2814	8.0793
4	1.5000	19.4292	20.1081	22.4562	21.6163
5	2.0000	50.5614	52.9779	61.5496	58.4462

MATLAB code on next page.

The code uses the anonymous function handle slope = @(t,y) 0.5 - t + 2y in the forward Euler method, so that $y_{n+1} = y_n + h*slope(t_n, y_n)$, and impSol = @(t,y) (y + 0.5h - ht)/(1-2h) so that $y_{n+1} = impSol(t_{n+1}, y_n)$.

```
clear;
%Change TableTimes[], StepSizes[], y(1) the initial value,
%       slope=dy/dt, T.Properties.VariableNames

% times to display the y values in the table
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
NumRows = length(TableTimes);
%incremental time values (=h) for computing y and y'
StepSizes = [0.025, 0.0125, 0.025, 0.0125];
NumSteps = length(StepSizes);
%initialize table array: columns for times + h values
V = zeros(NumRows, 1 + NumSteps);
V(1,1) = TableTimes(1);  %may not always specify initial value at t=0

Col = 1;   % initialize column index to array V for below loop
for h = StepSizes  %Compute values for each time given step size h
    Col = Col +1;
    Row = 1;   %start at row 1 for each column
    t=TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y
    y = zeros(length(t),1);
    y(1) = 1;   % really y0, but MATLAB not zero based

    if Col < 4  % columns 2, 3 for standard Euler method
        % slope = dy/dt = f(t,y)
        slope = @(t,y) 0.5 - t + 2*y;
        % pre-allocate memory for f
        f = zeros(length(t),1);
        f(1) = slope(t(1), y(1));
        V(Row,Col) = y(1);
        for i = 2:length(t)
            y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
            f(i) = slope(t(i), y(i));
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    else        % columns 4, 5 for backward method
        V(Row,Col) = y(1);
        % from implicit solution, get y_next
        recip = 1/(1 - 2*h); % decrease number of divisions
        impSol = @(t,y) (y + 0.5*h - h*t)*recip;
        for i = 2:length(t)
            y(i) = impSol(t(i), y(i-1));
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    end
end
format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h_025', 'h_0125', 'bk_025', 'bk_0125'}
```

9.

Forward formula: $y_{n+1} = y_n + h(5t_n - 3\sqrt{y_n})$

Backward formula: $y_{n+1} = y_n + h(5t_{n+1} - 3\sqrt{y_{n+1}})$

As in #3, $\therefore y_{n+1} + 3h\sqrt{y_{n+1}} = y_n + 5ht_{n+1}$

Starting with $y(0)=2$, the right-hand side is always positive. $\therefore$ Let $z = y_n + 5ht_{n+1}$ and let $y_{n+1} = x^2$.

$\therefore x^2 + 3hx - z = 0$, take $x = \dfrac{-3h + \sqrt{9h^2+4z}}{2}$

Then $y_{n+1} = x^2$, noting that $\sqrt{9h^2+4z} > 3h$, so x above is always positive.

The results are:

T = 5×5 table

	t	h_025	h_0125	bk_025	bk_0125
1	0	2.0000	2.0000	2.0000	2.0000
2	0.5000	0.8918	0.9089	0.9586	0.9423
3	1.0000	1.2523	1.2687	1.3179	1.3015
4	1.5000	2.3782	2.3934	2.4392	2.4239
5	2.0000	4.0726	4.0880	4.1347	4.1191

MATLAB code on next page.

```
clear;
%Change TableTimes[], StepSizes[], y(1) the initial value,
%       slope=dy/dt, T.Properties.VariableNames

% times to display the y values in the table
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
NumRows = length(TableTimes);
%incremental time values (=h) for computing y and y'
StepSizes = [0.025, 0.0125, 0.025, 0.0125];
NumSteps = length(StepSizes);
%initialize table array: columns for times + h values
V = zeros(NumRows, 1 + NumSteps);
V(1,1) = TableTimes(1);  %may not always specify initial value at t=0

Col = 1;   % initialize column index to array V for below loop
for h = StepSizes  %Compute values for each time given step size h
    Col = Col +1;
    Row = 1;   %start at row 1 for each column
    t=TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y
    y = zeros(length(t),1);
    y(1) = 2;   % really y0, but MATLAB not zero based

    if Col < 4  % columns 2, 3 for standard Euler method
        % slope = dy/dt = f(t,y)
        slope = @(t,y) 5*t - 3*sqrt(y);
        % pre-allocate memory for f
        f = zeros(length(t),1);
        f(1) = slope(t(1), y(1));
        V(Row,Col) = y(1);
        for i = 2:length(t)
            y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
            f(i) = slope(t(i), y(i));
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    else        % columns 4, 5 for backward method
        V(Row,Col) = y(1);
        for i = 2:length(t)
            z = y(i-1) + 5*h*t(i);
            x = (-3*h + sqrt(9*h^2 + 4*z))/2;
            y(i) = x^2;
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    end
end
format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h_025', 'h_0125', 'bk_025', 'bk_0125'}
```

10.

Forward formula: $y_{n+1} = y_n + h(2t_n + e^{-t_n y_n})$

Backward formula: $y_{n+1} = y_n + h(2t_{n+1} + e^{-t_{n+1} y_{n+1}})$

As in #5, no simple explicit formula for y_{n+1}, so use MATLAB's numerical vpasolve to solve.

The results are:

T = 5×5 table

	t	h_025	h_0125	bk_025	bk_0125
1	0	1.0000	1.0000	1.0000	1.0000
2	0.5000	1.6073	1.6100	1.6179	1.6153
3	1.0000	2.4683	2.4746	2.4936	2.4872
4	1.5000	3.7217	3.7336	3.7694	3.7574
5	2.0000	5.4596	5.4777	5.5322	5.5140

MATLAB code is on the next page.

```
clear;
%Change TableTimes[], StepSizes[], y(1) the initial value,
%       slope=dy/dt, T.Properties.VariableNames

% times to display the y values in the table
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
NumRows = length(TableTimes);
%incremental time values (=h) for computing y and y'
StepSizes = [0.025, 0.0125, 0.025, 0.0125];
NumSteps = length(StepSizes);
%initialize table array: columns for times + h values
V = zeros(NumRows, 1 + NumSteps);
V(1,1) = TableTimes(1);  %may not always specify initial value at t=0

Col = 1;   % initialize column index to array V for below loop
for h = StepSizes  %Compute values for each time given step size h
    Col = Col +1;
    Row = 1;   %start at row 1 for each column
    t=TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y
    y = zeros(length(t),1);
    y(1) = 1;   % really y0, but MATLAB not zero based

    if Col < 4  % columns 2, 3 for standard Euler method
        % slope = dy/dt = f(t,y)
        slope = @(t,y) 2*t + exp(-t*y);
        % pre-allocate memory for f
        f = zeros(length(t),1);
        f(1) = slope(t(1), y(1));
        V(Row,Col) = y(1);
        for i = 2:length(t)
            y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
            f(i) = slope(t(i), y(i));
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    else        % columns 4, 5 for backward method
        V(Row,Col) = y(1);
        % get y_next using vpasolve
        syms x
        for i = 2:length(t)
            eqn = x == y(i-1) + h*(2*t(i) + exp(-t(i)*x));
            y(i) = vpasolve(eqn,x);
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    end
end
format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h_025', 'h_0125', 'bk_025', 'bk_0125'}
```

11.

Forward formula: $y_{n+1} = y_n + h\left[\frac{4 - t_n y_n}{1 + y_n^2}\right]$

Backward formula: $y_{n+1} = y_n + h\left[\frac{4 - t_{n+1} y_{n+1}}{1 + y_{n+1}^2}\right]$

No simple explicit formula for y_{n+1}. ∴ Use MATLAB's numerical vpasolve to solve. Since the equation for y_{n+1} looks cubic, there may be 3 real solutions or 1 real and 2 complex solutions. Use the range option for vpasolve to return only real values close to y_n in the iterative process: reasonable since h is small.

T = 5×5 table

	t	h_025	h_0125	bk_025	bk_0125
1	0	-2.0000	-2.0000	-2.0000	-2.0000
2	0.5000	-1.4587	-1.4532	-1.4360	-1.4419
3	1.0000	-0.2175	-0.1808	-0.0682	-0.1057
4	1.5000	1.0572	1.0590	1.0649	1.0629
5	2.0000	1.4149	1.4124	1.4058	1.4079

MATLAB code next page.

```
clear;
%Change TableTimes[], StepSizes[], y(1) the initial value,
%       slope=dy/dt, T.Properties.VariableNames

% times to display the y values in the table
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
NumRows = length(TableTimes);
%incremental time values (=h) for computing y and y'
StepSizes = [0.025, 0.0125, 0.025, 0.0125];
NumSteps = length(StepSizes);
%initialize table array: columns for times + h values
V = zeros(NumRows, 1 + NumSteps);
V(1,1) = TableTimes(1);  %may not always specify initial value at t=0

Col = 1;   % initialize column index to array V for below loop
for h = StepSizes  %Compute values for each time given step size h
    Col = Col +1;
    Row = 1;   %start at row 1 for each column
    t=TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y
    y = zeros(length(t),1);
    y(1) = -2;   % really y0, but MATLAB not zero based

    if Col < 4  % columns 2, 3 for standard Euler method
        % slope = dy/dt = f(t,y)
        slope = @(t,y) (4 - t*y)/(1 + y^2);
        % pre-allocate memory for f
        f = zeros(length(t),1);
        f(1) = slope(t(1), y(1));
        V(Row,Col) = y(1);
        for i = 2:length(t)
            y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
            f(i) = slope(t(i), y(i));
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    else        % columns 4, 5 for backward method
        V(Row,Col) = y(1);
        % get y_next using vpasolve
        syms x
        for i = 2:length(t)
            eqn = x == y(i-1) + h*(4 - t(i)*x)/(1 + x^2);
            % guess 10*h as an interval to look for next y
            range = [y(i-1) - 5*h, y(i-1) + 5*h];
            y(i) = vpasolve(eqn,x,range);
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    end
end
format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h_025', 'h_0125', 'bk_025', 'bk_0125'}
```

12.

Forward formula: $y_{n+1} = y_n + h\left[\frac{y_n^2 + 2t_n y_n}{3 + t_n^2}\right]$

Backward formula: $y_{n+1} = y_n + h\left[\frac{y_{n+1}^2 + 2t_{n+1} y_{n+1}}{3 + t_{n+1}^2}\right]$

As in #6, $\left[\frac{h}{3 + t_{n+1}^2}\right] y_{n+1}^2 + \left[\frac{2h t_{n+1}}{3 + t_{n+1}^2} - 1\right] y_{n+1} + y_n = 0$

∴ Get y_{n+1} by solving quadratic equation, and for simplicity, use MATLAB vpasolve with specified range to get root y_{n+1} closest to y_n, reasonable since h is small.

The results are:

T = 5×5 table

	t	h_025	h_0125	bk_025	bk_0125
1	0	0.5000	0.5000	0.5000	0.5000
2	0.5000	0.5880	0.5894	0.5939	0.5924
3	1.0000	0.7916	0.7958	0.8087	0.8043
4	1.5000	1.1474	1.1569	1.1869	1.1766
5	2.0000	1.7097	1.7296	1.7929	1.7711

```
clear;
%Change TableTimes[], StepSizes[], y(1) the initial value,
%       slope=dy/dt, T.Properties.VariableNames

% times to display the y values in the table
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
NumRows = length(TableTimes);
%incremental time values (=h) for computing y and y'
StepSizes = [0.025, 0.0125, 0.025, 0.0125];
NumSteps = length(StepSizes);
%initialize table array: columns for times + h values
V = zeros(NumRows, 1 + NumSteps);
V(1,1) = TableTimes(1);  %may not always specify initial value at t=0

Col = 1;   % initialize column index to array V for below loop
for h = StepSizes  %Compute values for each time given step size h
    Col = Col +1;
    Row = 1;   %start at row 1 for each column
    t=TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y
    y = zeros(length(t),1);
    y(1) = 0.5;   % really y0, but MATLAB not zero based

    if Col < 4  % columns 2, 3 for standard Euler method
        % slope = dy/dt = f(t,y)
        slope = @(t,y) (y^2 + 2*t*y)/(3 + t^2);
        % pre-allocate memory for f
        f = zeros(length(t),1);
        f(1) = slope(t(1), y(1));
        V(Row,Col) = y(1);
        for i = 2:length(t)
            y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
            f(i) = slope(t(i), y(i));
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    else        % columns 4, 5 for backward method
        V(Row,Col) = y(1);
        % get y_next using vpasolve
        syms x
        for i = 2:length(t)
            a = h/(3+t(i)^2);  % specify quadratic coefficients
            b = 2*h*t(i)/(3+t(i)^2) - 1;
            c = y(i-1);
            eqn = a*x^2 + b*x + c == 0;
            % guess 10*h as an interval to look for next y
            range = [y(i-1) - 5*h, y(i-1) + 5*h];
            y(i) = vpasolve(eqn,x,range);
            % display table only at specified times
            if abs(t(i) - TableTimes(Row+1)) < 0.00001
                Row = Row + 1;
                V(Row,1) = t(i);
                V(Row,Col) = y(i);
            end
        end
    end
end
format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h_025', 'h_0125', 'bk_025', 'bk_0125'}
```

13.

With $y'(t) = f(t,y)$, $y''(t) = \frac{\partial f}{\partial t}(t,y) + \frac{\partial f}{\partial y}\cdot\frac{dy}{dt}$

Since $f(t,y) = 1 - t + 4y$, $y''(t) = -1 + 4y'(t) = -1 + 4f(t,y)$

$$\therefore y''(t) = -1 + 4(1 - t + 4y) = 3 - 4t + 16y$$

$$\therefore y(t_n + h) \simeq y(t_n) + y'(t_n)h + y''(t_n)\frac{h^2}{2}$$

$$\text{Or, } y_{n+1} = y_n + (1 - t_n + 4y_n)h + (3 - 4t_n + 16y_n)\frac{h^2}{2}$$

Using MATLAB, and modifying code in problem #1, using anonymous function handles for the Euler method and Taylor series method,

T = 3×4 table

	t	Euler_h_01	Taylor_h_01	Exact
1	0	1.0000	1.0000	1.0000
2	0.1000	1.5000	1.5950	1.6090
3	0.2000	2.1900	2.4636	2.5053

Code on next page.

```
clear;
%Change TableTimes[], y(1) the initial value,
%       eulerf, taylorf, exact solution,
%       T.Properties.VariableNames
% times to display the y values in the table
TableTimes = [0, 0.1, 0.2];
NumRows = length(TableTimes);
% initialize table array: 4 columns for time + Euler + Taylor + exact
V = zeros(NumRows,4);

% populate times (Col=1), exact values for solution (Col=4)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,4) = -3/16 + (1/4)*t + (19/16)*exp(4*t);
end

h = 0.1;  %Compute values for time step size h
t=TableTimes(1):h:TableTimes(end);

% eulerf = y(t) + h*f
eulerf = @(t,y) y + (1-t+4*y)*h;
% taylorf = y(t) + hy'(t) + (h^2/2)y"(t)
taylorf = @(t,y) y + (1-t+4*y)*h + (3-4*t+16*y)*h^2/2;
% pre-allocate memory for ye for Euler, yt for Taylor
ye = zeros(length(t),1);
yt = zeros(length(t),1);
Row = 1;   %start at row 1 for each column
ye(1) = 1;   %really y0, but MATLAB not zero based
yt(1) = 1;   %really y0, but MATLAB not zero based
V(Row,2) = ye(1);
V(Row,3) = yt(1);
for i = 2:length(t)
    ye(i) = eulerf(t(i-1),ye(i-1));   % new estimate of ye(t)
    yt(i) = taylorf(t(i-1),yt(i-1));  % new estimate of yt(t)
    Row = Row + 1;
    V(Row,2) = ye(i);
    V(Row,3) = yt(i);
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Euler_h_01', 'Taylor_h_01', 'Exact'}
```

Note the anonymous function handles using the @ symbol.

$$eulerf = @(t,y)\ y + (1-t+4y)h$$

$$taylorf = @(t,y)\ y + (1-t+4y)h + (3-4t+16y)\frac{h^2}{2}$$

14.

(a)

$$\phi''(t) = \frac{d}{dt}\phi'(t) = \frac{d}{dt}\left(2\phi(t) - 1\right) = 2\phi'(t) - 1$$

$$= 2\left(2\phi(t) - 1\right) = 4\phi(t) - 2$$

$$\therefore\ e_{n+1} = \frac{h^2}{2}\phi''(\bar{t}_n) = \frac{h^2}{2}\left(4\phi(\bar{t}_n) - 2\right) = h^2\left[2\phi(\bar{t}_n) - 1\right]$$

$$\therefore\ e_{n+1} = h^2\left[2\phi(\bar{t}_n) - 1\right],\ t_n \le \bar{t}_n \le t_{n+1}$$

(b)

$$|e_{n+1}| = \left|h^2\left(2\phi(\bar{t}_n) - 1\right)\right| = h^2\left|2\phi(\bar{t}_n) - 1\right|$$

$$\le h^2\left(2\left|\phi(\bar{t}_n)\right| + 1\right) \qquad |a+b| \le |a| + |b|$$

$$\le h^2\left(2M + 1\right),\ M = \max\left|\phi(t)\right|,\ 0 \le t \le 1$$

(c)

Solution: $(ye^{-2t})' = -e^{-2t}$, $ye^{-2t} = \frac{1}{2}e^{-2t} + c$,

$y = \frac{1}{2} + ce^{2t}$, $y(0) = 1 = \frac{1}{2} + c$, $c = \frac{1}{2}$

$\therefore \phi(t) = \frac{1}{2} + \frac{1}{2}e^{2t}$

$\therefore \phi'(t) = e^{2t}$, $\phi''(t) = 2e^{2t}$

$e_{n+1} = \frac{h^2}{2}\phi''(\bar{t}_n) = \underline{h^2 e^{2\bar{t}_n}}$, $t_n \le \bar{t}_n \le t_{n+1}$

(d)

$e_1 = h^2 e^{2\bar{t}_0}$, $t_0 < \bar{t}_0 < t_1 = h = 0.1$

$\therefore e_1 < (0.1)^2 e^{2(0.1)} = 0.01e^{0.2} = 0.01221$

$t_0 = 0$, $\therefore h^2 e^{2(0)} = 0.01(1) = 0.01$.

$\therefore \underline{0.01 \le e_1 \le 0.01221}$

$\phi(0.1) = \frac{1}{2} + \frac{1}{2}e^{2(0.1)} = 1.11070$

$y_1 = y_0 + hf(t_0, y_0) = 1 + 0.1[2(1) - 1] = 1.1$

$\therefore \underline{\text{Actual error}} = 1.11070 - 1.1 = \underline{0.01070}$

(e)

From (c), $e_4 = h^2 e^{2\bar{t}_3}$, $t_3 \le \bar{t}_3 \le t_4$

$t_3 = 3(h) = 3(0.1) = 0.3$ using $h = 0.1$, $t_0 = 0$

$t_4 = 0.4$

Since e^x is monotonic increasing, $e^{t_3} \le e^{\bar{t}_3} \le e^{t_4}$

$\therefore h^2 e^{2t_3} = (0.01)e^{2(0.3)} = 0.01822$

$h^2 e^{2t_4} = (0.01)e^{2(0.4)} = 0.2226$

$\therefore 0.01822 \le e_4 \le 0.2226$

15.

(a)

$$\phi''(t) = \frac{d}{dt}\phi'(t) = \frac{d}{dt}\left(\frac{1}{2} - t + 2\phi\right) = -1 + 2\phi'(t)$$

$$= -1 + 2\left(\frac{1}{2} - t + 2\phi(t)\right) = -2t + 4\phi(t)$$

$\therefore e_{n+1} = \frac{1}{2}h^2\phi''(\bar{t}_n) = \frac{1}{2}h^2\left[-2\bar{t}_n + 4\phi(\bar{t}_n)\right]$

$= h^2\left[2\phi(\bar{t}_n) - \bar{t}_n\right]$, $t_n \le \bar{t}_n \le t_{n+1}$

(b)

$$|e_{n+1}| = h^2 |2\phi(\bar{t}_n) - \bar{t}_n|$$

$$\leq 2h^2 |\phi(\bar{t}_n)| + h^2 |\bar{t}_n| \qquad |a+b| \leq |a| + |b|$$

$$\leq 2h^2 |\phi(t_n)| + h^2$$

$$\therefore |e_{n+1}| \leq h^2 (2M+1), \quad M = \max |\phi(t)| \text{ on } 0 \leq t \leq 1$$

(c)

Solution: $(ye^{-2t})' = \frac{1}{2}e^{-2t} - te^{-2t}$

$$ye^{-2t} = -\frac{1}{4}e^{-2t} + \frac{1}{2}te^{-2t} + \frac{1}{4}e^{-2t} + C$$

$$y = \frac{1}{2}t + Ce^{2t}, \quad y(0) = 1 \Rightarrow C = 1$$

$$\therefore \phi(t) = \frac{1}{2}t + e^{2t}$$

$$\therefore \phi'(t) = \frac{1}{2} + 2e^{2t}, \quad \phi''(t) = 4e^{2t}$$

$$\therefore e_{n+1} = \frac{1}{2}h^2\phi''(\bar{t}_n) = \frac{1}{2}h^2(4e^{2\bar{t}_n})$$

$$\therefore e_{n+1} = 2h^2 e^{2\bar{t}_n}, \quad t_n \leq \bar{t}_n \leq t_{n+1}$$

(d)

$$e_1 = 2(0.1)^2 e^{2\bar{t}_0}, \quad 0 \le \bar{t}_0 \le 0.1 \Rightarrow 1 \le e^{2\bar{t}_0} \le e^{0.2}$$

$$\therefore 0.02 \le 0.02e^{2\bar{t}_0} \le 0.02e^{0.2} = 0.02443$$

$$\therefore 0.02 \le e_1 \le 0.02443$$

Actual error: $\phi(0.1) = \frac{1}{2}(0.1) + e^{2(0.1)} = 1.27140$

$$y_1 = y_0 + h f(t_0, y_0) = y_0 + h\left(\tfrac{1}{2} \cdot t_0 + 2y_0\right)$$

$$= 1 + 0.1\left(\tfrac{1}{2} \cdot 0 + 2(1)\right) = 1.25$$

$$\therefore 1.27140 - 1.25 = 0.0214$$

(e)

From (c), $e_{n+1} = 2h^2 e^{2\bar{t}_n}$

$$\therefore e_4 = 2h^2 e^{2\bar{t}_3}, \quad t_3 \le \bar{t}_3 \le t_4$$

Using $t_0 = 0$, $h = 0.1$, $t_3 = 0.3$, $t_4 = 0.4$

$$\therefore 0.3 \le \bar{t}_3 \le 0.4, \quad 0.6 \le 2\bar{t}_3 \le 0.8$$

$$\therefore e^{0.6} \le e^{2\bar{t}_3} \le e^{0.8}, \quad 2h^2 e^{0.6} \le 2h^2 e^{2\bar{t}_3} \le 2h^2 e^{0.8}$$

$$\therefore \; 0.02(e^{0.6}) \le e_4 \le 0.02e^{0.8}$$

$$0.03644 \le e_4 \le 0.04451$$

16.

$$\phi'(t) = 5t - 3\sqrt{\phi(t)},$$

$$\phi''(t) = 5 - \tfrac{3}{2}(\phi(t))^{-\frac{1}{2}}\phi'(t) = 5 - \frac{3}{2}\frac{(5t - 3\sqrt{\phi(t)})}{\sqrt{\phi(t)}}$$

$$= \frac{-15t + 19\sqrt{\phi(t)}}{2\sqrt{\phi(t)}} = -\frac{15}{2}t\,\phi(t)^{-\frac{1}{2}} + \frac{19}{2}$$

$$e_{n+1} = \tfrac{1}{2}h^2\phi''(\bar{t}_n) = \frac{h^2}{4}\left[19 - 15\bar{t}_n\,\phi(\bar{t}_n)^{-\frac{1}{2}}\right]$$

17.

$$\phi'(t) = (t + \phi(t))^{\frac{1}{2}}$$

$$\phi''(t) = \tfrac{1}{2}(t + \phi(t))^{-\frac{1}{2}}\left[1 + \phi'(t)\right]$$

$$= \frac{1 + \sqrt{t + \phi(t)}}{2\sqrt{t + \phi(t)}} = \frac{1}{2} + \frac{1}{2}(t + \phi(t))^{-\frac{1}{2}}$$

$$e_{n+1} = \frac{1}{2} h^2 \phi''(\bar{t}_n) = \frac{h^2}{4}\left(1 + \left[\bar{t}_n + \phi(\bar{t}_n)\right]^{-\frac{1}{2}}\right)$$

18.

$$\phi'(t) = 2t + e^{-t\phi(t)}$$

$$\phi''(t) = 2 + e^{-t\phi(t)}\left[-t\phi'(t) - \phi(t)\right]$$

$$= 2 + e^{-t\phi(t)}\left[-2t^2 - te^{-t\phi(t)} - \phi(t)\right]$$

$$= 2 - 2t^2 e^{-t\phi(t)} - te^{-2t\phi(t)} - \phi(t)e^{-t\phi(t)}$$

$$= 2 - \left(2t^2 + \phi(t)\right)e^{-t\phi(t)} - te^{-2t\phi(t)}$$

$$e_{n+1} = \frac{1}{2} h^2 \phi''(\bar{t}_n)$$

$$= \frac{h^2}{2}\left[2 - \left(2\bar{t}_n^{\,2} + \phi(\bar{t}_n)\right)e^{-\bar{t}_n\phi(\bar{t}_n)} - \bar{t}_n e^{-2\bar{t}_n\phi(\bar{t}_n)}\right]$$

19.

(a)

Using MATLAB,

code next page

T = 4×2 table

	t	Euler_h_02
1	0	1.0000
2	0.2000	1.2000
3	0.4000	1.0000
4	0.6000	1.2000

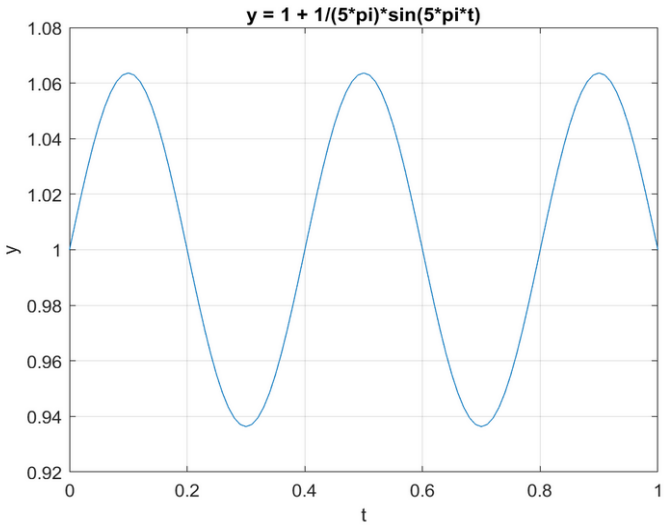
```
clear;
%Change TableTimes[], y(1) the initial value,
%       eulerf, taylorf, exact solution,
%       T.Properties.VariableNames
% times to display the y values in the table
TableTimes = [0, 0.2, 0.4, 0.6];
NumRows = length(TableTimes);
% initialize table array: 2 columns for time + Euler
V = zeros(NumRows,2);

h = 0.2;  %Compute values for time step size h
t=TableTimes(1):h:TableTimes(end);

% eulerf = y(t) + h*f
eulerf = @(t,y) y + h*cos(5*pi*t);
% pre-allocate memory for y used for computations
y = zeros(length(t),1);

Row = 1;    %start at row 1 for each column
y(1) = 1;   %really y0, but MATLAB not zero based
V(Row,1) = t(1);
V(Row,2) = y(1);
for i = 2:length(t)
    y(i) = eulerf(t(i-1),y(i-1));   % new estimate of ye(t)
    Row = Row + 1;
    V(Row,1) = t(i);
    V(Row,2) = y(i);
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Euler_h_02'}
```

(6)

$$\phi(t) = 1 + \frac{1}{5\pi}\sin(5\pi t)$$

Using MATLAB

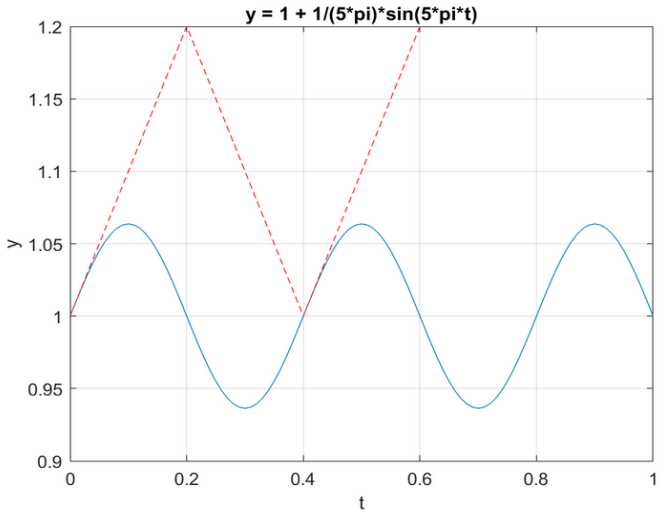
```
clear
t = 0:0.01:1;
y = 1 + (1/(5*pi))*sin(5*pi*t);
plot(t,y)
grid on
xlabel 't', ylabel 'y'
title 'y = 1 + 1/(5*pi)*sin(5*pi*t)'
```



(c)

Using MATLAB

```
clear
t = 0:0.01:1;
y = 1 + (1/(5*pi))*sin(5*pi*t);
plot(t,y)
grid on
xlabel 't', ylabel 'y'
title 'y = 1 + 1/(5*pi)*sin(5*pi*t)'

t = [0, 0.2, 0.4, 0.6];
% values from (a)
y = [1.0, 1.2, 1.0, 1.2];
hold on
plot(t,y, '--r')
```



(d)

Using MATLAB,

```
clear;
%Change TableTimes[], y(1) the initial value,
%       eulerf, taylorf, exact solution,
%       T.Properties.VariableNames
% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4];
NumRows = length(TableTimes);
% initialize table array: 2 columns for time + Euler
V = zeros(NumRows,2);

h = 0.1;  %Compute values for time step size h
t=TableTimes(1):h:TableTimes(end);

% eulerf = y(t) + h*f
eulerf = @(t,y) y + h*cos(5*pi*t);
% pre-allocate memory for y used for computations
y = zeros(length(t),1);

Row = 1;    %start at row 1 for each column
y(1) = 1;   %really y0, but MATLAB not zero based
V(Row,1) = t(1);
V(Row,2) = y(1);
for i = 2:length(t)
    y(i) = eulerf(t(i-1),y(i-1));   % new estimate of
ye(t)
    Row = Row + 1;
    V(Row,1) = t(i);
    V(Row,2) = y(i);
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Euler_h_01'}
```

T = 5×2 table

	t	Euler_h_01
1	0	1.0000
2	0.1000	1.1000
3	0.2000	1.1000
4	0.3000	1.0000
5	0.4000	1.0000

(e)

$$\phi'(t) = \cos(5\pi t) \qquad \phi''(t) = -5\pi \sin(5\pi t)$$

$$e_{n+1} = \frac{h^2}{2}\phi''(\bar{t}_n) = \frac{h^2}{2}(-5\pi \sin(5\pi \bar{t}_n))$$

$$\therefore |e_n| = \frac{5\pi h^2}{2}|\sin(5\pi \bar{t}_{n-1})| \leq \frac{5\pi h^2}{2}$$

For $h = 0.2$, $|e_n| < \frac{5\pi(0.2)^2}{2} = 0.314$

$h = 0.1$, $|e_n| < \frac{5\pi(0.1)^2}{2} = 0.079$

(f)

$$|e_{n+1}| = \frac{1}{2}h^2\phi''(\bar{t}_n), \quad \phi''(t) = -5\pi \sin(5\pi t)$$

On $[0,1]$, $\max|\sin(5\pi t)| = 1$

$$\therefore |e_{n+1}| \leq \frac{1}{2}h^2(5\pi)$$

$\therefore$ if $\frac{5\pi h^2}{2} < 0.05$, then $|e_{n+1}| < 0.05$

$$\therefore \frac{5\pi h^2}{2} < 0.05 \Rightarrow h^2 < \frac{0.1}{5\pi} = \frac{1}{50\pi}$$

$$\therefore \underline{h < \frac{1}{\sqrt{50\pi}} = 0.07979, \text{ or } h < 0.08}$$

20.

(a)

Equation (20) is: $\phi(t_{n+1}) = \phi(t_n) + hf(t_n, \phi(t_n)) + \frac{1}{2}\phi''(\bar{t}_n)h^2.$ (20)

By definition, $E_{n+1} = \phi(t_{n+1}) - y_{n+1}$ [1]

and $E_n = \phi(t_n) - y_n$ [2]

and by the Euler method $y_{n+1} = y_n + hf(t_n, y_n)$

$$\therefore E_{n+1} = \left[\phi(t_n) + hf(t_n, \phi(t_n)) + \tfrac{1}{2}h^2\phi''(\bar{t}_n)\right] - \left[y_n + hf(t_n, y_n)\right]$$

using [1], (20)

$$= \phi(t_n) - y_n + h\left[f(t_n, \phi(t_n)) - f(t_n, y_n)\right] + \frac{1}{2}h^2\phi''(\bar{t}_n)$$

$$= E_n + h\left[f(t_n, \phi(t_n)) - f(t_n, y_n)\right] + \frac{1}{2}h^2\phi''(\bar{t}_n)$$

using [2]

$$\therefore |E_{n+1}| = \left|E_n + h\left[f(t_n, \phi(t_n)) - f(t_n, y_n)\right] + \frac{1}{2}h^2\phi''(\bar{t}_n)\right|$$

$$\therefore |E_{n+1}| \le |E_n| + h\left|f(t_n, \phi(t_n)) - f(t_n, y_n)\right| + \frac{1}{2}h^2\left|\phi''(\bar{t}_n)\right| \quad [3]$$

using $|a+b+c| \le |a|+|b|+|c|$ and $h > 0$

There exists a constant, L, s.t.

$$\left|f(t_n, \phi(t_n)) - f(t_n, y_n)\right| \le L\left|\phi(t_n) - y_n\right| = L|E_n|$$

$\therefore$ From [3],

$$|E_{n+1}| \le |E_n| + hL\left|\phi(t_n) - y_n\right| + \frac{1}{2}h^2\left|\phi''(\bar{t}_n)\right|$$

$$= |E_n| + hL|E_n| + \frac{1}{2}h^2\left|\phi''(\bar{t}_n)\right|$$

$$= (1+hL)|E_n| + \frac{1}{2}h^2\left|\phi''(\bar{t}_n)\right|$$

$$\therefore |E_{n+1}| \le \alpha|E_n| + \frac{1}{2}h^2\left|\phi''(\bar{t}_n)\right| \qquad \alpha = (1+hL)$$

Let $\beta = \max \frac{1}{2}\left|\phi''(t)\right|$ on $t_0 \le t \le t_{n+1}$

$$\therefore \left|\phi''(\bar{t}_n)\right| \le \beta \quad \text{since} \quad t_0 \le t_n \le \bar{t}_n \le t_{n+1} = t_n + h$$

$$\therefore \underline{|E_{n+1}| \le \alpha|E_n| + h^2\beta}$$

(5)

From (31), $|E_1| \le \alpha |E_0| + h^2\beta = h^2\beta$

$\therefore |E_2| \le \alpha |E_1| + h^2\beta = \alpha(h^2\beta) + h^2\beta$

$|E_3| \le \alpha |E_2| + h^2\beta = \alpha^2(h^2\beta) + \alpha(h^2\beta) + (h^2\beta)$

$\therefore$ By induction,

$$|E_n| \le \alpha^{n-1}(h^2\beta) + \alpha^{n-2}(h^2\beta) + \ldots + \alpha(h^2\beta) + (h^2\beta)$$

$$= (\alpha^{n-1} + \alpha^{n-2} + \ldots + \alpha + 1)(h^2\beta)$$

But $\alpha^n - 1 = (\alpha - 1)(\alpha^{n-1} + \alpha^{n-2} + \ldots + \alpha + 1)$

$\therefore$ For $\alpha \neq 1$, $\dfrac{\alpha^n - 1}{\alpha - 1} = \alpha^{n-1} + \alpha^{n-2} + \ldots + \alpha + 1$

$$\therefore |E_n| \le \frac{\alpha^n - 1}{\alpha - 1}(h^2\beta)$$

Now using $\alpha = 1 + hL$, $|E_n| \le \dfrac{(1+hL)^n - 1}{(1+hL) - 1}\beta h^2$

Or, $|E_n| \le \dfrac{(1+hL)^n - 1}{L}\beta h$

(c)

Taylor expansion of $e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$

$\therefore$ For $x \geq 0$, $1 + x \leq e^x$

$\therefore$ Since $hL \geq 0$, $0 < 1 + hL \leq e^{hL}$

$\therefore$ For $n > 0$, $(1+hL)^n \leq e^{nhL}$

$\therefore (1+hL)^n - 1 \leq e^{nhL} - 1 \Rightarrow \frac{(1+hL)^n - 1}{L} \beta h \leq \frac{e^{nhL} - 1}{L} \beta h$

since $L > 0$, $\beta h > 0$

21.

Let $y = \phi(t)$ be the solution to $y'(t) = f(t, y)$,

$y(t_0) = y_0$. $\therefore \phi'(t) = f(t, \phi(t))$

Using a Taylor expansion for $\phi(t)$ about t_{n+1},

$$\phi(t) = \phi(t_{n+1}) + (t - t_{n+1})\phi'(t_{n+1}) + \frac{(t-t_{n+1})^2}{2}\phi''(\bar{t}_n)$$

where $\bar{t}_n$ is some number between t and t_{n+1}, and $\phi''(t)$ is continuous on some interval I about t_{n+1}, and $t \in I$. Let t_n be such that $t_n < t_{n+1}$, $t_n \in I$. Let $h = t_{n+1} - t_n > 0$, so $-h = t_n - t_{n+1}$

$$\therefore \phi(t_n) = \phi(t_{n+1}) - h\,\phi'(t_{n+1}) + \frac{h^2}{2}\phi''(\bar{t}_n)$$

where $t_n \le \bar{t}_n \le t_{n+1}$

$$\therefore \phi(t_{n+1}) = \phi(t_n) + hf(t_{n+1}, \phi(t_{n+1})) - \frac{h^2}{2}\phi''(\bar{t}_n) \quad [1]$$

Using the backward formula for the Euler method,

$$y_{n+1} = y_n + (t_{n+1} - t_n) f(t_{n+1}, y_{n+1})$$

Assume we know the correct value for y_n at t_n: $\phi(t_n)$

$$\therefore y_{n+1} = \phi(t_n) + h f(t_{n+1}, y_{n+1}) \quad [2]$$

Subtracting [1] − [2],

$$e_{n+1} = \phi(t_{n+1}) - y_{n+1} = h\left[f(t_{n+1}, \phi(t_{n+1})) - f(t_{n+1}, y_{n+1})\right] - \frac{h^2}{2}\phi''(\bar{t}_n)$$

To simplify, using the Mean Value Theorem,

$$f(t_{n+1}, \phi(t_{n+1})) - f(t_{n+1}, y_{n+1}) = [\phi(t_{n+1}) - y_{n+1}] f_y(t_{n+1}, y^*_{n+1})$$

$= e_{n+1}$

where y^*_{n+1} is between $\phi(t_{n+1})$ and y_{n+1}

$$\therefore e_{n+1} = (h)(e_{n+1}) f_y(t_{n+1}, y^*_{n+1}) - \frac{h^2}{2}\phi''(\bar{t}_n)$$

$$\text{Or, } e_{n+1} = \frac{-\frac{h^2}{2}\phi''(\bar{t}_n)}{1 - h f_y(t_{n+1}, y^*_{n+1})}$$

Using the geometric series expansion, $1 + r + r^2 \ldots = \frac{1}{1-r}$,

choosing h s.t. $h f_y(t_{n+1}, y^*_{n+1}) < 1$

$$e_{n+1} = -\frac{h^2}{2}\phi''(\bar{t}_n)\left[1 + h f_y + (h f_y)^2 + \ldots\right]$$

$$= -\frac{\phi''(\bar{t}_n)}{2}\left[h^2 + h^3 f_y + h^4 f_y^2 + \ldots\right]$$

Dropping $h^3 f_y$ and higher terms, since h is small

$$\boxed{e_{n+1} \simeq -\frac{h^2}{2}\phi''(\bar{t}_n)}$$

22.

(a)

Using MATLAB

T = 5×4 table

	t	digits_3	Euler_h_05	Exact
1	0	1.0000	1.0000	1.0000
2	0.1000	1.5500	1.5475	1.6090
3	0.2000	2.3400	2.3249	2.5053
4	0.3000	3.4600	3.4334	3.8301
5	0.4000	5.0700	5.0185	5.7942

Code is on next page. The code uses the anonymous function slope = @(t,y) 1-t+4y

Computations for 3 digit accuracy are:

$y_1 = y_0 + 0.05(1 - t_0 - 4y_0) = 1 + 0.05(1 - 0 + 4(1)) = 1.25$

$y_2 = 1.25 + 0.05(1 - 0.05 + 4(1.25)) = 1.55$

$y_3 = 1.55 + 0.05(1 - 0.1 + 4(1.55)) = 1.91$

$y_4 = 1.91 + 0.05(1 - 0.15 + 4(1.91)) = 2.34$

$y_5 = 2.34 + 0.05(1 - 0.2 + 4(2.34) = 2.85$

$y_6 = 2.85 + 0.05(1 - 0.25 + 4(2.85) = 3.46$

$y_7 = 3.46 + 0.05(1 - 0.3 + 4(3.46)) = 4.19$

$y_8 = 4.19 + 0.05(1 - 0.35 + 4(4.19)) = 5.07$

The code also uses a function called d3, which converts a number to 3-digit precision, as MATLAB's num2str(x,3) function sometimes rounds up, sometimes rounds down.

```
clear;
%Change TableTimes[], y(1) the initial value,
%       slope=dy/dt, solution, T.Properties.VariableNames
%times to display the y values in the table
TableTimes = [0 0.1 0.2 0.3 0.4];
NumRows = length(TableTimes);
% pre-allocate memory for 4 columns in table
V = zeros(NumRows, 4);

% populate table times (Col=1), exact values for solution (Col=4)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,4) = -3/16 + (1/4)*t + (19/16)*exp(4*t);
end

h = 0.05;
t = TableTimes(1):h:TableTimes(end);
% pre-allocate memory for y, f
y = zeros(length(t),1);
f = zeros(length(t),1);
y(1) = 1;   %really y0, but MATLAB not zero based

Col = 2;  % 3 digit calculations (2 decimal rounding)
slope = @(t,y) d3(1 - t + d3(4*y));  % slope = dy/dt = f(t,y)
Row = 1;  % start at row 1 for each column
f(1) = slope(t(1), y(1));  % initialize
V(Row,Col) = y(1);
for i = 2:length(t)
    y(i) = d3(y(i-1) + d3(f(i-1)*h));  %new estimate of y(t)
    f(i) = slope(t(i), y(i));
    %display table only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

Col = 3;   % standard Euler method
slope = @(t,y) 1 - t + 4*y;  % slope = dy/dt = f(t,y)
Row = 1;   % start at row 1 for each column
f(1) = slope(t(1), y(1));  % initialize
V(Row,Col) = y(1);
for i = 2:length(t)
    y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
    f(i) = slope(t(i), y(i));
    %display table only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'digits_3', 'Euler_h_05', 'Exact'}

function n3digit = d3(x)
    % num2str(x,3) has unpredictable rounding.  This
    %   code produces 3-digit precision predictably by
    %   adding 5 to the 4th sig digit, then truncating
    %   the 4th sig digit
    xstr = sprintf('%.3e',x);  % example: '4.225e-06'
    sn = sign(x);  % need to know if '-' in string
    l = length(xstr);
    ex = xstr(l-3:l);  % get exponent component, e.g. 'e-06'
    y = strcat('0.005', ex);  % create number to add to
                              % 4th sig digit, '0.005e-06'
    z = x + sn*str2double(y); % number rounded away from 0
    zstr = sprintf('%.3e',z); % put in exp form
    l = length(zstr);
    zstr = strcat(zstr(l-8:l-5), ex); % get rid of 4th sig digit
    n3digit = str2double(zstr);  % 3 digit precision
end
```

(b)

The solution: $\frac{d}{dt} ye^t = 3e^t + te^t$

$ye^t = 3e^t + te^t - e^t + C = 2e^t + te^t + C$

$y = 2 + t + ce^{-t}$, $y(0) = 1 \Rightarrow c = -1$

$\therefore y(t) = 2 + t - e^{-t}$

Using MATLAB, modifying code in (a).

T = 5×4 table

	t	digits_3	Euler_h_05	Exact
1	0	1.0000	1.0000	1.0000
2	0.1000	1.2000	1.1975	1.1952
3	0.2000	1.3900	1.3855	1.3813
4	0.3000	1.5700	1.5649	1.5592
5	0.4000	1.7400	1.7366	1.7297

Code on next page

```
clear;
%Change TableTimes[], y(1) the initial value,
%       slope=dy/dt, solution, T.Properties.VariableNames
%times to display the y values in the table
TableTimes = [0 0.1 0.2 0.3 0.4];
NumRows = length(TableTimes);
% pre-allocate memory for 4 columns in table
V = zeros(NumRows, 4);

% populate table times (Col=1), exact values for solution (Col=4)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,4) = 2 + t - exp(-t);
end

h = 0.05;
t = TableTimes(1):h:TableTimes(end);
% pre-allocate memory for y, f
y = zeros(length(t),1);
f = zeros(length(t),1);
y(1) = 1;   %really y0, but MATLAB not zero based

Col = 2;  % 3 digit calculations (2 decimal rounding)
slope = @(t,y) d3(3 + t - y);  % slope = dy/dt = f(t,y)
Row = 1;  % start at row 1 for each column
f(1) = slope(t(1), y(1));  % initialize
V(Row,Col) = y(1);
for i = 2:length(t)
    y(i) = d3(y(i-1) + d3(f(i-1)*h));  %new estimate of y(t)
    f(i) = slope(t(i), y(i));
    %display table only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

Col = 3;   % standard Euler method
slope = @(t,y) 3 + t - y;  % slope = dy/dt = f(t,y)
Row = 1;   % start at row 1 for each column
f(1) = slope(t(1), y(1));  % initialize
V(Row,Col) = y(1);
for i = 2:length(t)
    y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
    f(i) = slope(t(i), y(i));
    %display table only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'digits_3', 'Euler_h_05', 'Exact'}

function n3digit = d3(x)
    % num2str(x,3) has unpredictable rounding.  This
    %   code produces 3-digit precision predictably by
    %   adding 5 to the 4th sig digit, then truncating
    %   the 4th sig digit
    xstr = sprintf('%.3e',x);  % example: '4.225e-06'
    sn = sign(x);  % need to know if '-' in string
    l = length(xstr);
    ex = xstr(l-3:l);  % get exponent component, e.g. 'e-06'
    y = strcat('0.005', ex);  % create number to add to
                              % 4th sig digit, '0.005e-06'
    z = x + sn*str2double(y); % number rounded away from 0
    zstr = sprintf('%.3e',z); % put in exp form
    l = length(zstr);
    zstr = strcat(zstr(l-8:l-5), ex); % get rid of 4th sig digit
    n3digit = str2double(zstr);  % 3 digit precision
end
```

(c)

The solution: $\frac{d}{dt}(ye^{-2t}) = -3te^{-2t}$

$$ye^{-2t} = \frac{3}{2}te^{-2t} + \frac{3}{4}e^{-2t} + c$$

$$\therefore y = \frac{3}{2}t + \frac{3}{4} + ce^{2t}, \quad y(0) = 1 \Rightarrow c = \frac{1}{4}$$

$$\therefore y(t) = \frac{3}{4} + \frac{3}{2}t + \frac{1}{4}e^{2t}$$

Using MATLAB, modifying code in (a)

T = 5×4 table

	t	digits_3	Euler_h_05	Exact
1	0	1.0000	1.0000	1.0000
2	0.1000	1.2000	1.2025	1.2054
3	0.2000	1.4200	1.4160	1.4230
4	0.3000	1.6500	1.6429	1.6555
5	0.4000	1.9000	1.8859	1.9064

Code on next page

```
clear;
%Change TableTimes[], y(1) the initial value,
%       slope=dy/dt, solution, T.Properties.VariableNames
%times to display the y values in the table
TableTimes = [0 0.1 0.2 0.3 0.4];
NumRows = length(TableTimes);
% pre-allocate memory for 4 columns in table
V = zeros(NumRows, 4);

% populate table times (Col=1), exact values for solution (Col=4)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,4) = 3/4 + (3/2)*t + (1/4)*exp(2*t);
end

h = 0.05;
t = TableTimes(1):h:TableTimes(end);
% pre-allocate memory for y, f
y = zeros(length(t),1);
f = zeros(length(t),1);
y(1) = 1;   %really y0, but MATLAB not zero based

Col = 2;  % 3 digit calculations (2 decimal rounding)
slope = @(t,y) d3(d3(2*y) - d3(3*t));  % slope = dy/dt = f(t,y)
Row = 1;  % start at row 1 for each column
f(1) = slope(t(1), y(1));  % initialize
V(Row,Col) = y(1);
for i = 2:length(t)
    y(i) = d3(y(i-1) + d3(f(i-1)*h));  %new estimate of y(t)
    f(i) = slope(t(i), y(i));
    %display table only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

Col = 3;   % standard Euler method
slope = @(t,y) 2*y - 3*t;  % slope = dy/dt = f(t,y)
Row = 1;   % start at row 1 for each column
f(1) = slope(t(1), y(1));  % initialize
V(Row,Col) = y(1);
for i = 2:length(t)
    y(i) = y(i-1) + f(i-1)*h;  %new estimate of y(t)
    f(i) = slope(t(i), y(i));
    %display table only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'digits_3', 'Euler_h_05', 'Exact'}

function n3digit = d3(x)
    % num2str(x,3) has unpredictable rounding.  This
    %   code produces 3-digit precision predictably by
    %   adding 5 to the 4th sig digit, then truncating
    %   the 4th sig digit
    xstr = sprintf('%.3e',x);  % example: '4.225e-06'
    sn = sign(x);  % need to know if '-' in string
    l = length(xstr);
    ex = xstr(l-3:l);  % get exponent component, e.g. 'e-06'
    y = strcat('0.005', ex);  % create number to add to
                              % 4th sig digit, '0.005e-06'
    z = x + sn*str2double(y); % number rounded away from 0
    zstr = sprintf('%.3e',z); % put in exp form
    l = length(zstr);
    zstr = strcat(zstr(l-8:l-5), ex); % get rid of 4th sig digit
    n3digit = str2double(zstr);  % 3 digit precision
end
```

23.

(a)

$$1000\begin{vmatrix} 6.0 & 18 \\ 2.0 & 6.0 \end{vmatrix} = 1000(36-36) = \underline{0}$$

(b)

$$1000\begin{vmatrix} 6.01 & 18.0 \\ 2.00 & 6.00 \end{vmatrix} = 1000(36.06-36.00) = \underline{60}$$

(c)

$$1000\begin{vmatrix} 6.010 & 18.04 \\ 2.004 & 6.000 \end{vmatrix} = 1000(36.06-36.152) = \underline{-92.16}$$

24.

$$0.22(3.19-2.17) = 0.22(1.02) = 0.2244 \Rightarrow \underline{0.224}$$

$$(0.22)(3.19) - (0.22)(2.17) = 0.7018 - 0.4774$$

$$\Rightarrow 0.702 - 0.477 = \underline{0.225}$$

8.2 Improvements on the Euler Method

Note Title 4/13/2020

1.

For column 4 ($h = 0.025$)

$y_0 = 1$

$a = 1 - (0) + 4(1) = 5$ (t_0, y_0)

$b = 1 - (0 + 0.025) + 4(1 + 0.25(5)) = 5.475$ (t_0, h, y_0, h, a)

$y_1 = 1 + 0.5(0.025)(5 + 5.475) = 1.13094$ (y_0, $1/2$, h, $a + b$)

$a = 1 - (0.025) + 4(1.1309 4) = 5.49875$ (t_1, y_1)

$b = 1 - (0.025 + 0.025) + 4(1.13094 + 0.025(5.49875)) = 6.02363$ ($t_1 + h$, y_1, h, a)

$y_2 = 1.13094 + 0.5(0.025)(5.49875 + 6.02363) = 1.27497$ (y_1, $1/2$, h, $a + b$)

$a = 1 - (0.05) + 4(1.27497) = 6.04987$ (t_2, y_2)

$b = 1 - (0.05 + 0.025) + 4(1.27497 + 0.025(6.04987)) = 6.62986$ ($t_2 + h$, y_2, h, a)

$y_3 = 1.27497 + 0.5(0.025)(6.04987 + 6.62986) = 1.43346$ (y_2, $1/2$, h, $a + b$)

$a = 1 - (0.075) + 4(1.43346) = 6.65885$ (t_3, y_3)

$b = 1 - (0.075 + 0.025) + 4(1.43415 + 0.025(6.65885)) = 7.29974$ ($t_3 + h$, y_3, h, a)

$y_4 = \underset{y_3}{1.43346} + \underset{1/2}{0.5}\,\underset{h}{(0.025)}\,(\underset{a}{6.65885} + \underset{b}{7.29974}) = 1.60795$

∴ After the above computations, the first table entry for $t = 0.1$ is obtained: $y_4 = 1.60795$

The MATLAB code for all the calculations is shown on the next page. This was done using version R2020a, which provides for more flexibility in strings for table column headers, specified in "T.Properties.VariableNames".

In the code, use is made of an anonymous function slope = @(t,y) 1-t+4*y, a short, simple way to create a function, making coding easier.

	t	Euler h = 0.01	Euler h = 0.001	Imp Euler h = 0.025	Imp Euler h = 0.01	Exact
1	0	1	1	1	1	1
2	0.1000	1.5953	1.6076	1.6079	1.6089	1.6090
3	0.2000	2.4645	2.5011	2.5021	2.5048	2.5053
4	0.3000	3.7390	3.8207	3.8228	3.8289	3.8301
5	0.4000	5.6137	5.7755	5.7797	5.7918	5.7942
6	0.5000	8.3767	8.6771	8.6849	8.7075	8.7120
7	1.0000	60.0371	64.3826	64.4979	64.8307	64.8978
8	1.5000	426.4082	473.5598	474.8340	478.5159	479.2592
9	2.0000	3.0293e+03	3.4842e+03	3.4967e+03	3.5329e+03	3.5402e+03

```
clear
%Change TableTimes[], y0 the initial value,
%       slope=dy/dt, T.Properties.VariableNames

y0 = 1;
slope = @(t,y) 1 - t + 4*y;   % slope = dy/dt = f(t,y)
% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 1.5, 2.0];
NumRows = length(TableTimes);
%incremental time values (=h) for computing y and y'
EulerSteps = [0.01, 0.001];
ImpEulerSteps = [0.025, 0.01];
NumSteps = length(EulerSteps) + length(ImpEulerSteps);
%initialize table array: columns for times + h values + exact
V = zeros(NumRows, 2 + NumSteps);

% Populate table times (Col=1) and exact values for solution (Col=6)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,2+NumSteps) = -3/16 + (1/4)*t + (19/16)*exp(4*t);
end

Col = 2;   % start at Col = 2 for standard Euler method
for h = EulerSteps
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory for y
    y(1) = y0;               % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    for i = 2:length(t)
        y(i) = y(i-1) + slope(t(i-1), y(i-1))*h;  %new estimate of y(t)
        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,1) = t(i);
            V(Row,Col) = y(i);
        end
    end
    Col = Col + 1;  % go to next column
end

Col = 4;  % start at Col = 4 for Improved Euler Method
for h = ImpEulerSteps
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory for y
    y(1) = y0;               % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    for i = 2:length(t)
        a = slope(t(i-1), y(i-1));
        b = slope(t(i), y(i-1) + h*a);
        y(i) = y(i-1) + 0.5*(a+b)*h;  % new estimate of y(t)
        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,1) = t(i);
            V(Row,Col) = y(i);
        end
    end
    Col = Col + 1;  % go to next column
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Euler h = 0.01', 'Euler h = 0.001', ...
    'Imp Euler h = 0.025', 'Imp Euler h = 0.01', ...
    'Exact'}
```

2.

Exact solution: $\frac{d}{dt}(ye^t) = 3e^t + te^t$

$\therefore ye^t = 3e^t + (te^t - e^t) + c = 2e^t + te^t + c,$

$y = 2 + t + ce^{-t}, \quad y(0) = 1 \Rightarrow c = -1$

$\therefore y(t) = 2 + t - e^{-t}$

Improved Euler: Let $a = 3 + t_n - y_n$

$\therefore y_{n+1} = y_n + \frac{h}{2}\left[a + 3 + (t_n + h) - (y_n + ha)\right]$

Euler: $y_{n+1} = y_n + h(3 + t_n - y_n)$

Backward Euler: $y_{n+1} = y_n + h(3 + t_{n+1} - y_{n+1})$

$\therefore y_{n+1} = \frac{y_n + 3h + ht_{n+1}}{1+h} = \frac{y_n + h(3 + h + t_n)}{1+h}$

(a) $h = 0.05$

MATLAB code and results on next page.

```
clear
%Change TableTimes[], h, y(0) the initial value,
%       T.Properties.VariableNames, exact sol and @ functions

% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4];
NumRows = length(TableTimes);
%initialize table array: columns for times + ImpEuler + Euler + BkEuler + exact
V = zeros(NumRows, 5);

% Populate table times (Col=1) and exact values for solution (Col=5)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,5) = 2 + t - exp(-t);
end

h = 0.05;
t = TableTimes(1):h:TableTimes(end);
% pre-allocate memory for y values
y_im = zeros(length(t),1);
y_eu = zeros(length(t),1);
y_bk = zeros(length(t),1);
% initialize 1st entry to y(0)
y_im(1) = 1; y_eu(1) = 1; y_bk(1) = 1;
% create anonymous functions for easy coding below
fn = @(t,y) 3 + t - y;   % fn = dy/dt = f(t,y)
bk = @(t,y) (y + h*(3 + h + t))/(1+ h);  % backward

Row = 1;   %start at row 1 for each column
V(Row,2) = y_im(1);  % populate table array
V(Row,3) = y_eu(1);
V(Row,4) = y_bk(1);
for i = 2:length(t)
    y_eu(i) = y_eu(i-1) + h*fn(t(i-1), y_eu(i-1));
        a = fn(t(i-1), y_im(i-1));
    y_im(i) = y_im(i-1)+ (a + fn(t(i-1)+ h, y_im(i-1) + h*a))*h/2;
    y_bk(i) = bk(t(i-1), y_bk(i-1));
    % display table results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,2) = y_im(i);  % populate table array
        V(Row,3) = y_eu(i);
        V(Row,4) = y_bk(i);
    end
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Improved_Euler', 'Euler', ...
                              'Bkwd_Euler', 'Exact'}
```

T = 5×5 table

	t	Improved_Euler	Euler	Bkwd_Euler	Exact
1	0	1.000000000000000	1.0000000…	1.0000000000…	1.0000000…
2	0.10000000…	1.195123437500000	1.1975000…	1.1929705215…	1.1951625…
3	0.20000000…	1.381198406638184	1.3854937…	1.3772975252…	1.3812692…
4	0.30000000…	1.559085628829237	1.5649081…	1.5537846033…	1.5591817…
5	0.40000000…	1.729563950708151	1.7365795…	1.7231606379…	1.7296799…

$h = 0.05$

(b) $h = 0.025$

```
clear
%Change TableTimes[], h, y(0) the initial value,
%       T.Properties.VariableNames, exact sol and @ functions

% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4];
NumRows = length(TableTimes);
%initialize table array: columns for times + ImpEuler + Euler + BkEuler + exact
V = zeros(NumRows, 5);

% Populate table times (Col=1) and exact values for solution (Col=5)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,5) = 2 + t - exp(-t);
end

h = 0.025;
t = TableTimes(1):h:TableTimes(end);
% pre-allocate memory for y values
y_im = zeros(length(t),1);
y_eu = zeros(length(t),1);
y_bk = zeros(length(t),1);
% initialize 1st entry to y(0)
y_im(1) = 1; y_eu(1) = 1; y_bk(1) = 1;
% create anonymous functions for easy coding below
fn = @(t,y) 3 + t - y;   % fn = dy/dt = f(t,y)
bk = @(t,y) (y + h*(3 + h + t))/(1+ h);  % backward

Row = 1;   %start at row 1 for each column
V(Row,2) = y_im(1);  % populate table array
V(Row,3) = y_eu(1);
V(Row,4) = y_bk(1);
for i = 2:length(t)
    y_eu(i) = y_eu(i-1) + h*fn(t(i-1), y_eu(i-1));
        a = fn(t(i-1), y_im(i-1));
    y_im(i) = y_im(i-1)+ (a + fn(t(i-1)+ h, y_im(i-1) + h*a))*h/2;
    y_bk(i) = bk(t(i-1), y_bk(i-1));
    % display table results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,2) = y_im(i);  % populate table array
        V(Row,3) = y_eu(i);
        V(Row,4) = y_bk(i);
    end
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Improved_Euler', 'Euler', ...
                              'Bkwd_Euler', 'Exact'}
```

T = 5×5 table

	t	Improved_Euler	Euler	Bkwd_Euler	Exact
1	0	1.000000000000000	1.0000000…	1.0000000000…	1.0000000…
2	0.10000000…	1.195152978030386	1.1963121…	1.1940493552…	1.1951625…
3	0.20000000…	1.381251866832721	1.3833481…	1.3792534291…	1.3812692…
4	0.30000000…	1.559158189960407	1.5620016…	1.5564441149…	1.5591817…
5	0.40000000…	1.729651494435096	1.7330798…	1.7263750664…	1.7296799…

$h = 0.025$

(c) $h = 0.0125$

```
clear
%Change TableTimes[], h, y(0) the initial value,
%       T.Properties.VariableNames, exact sol and @ functions

% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4];
NumRows = length(TableTimes);
%initialize table array: columns for times + ImpEuler + Euler + BkEuler + exact
V = zeros(NumRows, 5);

% Populate table times (Col=1) and exact values for solution (Col=5)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,5) = 2 + t - exp(-t);
end

h = 0.0125;
t = TableTimes(1):h:TableTimes(end);
% pre-allocate memory for y values
y_im = zeros(length(t),1);
y_eu = zeros(length(t),1);
y_bk = zeros(length(t),1);
% initialize 1st entry to y(0)
y_im(1) = 1; y_eu(1) = 1; y_bk(1) = 1;
% create anonymous functions for easy coding below
fn = @(t,y) 3 + t - y;   % fn = dy/dt = f(t,y)
bk = @(t,y) (y + h*(3 + h + t))/(1+ h);  % backward

Row = 1;   %start at row 1 for each column
V(Row,2) = y_im(1);  % populate table array
V(Row,3) = y_eu(1);
V(Row,4) = y_bk(1);
for i = 2:length(t)
    y_eu(i) = y_eu(i-1) + h*fn(t(i-1), y_eu(i-1));
        a = fn(t(i-1), y_im(i-1));
    y_im(i) = y_im(i-1)+ (a + fn(t(i-1)+ h, y_im(i-1) + h*a))*h/2;
    y_bk(i) = bk(t(i-1), y_bk(i-1));
    % display table results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,2) = y_im(i);  % populate table array
        V(Row,3) = y_eu(i);
        V(Row,4) = y_bk(i);
    end
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Improved_Euler', 'Euler', ...
                              'Bkwd_Euler', 'Exact'}
```

T = 5×5 table

	t	Improved_Euler	Euler	Bkwd_Euler	Exact
1	0	1.000000000000000	1.0000000…	1.0000000000…	1.0000000…
2	0.10000000…	1.195160203412267	1.1957326…	1.1946015536…	1.1951625…
3	0.20000000…	1.381264942511069	1.3823006…	1.3802536533…	1.3812692…
4	0.30000000…	1.559175937122469	1.5605811…	1.5578029313…	1.5591817…
5	0.40000000…	1.729672905638597	1.7313677…	1.7280159272…	1.7296799…

$h = 0.0125$

3.

Exact solution: $\frac{d}{dt}(ye^{-2t}) = -3te^{-2t}$,

$\therefore ye^{-2t} = \frac{3}{2}te^{-2t} + \frac{3}{4}e^{-2t} + c$

$y(t) = \frac{3}{2}t + \frac{3}{4} + ce^{2t}$, $y(0) = 1 \Rightarrow c = \frac{1}{4}$

$\therefore y(t) = \frac{3}{4} + \frac{3}{2}t + \frac{1}{4}e^{2t}$

Euler: $y_{n+1} = y_n + h(2y_n - 3t_n)$

Backward: $y_{n+1} = y_n + h(2y_{n+1} - 3t_{n+1})$

$\therefore y_{n+1}(1-2h) = y_n - 3h(t_n + h)$

$$y_{n+1} = \frac{y_n - 3h(t_n + h)}{1-2h}$$

Improved: Let $a = 2y_n - 3t_n$

$$y_{n+1} = y_n + \frac{h}{2}[a + 2(y_n + ha) - 3(t_n + h)]$$

Modify MATLAB code slightly from #2 above to produce the 3 tables in one program.

```
ans = 'Table for h = 0.0500'
```

T = 5×5 table

	t	Improved_Euler	Euler	Bkwd_Euler	Exact
1	0	1.000000000000000	1.0000000…	1.0000000000…	1.0000000…
2	0.10000000…	1.205256250000000	1.2025000…	1.2086419753…	1.2053506…
3	0.20000000…	1.422725512656250	1.4160250…	1.4310394756…	1.4229561…
4	0.30000000…	1.655107169091098	1.6428902…	1.6704191057…	1.6555297…
5	0.40000000…	1.905697231139458	1.8858972…	1.9307643281…	1.9063852…

```
ans = 'Table for h = 0.0250'
```

T = 5×5 table

	t	Improved_Euler	Euler	Bkwd_Euler	Exact
1	0	1.000000000000000	1.0000000…	1.0000000000…	1.0000000…
2	0.10000000…	1.205326179785767	1.2038765…	1.2069344157…	1.2053506…
3	0.20000000…	1.422896304250281	1.4193638…	1.4268349423…	1.4229561…
4	0.30000000…	1.655420016131877	1.6489640…	1.6626544515…	1.6555297…
5	0.40000000…	1.906206614894072	1.8957186…	1.9180182952…	1.9063852…

```
ans = 'Table for h = 0.0125'
```

T = 5×5 table

	t	Improved_Euler	Euler	Bkwd_Euler	Exact
1	0	1.000000000000000	1.0000000…	1.0000000000…	1.0000000…
2	0.10000000…	1.205344446216269	1.2046007…	1.2061280203…	1.2053506…
3	0.20000000…	1.422940923340480	1.4211264…	1.4248574594…	1.4229561…
4	0.30000000…	1.655501758835131	1.6521814…	1.6590174879…	1.6555297…
5	0.40000000…	1.906339729208197	1.9009392…	1.9120724596…	1.9063852…

MATLAB code on next page.

Note: only code for exact solution, @ functions, and y(0) needs to be modified for problems #2 → #6.

```
clear
%Change TableTimes[], h, y(0) the initial value,
%       T.Properties.VariableNames, exact sol and @ functions

% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4];
NumRows = length(TableTimes);
%initialize table array: columns for times + ImpEuler + Euler + BkEuler + exact
V = zeros(NumRows, 5);

% Populate table times (Col=1) and exact values for solution (Col=5)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,5) = 3/4 + 3*t/2 + (1/4)*exp(2*t);  % Exact solution
end

for h = [0.05, 0.025, 0.0125]
    t = TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y values
    y_im = zeros(length(t),1);
    y_eu = zeros(length(t),1);
    y_bk = zeros(length(t),1);
    % initialize 1st entry to y(0)
    y_im(1) = 1; y_eu(1) = 1; y_bk(1) = 1;
    % create anonymous functions for easy coding below
    fn = @(t,y) 2*y - 3*t;   % fn = dy/dt = f(t,y)
    bk = @(t,y) (y - 3*h*(t + h))/(1 - 2*h);  % backward

    Row = 1;   %start at row 1 for each column
    V(Row,2) = y_im(1);  % populate table array
    V(Row,3) = y_eu(1);
    V(Row,4) = y_bk(1);
    for i = 2:length(t)
        y_eu(i) = y_eu(i-1) + h*fn(t(i-1), y_eu(i-1));
            a = fn(t(i-1), y_im(i-1));
        y_im(i) = y_im(i-1)+ (a + fn(t(i-1)+ h, y_im(i-1) + h*a))*h/2;
        y_bk(i) = bk(t(i-1), y_bk(i-1));
        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,2) = y_im(i);  % populate table array
            V(Row,3) = y_eu(i);
            V(Row,4) = y_bk(i);
        end
    end

    format long
    T = array2table(V);
    % label the table columms and display the table
    sprintf('Table for h = %.4f',h)
    T.Properties.VariableNames = {'t', 'Improved_Euler', 'Euler', ...
                                  'Bkwd_Euler', 'Exact'}
end
```

4.

No column for exact solution.

Euler: $y_{n+1} = y_n + h(2t_n + e^{-t_n y_n})$

Backward: $y_{n+1} = y_n + h(2t_{n+1} + e^{-t_{n+1} y_{n+1}})$

No explicit formula for y_{n+1}. Use MATLAB's "vpasolve" method to get y_{n+1} using y_n, t_n, h.

Note use of a range around y_n to get the estimate for y_{n+1} closest to y_n, as "vpasolve" may obtain several solutions.

Improved: Let $a = 2t_n + e^{-t_n y_n}$

$$\therefore y_{n+1} = y_n + \frac{h}{2}\left[a + 2(t_n + h) + e^{-(t_n+h)(y_n + ha)}\right]$$

Answers in table form on next page.

```
ans = 'Table for h = 0.0500'
```

T = 5×4 table

	t	Improved_Euler	Euler	Bkwd_Euler
1	0	1.000000000000000	1.0000000…	1.0000000000…
2	0.10000000…	1.104828922848498	1.1024427…	1.1071964200…
3	0.20000000…	1.218823869462594	1.2142558…	1.2233331744…
4	0.30000000…	1.341458888912524	1.3348419…	1.3479680819…
5	0.40000000…	1.472626670489802	1.4639918…	1.4811042567…

```
ans = 'Table for h = 0.0250'
```

T = 5×4 table

	t	Improved_Euler	Euler	Bkwd_Euler
1	0	1.000000000000000	1.0000000…	1.0000000000…
2	0.10000000…	1.104839333541998	1.1036484…	1.1060258674…
3	0.20000000…	1.218836490748672	1.2165597…	1.2210991342…
4	0.30000000…	1.341465302208264	1.3381703…	1.3447340434…
5	0.40000000…	1.472618923828322	1.4683213…	1.4768779725…

```
ans = 'Table for h = 0.0125'
```

T = 5×4 table

	t	Improved_Euler	Euler	Bkwd_Euler
1	0	1.000000000000000	1.0000000…	1.0000000000…
2	0.10000000…	1.104842078602127	1.1042471…	1.1054359454…
3	0.20000000…	1.218839885414212	1.2177032…	1.2199730556…
4	0.30000000…	1.341467203856788	1.3398229…	1.3431049409…
5	0.40000000…	1.472617312810881	1.4704733…	1.4747517324…

MATLAB code on next page.

```
clear
%Change TableTimes[], h, y(0) the initial value,
%       T.Properties.VariableNames, exact sol and @ functions

% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4];
NumRows = length(TableTimes);
%initialize table array: columns for times + ImpEuler + Euler + BkEuler + exact
V = zeros(NumRows, 4);  % 5 columns if Exact is known

% Populate table times (Col=1) and exact values for solution (Col=5)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    %V(i,5) = ;  % Exact solution
end
syms x  % for backward method, using vpasolve

for h = [0.05, 0.025, 0.0125]
    t = TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y values
    y_im = zeros(length(t),1);
    y_eu = zeros(length(t),1);
    y_bk = zeros(length(t),1);
    % initialize 1st entry to y(0)
    y_im(1) = 1; y_eu(1) = 1; y_bk(1) = 1;
    % create anonymous functions for easy coding below
    fn = @(t,y) 2*t + exp(-t*y);   % fn = dy/dt = f(t,y)
    %bk = @(t,y) ;  % backward

    Row = 1;   %start at row 1 for each column
    V(Row,2) = y_im(1);  % populate table array
    V(Row,3) = y_eu(1);
    V(Row,4) = y_bk(1);
    for i = 2:length(t)
        y_eu(i) = y_eu(i-1) + h*fn(t(i-1), y_eu(i-1));
            a = fn(t(i-1), y_im(i-1));
        y_im(i) = y_im(i-1)+ (a + fn(t(i-1)+ h, y_im(i-1) + h*a))*h/2;

        %y_bk(i) = bk(t(i-1), y_bk(i-1));
        eqn = x == y_bk(i-1) + h*(2*t(i) + exp(-t(i)*x));
        % guess 10*h as an interval to look for next y
        range = [y_bk(i-1) - 5*h, y_bk(i-1) + 5*h];
        y_bk(i) = vpasolve(eqn,x,range);

        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,2) = y_im(i);  % populate table array
            V(Row,3) = y_eu(i);
            V(Row,4) = y_bk(i);
        end
    end

    format long
    T = array2table(V);
    % label the table columms and display the table
    sprintf('Table for h = %.4f',h)
    T.Properties.VariableNames = {'t', 'Improved_Euler', 'Euler', ...
                                  'Bkwd_Euler'}
end
```

5.

Euler: $y_{n+1} = y_n + h\left[\frac{y_n^2 + 2t_n y_n}{3 + t_n^2}\right]$

Backward: Using #12, Section 8.1,

$$y_{n+1} = y_n + h\left[\frac{y_{n+1}^2 + 2t_{n+1} y_{n+1}}{3 + t_{n+1}^2}\right]$$

$$\therefore \left[\frac{h}{3 + t_{n+1}^2}\right] y_{n+1}^2 + \left[\frac{2h t_{n+1}}{3 + t_{n+1}^2} - 1\right] y_{n+1} + y_n = 0$$

Get y_{n+1} solving quadratic equation, using MATLAB's "vpasolve" with a range around y_n to get root of y_{n+1} closest to y_n.

Improved Euler: Let $f(t,y) = \frac{y^2 + 2ty}{3 + t^2}$

Let $a = f(t_n, y_n)$

$$\therefore y_{n+1} = y_n + \frac{h}{2}\left[a + f(t_n + h, y_n + ha)\right]$$

No exact solution. Answers (tables) on next page.

```
ans = 'Table for h = 0.0500'
```

T = 5×4 table

	t	Improved_Euler	Euler	Bkwd_Euler
1	0	0.500000000000000	0.5000000…	0.5000000000…
2	0.10000000…	0.510164405455806	0.5092391…	0.5111273151…
3	0.20000000…	0.524125577399366	0.5221872…	0.5261547165…
4	0.30000000…	0.542083123398377	0.5390229…	0.5453061233…
5	0.40000000…	0.564250888364894	0.5599363…	0.5688229487…

```
ans = 'Table for h = 0.0250'
```

T = 5×4 table

	t	Improved_Euler	Euler	Bkwd_Euler
1	0	0.500000000000000	0.5000000…	0.5000000000…
2	0.10000000…	0.510168261195135	0.5097010…	0.5106447800…
3	0.20000000…	0.524134923538198	0.5231547…	0.5251375940…
4	0.30000000…	0.542099845806299	0.5405501…	0.5436900220…
5	0.40000000…	0.564277157257266	0.5620888…	0.5665294580…

```
ans = 'Table for h = 0.0125'
```

T = 5×4 table

	t	Improved_Euler	Euler	Bkwd_Euler
1	0	0.500000000000000	0.5000000…	0.5000000000…
2	0.10000000…	0.510169189206617	0.5099344…	0.5104062548…
3	0.20000000…	0.524137189531322	0.5236443…	0.5246356408…
4	0.30000000…	0.542103923944729	0.5413240…	0.5428938361…
5	0.40000000…	0.564283594335094	0.5631816…	0.5654015498…

MATLAB code on next page

```
clear
%Change TableTimes[], h, y(0) the initial value,
%       T.Properties.VariableNames, exact sol and @ functions

% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4];
NumRows = length(TableTimes);
%initialize table array: columns for times + ImpEuler + Euler + BkEuler + exact
V = zeros(NumRows, 4);  % 5 columns if Exact is known

% Populate table times (Col=1) and exact values for solution (Col=5)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    %V(i,5) = ;  % Exact solution
end
syms x  % for backward method, using vpasolve

for h = [0.05, 0.025, 0.0125]
    t = TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y values
    y_im = zeros(length(t),1);
    y_eu = zeros(length(t),1);
    y_bk = zeros(length(t),1);
    % initialize 1st entry to y(0)
    y_im(1) = 0.5; y_eu(1) = 0.5; y_bk(1) = 0.5;
    % create anonymous functions for easy coding below
    fn = @(t,y) (y^2 + 2*t*y)/(3 + t^2);   % fn = dy/dt = f(t,y)
    %bk = @(t,y) ;  % backward

    Row = 1;   %start at row 1 for each column
    V(Row,2) = y_im(1);  % populate table array
    V(Row,3) = y_eu(1);
    V(Row,4) = y_bk(1);
    for i = 2:length(t)
        y_eu(i) = y_eu(i-1) + h*fn(t(i-1), y_eu(i-1));
            a = fn(t(i-1), y_im(i-1));
        y_im(i) = y_im(i-1)+ (a + fn(t(i-1)+ h, y_im(i-1) + h*a))*h/2;

        %y_bk(i) = bk(t(i-1), y_bk(i-1));
        a = h/(3+t(i)^2);  % specify quadratic coefficients
        b = 2*h*t(i)/(3+t(i)^2) - 1;
        c = y_bk(i-1);
        eqn = a*x^2 + b*x + c == 0;
        % guess 10*h as an interval to look for next y
        range = [y_bk(i-1) - 5*h, y_bk(i-1) + 5*h];
        y_bk(i) = vpasolve(eqn,x,range);

        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,2) = y_im(i);  % populate table array
            V(Row,3) = y_eu(i);
            V(Row,4) = y_bk(i);
        end
    end

    format long
    T = array2table(V);
    % label the table columms and display the table
    sprintf('Table for h = %.4f',h)
    T.Properties.VariableNames = {'t', 'Improved_Euler', 'Euler', ...
                                  'Bkwd_Euler'}
end
```

6.

Euler: $y_{n+1} = y_n + h\left[(t_n^2 - y_n^2)\sin(y_n)\right]$

Backward: $y_{n+1} = y_n + h\left[(t_{n+1}^2 - y_{n+1}^2)\sin(y_{n+1})\right]$

No explicit formula. Use MATLAB's "vpasolve" to get y_{n+1} from y_n, t_n, h, noting $t_{n+1} = t_n + h$, and use a range around y_n to get the y_{n+1} closest to y_n.

Improved Euler: Let $a = f(t_n, y_n) = (t_n^2 - y_n^2)\sin(y_n)$

$\therefore y_{n+1} = y_n + \frac{h}{2}\left[a + f(t_n + h, y_n + ha)\right]$

ans = 'Table for h = 0.0500'

T = 5×4 table

	t	Improved_Euler	Euler	Bkwd_Euler
1	0	-1.0000000000000...	-1.0000000...	-1.000000000...
2	0.10000000...	-0.9246496575239...	-0.9204978...	-0.928058829...
3	0.20000000...	-0.8643379978643...	-0.8575380...	-0.870054171...
4	0.30000000...	-0.8166421419442...	-0.8080301...	-0.824020997...
5	0.40000000...	-0.7800082959795...	-0.7700377...	-0.788685901...

ans = 'Table for h = 0.0250'

T = 5×4 table

	t	Improved_Euler	Euler	Bkwd_Euler
1	0	-1.0000000000000...	-1.0000000...	-1.000000000...
2	0.10000000...	-0.9245497576669...	-0.9225751...	-0.926340535...
3	0.20000000...	-0.8641774157984...	-0.8609229...	-0.867163191...
4	0.30000000...	-0.8164422966826...	-0.8122997...	-0.820278744...
5	0.40000000...	-0.7797807583424...	-0.7749651...	-0.784275183...

MATLAB code on next page

```
ans = 'Table for h = 0.0125'
```

T = 5×4 table

	t	Improved_Euler	Euler	Bkwd_Euler
1	0	-1.0000000000000…	-1.0000000…	-1.000000000…
2	0.10000000…	-0.9245252514521…	-0.9235619…	-0.925442784…
3	0.20000000…	-0.8641379250416…	-0.8625454…	-0.865663418…
4	0.30000000…	-0.8163930357497…	-0.8143611…	-0.818348620…
5	0.40000000…	-0.7797245486533…	-0.7773578…	-0.782011159…

```
clear
%Change TableTimes[], h, y0 the initial value,
%       T.Properties.VariableNames, exact sol and @ functions

% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4];
NumRows = length(TableTimes);
%initialize table array: columns for times + ImpEuler + Euler + BkEuler
V = zeros(NumRows, 4);

% Populate table times (Col=1) and exact values for solution (Col=5)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end
syms x  % for backward method, using vpasolve

y0 = -1;  % y(0) initial value
for h = [0.05, 0.025, 0.0125]
    t = TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y values
    y_im = zeros(length(t),1);
    y_eu = zeros(length(t),1);
    y_bk = zeros(length(t),1);
    % initialize 1st entry to y(0)
    y_im(1) = y0; y_eu(1) = y0; y_bk(1) = y0;
    % create anonymous functions for easy coding below
    fn = @(t,y) (t^2 - y^2)*sin(y);   % fn = dy/dt = f(t,y)

    Row = 1;   %start at row 1 for each column
    V(Row,2) = y_im(1);  % populate table array
    V(Row,3) = y_eu(1);
    V(Row,4) = y_bk(1);
    for i = 2:length(t)
        y_eu(i) = y_eu(i-1) + h*fn(t(i-1), y_eu(i-1));
            a = fn(t(i-1), y_im(i-1));
        y_im(i) = y_im(i-1)+ (a + fn(t(i-1)+ h, y_im(i-1) + h*a))*h/2;

        eqn = x == y_bk(i-1) + h*(t(i)^2 - x^2)*sin(x);
        % guess 10*h as an interval to look for next y
        range = [y_bk(i-1) - 5*h, y_bk(i-1) + 5*h];
        y_bk(i) = vpasolve(eqn,x,range);

        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,2) = y_im(i);  % populate table array
            V(Row,3) = y_eu(i);
            V(Row,4) = y_bk(i);
        end
    end

    format long
    T = array2table(V);
    % label the table columms and display the table
    sprintf('Table for h = %.4f',h)
    T.Properties.VariableNames = {'t', 'Improved_Euler', 'Euler', 'Bkwd_Euler'}
end
```

7.

Using MATLAB,

Version R2020a allows "h = 0.025" column header

T = 5×3 table

	t	h = 0.025	h = 0.0125
1	0	1.0000	1.0000
2	0.5000	2.9672	2.9680
3	1.0000	7.8831	7.8875
4	1.5000	20.8114	20.8294
5	2.0000	55.5106	55.5758

```
clear
%Edit TableTimes[], h = StepSizes, y0 = y(0),
%     @ function, T.Properties.VariableNames

TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
StepSizes = [0.025, 0.0125];
y0 = 1;  % y(0) initial value
% create anonymous function for easy coding below
fn = @(t,y) 0.5 - t + 2*y;   % fn = dy/dt = f(t,y)

NumRows = length(TableTimes);
%initialize table array: columns for times + StepSizes
V = zeros(NumRows, 1 + length(StepSizes));
% Populate table times (Col=1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

Col = 1;  % initialize for V(,Col)
for h = StepSizes
    Col = Col + 1;
    t = TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y values
    y = zeros(length(t),1);
    y(1) = y0;  % initialize 1st entry to y(0)

    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);  % populate table array
    for i = 2:length(t)
        a = fn(t(i-1), y(i-1));  % for Improved Euler formula
        y(i) = y(i-1)+ (a + fn(t(i-1)+ h, y(i-1) + h*a))*h/2;

        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);  % populate table array
        end
    end
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h = 0.025', 'h = 0.0125'}
```

8.

Using MATLAB, version R2020a

T = 5×3 table

	t	h = 0.025	h = 0.0125
1	0	2.0000	2.0000
2	0.5000	0.9261	0.9258
3	1.0000	1.2856	1.2853
4	1.5000	2.4090	2.4087
5	2.0000	4.1039	4.1036

```
clear
%Edit TableTimes[], h = StepSizes, y0 = y(0),
%     @ function, T.Properties.VariableNames

TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
StepSizes = [0.025, 0.0125];
y0 = 2;  % y(0) initial value
% create anonymous function for easy coding below
fn = @(t,y) 5*t - 3*sqrt(y);   % fn = dy/dt = f(t,y)

NumRows = length(TableTimes);
%initialize table array: columns for times + StepSizes
V = zeros(NumRows, 1 + length(StepSizes));
% Populate table times (Col=1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

Col = 1;  % initialize for V(,Col)
for h = StepSizes
    Col = Col + 1;
    t = TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y values
    y = zeros(length(t),1);
    y(1) = y0;  % initialize 1st entry to y(0)

    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);  % populate table array
    for i = 2:length(t)
        a = fn(t(i-1), y(i-1));  % for Improved Euler formula
        y(i) = y(i-1)+ (a + fn(t(i-1)+ h, y(i-1) + h*a))*h/2;

        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);  % populate table array
        end
    end
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h = 0.025', 'h = 0.0125'}
```

9.

Using MATLAB,

using Format Long

T = 5×3 table

	t	h = 0.025	h = 0.0125
1	0	3.00000000000…	3.000000000000…
2	0.50000000…	3.96216712110…	3.962181609358…
3	1.00000000…	5.10886638094…	5.108893616575…
4	1.50000000…	6.43133830104…	6.431377228403…
5	2.00000000…	7.92331820767…	7.923368117951…

```
clear
%Edit TableTimes[], h = StepSizes, y0 = y(0),
%     @ function, T.Properties.VariableNames

TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
StepSizes = [0.025, 0.0125];
y0 = 3;  % y(0) initial value
% create anonymous function for easy coding below
fn = @(t,y) sqrt(t + y);   % fn = dy/dt = f(t,y)

NumRows = length(TableTimes);
%initialize table array: columns for times + StepSizes
V = zeros(NumRows, 1 + length(StepSizes));
% Populate table times (Col=1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

Col = 1;  % initialize for V(,Col)
for h = StepSizes
    Col = Col + 1;
    t = TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y values
    y = zeros(length(t),1);
    y(1) = y0;  % initialize 1st entry to y(0)

    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);  % populate table array
    for i = 2:length(t)
        a = fn(t(i-1), y(i-1));  % for Improved Euler formula
        y(i) = y(i-1)+ (a + fn(t(i-1)+ h, y(i-1) + h*a))*h/2;

        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);  % populate table array
        end
    end
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h = 0.025', 'h = 0.0125'}
```

10.

Using MATLAB,

T = 5×3 table

	t	h = 0.025	h = 0.0125
1	0	1.00000000000…	1.000000000000…
2	0.50000000…	1.61263136630…	1.612624508858…
3	1.00000000…	2.48096653283…	2.480924283753…
4	1.50000000…	3.74555736922…	3.745498974732…
5	2.00000000…	5.49595259178…	5.495892945186…

```
clear
%Edit TableTimes[], h = StepSizes, y0 = y(0),
%     @ function, T.Properties.VariableNames

TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
StepSizes = [0.025, 0.0125];
y0 = 1;  % y(0) initial value
% create anonymous function for easy coding below
fn = @(t,y) 2*t + exp(-t*y);   % fn = dy/dt = f(t,y)

NumRows = length(TableTimes);
%initialize table array: columns for times + StepSizes
V = zeros(NumRows, 1 + length(StepSizes));
% Populate table times (Col=1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

Col = 1;  % initialize for V(,Col)
for h = StepSizes
    Col = Col + 1;
    t = TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y values
    y = zeros(length(t),1);
    y(1) = y0;  % initialize 1st entry to y(0)

    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);  % populate table array
    for i = 2:length(t)
        a = fn(t(i-1), y(i-1));  % for Improved Euler formula
        y(i) = y(i-1)+ (a + fn(t(i-1)+ h, y(i-1) + h*a))*h/2;

        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);  % populate table array
        end
    end
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h = 0.025', 'h = 0.0125'}
```

11.

Using MATLAB,

T = 5×3 table

	t	h = 0.025	h = 0.0125
1	0	0.50000000000…	0.500000000000…
2	0.50000000…	0.59089655473…	0.590905979544…
3	1.00000000…	0.79995006612…	0.799987534668…
4	1.50000000…	1.16653147636…	1.166632817107…
5	2.00000000…	1.74968914279…	1.749922034332…

```
clear
%Edit TableTimes[], h = StepSizes, y0 = y(0),
%     @ function, T.Properties.VariableNames

TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
StepSizes = [0.025, 0.0125];
y0 = 0.5;  % y(0) initial value
% create anonymous function for easy coding below
fn = @(t,y) (y^2 + 2*t*y)/(3 + t^2);   % fn = dy/dt = f(t,y)

NumRows = length(TableTimes);
%initialize table array: columns for times + StepSizes
V = zeros(NumRows, 1 + length(StepSizes));
% Populate table times (Col=1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

Col = 1;  % initialize for V(,Col)
for h = StepSizes
    Col = Col + 1;
    t = TableTimes(1):h:TableTimes(end);
    % pre-allocate memory for y values
    y = zeros(length(t),1);
    y(1) = y0;  % initialize 1st entry to y(0)

    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);  % populate table array
    for i = 2:length(t)
        a = fn(t(i-1), y(i-1));  % for Improved Euler formula
        y(i) = y(i-1)+ (a + fn(t(i-1)+ h, y(i-1) + h*a))*h/2;

        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);  % populate table array
        end
    end
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'h = 0.025', 'h = 0.0125'}
```

12.

(a)

From equation (5),

$$y_{n+1} = y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_n + h,\ y_n + h f(t_n, y_n)) \right] \quad [1]$$

From Taylor expansion,

$$\phi(t_{n+1}) = \phi(t_n) + \phi'(t_n)h + \phi''(t_n)\frac{h^2}{2!} + \phi'''(\bar{t}_n)\frac{h^3}{3!} \quad [2]$$

Subtracting [2] - [1], noting $y_n = \phi(t_n)$ and

$$\phi'(t_n) = f(t_n, \phi(t_n)) = f(t_n, y_n),$$

$$e_{n+1} = \phi(t_{n+1}) - y_{n+1}$$

$$= \phi(t_n) - y_n + \phi'(t_n)h + \phi''(t_n)\frac{h^2}{2!} - \frac{h}{2}\left[f(t_n, y_n) + f(t_n + h,\ y_n + h f(t_n, y_n)) \right] + \phi'''(\bar{t}_n)\frac{h^3}{3!}$$

($\phi(t_n) - y_n = 0$; $\phi'(t_n)h = f(t_n, y_n)h = \frac{2f(t_n, y_n)h}{2}$)

$$= \frac{2f(t_n, y_n)h + \phi''(t_n)h^2 - f(t_n, y_n)h - f(t_n + h, y_n + hf(t_n, y_n))h}{2!}$$

$$+ \phi'''(\bar{t}_n)\frac{h^3}{3!}$$

$$= \frac{\phi''(t_n)h^2 + f(t_n, y_n)h - f(t_n + h, y_n + hf(t_n, y_n))h}{2!}$$

$$+ \phi'''(\bar{t}_n)\frac{h^3}{3!}$$

$$= \frac{\phi''(t_n)h - \left[f(t_n + h, y_n + hf(t_n, y_n)) - f(t_n, y_n)\right]}{2!}h$$

$$+ \phi'''(\bar{t}_n)\frac{h^3}{3!}$$

(6)

$$\phi''(t_n) = f_t(t_n, \phi(t_n)) + f_y(t_n, \phi(t_n))\phi'(t_n)$$

$$= f_t(t_n, y_n) + f_y(t_n, y_n)f(t_n, y_n)$$

since $y_n = \phi(t_n)$, and $\phi'(t_n) = f(t_n, \phi(t_n)) = f(t_n, y_n)$

since ϕ is a solution to $y' = f(t,y)$

$\therefore \phi''(t_n)\,h = f_t(t_n, y_n)h + f_y(t_n, y_n)\,h\,f(t_n, y_n)$ [3]

Rewrite the Taylor expansion for two variables using $a = t_n$, $b = y_n$, $k = h f(t_n, y_n)$, so that

$$f(t_n + h, y_n + h f(t_n, y_n)) = \underbrace{f(t_n, y_n) + f_t(t_n, y_n)h + f_y(t_n, y_n)\,h\,f(t_n, y_n)}_{= \phi''(t_n)h}$$
$$+ \frac{1}{2!}\left[h^2 f_{tt} + 2h^2 f(t_n, y_n) f_{ty} + h^2 f^2(t_n, y_n) f_{yy}\right]\Big|_{t=\epsilon,\, y=\eta}$$

where $t_n \le \epsilon \le t_n + h$, $y_n \le \eta \le y_n + hf(t_n, y_n)$

$\therefore$ Using [3], and rearranging,

$$\phi''(t_n)h - \left[f(t_n + h, y_n + h f(t_n, y_n)) - f(t_n, y_n)\right]$$
$$= -\frac{1}{2!}\left[h^2 f_{tt} + 2h^2 f(t_n, y_n) f_{ty} + h^2 f^2(t_n, y_n) f_{yy}\right]\Big|_{t=\epsilon,\, y=\eta}$$
$$= -\frac{h^2}{2!}\left[f_{tt} + 2f(t_n, y_n) f_{ty} + f^2(t_n, y_n) f_{yy}\right]\Big|_{t=\epsilon,\, y=\eta}$$

Substituting this into the result in (a),

$$e_{n+1} = -\frac{h^3}{2!}\left[f_{tt}(\epsilon,\eta) + 2f(t_n,y_n)f_{ty}(\epsilon,\eta) + f^2(t_n,y_n)f_{yy}(\epsilon,\eta)\right] + \frac{h^3}{3!}\phi'''(\bar{t}_n)$$

$$\therefore \boxed{e_{n+1} = h^3\left[\frac{\phi'''(\bar{t}_n)}{3!} - \frac{\left[f_{tt} + 2f(t_n,y_n)f_{ty} + f^2(t_n,y_n)f_{yy}\right]}{2!}\right]}$$

(c)

If $f(t,y) = at + by$, a, b some constants,

then $f_t = a$, $f_y = b$, $f_{tt} = 0$, $f_{ty} = 0$, $f_{yy} = 0$

$\therefore$ From (b), $e_{n+1} = \frac{h^3}{6}\phi'''(\bar{t}_n)$

13.

(a)

Exact solution: $\phi(t) = -\frac{3}{16} + \frac{t}{4} + \frac{19}{16}e^{4t}$

$\therefore \phi' = \frac{1}{4} + \frac{19}{4}e^{4t}$, $\phi'' = 19e^{4t}$, $\phi'''(t) = 76e^{4t}$

$$\therefore e_{n+1} = \frac{h^3}{6}\left(76 e^{4\bar{t}_n}\right) = \frac{38}{3} h^3 e^{4\bar{t}_n}$$

Since e^x is monotonic increasing, on $[0,2]$,

$$e_{n+1} \le \frac{38}{3} h^3 e^{4(2)} \approx \frac{38}{3} h^3 (2981) \approx 37{,}758.8\, h^3$$

$$\therefore e_{n+1} \le 37{,}758.8\, h^3 \text{ on } [0,2]$$

(b)

In Section 8.1, $e_{n+1} = \frac{19 e^{4\bar{t}_n} h^2}{2}, \quad t_n < \bar{t}_n < t_n + h.$ (27)

$$\therefore \frac{19}{2} h^2 e^{4\bar{t}_n} \text{ vs. } \frac{38}{3} h^3 e^{4\bar{t}_n}$$

(c)

For e_1, $n=0$. $\therefore e_1 \le \frac{38}{3}(0.05)^3 e^{4(0.05)} = 0.00193$

From Section 8.1, $e_1 \le 0.02901$

14.

(a) Exact solution: $\frac{d}{dt}\left(ye^{-2t}\right) = -e^{-2t}$, $ye^{-2t} = \frac{1}{2}e^{-2t} + C$,

$y = \frac{1}{2} + Ce^{2t}$, $y(0) = 1 \Rightarrow C = \frac{1}{2}$. $\therefore \phi(t) = \frac{1}{2} + \frac{1}{2}e^{2t}$

$\therefore \phi'(t) = e^{2t}$, $\phi''(t) = 2e^{2t}$, $\phi'''(t) = 4e^{2t}$

Since $2y-1$ is linear in y and t, from #12(c),

$e_{n+1} = \frac{1}{6}\phi'''(\bar{t}_n)h^3 = \frac{1}{6}(4e^{2\bar{t}_n})h^3 = \frac{2}{3}e^{2\bar{t}_n}h^3$

$\therefore \underline{e_{n+1} = \frac{2}{3}e^{2\bar{t}_n}h^3}$, $t_n \le \bar{t}_n \le t_{n+1}$

Since $\phi'''(t)$ is monotonic increasing, $e^{2\bar{t}_n} \le e^2$

on $[0,1]$. $\therefore \underline{|e_{n+1}| \le \frac{2}{3}e^2h^3 \simeq (4.92604)h^3}$

(b) For e_1 and $h = 0.1$, $t_0 = 0$, $t_1 = 0.1$ on $[0,1]$

$\therefore 0 \le \bar{t}_1 \le 0.1$, so $|e_1| \le \frac{2}{3}e^{2(0.1)}(0.1)^3$

or, $\underline{|e_1| \le 0.00081427}$

15.

(a)

$$\frac{d}{dt}\left(ye^{-2t}\right) = 0.5e^{-2t} - te^{-2t}$$

$$ye^{-2t} = -\frac{1}{4}e^{-2t} + \frac{1}{2}te^{-2t} + \frac{1}{4}e^{-2t} + C, \quad y = \frac{1}{2}t + Ce^{2t}$$

$$y(0) = 1 \Rightarrow C = 1 \quad \therefore \phi(t) = \frac{1}{2}t + e^{2t}$$

$$\therefore \phi' = \frac{1}{2} + 2e^{2t}, \quad \phi'' = 4e^{2t}, \quad \phi''' = 8e^{2t}$$

$0.5 - t + 2y$ is linear in t, y. $\therefore$ By #12(c),

$$e_{n+1} = \frac{1}{6}\phi'''(\bar{t}_n)h^3 = \underline{\underline{\frac{4}{3}e^{2\bar{t}_n}h^3}}, \quad t_n \le \bar{t}_n \le t_{n+1}$$

e^{2t} is monotonic increasing, so $e^{2t} \le e^2$ on $[0,1]$

$$\therefore \underline{\underline{|e_{n+1}| \le \frac{4}{3}e^2h^3 \approx (9.85207)h^3}}$$

(b)

For e_1 and $h = 0.1$, $t_0 = 0$, $t_1 = 0.1$ on $[0,1]$

$$\therefore |e_1| \le \frac{4}{3}e^{2(0.1)}(0.1)^3 = \underline{\underline{0.0016285}}$$

16.

(a) Euler: $y_1 = y_0 + h(0.5 - t_0 + 2y_0)$

$= 1 + 0.1(0.5 - 0 + 2(1)) = \underline{1.25}$

Improved: Let $a = f(t_0, y_0) = f(0,1) = 0.5 - 0 + 2(1) = 2.5$

$\therefore y_1 = y_0 + \frac{h}{2}[a + f(t_0 + h, y_0 + ha)]$

$= 1 + \frac{0.1}{2}[2.5 + [0.5 - (0+0.1) + 2(1 + 0.1(2.5))]]$

$= \underline{1.27}$

Using MATLAB,

```
clear
fn = @(t,y) 0.5 - t + 2*y;  % f(t,y)
t0 = 0;
y0 = 1;
h = 0.1;

a = fn(t0,y0);
y_Euler = y0 + h*a
y_Improved = y0 + h/2*(a + fn(t0+h, y0+h*a))
```

```
y_Euler = 1.2500
y_Improved = 1.2700
```

(b) Local truncation error is proportional to h^2 for Euler method. $\therefore e_1 = kh^2$, k some constant.

Assume y_Improved is the "true" value, or very close to it. Then $e_1 \approx 1.27 - 1.25 = 0.02$

For error $\epsilon = 0.0025$, $\frac{0.0025}{0.02} = \frac{K h^2}{K(0.1)^2}$,

$$h = 0.1\sqrt{\frac{0.0025}{0.02}} = 0.03536$$

∴ Step size, for Euler method, should be

$h \approx 0.035$ for $\epsilon = 0.0025$

17.

Using MATLAB,

```
clear
fn = @(t,y) 5*t - 3*sqrt(y);  % f(t,y)
t0 = 0;    % initial conditions
y0 = 2;
h = 0.1;

a = fn(t0,y0);
y_Euler = y0 + h*a
y_Improved = y0 + h/2*(a + fn(t0+h, y0+h*a))

e_max = 0.0025;  % specified max error
error = abs(y_Improved - y_Euler);
NewStep = h*sqrt(e_max/error)
```

```
y_Euler = 1.5757
y_Improved = 1.6246

NewStep = 0.0226
```

∴ Euler: $y_1 = 1.5757$ Improved: 1.6246

Step size for Euler $\epsilon \leq 0.0025$: $h \approx 0.0226$

18.

Using MATLAB,

```
clear
fn = @(t,y) sqrt(t + y);  % f(t,y)
t0 = 0;    % initial conditions
y0 = 3;
h = 0.1;

a = fn(t0,y0);
y_Euler = y0 + h*a
y_Improved = y0 + h/2*(a + fn(t0+h, y0+h*a))

e_max = 0.0025;  % specified max error
error = abs(y_Improved - y_Euler);
NewStep = h*sqrt(e_max/error)
```

```
y_Euler = 3.1732
y_Improved = 3.1771

NewStep = 0.0805
```

$\therefore$ Euler: $y_1 = \underline{3.1732}$

Improved: $y_1 = \underline{3.1771}$

Step size for Euler $\epsilon \leq 0.0025$: $h \approx \underline{0.0805}$

19.

Using MATLAB,

```
clear
fn = @(t,y) (y^2 + 2*t*y)/(3 + t^2);  % f(t,y)
t0 = 0;    % initial conditions
y0 = 0.5;
h = 0.1;

a = fn(t0,y0);
y_Euler = y0 + h*a
y_Improved = y0 + h/2*(a + fn(t0+h, y0+h*a))

e_max = 0.0025;  % specified max error
error = abs(y_Improved - y_Euler);
NewStep = h*sqrt(e_max/error)
```

```
y_Euler = 0.5083
y_Improved = 0.5101

NewStep = 0.1174
```

$\therefore$ Euler: $y_1 = 0.5083$

Improved: $y_1 = 0.5101$

Step size for Euler $\epsilon \leq 0.0025$: $h \approx 0.1174$

20.

The $\frac{1}{2}h$ seems a technical problem at first, but can be handled via substitution. Much of the proof is dependent on the Taylor expansion for 2 variables, and the $\frac{1}{2}h$ factor can be handled there.

From #12, $$F(a+h, b+k) = F(a,b) + F_t(a,b)h + F_y(a,b)k + \frac{1}{2!}(h^2 F_{tt} + 2hkF_{ty} + k^2 F_{yy})\Big|_{t=\xi, y=\eta},$$

Let "a" $= t_n$, "h" $= \frac{1}{2}h$, "b" $= y_n$, "k" $= \frac{1}{2}h f(t_n, y_n)$

$\therefore f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}h f(t_n, y_n)) =$

$= \phi''(t_n)(\frac{1}{2}h)$ shown below

$$f(t_n, y_n) + f_t(t_n, y_n)(\tfrac{1}{2}h) + f_y(t_n, y_n)\tfrac{1}{2}h f(t_n, y_n) + \frac{1}{2!}\left[\frac{h^2}{4}f_{tt} + 2\left(\frac{1}{2}\right)\left(\tfrac{1}{2}h f(t_n, y_n)\right)f_{ty} + \tfrac{1}{4}h^2 f^2(t_n, y_n) f_{yy}\right]\Bigg|_{\substack{t=\xi \\ y=\eta}}$$

where $t_n \le \epsilon \le t_n + \frac{1}{2}h$, $y_n \le \eta \le y_n + \frac{1}{2}h f(t_n, y_n)$

Now bring in $\phi''(t_n)$ to deal with f_t and f_y above.

Since $\phi'(t) = f(t, \phi(t))$, $\phi'' = f_t(t, \phi) + f_y(t, \phi)\phi'$

$$\therefore \phi''(t_n) = f_t(t_n, \phi(t_n)) + f_y(t_n, \phi(t_n))\phi'(t_n)$$

$$= f_t(t_n, y_n) + f_y(t_n, y_n) f(t_n, y_n)$$

$$\therefore \phi''(t_n)\left(\tfrac{1}{2}h\right) = f_t(t_n, y_n)\left(\tfrac{1}{2}h\right) + f_y(t_n, y_n)\left(\tfrac{1}{2}h\right) f(t_n, y_n)$$

Using this $\phi''(t_n)\left(\frac{1}{2}h\right)$ in the above Taylor expansion,

$$f\left(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}h f(t_n, y_n)\right) = f(t_n, y_n) + \phi''(t_n)\left(\tfrac{1}{2}h\right)$$

$$+ \frac{1}{2!}\left[\frac{h^2}{4} f_{tt} + \frac{h^2}{2} f(t_n, y_n) f_{ty} + \frac{1}{4}h^2 f^2(t_n, y_n) f_{yy}\right]$$

where f_{tt}, f_{ty}, f_{yy} are evaluated at (ϵ, η)

$$\therefore (2h) f\left(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}h f(t_n, y_n)\right) = (2h) f(t_n, y_n) + \phi''(t_n) h^2$$

$$+ \frac{h^3}{4}\left[f_{tt} + 4 f(t_n, y_n) f_{ty} + f^2(t_n, y_n) f_{yy}\right] \qquad [1]$$

From a Taylor expansion of $\phi(t)$, $t_n \le \bar{t}_n \le t_{n+1}$

$$\phi(t_{n+1}) = \phi(t_n) + \phi'(t_n) h + \phi''(t_n)\frac{h^2}{2!} + \phi'''(\bar{t}_n)\frac{h^3}{3!} \qquad [2]$$

Noting $y_n = \phi(t_n)$, $\phi'(t_n) = f(t_n, \phi(t_n)) = f(t_n, y_n)$

and subtracting the modified formula from [2],

$$e_{n+1} = \phi(t_{n+1}) - y_{n+1}$$

$$= \underbrace{\phi(t_n) - y_n}_{=0} + \phi'(t_n)h - \underbrace{f(t_n, y_n)h}_{=\frac{2f(t_n,y_n)h}{2}} + \phi''(t_n)\frac{h^2}{2!} - hf\left(t_n + \tfrac{1}{2}h,\ y_n + \tfrac{1}{2}hf(t_n, y_n)\right)$$

$$+ \phi'''(\bar{t}_n)\frac{h^3}{3!}$$

$$= \frac{\phi''(t_n)h^2 + 2f(t_n, y_n)h - (2h)f\left(t_n + \frac{1}{2}h,\ y_n + \frac{1}{2}hf(t_n, y_n)\right)}{2}$$

$$+ \phi'''(\bar{t}_n)\frac{h^3}{3!}$$

Now substituting [1] into the above,

$$e_{n+1} = \frac{\phi''(t_n)h^2 + (2h)f(t_n, y_n) - \overbrace{\left[(2h)f(t_n, y_n) + \phi''(t_n)h^2\right]}^{=0}}{2}$$

$$- \frac{1}{2}\cdot\frac{h^3}{4}\left[f_{tt} + 4f(t_n, y_n)f_{ty} + f^2(t_n, y_n)f_{yy}\right]$$

$$+ \phi'''(\bar{t}_n)\frac{h^3}{3!}$$

$$\therefore e_{n+1} = h^3 \left[\frac{\phi'''(\bar{t}_n)}{3!} - \frac{1}{8} f_{tt} - \frac{1}{2} f(t_n, y_n) f_{ty} - \frac{1}{8} f^2(t_n, y_n) f_{yy} \right]$$

$\therefore e_{n+1}$ is proportional to h^3

21.

To compare with Problem #2, create table showing both Improved and Modified Euler methods. Using MATLAB, code on next page.

```
ans = 'Table for h = 0.0500'
```

T = 5×3 table

	t	Improved_Euler	Modified_Euler
1	0	1.000000000000000	1.000000000000000
2	0.10000000…	1.195123437500000	1.195123437500000
3	0.20000000…	1.381198406638184	1.381198406638184
4	0.30000000…	1.559085628829237	1.559085628829237
5	0.40000000…	1.729563950708151	1.729563950708151

The results of the Modified Euler method are identical to the Improved Euler method, to 15 decimals.

```
clear
%Change TableTimes[], y(0) the initial value, @ function

TableTimes = [0, 0.1, 0.2, 0.3, 0.4];  % times for y values in table
y0 = 1;
h = 0.05;
% create anonymous function for easy coding below
fn = @(t,y) 3 + t - y;   % fn = dy/dt = f(t,y)

NumRows = length(TableTimes);
%initialize table array: columns for times + ImpEuler + ModifiedEuler
V = zeros(NumRows, 3);
% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

t = TableTimes(1):h:TableTimes(end);
% pre-allocate memory for y values
y_imp = zeros(length(t),1);
y_mod = zeros(length(t),1);
% initialize 1st row entry to y(0)
y_imp(1) = y0;
y_mod(1) = y0;

Row = 1;   %start at row 1 for each column
V(Row,2) = y_imp(1);  % populate 1st table row entry
V(Row,3) = y_mod(1);
for i = 2:length(t)
        a = fn(t(i-1), y_imp(i-1));  % prelim value for below
    y_imp(i) = y_imp(i-1)+ h/2*(a + fn(t(i-1)+ h, y_imp(i-1) + h*a));
        a = fn(t(i-1), y_mod(i-1));  % prelim value for below
    y_mod(i) = y_mod(i-1) + h*fn(t(i-1)+ h/2, y_mod(i-1) + h/2*a);

    % display table results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,2) = y_imp(i);  % populate table array
        V(Row,3) = y_mod(i);
    end
end

format long
T = array2table(V);
% label the table columms and display the table
sprintf('Table for h = %.4f',h)
T.Properties.VariableNames = {'t', 'Improved_Euler', 'Modified_Euler'}
```

22.

Use MATLAB to compare Modified with Improved Euler methods.

```
ans = 'Table for h = 0.0500'

T =  5×3 table
```

	t	Improved_Euler	Modified_Euler
1	0	2.000000000000000	2.000000000000000
2	0.10000000...	1.622831707983092	1.622680180372303
3	0.20000000...	1.334604231198154	1.334347614870263
4	0.30000000...	1.128203984456552	1.127894096427027
5	0.40000000...	0.995445044510686	0.995129859049131

```
clear
%Change TableTimes[], y(0) the initial value, @ function

TableTimes = [0, 0.1, 0.2, 0.3, 0.4];  % times for y values in table
y0 = 2;
h = 0.05;
% create anonymous function for easy coding below
fn = @(t,y) 5*t - 3*sqrt(y);   % fn = dy/dt = f(t,y)

NumRows = length(TableTimes);
%initialize table array: columns for times + ImpEuler + ModifiedEuler
V = zeros(NumRows, 3);
% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

t = TableTimes(1):h:TableTimes(end);
% pre-allocate memory for y values
y_imp = zeros(length(t),1);
y_mod = zeros(length(t),1);
% initialize 1st row entry to y(0)
y_imp(1) = y0;
y_mod(1) = y0;

Row = 1;   %start at row 1 for each column
V(Row,2) = y_imp(1);  % populate 1st table row entry
V(Row,3) = y_mod(1);
for i = 2:length(t)
        a = fn(t(i-1), y_imp(i-1));  % prelim value for below
    y_imp(i) = y_imp(i-1)+ h/2*(a + fn(t(i-1)+ h, y_imp(i-1) + h*a));
        a = fn(t(i-1), y_mod(i-1));  % prelim value for below
    y_mod(i) = y_mod(i-1) + h*fn(t(i-1)+ h/2, y_mod(i-1) + h/2*a);

    % display table results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,2) = y_imp(i);  % populate table array
        V(Row,3) = y_mod(i);
    end
end

format long
T = array2table(V);
% label the table columms and display the table
sprintf('Table for h = %.4f',h)
T.Properties.VariableNames = {'t', 'Improved_Euler', 'Modified_Euler'}
```

23.

Use MATLAB to compare Modified with Improved Euler methods.

Results are identical to 15 decimals.

```
ans = 'Table for h = 0.0500'

T =  5×3 table
```

	t	Improved_Euler	Modified_Euler
1	0	1.000000000000000	1.000000000000000
2	0.10000000...	1.205256250000000	1.205256250000000
3	0.20000000...	1.422725512656250	1.422725512656250
4	0.30000000...	1.655107169091098	1.655107169091098
5	0.40000000...	1.905697231139458	1.905697231139458

```
clear
%Change TableTimes[], y(0) the initial value, @ function

TableTimes = [0, 0.1, 0.2, 0.3, 0.4];  % times for y values in table
y0 = 1;
h = 0.05;
% create anonymous function for easy coding below
fn = @(t,y) 2*y - 3*t;   % fn = dy/dt = f(t,y)

NumRows = length(TableTimes);
%initialize table array: columns for times + ImpEuler + ModifiedEuler
V = zeros(NumRows, 3);
% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

t = TableTimes(1):h:TableTimes(end);
% pre-allocate memory for y values
y_imp = zeros(length(t),1);
y_mod = zeros(length(t),1);
% initialize 1st row entry to y(0)
y_imp(1) = y0;
y_mod(1) = y0;

Row = 1;   %start at row 1 for each column
V(Row,2) = y_imp(1);  % populate 1st table row entry
V(Row,3) = y_mod(1);
for i = 2:length(t)
        a = fn(t(i-1), y_imp(i-1));  % prelim value for below
    y_imp(i) = y_imp(i-1)+ h/2*(a + fn(t(i-1)+ h, y_imp(i-1) + h*a));
        a = fn(t(i-1), y_mod(i-1));  % prelim value for below
    y_mod(i) = y_mod(i-1) + h*fn(t(i-1)+ h/2, y_mod(i-1) + h/2*a);

    % display table results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,2) = y_imp(i);  % populate table array
        V(Row,3) = y_mod(i);
    end
end

format long
T = array2table(V);
% label the table columms and display the table
sprintf('Table for h = %.4f',h)
T.Properties.VariableNames = {'t', 'Improved_Euler', 'Modified_Euler'}
```

24.

Use MATLAB to compare Modified with Improved Euler methods.

```
ans = 'Table for h = 0.0500'

T =  5×3 table
```

	t	Improved_Euler	Modified_Euler
1	0	1.000000000000000	1.000000000000000
2	0.10000000…	1.104828922848498	1.104853513564417
3	0.20000000…	1.218823869462594	1.218860436846571
4	0.30000000…	1.341458888912524	1.341492834649174
5	0.40000000…	1.472626670489802	1.472643426810588

```
clear
%Change TableTimes[], y(0) the initial value, @ function

TableTimes = [0, 0.1, 0.2, 0.3, 0.4];  % times for y values in table
y0 = 1;
h = 0.05;
% create anonymous function for easy coding below
fn = @(t,y) 2*t + exp(-t*y);   % fn = dy/dt = f(t,y)

NumRows = length(TableTimes);
%initialize table array: columns for times + ImpEuler + ModifiedEuler
V = zeros(NumRows, 3);
% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

t = TableTimes(1):h:TableTimes(end);
% pre-allocate memory for y values
y_imp = zeros(length(t),1);
y_mod = zeros(length(t),1);
% initialize 1st row entry to y(0)
y_imp(1) = y0;
y_mod(1) = y0;

Row = 1;   %start at row 1 for each column
V(Row,2) = y_imp(1);  % populate 1st table row entry
V(Row,3) = y_mod(1);
for i = 2:length(t)
        a = fn(t(i-1), y_imp(i-1));  % prelim value for below
    y_imp(i) = y_imp(i-1)+ h/2*(a + fn(t(i-1)+ h, y_imp(i-1) + h*a));
        a = fn(t(i-1), y_mod(i-1));  % prelim value for below
    y_mod(i) = y_mod(i-1) + h*fn(t(i-1)+ h/2, y_mod(i-1) + h/2*a);

    % display table results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,2) = y_imp(i);  % populate table array
        V(Row,3) = y_mod(i);
    end
end

format long
T = array2table(V);
% label the table columms and display the table
sprintf('Table for h = %.4f',h)
T.Properties.VariableNames = {'t', 'Improved_Euler', 'Modified_Euler'}
```

25.

If f is linear in both t and y, then

$$f_t(t,y) = a, \text{ a constant} \qquad [1]$$

and $$f_y(t,y) = b, \text{ a constant} \qquad [2]$$

$$\therefore \; f_{tt} = 0, \; f_{ty} = 0, \; f_{yy} = 0 \qquad [3]$$

As shown in #20, the Taylor expansion of

$$f\left(t_n + \tfrac{1}{2}h, \; y_n + \tfrac{1}{2}h f(t_n, y_n)\right) =$$

$$f(t_n, y_n) + f_t(t_n, y_n)\left(\tfrac{1}{2}h\right) + f_y(t_n, y_n)\tfrac{1}{2}h f(t_n, y_n)$$

$$+ \frac{1}{2!}\left[\frac{h^2}{4}f_{tt} + 2\left(\frac{h}{2}\right)\left(\tfrac{1}{2}h f(t_n, y_n)\right)f_{ty} + \tfrac{1}{4}h^2 f^2(t_n, y_n) f_{yy}\right]\Bigg|_{\substack{t=c \\ y=n}}$$

where $t_n \le c \le t_n + \frac{1}{2}h$, $y_n \le n \le y_n + \frac{1}{2}h f(t_n, y_n)$

$$\therefore \; f\left(t_n + \tfrac{1}{2}h, \; y_n + \tfrac{1}{2}h f(t_n, y_n)\right) =$$

$$f(t_n, y_n) + a\left(\tfrac{1}{2}h\right) + b\left(\tfrac{1}{2}h\right) f(t_n, y_n)$$

$$= f(t_n, y_n) + \left(\tfrac{1}{2}h\right)\left[a + b f(t_n, y_n)\right]$$

$\therefore$ Modified Euler formula becomes:

$$y_{n+1} = y_n + h f(t_n, y_n) + \tfrac{1}{2}h^2\left[a + b f(t_n, y_n)\right] \qquad [4]$$

For the Improved Euler method, #12(6) above showed: $f(t_n+h, y_n + hf(t_n,y_n)) =$

$$f(t_n,y_n) + f_t(t_n,y_n)h + f_y(t_n,y_n)hf(t_n,y_n)$$
$$+ \frac{1}{2!}\left[h^2 f_{tt} + 2h^2 f(t_n,y_n) f_{ty} + h^2 f^2(t_n,y_n) f_{yy}\right]\Big|_{t=c,\,y=\eta}$$

where $t_n \le c \le t_n + h$, $y_n \le \eta \le y_n + hf(t_n,y_n)$

Using [1], [2], and [3]

$f(t_n+h, y_n + hf(t_n,y_n)) =$

$$f(t_n,y_n) + ah + bhf(t_n,y_n)$$
$$= f(t_n,y_n) + h[a + bf(t_n,y_n)]$$

$\therefore$ Improved formula becomes:

$$y_{n+1} = y_n + \frac{h}{2}\left[f(t_n,y_n) + f(t_n+h,\, y_n + hf(t_n,y_n))\right]$$
$$= y_n + \frac{h}{2}\left[f(t_n,y_n) + f(t_n,y_n) + h[a + bf(t_n,y_n)]\right]$$

$$\therefore y_{n+1} = y_n + hf(t_n,y_n) + \frac{h^2}{2}\left[a + bf(t_n,y_n)\right] \quad [5]$$

Since [4] = [5], the modified and improved Euler methods are identical.

8.3 The Runge-Kutta Method

Note Title 4/28/2020

1.

MATLAB code, next page, is used to fill out the table. Use is made of an anonymous function "slope" defined as @(t,y) 1-t+4y, used for both the improved Euler method and Runge-Kutta method. R2020a is used, which allows strings to be used as table column headers in the T.Properties.VariableNames statement. Note the first step size, h=0.2, is larger than the initial table times (0.1, 0.2, 0.3, 0.4, 0.5) increments.

T = 9×6 table

	t	ImpEuler h = 0.025	R-K h = 0.2	R-K h = 0.1	R-K h = 0.05	Exact
1	0	1	1	1	1	1
2	0.1000	1.6079	0	1.6089	1.6090	1.6090
3	0.2000	2.5021	2.5016	2.5050	2.5053	2.5053
4	0.3000	3.8228	0	3.8294	3.8301	3.8301
5	0.4000	5.7797	5.7776	5.7928	5.7941	5.7942
6	0.5000	8.6849	0	8.7093	8.7118	8.7120
7	1.0000	64.4979	64.4416	64.8581	64.8949	64.8978
8	1.5000	474.8340	0	478.8193	479.2267	479.2592
9	2.0000	3.4967e+03	3.4906e+03	3.5359e+03	3.5399e+03	3.5402e+03

```
clear
% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 1.5, 2.0];
NumRows = length(TableTimes);
%initialize table array, 6 columns: time + ImpEuler + 3 R-K + Exact
V = zeros(NumRows, 6);

% Populate table times (Col=1) and exact values for solution (Col=6)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,6) = -3/16 + (1/4)*t + (19/16)*exp(4*t);
end

slope = @(t,y) 1 - t + 4*y;   % slope = dy/dt = f(t,y)

Col = 2;  % start at Col = 2 for Improved Euler Method
h = 0.025;
t = TableTimes(1):h:TableTimes(end);
y = zeros(length(t),1);  % pre-allocate memory for y
y(1) = 1;                % MATLAB not zero based
Row = 1;   %start at row 1 for each column
V(Row,Col) = y(1);
for i = 2:length(t)
    a = slope(t(i-1), y(i-1));
    b = slope(t(i), y(i-1) + h*a);
    y(i) = y(i-1) + 0.5*(a+b)*h;  % new estimate of y(t)
    % display table results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end
Col = 3;  % start of Runge-Kutta Method
for h = [0.2, 0.1, 0.05]  % step sizes for R-K
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory for y
    y(1) = 1;                % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    h2 = h/2;  % shorten calcs below
    for i = 2:length(t)
        k1 = slope(t(i-1), y(i-1));
        k2 = slope(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = slope(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = slope(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        % display table results only at specified times
        for Row = 1:NumRows  % find the correct Row
            if abs(t(i) - TableTimes(Row)) < 0.00001
                V(Row,Col) = y(i);
            end
        end
    end
    Col = Col + 1;  % go to next table column
end

format
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'ImpEuler h = 0.025', 'R-K h = 0.2', ...
                              'R-K h = 0.1', 'R-K h = 0.05', 'Exact'}
```

2.

Exact solution: $\frac{d}{dt}(ye^t) = 3e^t + te^t$, $ye^t = 3e^t + te^t - e^t + c$

$\therefore y = 2 + t + ce^{-t}$, $y(0) = 1 \Rightarrow c = -1$. $\therefore y(t) = 2 + t - e^{-t}$

Compare with improved Euler method at $h = 0.1, 0.05$

Use MATLAB (R2020a), code on next page.

T = 5×6 table

	t	ImpEuler h = 0.1	ImpEuler h = 0.05	R-K h = 0.1	R-K h = 0.05	Exact
1	0	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.1000	1.1950	1.1951	1.1952	1.1952	1.1952
3	0.2000	1.3810	1.3812	1.3813	1.3813	1.3813
4	0.3000	1.5588	1.5591	1.5592	1.5592	1.5592
5	0.4000	1.7292	1.7296	1.7297	1.7297	1.7297

```
clear
% Change slope = @(t,y), y0, exact solution

y0 = 1;
slope = @(t,y) 3 + t - y;   % slope = dy/dt = f(t,y)

% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4];
StepSizes = [0.1, 0.05];
NumRows = length(TableTimes);
%initialize table array, 6 columns: time +  2 ImpEuler + 2 R-K + Exact
V = zeros(NumRows, 6);

% Populate table times (Col=1) and exact values for solution (Col=6)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,6) = 2 + t - exp(-t);
end

Col = 2;  % start at Col = 2 for Improved Euler Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(NumRows,1);  % pre-allocate memory for y
    y(1) = y0;             % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    for i = 2:length(t)
        a = slope(t(i-1), y(i-1));
        b = slope(t(i), y(i-1) + h*a);
        y(i) = y(i-1) + 0.5*(a+b)*h;  % new estimate of y(t)
        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
    Col = Col + 1;  % go to next table column
end

Col = 4;  % start of Runge-Kutta Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(NumRows,1);  % pre-allocate memory for y
    y(1) = y0;             % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    h2 = h/2;  % shorten calcs below
    for i = 2:length(t)
        k1 = slope(t(i-1), y(i-1));
        k2 = slope(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = slope(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = slope(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        % display table results only at specified times
        for Row = 1:NumRows  % find the correct Row
            if abs(t(i) - TableTimes(Row)) < 0.00001
                V(Row,Col) = y(i);
            end
        end
    end
    Col = Col + 1;  % go to next table column
end

format
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'ImpEuler h = 0.1', 'ImpEuler h = 0.05', ...
                              'R-K h = 0.1', 'R-K h = 0.05', 'Exact'}
```

3.

Using MATLAB,

Using improved Euler method for comparison.

```
clear
% Change slope = @(t,y), y0
y0 = 2;
slope = @(t,y) 5*t - 3*sqrt(y);   % slope = dy/dt = f(t,y)

% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4];
StepSizes = [0.1, 0.05];
NumRows = length(TableTimes);
% initialize table array, 5 columns:
%   time +  2 ImpEuler + 2 R-K
V = zeros(NumRows, 5);

% Populate table times (Col=1)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
end

Col = 2;  % start at Col = 2 for Improved Euler Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory for y
    y(1) = y0;               % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    for i = 2:length(t)
        a = slope(t(i-1), y(i-1));
        b = slope(t(i), y(i-1) + h*a);
        y(i) = y(i-1) + 0.5*(a+b)*h;  % new estimate of y(t)
        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
    Col = Col + 1;  % go to next table column
end

Col = 4;  % start of Runge-Kutta Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory for y
    y(1) = y0;               % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    h2 = h/2;  % shorten calcs below
    for i = 2:length(t)
        k1 = slope(t(i-1), y(i-1));
        k2 = slope(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = slope(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = slope(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        % display table results only at specified times
        for Row = 1:NumRows  % find the correct Row
            if abs(t(i) - TableTimes(Row)) < 0.00001
                V(Row,Col) = y(i);
            end
        end
    end
    Col = Col + 1;  % go to next table column
end

format
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'ImpEuler h = 0.1', ...
                              'ImpEuler h = 0.05', ...
                              'R-K h = 0.1', ...
                              'R-K h = 0.05'}
```

T = 5×5 table

	t	ImpEuler h = 0.1	ImpEuler h = 0.05	R-K h = 0.1	R-K h = 0.05
1	0	2.0000	2.0000	2.0000	2.0000
2	0.1000	1.6246	1.6228	1.6223	1.6223
3	0.2000	1.3379	1.3346	1.3336	1.3336
4	0.3000	1.1327	1.1282	1.1269	1.1268
5	0.4000	1.0008	0.9954	0.9938	0.9938

4.

Using MATLAB,

Using improved Euler method for comparison.

```
clear
% Change slope = @(t,y), y0
y0 = 1;
slope = @(t,y) 2*t + exp(-t*y);   % slope = dy/dt = f(t,y)

% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4];
StepSizes = [0.1, 0.05];
NumRows = length(TableTimes);
% initialize table array, 5 columns:
%   time +  2 ImpEuler + 2 R-K
V = zeros(NumRows, 5);

% Populate table times (Col=1)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
end

Col = 2;  % start at Col = 2 for Improved Euler Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory for y
    y(1) = y0;               % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    for i = 2:length(t)
        a = slope(t(i-1), y(i-1));
        b = slope(t(i), y(i-1) + h*a);
        y(i) = y(i-1) + 0.5*(a+b)*h;  % new estimate of y(t)
        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
    Col = Col + 1;  % go to next table column
end

Col = 4;  % start of Runge-Kutta Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory for y
    y(1) = y0;               % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    h2 = h/2;  % shorten calcs below
    for i = 2:length(t)
        k1 = slope(t(i-1), y(i-1));
        k2 = slope(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = slope(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = slope(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        % display table results only at specified times
        for Row = 1:NumRows  % find the correct Row
            if abs(t(i) - TableTimes(Row)) < 0.00001
                V(Row,Col) = y(i);
            end
        end
    end
    Col = Col + 1;  % go to next table column
end

format
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'ImpEuler h = 0.1', ...
                              'ImpEuler h = 0.05', ...
                              'R-K h = 0.1', ...
                              'R-K h = 0.05'}
```

T = 5×5 table

	t	ImpEuler h = 0.1	ImpEuler h = 0.05	R-K h = 0.1	R-K h = 0.05
1	0	1.0000	1.0000	1.0000	1.0000
2	0.1000	1.1048	1.1048	1.1048	1.1048
3	0.2000	1.2188	1.2188	1.2188	1.2188
4	0.3000	1.3414	1.3415	1.3415	1.3415
5	0.4000	1.4727	1.4726	1.4726	1.4726

5.

Using MATLAB,

Using improved Euler method for comparison.

```
clear
% Change slope = @(t,y), y0
y0 = 0.5;
slope = @(t,y) (y^2 + 2*t*y)/(3+t^2);   % slope = dy/dt = f(t,y)

% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4];
StepSizes = [0.1, 0.05];
NumRows = length(TableTimes);
% initialize table array, 5 columns:
%   time +  2 ImpEuler + 2 R-K
V = zeros(NumRows, 5);

% Populate table times (Col=1)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
end

Col = 2;  % start at Col = 2 for Improved Euler Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory for y
    y(1) = y0;               % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    for i = 2:length(t)
        a = slope(t(i-1), y(i-1));
        b = slope(t(i), y(i-1) + h*a);
        y(i) = y(i-1) + 0.5*(a+b)*h;  % new estimate of y(t)
        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
    Col = Col + 1;  % go to next table column
end

Col = 4;  % start of Runge-Kutta Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory for y
    y(1) = y0;               % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    h2 = h/2;  % shorten calcs below
    for i = 2:length(t)
        k1 = slope(t(i-1), y(i-1));
        k2 = slope(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = slope(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = slope(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        % display table results only at specified times
        for Row = 1:NumRows  % find the correct Row
            if abs(t(i) - TableTimes(Row)) < 0.00001
                V(Row,Col) = y(i);
            end
        end
    end
    Col = Col + 1;  % go to next table column
end

format
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'ImpEuler h = 0.1', ...
                              'ImpEuler h = 0.05', ...
                              'R-K h = 0.1', ...
                              'R-K h = 0.05'}
```

T = 5×5 table

	t	ImpEuler h = 0.1	ImpEuler h = 0.05	R-K h = 0.1	R-K h = 0.05
1	0	0.5000	0.5000	0.5000	0.5000
2	0.1000	0.5101	0.5102	0.5102	0.5102
3	0.2000	0.5241	0.5241	0.5241	0.5241
4	0.3000	0.5420	0.5421	0.5421	0.5421
5	0.4000	0.5641	0.5643	0.5643	0.5643

6.

Using MATLAB, Using improved Euler method for comparison.

```
clear
% Change slope = @(t,y), y0
y0 = -1;
slope = @(t,y) (t^2 - y^2)*sin(y);   % slope = dy/dt = f(t,y)

% times to display the y values in the table
TableTimes = [0, 0.1, 0.2, 0.3, 0.4];
StepSizes = [0.1, 0.05];
NumRows = length(TableTimes);
% initialize table array, 5 columns:
%   time +  2 ImpEuler + 2 R-K
V = zeros(NumRows, 5);

% Populate table times (Col=1)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
end

Col = 2;  % start at Col = 2 for Improved Euler Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory for y
    y(1) = y0;               % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    for i = 2:length(t)
        a = slope(t(i-1), y(i-1));
        b = slope(t(i), y(i-1) + h*a);
        y(i) = y(i-1) + 0.5*(a+b)*h;  % new estimate of y(t)
        % display table results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
    Col = Col + 1;  % go to next table column
end

Col = 4;  % start of Runge-Kutta Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory for y
    y(1) = y0;               % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    h2 = h/2;  % shorten calcs below
    for i = 2:length(t)
        k1 = slope(t(i-1), y(i-1));
        k2 = slope(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = slope(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = slope(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        % display table results only at specified times
        for Row = 1:NumRows  % find the correct Row
            if abs(t(i) - TableTimes(Row)) < 0.00001
                V(Row,Col) = y(i);
            end
        end
    end
    Col = Col + 1;  % go to next table column
end

format
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'ImpEuler h = 0.1', ...
                              'ImpEuler h = 0.05', ...
                              'R-K h = 0.1', ...
                              'R-K h = 0.05'}
```

T = 5×5 table

	t	ImpEuler h = 0.1	ImpEuler h = 0.05	R-K h = 0.1	R-K h = 0.05
1	0	-1.0000	-1.0000	-1.0000	-1.0000
2	0.1000	-0.9251	-0.9246	-0.9245	-0.9245
3	0.2000	-0.8650	-0.8643	-0.8641	-0.8641
4	0.3000	-0.8175	-0.8166	-0.8164	-0.8164
5	0.4000	-0.7809	-0.7800	-0.7797	-0.7797

7.

Exact solution: $\frac{d}{dt}(ye^{-2t}) = 0.5e^{-2t} - te^{-2t}$

$$ye^{-2t} = -\frac{1}{4}e^{-2t} + \frac{1}{2}te^{-2t} + \frac{1}{4}e^{-2t} + C$$

$$y = \frac{1}{2}t + ce^{2t}, \quad y(0) = 1 \Rightarrow y(t) = \frac{1}{2}t + e^{2t}$$

Use MATLAB, make use anonymous function

```
fn = @(t,y) 0.5 - t + 2y
```

Compare with improved Euler method

T = 5×6 table

	t	ImpEuler h = 0.1	ImpEuler h = 0.05	R-K h = 0.1	R-K h = 0.05	Exact
1	0	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.5000	2.9527	2.9641	2.9683	2.9683	2.9683
3	1.0000	7.8046	7.8662	7.8889	7.8890	7.8891
4	1.5000	20.4923	20.7426	20.8349	20.8355	20.8355
5	2.0000	54.3576	55.2614	55.5957	55.5980	55.5982

Code on next page.

```
clear
% Change slope = @(t,y), y(0), ? exact solution
y0 = 1;
fn = @(t,y) 0.5 - t + 2*y;   % fn = dy/dt = f(t,y)

% times to display the y values in the table
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
StepSizes = [0.1, 0.05];
NumRows = length(TableTimes);
% initialize table array, 6 columns:
%   time +  2 ImpEuler + 2 R-K + exact
V = zeros(NumRows, 6);

% Populate table times (Col=1), exact (Col=6)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,6) = 0.5*t + exp(2*t);
end

% start at Col = 2 for Improved Euler Method
Col = 2;
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory for y
    y(1) = y0;               % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    for i = 2:length(t)
        a = fn(t(i-1), y(i-1));
        b = fn(t(i), y(i-1) + h*a);
        % new estimate of y(t)
        y(i) = y(i-1) + 0.5*(a+b)*h;
        % display results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
    Col = Col + 1;  % go to next table column
end

Col = 4;  % start of Runge-Kutta Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory for y
    y(1) = y0;               % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    h2 = h/2;  % shorten calcs below
    for i = 2:length(t)
        k1 = fn(t(i-1), y(i-1));
        k2 = fn(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = fn(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = fn(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        % display results only at specified times
        for Row = 1:NumRows  % find the correct Row
            if abs(t(i) - TableTimes(Row)) < 0.00001
                V(Row,Col) = y(i);
            end
        end
    end
    Col = Col + 1;  % go to next table column
end

format
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', ...
                              'ImpEuler h = 0.1', ...
                              'ImpEuler h = 0.05', ...
                              'R-K h = 0.1', ...
                              'R-K h = 0.05', ...
                              'Exact'}
```

8.

Using MATLAB,

T = 5×5 table

	t	ImpEuler h = 0.1	ImpEuler h = 0.05	R-K h = 0.1	R-K h = 0.05
1	0	2.0000	2.0000	2.0000	2.0000
2	0.5000	0.9334	0.9275	0.9257	0.9257
3	1.0000	1.2927	1.2869	1.2852	1.2851
4	1.5000	2.4152	2.4102	2.4086	2.4086
5	2.0000	4.1096	4.1050	4.1035	4.1035

```
clear
% Change slope = @(t,y), y(0), ? exact solution
y0 = 2;
fn = @(t,y) 5*t - 3*sqrt(y);   % fn = dy/dt = f(t,y)

% times to display the y values in the table
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
StepSizes = [0.1, 0.05];
NumRows = length(TableTimes);
% initialize table array, 5 columns:
%   time +  2 ImpEuler + 2 R-K
V = zeros(NumRows, 5);

% Populate table times (Col=1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% start at Col = 2 for Improved Euler Method
Col = 2;
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1); % pre-allocate memory
    y(1) = y0;              % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    for i = 2:length(t)
        a = fn(t(i-1), y(i-1));
        b = fn(t(i), y(i-1) + h*a);
        % new estimate of y(t)
        y(i) = y(i-1) + 0.5*(a+b)*h;
        % display results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
    Col = Col + 1;  % go to next table column
end

Col = 4;  % start of Runge-Kutta Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1); % pre-allocate memory
    y(1) = y0;              % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    h2 = h/2;  % shorten calcs below
    for i = 2:length(t)
        k1 = fn(t(i-1), y(i-1));
        k2 = fn(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = fn(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = fn(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        % display results only at specified times
        for Row = 1:NumRows  % find the correct Row
            if abs(t(i) - TableTimes(Row)) < 0.00001
                V(Row,Col) = y(i);
            end
        end
    end
    Col = Col + 1;  % go to next table column
end

format
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', ...
                              'ImpEuler h = 0.1', ...
                              'ImpEuler h = 0.05', ...
                              'R-K h = 0.1', ...
                              'R-K h = 0.05'}
```

9.

Using MATLAB,

T = 5×5 table

	t	ImpEuler h = 0.1	ImpEuler h = 0.05	R-K h = 0.1	R-K h = 0.05
1	0	3.0000	3.0000	3.0000	3.0000
2	0.5000	3.9619	3.9621	3.9622	3.9622
3	1.0000	5.1083	5.1088	5.1089	5.1089
4	1.5000	6.4306	6.4312	6.4314	6.4314
5	2.0000	7.9223	7.9231	7.9234	7.9234

```
clear
% Change slope = @(t,y), y(0), ? exact solution
y0 = 3;
fn = @(t,y) sqrt(t+y);   % fn = dy/dt = f(t,y)

% times to display the y values in the table
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
StepSizes = [0.1, 0.05];
NumRows = length(TableTimes);
% initialize table array, 5 columns:
%   time +  2 ImpEuler + 2 R-K
V = zeros(NumRows, 5);

% Populate table times (Col=1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% start at Col = 2 for Improved Euler Method
Col = 2;
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1); % pre-allocate memory
    y(1) = y0;              % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    for i = 2:length(t)
        a = fn(t(i-1), y(i-1));
        b = fn(t(i), y(i-1) + h*a);
        % new estimate of y(t)
        y(i) = y(i-1) + 0.5*(a+b)*h;
        % display results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
    Col = Col + 1;  % go to next table column
end

Col = 4;  % start of Runge-Kutta Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1); % pre-allocate memory
    y(1) = y0;              % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    h2 = h/2;  % shorten calcs below
    for i = 2:length(t)
        k1 = fn(t(i-1), y(i-1));
        k2 = fn(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = fn(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = fn(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        % display results only at specified times
        for Row = 1:NumRows  % find the correct Row
            if abs(t(i) - TableTimes(Row)) < 0.00001
                V(Row,Col) = y(i);
            end
        end
    end
    Col = Col + 1;  % go to next table column
end

format
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', ...
                              'ImpEuler h = 0.1', ...
                              'ImpEuler h = 0.05', ...
                              'R-K h = 0.1', ...
                              'R-K h = 0.05'}
```

10.

Using MATLAB,

T = 5×5 table

	t	ImpEuler h = 0.1	ImpEuler h = 0.05	R-K h = 0.1	R-K h = 0.05
1	0	1.0000	1.0000	1.0000	1.0000
2	0.5000	1.6128	1.6127	1.6126	1.6126
3	1.0000	2.4818	2.4811	2.4809	2.4809
4	1.5000	3.7467	3.7458	3.7455	3.7455
5	2.0000	5.4971	5.4962	5.4959	5.4959

```
clear
% Change slope = @(t,y), y(0), ? exact solution
y0 = 1;
fn = @(t,y) 2*t + exp(-t*y);   % fn = dy/dt = f(t,y)

% times to display the y values in the table
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
StepSizes = [0.1, 0.05];
NumRows = length(TableTimes);
% initialize table array, 5 columns:
%   time +  2 ImpEuler + 2 R-K
V = zeros(NumRows, 5);

% Populate table times (Col=1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% start at Col = 2 for Improved Euler Method
Col = 2;
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1); % pre-allocate memory
    y(1) = y0;              % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    for i = 2:length(t)
        a = fn(t(i-1), y(i-1));
        b = fn(t(i), y(i-1) + h*a);
        % new estimate of y(t)
        y(i) = y(i-1) + 0.5*(a+b)*h;
        % display results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
    Col = Col + 1;  % go to next table column
end
```

```
Col = 4;  % start of Runge-Kutta Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1); % pre-allocate memory
    y(1) = y0;              % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    h2 = h/2;  % shorten calcs below
    for i = 2:length(t)
        k1 = fn(t(i-1), y(i-1));
        k2 = fn(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = fn(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = fn(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        % display results only at specified times
        for Row = 1:NumRows  % find the correct Row
            if abs(t(i) - TableTimes(Row)) < 0.00001
                V(Row,Col) = y(i);
            end
        end
    end
    Col = Col + 1;  % go to next table column
end

format
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', ...
                              'ImpEuler h = 0.1', ...
                              'ImpEuler h = 0.05', ...
                              'R-K h = 0.1', ...
                              'R-K h = 0.05'}
```

11.

Using MATLAB,

T = 5×5 table

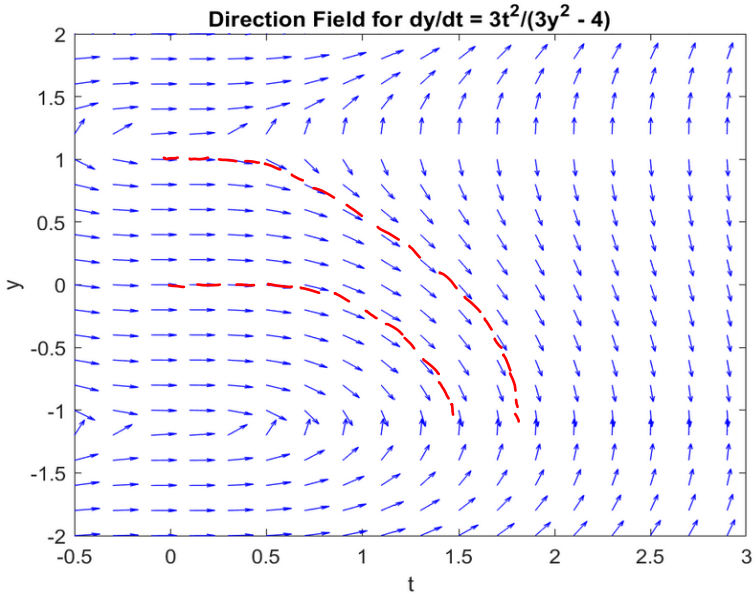
	t	ImpEuler h = 0.1	ImpEuler h = 0.05	R-K h = 0.1	R-K h = 0.05
1	0	0.5000	0.5000	0.5000	0.5000
2	0.5000	0.5907	0.5909	0.5909	0.5909
3	1.0000	0.7992	0.7998	0.8000	0.8000
4	1.5000	1.1645	1.1661	1.1667	1.1667
5	2.0000	1.7451	1.7488	1.7500	1.7500

```
clear
% Change slope = @(t,y), y(0), ? exact solution
y0 = 0.5;
fn = @(t,y) (y^2 + 2*t*y)/(3+t^2); % fn = dy/dt = f(t,y)

% times to display the y values in the table
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
StepSizes = [0.1, 0.05];
NumRows = length(TableTimes);
% initialize table array, 5 columns:
%   time +  2 ImpEuler + 2 R-K
V = zeros(NumRows, 5);
y = zeros(NumRows,1);  % pre-allocate memory for y
y(1) = y0;             % y0, MATLAB not zero based

% Populate table times (Col=1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% start at Col = 2 for Improved Euler Method
Col = 2;
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1); % pre-allocate memory
    y(1) = y0;              % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    for i = 2:length(t)
        a = fn(t(i-1), y(i-1));
        b = fn(t(i), y(i-1) + h*a);
        % new estimate of y(t)
        y(i) = y(i-1) + 0.5*(a+b)*h;
        % display results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
    Col = Col + 1;  % go to next table column
end

Col = 4;  % start of Runge-Kutta Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1); % pre-allocate memory
    y(1) = y0;              % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    h2 = h/2;  % shorten calcs below
    for i = 2:length(t)
        k1 = fn(t(i-1), y(i-1));
        k2 = fn(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = fn(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = fn(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        % display results only at specified times
        for Row = 1:NumRows  % find the correct Row
            if abs(t(i) - TableTimes(Row)) < 0.00001
                V(Row,Col) = y(i);
            end
        end
    end
    Col = Col + 1;  % go to next table column
end

format
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', ...
                              'ImpEuler h = 0.1', ...
                              'ImpEuler h = 0.05', ...
                              'R-K h = 0.1', ...
                              'R-K h = 0.05'}
```

12.

(a) Using MATLAB,

```
clear,clc;
Xmin = -0.5; Xmax = 3;
Ymin = -2; Ymax = 2;
[t, y] = meshgrid(Xmin:0.2:Xmax, Ymin:0.2:Ymax);
% S = dy/dt = slope at point (t,y)
S = 3*t.^2./(3*y.^2 - 4);
L = sqrt(1 + S.^2); %length of the slope vector
% Create a unit vector (hypotenuse) from L,S
% scale by 0.5, color blue
q = quiver(t,y, 1./L, S./L, 0.5, 'b');
axis([Xmin Xmax Ymin Ymax]);
xlabel 't', ylabel 'y'
title 'Direction Field for dy/dt = 3t^2/(3y^2 - 4)'
```



(b) $t_m \approx 1.5$ At t_m, $y'(t) \to -\infty$

Thus, as $t \to t_m^-$, $y(t)$ reaches a point of discontinuity: $y(t)$ not defined at $t > t_m$ as t approaches t_m from the left.

(c) Use MATLAB, using trial-and-error to find t_m to greater precision. $t_m \approx 1.45$

T = 8×3 table

	t	h = 0.001	h = 0.0005
1	0	0	0
2	0.5000	-0.0313	-0.0313
3	1.4000	-0.8278	-0.8278
4	1.4500	-1.0594	-1.0594
5	1.4600	-1.1768	-1.0070
6	1.4700	-0.9912	-1.1063
7	1.5000	-0.4108	-0.9349
8	2.0000	-0.1580	-2.8367

Values are consistent up to $t = 1.45$. After that, values diverge for steps sizes of very small $h = 0.001$ and $h = 0.0005$.

```
clear
% Change slope = @(t,y), y(0)
y0 = 0;
fn = @(t,y) 3*t^2/(3*y^2 - 4); % fn = dy/dt = f(t,y)

% times to display the y values in the table
TableTimes = [0, 0.5, 1.4, 1.45, 1.46, 1.47, 1.5, 2.0];
StepSizes = [0.001, 0.0005];
NumRows = length(TableTimes);
% initialize table array, 3 columns:
%   time +  2 R-K
V = zeros(NumRows, 3);

% Populate table times (Col=1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

Col = 2;  % start of Runge-Kutta Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1); % pre-allocate memory
    y(1) = y0;              % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    h2 = h/2;  % shorten calcs below
    for i = 2:length(t)
        k1 = fn(t(i-1), y(i-1));
        k2 = fn(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = fn(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = fn(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        % display results only at specified times
        for Row = 1:NumRows  % find the correct Row
            if abs(t(i) - TableTimes(Row)) < 0.00001
                V(Row,Col) = y(i);
            end
        end
    end
    Col = Col + 1;  % go to next table column
end

format
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', ...
                              'h = 0.001', ...
                              'h = 0.0005'}
```

(d) Values for $t > t_m$ have no significance for the solution for $y(0) = 0$. From the direction field, all y values seem to be bounded by the line $y \approx -1.1$. This value is approached under continuous conditions. The "weighting" approach of Runge-Kutta does not work beyond the interval of continuity. ∴ The calculated values using Runge-Kutta should have no significance, since a calculation of $t_n + \frac{1}{2}h > t_m$ should have no meaning.

(e) $t_m \approx 1.8$ using $y(0) = 1$ and direction field.

Using MATLAB, $t_m \approx 1.82$ using Runge-Kutta

Code next page.

T = 9×3 table

	t	h = 0.001	h = 0.0005
1	0	1.0000	1.0000
2	0.5000	0.9026	0.9026
3	1.5000	-0.0940	-0.0940
4	1.8000	-0.8761	-0.8761
5	1.8200	-1.0316	-1.0316
6	1.8300	-1.1268	-0.9453
7	1.8400	-1.4110	-1.0432
8	1.8600	-1.2594	-0.3724
9	2.0000	-3.1750	-1.0032

```
clear
% Change slope = @(t,y), y(0)
y0 = 1;
fn = @(t,y) 3*t^2/(3*y^2 - 4); % fn = dy/dt = f(t,y)

% times to display the y values in the table
TableTimes = [0, 0.5, 1.5, 1.8, 1.82, 1.83, 1.84, 1.86, 2.0];
StepSizes = [0.001, 0.0005];
NumRows = length(TableTimes);
% initialize table array, 3 columns:
%   time +  2 R-K
V = zeros(NumRows, 3);

% Populate table times (Col=1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

Col = 2;  % start of Runge-Kutta Method
for h = StepSizes
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1); % pre-allocate memory
    y(1) = y0;              % MATLAB not zero based
    Row = 1;   %start at row 1 for each column
    V(Row,Col) = y(1);
    h2 = h/2;  % shorten calcs below
    for i = 2:length(t)
        k1 = fn(t(i-1), y(i-1));
        k2 = fn(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = fn(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = fn(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
        % display results only at specified times
        for Row = 1:NumRows  % find the correct Row
            if abs(t(i) - TableTimes(Row)) < 0.00001
                V(Row,Col) = y(i);
            end
        end
    end
    Col = Col + 1;  % go to next table column
end

format
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', ...
                              'h = 0.001', ...
                              'h = 0.0005'}
```

Using very small step sizes, $h = 0.001$ and $h = 0.0005$, values consistent up to 1.82. After that, calculation at $t_n + \frac{1}{2}h > t_m$ yields wild R-K values.

8.4 Multistep Methods

Note Title 5/5/2020

MATLAB is used to solve the problems below. The initial values for y are generated using the Runga-Kutta method. No attempt is made to provide the actual solution. A numerical solving method, vpasolve, is used to solve implicit equations, even if f(t,y) is linear and soluable by ordinary algebraic means, just for uniformity. Use is made of the anonymous function, afn = @(t,y) f(t,y). Results of the 3 methods are combined into one table.

/.

T = 2×4 table

	t	Predictor_corrector	Adam_Moulton	Bkward_Differentiation
1	0.40000000...	1.729680081756054	1.72967996678...	1.729680465441545
2	0.50000000...	1.893469731589717	1.89346952390...	1.893471130180784

```
clear
% Change afn = @(t,y) f(t,y), y(0)
y0 = 1;
afn = @(t,y) 3 + t - y;  % afn = dy/dt = f(t,y)
h = 0.1;
% table display times
TableTimes = [0, 0.1, 0.2, 0.3, 0.4, 0.5];
NumRows = length(TableTimes);

%initialize table array, 4 columns:
%   time + pre-corr + Adam-M + BkDiff
V = zeros(2, 4);  % Only show rows t = 0.4, 0.5

% Populate table times (Col = 1)
for i = 1:2  % values for rows 1,2
    V(i,1) = TableTimes(i+4);  % t = 0.4, 0.5
end

% times to compute the y values
t = TableTimes(1):h:TableTimes(end);
y = zeros(length(t),1);  % pre-allocate memory

% get initial y values frm Runge-Kutta
y(1) = y0;   % MATLAB not zero based
h2 = h/2;    % shorten calcs below
for i = 2:4  % t = 0.1, 0.2, 0.3
    k1 = afn(t(i-1), y(i-1));
    k2 = afn(t(i-1) + h2, y(i-1) + h2*k1);
    k3 = afn(t(i-1) + h2, y(i-1) + h2*k2);
    k4 = afn(t(i-1) + h, y(i-1) + h*k3);
    y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
end
```

```
Col = 2;  % Predictor-corrector
for i = 5:6
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    fn3 = afn(t(i-4),y(i-4));   % f n-3
    % Adams-Bashforth to get y_n+1
    yn1 = y(i-1) + (h/24)*(55*fn0 - 59*fn1 + 37*fn2 - 9*fn3);
    k = afn(t(i),yn1);
    % y_n+1 now explicitly calculated/corrected
    y(i) = y(i-1) + (h/24)*(9*k + 19*fn0 - 5*fn1 + fn2);
    V(i-4,Col) = y(i);
end

Col = 3;  % Adam-M
syms k  % needed to use vpasolve to get implicit value
for i = 5:6
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    eqn = k == y(i-1) + ...
              (h/24)*(9*afn(t(i),k) + 19*fn0 - 5*fn1 + fn2);
    y(i) = vpasolve(eqn, k);  % solve implicit equation
    V(i-4,Col) = y(i);
end

Col = 4;  % Backward-differentiation
for i = 5:6
    % use initial seeds of y from R-K to get y(5), y(6)
    eqn = k == (1/25)*(48*y(i-1) - 36*y(i-2) + 16*y(i-3) ...
                       - 3*y(i-4) + 12*h*afn(t(i),k));
    y(i) = vpasolve(eqn, k);  % solve implicit equation
    V(i-4,Col) = y(i);
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Predictor_corrector', ...
                              'Adam_Moulton', ...
                              'Bkward_Differentiation'}
```

2.

T = 2×4 table

	t	Predictor_corrector	Adam_Moulton	Bkward_Differentiation
1	0.40000000...	0.993851994061415	0.99384662206...	0.993869290303869
2	0.50000000...	0.925763539158337	0.92574646644...	0.925837011284843

```
clear
% Change afn = @(t,y) f(t,y), y(0)

y0 = 2;
% afn = dy/dt = f(t,y)
afn = @(t,y) 5*t - 3*sqrt(y);
h = 0.1;
% table display times
TableTimes = [0, 0.1, 0.2, 0.3, 0.4, 0.5];
NumRows = length(TableTimes);

%initialize table array, 4 columns:
%   time + pre-corr + Adam-M + BkDiff
V = zeros(2, 4);  % Only show rows t = 0.4, 0.5

% Populate table times (Col = 1)
for i = 1:2  % values for rows 1,2
    V(i,1) = TableTimes(i+4);  % t = 0.4, 0.5
end

% times to compute the y values
t = TableTimes(1):h:TableTimes(end);
y = zeros(length(t),1);  % pre-allocate memory

% get initial y values frm Runge-Kutta
y(1) = y0;   % MATLAB not zero based
h2 = h/2;    % shorten calcs below
for i = 2:4  % t = 0.1, 0.2, 0.3
    k1 = afn(t(i-1), y(i-1));
    k2 = afn(t(i-1) + h2, y(i-1) + h2*k1);
    k3 = afn(t(i-1) + h2, y(i-1) + h2*k2);
    k4 = afn(t(i-1) + h, y(i-1) + h*k3);
    y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
end
```

```
Col = 2;  % Predictor-corrector
for i = 5:6
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    fn3 = afn(t(i-4),y(i-4));   % f n-3
    % Adams-Bashforth to get y_n+1
    yn1 = y(i-1) + (h/24)*(55*fn0 - 59*fn1 + 37*fn2 - 9*fn3);
    k = afn(t(i),yn1);
    % y_n+1 now explicitly calculated/corrected
    y(i) = y(i-1) + (h/24)*(9*k + 19*fn0 - 5*fn1 + fn2);
    V(i-4,Col) = y(i);
end

syms k  % needed for vpasolve to get implicit value

Col = 3;  % Adam-M
for i = 5:6
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    eqn = k == y(i-1) + ...
              (h/24)*(9*afn(t(i),k) + 19*fn0 - 5*fn1 + fn2);
    y(i) = vpasolve(eqn, k);  % solve implicit equation
    V(i-4,Col) = y(i);
end

Col = 4;  % Backward-differentiation
for i = 5:6
    % use initial seeds of y from R-K to get y(5), y(6)
    eqn = k == (1/25)*(48*y(i-1) - 36*y(i-2) + 16*y(i-3) ...
                       - 3*y(i-4) + 12*h*afn(t(i),k));
    y(i) = vpasolve(eqn, k);  % solve implicit equation
    V(i-4,Col) = y(i);
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Predictor_corrector', ...
                              'Adam_Moulton', ...
                              'Bkward_Differentiation'}
```

3.

T = 2×4 table

	t	Predictor_corrector	Adam_Moulton	Bkward_Differentiation
1	0.40000000...	1.472617296809064	1.47261713293...	1.472619917661099
2	0.50000000...	1.612621521710783	1.61262138734...	1.612625562132374

```
clear
% Change afn = @(t,y) f(t,y), y(0)

y0 = 1;
% afn = dy/dt = f(t,y)
afn = @(t,y) 2*t + exp(-t*y);
h = 0.1;
% table display times
TableTimes = [0, 0.1, 0.2, 0.3, 0.4, 0.5];
NumRows = length(TableTimes);

%initialize table array, 4 columns:
%   time + pre-corr + Adam-M + BkDiff
V = zeros(2, 4);  % Only show rows t = 0.4, 0.5

% Populate table times (Col = 1)
for i = 1:2  % values for rows 1,2
    V(i,1) = TableTimes(i+4);  % t = 0.4, 0.5
end

% times to compute the y values
t = TableTimes(1):h:TableTimes(end);
y = zeros(length(t),1);  % pre-allocate memory

% get initial y values frm Runge-Kutta
y(1) = y0;   % MATLAB not zero based
h2 = h/2;    % shorten calcs below
for i = 2:4  % t = 0.1, 0.2, 0.3
    k1 = afn(t(i-1), y(i-1));
    k2 = afn(t(i-1) + h2, y(i-1) + h2*k1);
    k3 = afn(t(i-1) + h2, y(i-1) + h2*k2);
    k4 = afn(t(i-1) + h, y(i-1) + h*k3);
    y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
end
```

```
Col = 2;  % Predictor-corrector
for i = 5:6
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    fn3 = afn(t(i-4),y(i-4));   % f n-3
    % Adams-Bashforth to get y_n+1
    yn1 = y(i-1) + (h/24)*(55*fn0 - 59*fn1 + 37*fn2 - 9*fn3);
    k = afn(t(i),yn1);
    % y_n+1 now explicitly calculated/corrected
    y(i) = y(i-1) + (h/24)*(9*k + 19*fn0 - 5*fn1 + fn2);
    V(i-4,Col) = y(i);
end

syms k  % needed for vpasolve to get implicit value

Col = 3;  % Adam-M
for i = 5:6
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    eqn = k == y(i-1) + ...
              (h/24)*(9*afn(t(i),k) + 19*fn0 - 5*fn1 + fn2);
    y(i) = vpasolve(eqn, k);  % solve implicit equation
    V(i-4,Col) = y(i);
end

Col = 4;  % Backward-differentiation
for i = 5:6
    % use initial seeds of y from R-K to get y(5), y(6)
    eqn = k == (1/25)*(48*y(i-1) - 36*y(i-2) + 16*y(i-3) ...
                       - 3*y(i-4) + 12*h*afn(t(i),k));
    y(i) = vpasolve(eqn, k);  % solve implicit equation
    V(i-4,Col) = y(i);
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Predictor_corrector', ...
                              'Adam_Moulton', ...
                              'Bkward_Differentiation'}
```

4.

T = 2×4 table

	t	Predictor_corrector	Adam_Moulton	Bkward_Differentiation
1	0.40000000...	0.564285771227386	0.56428578181...	0.564285877514270
2	0.50000000...	0.590909178807777	0.59090920309...	0.590909522019650

```
clear
% Change afn = @(t,y) f(t,y), y(0)

y0 = 0.5;
% afn = dy/dt = f(t,y)
afn = @(t,y) (y^2 + 2*t*y)/(3 + t^2);
h = 0.1;
% table display times
TableTimes = [0, 0.1, 0.2, 0.3, 0.4, 0.5];
NumRows = length(TableTimes);

%initialize table array, 4 columns:
%   time + pre-corr + Adam-M + BkDiff
V = zeros(2, 4);  % Only show rows t = 0.4, 0.5

% Populate table times (Col = 1)
for i = 1:2  % values for rows 1,2
    V(i,1) = TableTimes(i+4);  % t = 0.4, 0.5
end

% times to compute the y values
t = TableTimes(1):h:TableTimes(end);
y = zeros(length(t),1);  % pre-allocate memory

% get initial y values frm Runge-Kutta
y(1) = y0;   % MATLAB not zero based
h2 = h/2;    % shorten calcs below
for i = 2:4  % t = 0.1, 0.2, 0.3
    k1 = afn(t(i-1), y(i-1));
    k2 = afn(t(i-1) + h2, y(i-1) + h2*k1);
    k3 = afn(t(i-1) + h2, y(i-1) + h2*k2);
    k4 = afn(t(i-1) + h, y(i-1) + h*k3);
    y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
end

Col = 2;  % Predictor-corrector
for i = 5:6
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    fn3 = afn(t(i-4),y(i-4));   % f n-3
    % Adams-Bashforth to get y_n+1
    yn1 = y(i-1) + (h/24)*(55*fn0 - 59*fn1 + 37*fn2 - 9*fn3);
    k = afn(t(i),yn1);
    % y_n+1 now explicitly calculated/corrected
    y(i) = y(i-1) + (h/24)*(9*k + 19*fn0 - 5*fn1 + fn2);
    V(i-4,Col) = y(i);
end

syms k  % needed for vpasolve to get implicit value

Col = 3;  % Adam-M
for i = 5:6
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    eqn = k == y(i-1) + ...
              (h/24)*(9*afn(t(i),k) + 19*fn0 - 5*fn1 + fn2);
    % guess 6*h as an interval to look for next y
    range = [y(i-1) - 3*h, y(i-1) + 3*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    V(i-4,Col) = y(i);
end

Col = 4;  % Backward-differentiation
for i = 5:6
    % use initial seeds of y from R-K to get y(5), y(6)
    eqn = k == (1/25)*(48*y(i-1) - 36*y(i-2) + 16*y(i-3) ...
                       - 3*y(i-4) + 12*h*afn(t(i),k));
    % guess 6*h as an interval to look for next y
    range = [y(i-1) - 3*h, y(i-1) + 3*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    V(i-4,Col) = y(i);
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Predictor_corrector', ...
                              'Adam_Moulton', ...
                              'Bkward_Differentiation'}
```

Note use of "range" when using "vpasolve", which can give multiple answers. A range confines next answer to be close to previous estimate.

5.

T = 2×4 table

	t	Predictor_corrector	Adam_Moulton	Bkward_Differentiation
1	0.40000000...	-0.779693259811017	-0.77970024903...	-0.779679679558112
2	0.50000000...	-0.753134541140855	-0.75314421203...	-0.753089253164558

```
clear
% Change afn = @(t,y) f(t,y), y(0)

y0 = -1;
% afn = dy/dt = f(t,y)
afn = @(t,y) (t^2 - y^2)*sin(y);
h = 0.1;
% table display times
TableTimes = [0, 0.1, 0.2, 0.3, 0.4, 0.5];
NumRows = length(TableTimes);

%initialize table array, 4 columns:
%   time + pre-corr + Adam-M + BkDiff
V = zeros(2, 4);  % Only show rows t = 0.4, 0.5

% Populate table times (Col = 1)
for i = 1:2  % values for rows 1,2
    V(i,1) = TableTimes(i+4);  % t = 0.4, 0.5
end

% times to compute the y values
t = TableTimes(1):h:TableTimes(end);
y = zeros(length(t),1);  % pre-allocate memory

% get initial y values frm Runge-Kutta
y(1) = y0;   % MATLAB not zero based
h2 = h/2;    % shorten calcs below
for i = 2:4  % t = 0.1, 0.2, 0.3
    k1 = afn(t(i-1), y(i-1));
    k2 = afn(t(i-1) + h2, y(i-1) + h2*k1);
    k3 = afn(t(i-1) + h2, y(i-1) + h2*k2);
    k4 = afn(t(i-1) + h, y(i-1) + h*k3);
    y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
end
```

```
Col = 2;  % Predictor-corrector
for i = 5:6
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    fn3 = afn(t(i-4),y(i-4));   % f n-3
    % Adams-Bashforth to get y_n+1
    yn1 = y(i-1) + (h/24)*(55*fn0 - 59*fn1 + 37*fn2 - 9*fn3);
    k = afn(t(i),yn1);
    % y_n+1 now explicitly calculated/corrected
    y(i) = y(i-1) + (h/24)*(9*k + 19*fn0 - 5*fn1 + fn2);
    V(i-4,Col) = y(i);
end

syms k  % needed for vpasolve to get implicit value

Col = 3;  % Adam-M
for i = 5:6
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    eqn = k == y(i-1) + ...
                (h/24)*(9*afn(t(i),k) + 19*fn0 - 5*fn1 + fn2);
    % guess 6*h as an interval to look for next y
    range = [y(i-1) - 3*h, y(i-1) + 3*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    V(i-4,Col) = y(i);
end

Col = 4;  % Backward-differentiation
for i = 5:6
    % use initial seeds of y from R-K to get y(5), y(6)
    eqn = k == (1/25)*(48*y(i-1) - 36*y(i-2) + 16*y(i-3) ...
                        - 3*y(i-4) + 12*h*afn(t(i),k));
    % guess 6*h as an interval to look for next y
    range = [y(i-1) - 3*h, y(i-1) + 3*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    V(i-4,Col) = y(i);
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Predictor_corrector', ...
                              'Adam_Moulton', ...
                              'Bkward_Differentiation'}
```

As in #4, note use of "range" when using "vpasolve" to confine next estimate to be close to last estimate.

6.

Using MATLAB (code on next page)

T = 5×4 table

	t	Predictor_corrector	Adams_Moulton	Bkward_Differentiation
1	0	1.0000	1.0000	1.0000
2	0.5000	2.9683	2.9683	2.9683
3	1.0000	7.8891	7.8891	7.8893
4	1.5000	20.8356	20.8357	20.8364
5	2.0000	55.5984	55.5986	55.6015

The exact solution, from #7 of Section 8.3 is below.

T = 5×6 table

	t	ImpEuler h = 0.1	ImpEuler h = 0.05	R-K h = 0.1	R-K h = 0.05	Exact
1	0	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.5000	2.9527	2.9641	2.9683	2.9683	2.9683
3	1.0000	7.8046	7.8662	7.8889	7.8890	7.8891
4	1.5000	20.4923	20.7426	20.8349	20.8355	20.8355
5	2.0000	54.3576	55.2614	55.5957	55.5980	55.5982

From #7 of Section 8.3, the exact solution is

$$y(t) = \frac{1}{2}t + e^{2t}$$

```
clear
% Change afn = @(t,y) f(t,y), y(0)

y0 = 1;
% afn = dy/dt = f(t,y)
afn = @(t,y) 0.5 - t + 2*y;
h = 0.05;
% table display times
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
NumRows = length(TableTimes);
%initialize table array, 4 columns:
%   time + pre-corr + Adam-M + BkDiff
V = zeros(NumRows, 4);

% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% times to compute the y values
t = TableTimes(1):h:TableTimes(end);
y = zeros(length(t),1);  % pre-allocate memory

% get initial y values frm Runge-Kutta
y(1) = y0;   % MATLAB not zero based
h2 = h/2;    % shorten calcs below
for i = 2:4  % t = 0.05, 0.10, 0.15
    k1 = afn(t(i-1), y(i-1));
    k2 = afn(t(i-1) + h2, y(i-1) + h2*k1);
    k3 = afn(t(i-1) + h2, y(i-1) + h2*k2);
    k4 = afn(t(i-1) + h, y(i-1) + h*k3);
    y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
end
```

```
Col = 2;  % Predictor-corrector
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    fn3 = afn(t(i-4),y(i-4));   % f n-3
    % Adams-Bashforth to get y_n+1
    yn1 = y(i-1) + ...
          (h/24)*(55*fn0 - 59*fn1 + 37*fn2 - 9*fn3);
    k = afn(t(i),yn1);
    % y_n+1 now explicitly calculated/corrected
    y(i) = y(i-1) + ...
           (h/24)*(9*k + 19*fn0 - 5*fn1 + fn2);
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

syms k  % needed for vpasolve to get implicit value

Col = 3;  % Adams-Moulton
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    eqn = k == y(i-1) + ...
              (h/24)*(9*afn(t(i),k) + 19*fn0 - 5*fn1 + fn2);
    % guess 400*h as an interval to look for next y
    range = [y(i-1) - 200*h, y(i-1) + 200*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

Col = 4;  % Backward-differentiation
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    % use initial seeds of y from R-K to get y(5), y(6)
    eqn = k == (1/25)*(48*y(i-1) - 36*y(i-2) + 16*y(i-3) ...
                       - 3*y(i-4) + 12*h*afn(t(i),k));
    % guess 400*h as an interval to look for next y
    range = [y(i-1) - 200*h, y(i-1) + 200*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Predictor_corrector', ...
                              'Adams_Moulton', ...
                              'Bkward_Differentiation'}
```

The "range" option is needed when using "vpasolve" as it will sometimes yield more than one solution to the implicit equation. The range used here, ± 200h, is found from trial and error. ± 5h is too small in this case.

7.

Using MATLAB,

T = 5×4 table

	t	Predictor_corrector	Adams_Moulton	Bkward_Differentiation
1	0	2.000000000000000	2.000000000000…	2.000000000000000
2	0.50000000…	0.925713321960374	0.925712481065…	0.925724837115900
3	1.00000000…	1.285148752976239	1.285148126086…	1.285158096635591
4	1.50000000…	2.408595063482623	2.408595005911…	2.408594108237475
5	2.00000000…	4.103494758917420	4.103494765812…	4.103492843740390

```
clear
% Change afn = @(t,y) f(t,y), y(0)

y0 = 2;
% afn = dy/dt = f(t,y)
afn = @(t,y) 5*t - 3*sqrt(y);
h = 0.05;
% table display times
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
NumRows = length(TableTimes);
%initialize table array, 4 columns:
%   time + pre-corr + Adam-M + BkDiff
V = zeros(NumRows, 4);

% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% times to compute the y values
t = TableTimes(1):h:TableTimes(end);
y = zeros(length(t),1);  % pre-allocate memory

% get initial y values frm Runge-Kutta
y(1) = y0;   % MATLAB not zero based
h2 = h/2;    % shorten calcs below
for i = 2:4  % t = 0.05, 0.10, 0.15
    k1 = afn(t(i-1), y(i-1));
    k2 = afn(t(i-1) + h2, y(i-1) + h2*k1);
    k3 = afn(t(i-1) + h2, y(i-1) + h2*k2);
    k4 = afn(t(i-1) + h, y(i-1) + h*k3);
    y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
end

Col = 2;  % Predictor-corrector
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    fn3 = afn(t(i-4),y(i-4));   % f n-3
    % Adams-Bashforth to get y_n+1
    yn1 = y(i-1) + ...
        (h/24)*(55*fn0 - 59*fn1 + 37*fn2 - 9*fn3);
    k = afn(t(i),yn1);
    % y_n+1 now explicitly calculated/corrected
    y(i) = y(i-1) + ...
           (h/24)*(9*k + 19*fn0 - 5*fn1 + fn2);
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end
```

```
syms k  % needed for vpasolve to get implicit value

Col = 3;  % Adams-Moulton
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    eqn = k == y(i-1) + ...
              (h/24)*(9*afn(t(i),k) + 19*fn0 - 5*fn1 + fn2);
    % guess 400*h as an interval to look for next y
    range = [y(i-1) - 200*h, y(i-1) + 200*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

Col = 4;  % Backward-differentiation
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    % use initial seeds of y from R-K to get y(5), y(6)
    eqn = k == (1/25)*(48*y(i-1) - 36*y(i-2) + 16*y(i-3) ...
                       - 3*y(i-4) + 12*h*afn(t(i),k));
    % guess 400*h as an interval to look for next y
    range = [y(i-1) - 200*h, y(i-1) + 200*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Predictor_corrector', ...
                              'Adams_Moulton', ...
                              'Bkward_Differentiation'}
```

8.

Using MATLAB,

T = 5×4 table

	t	Predictor_corrector	Adams_Moulton	Bkward_Differentiation
1	0	3.000000000000000	3.000000000000…	3.000000000000000
2	0.50000000…	3.962186453059349	3.962186452469…	3.962186399603712
3	1.00000000…	5.108902720227190	5.108902719227…	5.108902620655727
4	1.50000000…	6.431390239342711	6.431390238107…	6.431390111505507
5	2.00000000…	7.923384799036640	7.923384797628…	7.923384650275389

```
clear
% Change afn = @(t,y) f(t,y), y(0)

y0 = 3;
% afn = dy/dt = f(t,y)
afn = @(t,y) sqrt(t + y);
h = 0.05;
% table display times
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
NumRows = length(TableTimes);
%initialize table array, 4 columns:
%   time + pre-corr + Adam-M + BkDiff
V = zeros(NumRows, 4);

% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% times to compute the y values
t = TableTimes(1):h:TableTimes(end);
y = zeros(length(t),1);  % pre-allocate memory

% get initial y values frm Runge-Kutta
y(1) = y0;   % MATLAB not zero based
h2 = h/2;    % shorten calcs below
for i = 2:4  % t = 0.05, 0.10, 0.15
    k1 = afn(t(i-1), y(i-1));
    k2 = afn(t(i-1) + h2, y(i-1) + h2*k1);
    k3 = afn(t(i-1) + h2, y(i-1) + h2*k2);
    k4 = afn(t(i-1) + h, y(i-1) + h*k3);
    y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
end

Col = 2;  % Predictor-corrector
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    fn3 = afn(t(i-4),y(i-4));   % f n-3
    % Adams-Bashforth to get y_n+1
    yn1 = y(i-1) + ...
        (h/24)*(55*fn0 - 59*fn1 + 37*fn2 - 9*fn3);
    k = afn(t(i),yn1);
    % y_n+1 now explicitly calculated/corrected
    y(i) = y(i-1) + ...
           (h/24)*(9*k + 19*fn0 - 5*fn1 + fn2);
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

syms k  % needed for vpasolve to get implicit value

Col = 3;  % Adams-Moulton
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    eqn = k == y(i-1) + ...
              (h/24)*(9*afn(t(i),k) + 19*fn0 - 5*fn1 + fn2);
    % guess 400*h as an interval to look for next y
    range = [y(i-1) - 200*h, y(i-1) + 200*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

Col = 4;  % Backward-differentiation
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    % use initial seeds of y from R-K to get y(5), y(6)
    eqn = k == (1/25)*(48*y(i-1) - 36*y(i-2) + 16*y(i-3) ...
                       - 3*y(i-4) + 12*h*afn(t(i),k));
    % guess 400*h as an interval to look for next y
    range = [y(i-1) - 200*h, y(i-1) + 200*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Predictor_corrector', ...
                              'Adams_Moulton', ...
                              'Bkward_Differentiation'}
```

9.

Using MATLAB,

T = 5×4 table

	t	Predictor_corrector	Adams_Moulton	Bkward_Differentiation
1	0	1.000000000000000	1.000000000000...	1.000000000000000
2	0.50000000...	1.612622311083164	1.612622306101...	1.612623072598190
3	1.00000000...	2.480909358465159	2.480909387720...	2.480904658372030
4	1.50000000...	3.745478650414062	3.745478681716...	3.745473370023594
5	2.00000000...	5.495872433667518	5.495872464846...	5.495868941540706

```
clear
% Change afn = @(t,y) f(t,y), y(0)

y0 = 1;
% afn = dy/dt = f(t,y)
afn = @(t,y) 2*t + exp(-t*y);

h = 0.05;
% table display times
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
NumRows = length(TableTimes);
%initialize table array, 4 columns:
%   time + pre-corr + Adam-M + BkDiff
V = zeros(NumRows, 4);

% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% times to compute the y values
t = TableTimes(1):h:TableTimes(end);
y = zeros(length(t),1);  % pre-allocate memory

% get initial y values frm Runge-Kutta
y(1) = y0;   % MATLAB not zero based
h2 = h/2;    % shorten calcs below
for i = 2:4  % t = 0.05, 0.10, 0.15
    k1 = afn(t(i-1), y(i-1));
    k2 = afn(t(i-1) + h2, y(i-1) + h2*k1);
    k3 = afn(t(i-1) + h2, y(i-1) + h2*k2);
    k4 = afn(t(i-1) + h, y(i-1) + h*k3);
    y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
end

Col = 2;  % Predictor-corrector
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    fn3 = afn(t(i-4),y(i-4));   % f n-3
    % Adams-Bashforth to get y_n+1
    yn1 = y(i-1) + ...
        (h/24)*(55*fn0 - 59*fn1 + 37*fn2 - 9*fn3);
    k = afn(t(i),yn1);
    % y_n+1 now explicitly calculated/corrected
    y(i) = y(i-1) + ...
           (h/24)*(9*k + 19*fn0 - 5*fn1 + fn2);
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end
```

```
syms k  % needed for vpasolve to get implicit value

Col = 3;  % Adams-Moulton
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    eqn = k == y(i-1) + ...
              (h/24)*(9*afn(t(i),k) + 19*fn0 - 5*fn1 + fn2);
    % guess 400*h as an interval to look for next y
    range = [y(i-1) - 200*h, y(i-1) + 200*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

Col = 4;  % Backward-differentiation
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    % use initial seeds of y from R-K to get y(5), y(6)
    eqn = k == (1/25)*(48*y(i-1) - 36*y(i-2) + 16*y(i-3) ...
                       - 3*y(i-4) + 12*h*afn(t(i),k));
    % guess 400*h as an interval to look for next y
    range = [y(i-1) - 200*h, y(i-1) + 200*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Predictor_corrector', ...
                              'Adams_Moulton', ...
                              'Bkward_Differentiation'}
```

10.

Using MATLAB,

T = 5×4 table

	t	Predictor_corrector	Adams_Moulton	Bkward_Differentiation
1	0	0.500000000000000	0.500000000000…	0.500000000000000
2	0.50000000…	0.590909099320276	0.590909100590…	0.590909150549036
3	1.00000000…	0.800000029860729	0.800000036206…	0.800000246052543
4	1.50000000…	1.166666746111970	1.166666766388…	1.166667360045700
5	2.00000000…	1.750000193965070	1.750000249356…	1.750001748401531

```
clear
% Change afn = @(t,y) f(t,y), y(0)

y0 = 0.5;
% afn = dy/dt = f(t,y)
afn = @(t,y) (y^2 + 2*t*y)/(3 + t^2);

h = 0.05;
% table display times
TableTimes = [0, 0.5, 1.0, 1.5, 2.0];
NumRows = length(TableTimes);
%initialize table array, 4 columns:
%   time + pre-corr + Adam-M + BkDiff
V = zeros(NumRows, 4);

% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% times to compute the y values
t = TableTimes(1):h:TableTimes(end);
y = zeros(length(t),1);  % pre-allocate memory

% get initial y values frm Runge-Kutta
y(1) = y0;   % MATLAB not zero based
h2 = h/2;    % shorten calcs below
for i = 2:4  % t = 0.05, 0.10, 0.15
    k1 = afn(t(i-1), y(i-1));
    k2 = afn(t(i-1) + h2, y(i-1) + h2*k1);
    k3 = afn(t(i-1) + h2, y(i-1) + h2*k2);
    k4 = afn(t(i-1) + h, y(i-1) + h*k3);
    y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
end

Col = 2;  % Predictor-corrector
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    fn3 = afn(t(i-4),y(i-4));   % f n-3
    % Adams-Bashforth to get y_n+1
    yn1 = y(i-1) + ...
        (h/24)*(55*fn0 - 59*fn1 + 37*fn2 - 9*fn3);
    k = afn(t(i),yn1);
    % y_n+1 now explicitly calculated/corrected
    y(i) = y(i-1) + ...
           (h/24)*(9*k + 19*fn0 - 5*fn1 + fn2);
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end
```

```
syms k  % needed for vpasolve to get implicit value

Col = 3;  % Adams-Moulton
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    fn0 = afn(t(i-1),y(i-1));   % f n
    fn1 = afn(t(i-2),y(i-2));   % f n-1
    fn2 = afn(t(i-3),y(i-3));   % f n-2
    eqn = k == y(i-1) + ...
              (h/24)*(9*afn(t(i),k) + 19*fn0 - 5*fn1 + fn2);
    % guess 400*h as an interval to look for next y
    range = [y(i-1) - 200*h, y(i-1) + 200*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

Col = 4;  % Backward-differentiation
Row = 1;
V(Row,Col) = y(1);
for i = 5:length(t)
    % use initial seeds of y from R-K to get y(5), y(6)
    eqn = k == (1/25)*(48*y(i-1) - 36*y(i-2) + 16*y(i-3) ...
                       - 3*y(i-4) + 12*h*afn(t(i),k));
    % guess 400*h as an interval to look for next y
    range = [y(i-1) - 200*h, y(i-1) + 200*h];
    y(i) = vpasolve(eqn,k,range);  % solve implicit equation
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = y(i);
    end
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Predictor_corrector', ...
                              'Adams_Moulton', ...
                              'Bkward_Differentiation'}
```

11.

(a) From $\phi(t_{n+1}) - \phi(t_n) = \int_{t_n}^{t_{n+1}} \phi'(t)\,dt$,

let $\phi'(t) = A$, a constant (zero degree polynomial).

$\therefore \phi'(t) = A = f(t_n, y_n)$, from the left-side of $\phi'(t)$ on $[t_n, t_{n+1}]$. Also, $y_n = \phi(t_n)$ and $y_{n+1} = \phi(t_{n+1})$

$$\therefore y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} A\,dt = A(t_{n+1} - t_n) = Ah$$

$$\therefore y_{n+1} - y_n = f(t_n, y_n)\,h$$

$$\therefore \underline{y_{n+1} = y_n + h f(t_n, y_n)}$$

(b)

As in (a), let $\phi'(t) = A$, but this time

$A = f(t_{n+1}, y_{n+1})$, the right-side of $\phi'(t)$ on $[t_n, t_{n+1}]$

$$\therefore y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} A\,dt = A(t_{n+1} - t_n) = Ah = f(t_{n+1}, y_{n+1})h$$

$$\therefore \underline{y_{n+1} = y_n + h f(t_{n+1}, y_{n+1})}$$

12.

Let $\phi'(t) = P_2(t) = At^2 + Bt + C$

Since $\phi'(t) = F(t, y)$, then

$$At_n^2 + Bt_n + C = F(t_n, y_n) = f_n$$

$$At_{n-1}^2 + Bt_{n-1} + C = f(t_{n-1}, y_{n-1}) = f_{n-1}$$

$$At_{n-2}^2 + Bt_{n-2} + C = f(t_{n-2}, y_{n-2}) = f_{n-2}$$

Using matrix algebra,

$$\begin{bmatrix} t_n^2 & t_n & 1 \\ t_{n-1}^2 & t_{n-1} & 1 \\ t_{n-2}^2 & t_{n-2} & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} f_n \\ f_{n-1} \\ f_{n-2} \end{bmatrix}$$

Solving this by hand gets messy.

∴ Use MATLAB to do the necessary calculations and simplifications. The key is to keep the number of symbol variables to a minimum by using $t_{n-1} = t_n - h$, $t_{n-2} = t_n - 2h$, since the

step size is uniform. MATLAB will solve for the coefficients A, B, C, and then use MATLAB to do the definite integral

$$\phi(t_{n+1}) - \phi(t_n) \approx \int_{t_n}^{t_n+h} At^2 + Bt + C \, dt$$

Then use $y_{n+1} = \phi(t_{n+1})$, $y_n = \phi(t_n)$.

```
clear
syms t tn h fn fn1 fn2
E = [ tn^2,      tn,     1;
     (tn-h)^2,   tn-h,   1;
     (tn-2*h)^2, tn-2*h, 1];
F = [fn; fn1; fn2];
x = E\F  % solve Ex = F
% x holds coefficients of the polynomial
% integrate the polynomial from tn to tn+h
int(x(1)*t^2 + x(2)*t + x(3), t, tn, tn+h)
```

x =

$$\begin{pmatrix} \frac{fn - 2\,fn_1 + fn_2}{2\,h^2} \\ \frac{3\,fn\,h - 4\,fn_1\,h + fn_2\,h - 2\,fn\,tn + 4\,fn_1\,tn - 2\,fn_2\,tn}{2\,h^2} \\ \frac{2\,fn\,h^2 + fn\,tn^2 - 2\,fn_1\,tn^2 + fn_2\,tn^2 - 3\,fn\,h\,tn + 4\,fn_1\,h\,tn - fn_2\,h\,tn}{2\,h^2} \end{pmatrix}$$

$x(1) = A$

$x(2) = B$

$x(3) = C$

ans =

$$\frac{h\,(23\,fn - 16\,fn_1 + 5\,fn_2)}{12}$$

$$\therefore y_{n+1} - y_n = \int_{t_n}^{t_n+h} (At^2 + Bt + C)\,dt = \frac{h}{12}\left(23 f_n - 16 f_{n-1} + 5 f_{n-2}\right)$$

13.

As in #12, use MATLAB to do the messy computations.

Let $\phi'(t) = At^2 + Bt + C = f(t, y)$

This time, $\begin{bmatrix} t_{n+1}^2 & t_{n+1} & 1 \\ t_n^2 & t_n & 1 \\ t_{n-1} & t_{n-1} & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} f_{n+1} \\ f_n \\ f_{n-1} \end{bmatrix}$

Integration is still performed over $[t_n, t_{n+1}]$

use $t_{n+1} = t_n + h$
$t_{n-1} = t_n - h$

```
clear
syms t tn h fp1 fn fn1
E = [(tn+h)^2,   tn+h,   1;
       tn^2,      tn,    1;
     (tn-h)^2,   tn-h,   1];
F = [fp1; fn; fn1];
x = E\F  % solve Ex = F
% x holds coefficients of the polynomial
% integrate the polynomial from tn to tn+h
int(x(1)*t^2 + x(2)*t + x(3), t, tn, tn+h)
```

x =

$$\begin{pmatrix} \frac{\mathrm{fn}_1 - 2\,\mathrm{fn} + \mathrm{fp}_1}{2h^2} \\ -\frac{\mathrm{fn}_1 h - \mathrm{fp}_1 h - 4\,\mathrm{fn}\,\mathrm{tn} + 2\,\mathrm{fn}_1\,\mathrm{tn} + 2\,\mathrm{fp}_1\,\mathrm{tn}}{2h^2} \\ \frac{2\,\mathrm{fn}\,h^2 - 2\,\mathrm{fn}\,\mathrm{tn}^2 + \mathrm{fn}_1\,\mathrm{tn}^2 + \mathrm{fp}_1\,\mathrm{tn}^2 + \mathrm{fn}_1 h\,\mathrm{tn} - \mathrm{fp}_1 h\,\mathrm{tn}}{2h^2} \end{pmatrix}$$

$x(1) = A$ $\quad fp1 \equiv f_{n+1}$

$x(2) = B$ $\quad fn \equiv f_n$

$x(3) = C$ $\quad fn1 \equiv f_{n-1}$

ans =

$$\frac{h\,(8\,\mathrm{fn} - \mathrm{fn}_1 + 5\,\mathrm{fp}_1)}{12}$$

$$\therefore\ y_{n+1} - y_n = \int_{t_n}^{t_n+h} (At^2 + Bt + C)\,dt = \frac{h}{12}\left(5f_{n+1} + 8f_n - f_{n-1}\right)$$

Backward differentiation uses $P_k(t)$ to approximate $\phi(t)$ rather than $\phi'(t)$, which Adams methods use. Similar to Adams-Moulton, it uses the right side of $P_k(t)$ on $[t_n, t_{n+1}]$. $\therefore$ From $\phi'(t) = f(t,y)$, $P_k'(t) = f(t,y)$. $\therefore$ For $P_2(t) = At^2 + Bt + C$,

$P_2'(t) = f(t,y) = 2At + B$

$\therefore P_2'(t_{n+1}) = f(t_{n+1}, y_{n+1}) = 2A t_{n+1} + B$.

$\therefore$ Get A, B coefficients by solving

$$\begin{bmatrix} t_{n+1}^2 & t_{n+1} & 1 \\ t_n^2 & t_n & 1 \\ t_{n-1}^2 & t_{n-1} & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} y_{n+1} \\ y_n \\ y_{n-1} \end{bmatrix}$$

Note A and B will be in terms of y_{n+1}, y_n, y_{n-1}.

Then solve $f(t_{n+1}, y_{n+1}) = 2A t_{n+1} + B$ for y_{n+1}

To simplify the algebraic computations, minimize

number of variables by setting $t_{n+1} = t_n + h$, $t_{n-1} = t_n - h$.

```
clear
syms t tn h yp1 yn yn1 fp1
E = [(tn+h)^2,   tn+h,   1;
       tn^2,       tn,     1;
     (tn-h)^2,   tn-h,   1];
F = [yp1; yn; yn1];
x = E\F  % solve Ex = F
% x holds coefficients of the polynomial
eqn = 2*x(1)*(tn + h) + x(2) == fp1;
solve(eqn, yp1)
```

x =

$$\begin{pmatrix} \frac{yn_1 - 2\,yn + yp_1}{2\,h^2} \\ -\frac{h\,yn_1 - h\,yp_1 - 4\,tn\,yn + 2\,tn\,yn_1 + 2\,tn\,yp_1}{2\,h^2} \\ \frac{2\,h^2\,yn - 2\,tn^2\,yn + tn^2\,yn_1 + tn^2\,yp_1 + h\,tn\,yn_1 - h\,tn\,yp_1}{2\,h^2} \end{pmatrix}$$

$x(1) = A$

$x(2) = B$

$x(3) = C$

ans =

$$\frac{4\,yn}{3} - \frac{yn_1}{3} + \frac{2\,fp_1\,h}{3}$$

$yn = y_n$, $yn1 = y_{n-1}$

$yp1 = y_{n+1}$, $fp1 = f_{n+1}$

$$\therefore\ y_{n+1} = \frac{1}{3}\left(4y_n - y_{n-1} + 2hf_{n+1}\right)$$

8.5 Systems of First-order Equations

Note Title 5/12/2020

Use MATLAB, and make use of an anonymous function that returns a 1x2 vector. Display results of (a), (b), and (c) in one table. 1x2 vectors used instead of 1x2 vectors since the display table is composed of colums of 1x2 vectors. In the code, temporary vector holder Z is used in the R-k method to make coding more succinct.

1.

T = 6×7 table

	t	Euler_h01_x	Euler_h01_y	R_K_h02_x	R_K_h02_y	R_K_h01_x	R_K_h01_y
1	0	1.0000	0	1.0000	0	1.0000	0
2	0.2000	1.2600	0.7600	1.3249	0.7589	1.3249	0.7595
3	0.4000	1.7714	1.4824	1.9368	1.5792	1.9369	1.5800
4	0.6000	2.5899	2.3703	2.9341	2.6610	2.9346	2.6620
5	0.8000	3.8237	3.6041	4.4832	4.2264	4.4842	4.2278
6	1.0000	5.6425	5.3888	6.8424	6.5645	6.8444	6.5668

MATLAB code on next page.

```
clear
% Change fn = @(t,x,y) [f(t,x,y);g(t,x,y)], x(0), y(0)

x0 = 1;
y0 = 0;
% fn = [x' = f(t,x,y), y' = g(t,x,y)]
fn = @(t,x,y) [x + y + t, 4*x - 2*y];

% table display times
TableTimes = [0, 0.2, 0.4, 0.6, 0.8, 1.0];
NumRows = length(TableTimes);
%initialize table array, 7 columns:
%  times plus x,y for Euler, R-K h=0.2, R-K h=0.1
V = zeros(NumRows, 7);

% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% Euler Method
Col = 2;
h = 0.1;
% times to compute the X vector values
t = TableTimes(1):h:TableTimes(end);
X = zeros(length(t),2);  % pre-allocate memory
X(1,:) = [x0,y0];  % X0, MATLAB not zero based
Row = 1;
V(Row,Col:Col+1) = X(1,:);
for i = 2:length(t)
    X(i,:) = X(i-1,:) + h*fn(t(i-1),X(i-1,1),X(i-1,2));
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = X(i,1);
        V(Row,Col+1) = X(i,2);
    end
end

% Runge-Kutta Method
Col = 4;
StepSizes = [0.2, 0.1];
for h = StepSizes
    % times to compute the X vector values
    t = TableTimes(1):h:TableTimes(end);
    X = zeros(length(t),2);  % pre-allocate memory
    X(1,:) = [x0,y0];  % X0, MATLAB not zero based
    Row = 1;
    V(Row,Col:Col+1) = X(1,:);
    for i = 2:length(t)
        h2 = h/2;    % shorten calcs below
        k1 = fn(t(i-1), X(i-1,1), X(i-1,2));
            z = X(i-1,:) + h2*k1;  % temp holder
        k2 = fn(t(i-1) + h2, z(1), z(2));
            z = X(i-1,:) + h2*k2;
        k3 = fn(t(i-1)+h2, z(1), z(2));
            z = X(i-1,:) + h*k3;
        k4 = fn(t(i-1)+h, z(1), z(2));
        X(i,:) = X(i-1,:) + h*(k1 + 2*k2 + 2*k3 + k4)/6;

        % display results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = X(i,1);
            V(Row,Col+1) = X(i,2);
        end
    end
    Col = Col + 2;  % next col for next R-K step size
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Euler_h01_x', ...
                                   'Euler_h01_y', ...
                                   'R_K_h02_x', ...
                                   'R_K_h02_y', ...
                                   'R_K_h01_x', ...
                                   'R_K_h01_y'}
```

2.

MATLAB used, code below.

T = 6×7 table

	t	Euler_h01_x	Euler_h01_y	R_K_h02_x	R_K_h02_y	R_K_h01_x	R_K_h01_y
1	0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.2000	0.5820	1.1800	0.5685	1.1577	0.5684	1.1578
3	0.4000	0.1180	1.2734	0.1098	1.2256	0.1098	1.2256
4	0.6000	-0.3369	1.2738	-0.3221	1.2035	-0.3221	1.2035
5	0.8000	-0.7300	1.1857	-0.6813	1.1016	-0.6813	1.1016
6	1.0000	-1.0213	1.0237	-0.9379	0.9379	-0.9378	0.9378

```
clear
% Change fn = @(t,x,y) [f(t,x,y);g(t,x,y)], x(0), y(0)

x0 = 1;
y0 = 1;
% fn = [x' = f(t,x,y), y' = g(t,x,y)]
fn = @(t,x,y) [-t*x - y - 1, x];

% table display times
TableTimes = [0, 0.2, 0.4, 0.6, 0.8, 1.0];
NumRows = length(TableTimes);
%initialize table array, 7 columns:
%  times plus x,y for Euler, R-K h=0.2, R-K h=0.1
V = zeros(NumRows, 7);

% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% Euler Method
Col = 2;
h = 0.1;
% times to compute the X vector values
t = TableTimes(1):h:TableTimes(end);
X = zeros(length(t),2);  % pre-allocate memory
X(1,:) = [x0,y0];  % X0, MATLAB not zero based
Row = 1;
V(Row,Col:Col+1) = X(1,:);
for i = 2:length(t)
    X(i,:) = X(i-1,:) + h*fn(t(i-1),X(i-1,1),X(i-1,2));
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = X(i,1);
        V(Row,Col+1) = X(i,2);
    end
end
```

```
% Runge-Kutta Method
Col = 4;
StepSizes = [0.2, 0.1];
for h = StepSizes
    % times to compute the X vector values
    t = TableTimes(1):h:TableTimes(end);
    X = zeros(length(t),2);  % pre-allocate memory
    X(1,:) = [x0,y0];  % X0, MATLAB not zero based
    Row = 1;
    V(Row,Col:Col+1) = X(1,:);
    for i = 2:length(t)
        h2 = h/2;    % shorten calcs below
        k1 = fn(t(i-1), X(i-1,1), X(i-1,2));
            z = X(i-1,:) + h2*k1;  % temp holder
        k2 = fn(t(i-1) + h2, z(1), z(2));
            z = X(i-1,:) + h2*k2;
        k3 = fn(t(i-1)+h2, z(1), z(2));
            z = X(i-1,:) + h*k3;
        k4 = fn(t(i-1)+h, z(1), z(2));
        X(i,:) = X(i-1,:) + h*(k1 + 2*k2 + 2*k3 + k4)/6;

        % display results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = X(i,1);
            V(Row,Col+1) = X(i,2);
        end
    end
    Col = Col + 2;  % next col for next R-K step size
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Euler_h01_x', ...
                              'Euler_h01_y', ...
                              'R_K_h02_x', ...
                              'R_K_h02_y', ...
                              'R_K_h01_x', ...
                              'R_K_h01_y'}
```

3.

Using MATLAB,

T = 6×7 table

	t	Euler_h01_x	Euler_h01_y	R_K_h02_x	R_K_h02_y	R_K_h01_x	R_K_h01_y
1	0	0	1.0000	0	1.0000	0	1.0000
2	0.2000	-0.1980	0.6180	-0.1969	0.6309	-0.1969	0.6309
3	0.4000	-0.3788	0.2833	-0.3726	0.2989	-0.3727	0.2989
4	0.6000	-0.5193	-0.0321	-0.5013	-0.0111	-0.5013	-0.0112
5	0.8000	-0.5943	-0.3268	-0.5613	-0.2889	-0.5613	-0.2891
6	1.0000	-0.5883	-0.5755	-0.5471	-0.5083	-0.5470	-0.5084

```
clear
% Change fn = @(t,x,y) [f(t,x,y);g(t,x,y)],
%        x(0), y(0)
x0 = 0;
y0 = 1;
% fn = [x' = f(t,x,y), y' = g(t,x,y)]
fn = @(t,x,y) [x - y + x*y, 3*x - 2*y - x*y];

% table display times
TableTimes = [0, 0.2, 0.4, 0.6, 0.8, 1.0];
NumRows = length(TableTimes);
%initialize table array, 7 columns:
%  times plus x,y for Euler, R-K h=0.2, R-K h=0.1
V = zeros(NumRows, 7);

% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% Euler Method
Col = 2;
h = 0.1;
% times to compute the X vector values
t = TableTimes(1):h:TableTimes(end);
X = zeros(length(t),2);  % pre-allocate memory
X(1,:) = [x0,y0];  % X0, MATLAB not zero based
Row = 1;
V(Row,Col:Col+1) = X(1,:);
for i = 2:length(t)
    X(i,:) = X(i-1,:) + ...
             h*fn(t(i-1),X(i-1,1),X(i-1,2));
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = X(i,1);
        V(Row,Col+1) = X(i,2);
    end
end
```

```
% Runge-Kutta Method
Col = 4;
StepSizes = [0.2, 0.1];
for h = StepSizes
    % times to compute the X vector values
    t = TableTimes(1):h:TableTimes(end);
    X = zeros(length(t),2);  % pre-allocate memory
    X(1,:) = [x0,y0];  % X0, MATLAB not zero based
    Row = 1;
    V(Row,Col:Col+1) = X(1,:);
    for i = 2:length(t)
        h2 = h/2;    % shorten calcs below
        k1 = fn(t(i-1), X(i-1,1), X(i-1,2));
            z = X(i-1,:) + h2*k1;  % temp holder
        k2 = fn(t(i-1) + h2, z(1), z(2));
            z = X(i-1,:) + h2*k2;
        k3 = fn(t(i-1)+h2, z(1), z(2));
            z = X(i-1,:) + h*k3;
        k4 = fn(t(i-1)+h, z(1), z(2));
        X(i,:) = X(i-1,:) + h*(k1 + 2*k2 + 2*k3 + k4)/6;

        % display results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = X(i,1);
            V(Row,Col+1) = X(i,2);
        end
    end
    Col = Col + 2;  % next col for next R-K step size
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Euler_h01_x', ...
                              'Euler_h01_y', ...
                              'R_K_h02_x', ...
                              'R_K_h02_y', ...
                              'R_K_h01_x', ...
                              'R_K_h01_y'}
```

4.

Using MATLAB,

T = 6×7 table

	t	Euler_h01_x	Euler_h01_y	R_K_h02_x	R_K_h02_y	R_K_h01_x	R_K_h01_y
1	0	4.0000	1.0000	4.0000	1.0000	4.0000	1.0000
2	0.2000	2.9623	1.3454	3.0634	1.3486	3.0631	1.3490
3	0.4000	2.3412	1.6712	2.4450	1.6864	2.4446	1.6870
4	0.6000	1.9024	1.9716	1.9911	2.0004	1.9908	2.0011
5	0.8000	1.5660	2.2390	1.6382	2.2798	1.6378	2.2806
6	1.0000	1.2977	2.4673	1.3555	2.5175	1.3551	2.5183

```
clear
% Change fn = @(t,x,y) [f(t,x,y);g(t,x,y)],
%        x(0), y(0)
x0 = 4;
y0 = 1;
% fn = [x' = f(t,x,y), y' = g(t,x,y)]
fn = @(t,x,y) [x*(1 - 0.5*x - 0.5*y), ...
               y*(-0.25 + 0.5*x)];

% table display times
TableTimes = [0, 0.2, 0.4, 0.6, 0.8, 1.0];
NumRows = length(TableTimes);
%initialize table array, 7 columns:
%  times, x,y for Euler, R-K h=0.2, R-K h=0.1
V = zeros(NumRows, 7);

% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% Euler Method
Col = 2;
h = 0.1;
% times to compute the X vector values
t = TableTimes(1):h:TableTimes(end);
X = zeros(length(t),2);  % pre-allocate memory
X(1,:) = [x0,y0];  % X0, MATLAB not zero based
Row = 1;
V(Row,Col:Col+1) = X(1,:);
for i = 2:length(t)
    X(i,:) = X(i-1,:) + ...
             h*fn(t(i-1),X(i-1,1),X(i-1,2));
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = X(i,1);
        V(Row,Col+1) = X(i,2);
    end
end
```

```
% Runge-Kutta Method
Col = 4;
StepSizes = [0.2, 0.1];
for h = StepSizes
    % times to compute the X vector values
    t = TableTimes(1):h:TableTimes(end);
    X = zeros(length(t),2);  % pre-allocate memory
    X(1,:) = [x0,y0];  % X0, MATLAB not zero based
    Row = 1;
    V(Row,Col:Col+1) = X(1,:);
    for i = 2:length(t)
        h2 = h/2;    % shorten calcs below
        k1 = fn(t(i-1), X(i-1,1), X(i-1,2));
            z = X(i-1,:) + h2*k1;  % temp holder
        k2 = fn(t(i-1) + h2, z(1), z(2));
            z = X(i-1,:) + h2*k2;
        k3 = fn(t(i-1)+h2, z(1), z(2));
            z = X(i-1,:) + h*k3;
        k4 = fn(t(i-1)+h, z(1), z(2));
        X(i,:) = X(i-1,:) + h*(k1 + 2*k2 + 2*k3 + k4)/6;

        % display results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = X(i,1);
            V(Row,Col+1) = X(i,2);
        end
    end
    Col = Col + 2;  % next col for next R-K step size
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Euler_h01_x', ...
                              'Euler_h01_y', ...
                              'R_K_h02_x', ...
                              'R_K_h02_y', ...
                              'R_K_h01_x', ...
                              'R_K_h01_y'}
```

5.

Using MATLAB,

T = 6×7 table

	t	Euler_h01_x	Euler_h01_y	R_K_h02_x	R_K_h02_y	R_K_h01_x	R_K_h01_y
1	0	1.0000	2.0000	1.0000	2.0000	1.0000	2.0000
2	0.2000	1.4239	2.1896	1.4151	2.1870	1.4151	2.1870
3	0.4000	1.8223	2.3679	1.8121	2.3623	1.8121	2.3623
4	0.6000	2.2173	2.5333	2.2064	2.5258	2.2064	2.5258
5	0.8000	2.6112	2.6876	2.5983	2.6794	2.5983	2.6794
6	1.0000	2.9955	2.8335	2.9781	2.8249	2.9781	2.8249

```
clear
% Change fn = @(t,x,y) [f(t,x,y);g(t,x,y)],
%        x(0), y(0)
x0 = 1;
y0 = 2;
% fn = [x' = f(t,x,y), y' = g(t,x,y)]
fn = @(t,x,y) [exp(-x + y) - cos(x), ...
               sin(x - 3*y)];

% table display times
TableTimes = [0, 0.2, 0.4, 0.6, 0.8, 1.0];
NumRows = length(TableTimes);
%initialize table array, 7 columns:
%  times, x,y for Euler, R-K h=0.2, R-K h=0.1
V = zeros(NumRows, 7);

% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% Euler Method
Col = 2;
h = 0.1;
% times to compute the X vector values
t = TableTimes(1):h:TableTimes(end);
X = zeros(length(t),2);  % pre-allocate memory
X(1,:) = [x0,y0];  % X0, MATLAB not zero based
Row = 1;
V(Row,Col:Col+1) = X(1,:);
for i = 2:length(t)
    X(i,:) = X(i-1,:) + ...
             h*fn(t(i-1),X(i-1,1),X(i-1,2));
    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = X(i,1);
        V(Row,Col+1) = X(i,2);
    end
end
```

```
% Runge-Kutta Method
Col = 4;
StepSizes = [0.2, 0.1];
for h = StepSizes
    % times to compute the X vector values
    t = TableTimes(1):h:TableTimes(end);
    X = zeros(length(t),2);  % pre-allocate memory
    X(1,:) = [x0,y0];  % X0, MATLAB not zero based
    Row = 1;
    V(Row,Col:Col+1) = X(1,:);
    for i = 2:length(t)
        h2 = h/2;    % shorten calcs below
        k1 = fn(t(i-1), X(i-1,1), X(i-1,2));
            z = X(i-1,:) + h2*k1;  % temp holder
        k2 = fn(t(i-1) + h2, z(1), z(2));
            z = X(i-1,:) + h2*k2;
        k3 = fn(t(i-1)+h2, z(1), z(2));
            z = X(i-1,:) + h*k3;
        k4 = fn(t(i-1)+h, z(1), z(2));
        X(i,:) = X(i-1,:) + h*(k1 + 2*k2 + 2*k3 + k4)/6;

        % display results only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = X(i,1);
            V(Row,Col+1) = X(i,2);
        end
    end
    Col = Col + 2;  % next col for next R-K step size
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Euler_h01_x', ...
                              'Euler_h01_y', ...
                              'R_K_h02_x', ...
                              'R_K_h02_y', ...
                              'R_K_h01_x', ...
                              'R_K_h01_y'}
```

For comparison, MATLAB code using "sprintf" is used, allowing control over decimal precision (5 digits after the decimal is used here) and table headers, unavailable with "array2table". The code is the same as above, and is deleted, except for the code displaying the tables. The code shows the use of the \n (for new line) and \t (for tab) to align the column output.

```
format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Euler_h01_x', ...
                                   'Euler_h01_y', ...
                                   'R_K_h02_x', ...
                                   'R_K_h02_y', ...
                                   'R_K_h01_x', ...
                                   'R_K_h01_y'}

Header1 = 'time\tEuler h=0.1\tEuler h=0.1\tR-K h=0.2\tR-K h=0.2\tR-K h=0.1\tR-K h=0.1 \n';
Header2 = '\t\t     x \t\t     y\t\t     x \t\t     y\t\t     x \t\t     y \n';
stHeader = strcat(Header1,Header2);
stFormatSpec = '%.2f\t %.5f\t %.5f\t %.5f\t %.5f\t %.5f\t %.5f\n';
stV = sprintf(stFormatSpec, V(1,:), V(2,:), V(3,:), V(4,:), V(5,:), V(6,:));
s = strcat(stHeader,stV);
sprintf(s)
```

T = 6×7 table

	t	Euler_h01_x	Euler_h01_y	R_K_h02_x	R_K_h02_y	R_K_h01_x	R_K_h01_y
1	0	1.0000	2.0000	1.0000	2.0000	1.0000	2.0000
2	0.2000	1.4239	2.1896	1.4151	2.1870	1.4151	2.1870
3	0.4000	1.8223	2.3679	1.8121	2.3623	1.8121	2.3623
4	0.6000	2.2173	2.5333	2.2064	2.5258	2.2064	2.5258
5	0.8000	2.6112	2.6876	2.5983	2.6794	2.5983	2.6794
6	1.0000	2.9955	2.8335	2.9781	2.8249	2.9781	2.8249

```
ans =
    'time       Euler h=0.1       Euler h=0.1       R-K h=0.2       R-K h=0.2       R-K h=0.1       R-K h=0.1
                        x                 y               x               y               x               y
     0.00       1.00000           2.00000         1.00000         2.00000         1.00000         2.00000
     0.20       1.42386           2.18957         1.41513         2.18699         1.41513         2.18699
     0.40       1.82234           2.36791         1.81208         2.36233         1.81209         2.36233
     0.60       2.21728           2.53329         2.20635         2.52580         2.20635         2.52581
     0.80       2.61118           2.68763         2.59826         2.67940         2.59826         2.67941
     1.00       2.99550           2.83354         2.97806         2.82487         2.97806         2.82488'
```

Use of "array2table" is much easier.

6.

The exact solution is: $\phi(t) = \frac{e^{-t} + e^{3t}}{2}$ and $\psi(t) = \frac{e^{-t} - e^{3t}}{4}$. (8)

$\phi(1.0) = 10.227$, $\psi(1.0) = -4.9294$

∴ Look for answers in which $|x_{h_1} - x_{h_2}| < 0.001$ and $|y_{h_1} - y_{h_2}| < 0.0001$, since x corresponds to ϕ and y corresponds to ψ.

Using MATLAB, the last step size displayed is for $h = 0.025$, indicating that step $h = 0.05$ agrees with $h = 0.025$ up to 5 digits. The values of x and y are displayed for this last step, and they agree with $\phi(1.0) = 10.227$, $\psi(1.0) = -4.9294$.

```
h = 0.2000
xLast = 10.2069
yLast = -4.9195
h = 0.1000
xLast = 10.2251
yLast = -4.9286
h = 0.0500
xLast = 10.2266
yLast = -4.9294
h = 0.0250
xLast = 10.2267
yLast = -4.9294

x = 10.2270

y = -4.9294
```

Code next page.

```
clear
% Change fn = @(t,x,y) [f(t,x,y);g(t,x,y)],
%        x(0), y(0)
x0 = 1;
y0 = 0;
% fn = [x' = f(t,x,y), y' = g(t,x,y)]
fn = @(t,x,y) [x - 4*y, -x + y];

format
xDiff = 1;  % initialize for while loop
yDiff = 1;
xLast = x0;
yLast = y0;
h = 0.2;
while (xDiff >= 0.001) && (yDiff >= 0.0001)
    % times to compute the X vector values
    t = 0:h:1;
    X = zeros(length(t),2);  % pre-allocate memory
    X(1,:) = [x0,y0];  % X0, MATLAB not zero based

    % Runge-Kutta Method
    for i = 2:length(t)
        h2 = h/2;    % shorten calcs below
        k1 = fn(t(i-1), X(i-1,1), X(i-1,2));
            z = X(i-1,:) + h2*k1;  % temp holder
        k2 = fn(t(i-1) + h2, z(1), z(2));
            z = X(i-1,:) + h2*k2;
        k3 = fn(t(i-1)+h2, z(1), z(2));
            z = X(i-1,:) + h*k3;
        k4 = fn(t(i-1)+h, z(1), z(2));
        X(i,:) = X(i-1,:) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
    end

    % compare current h results with last h value
    xDiff = abs(X(end,1) - xLast);
    yDiff = abs(X(end,2) - yLast);

    % get ready for next while loop if needed
    h  % display current step size and values
    xLast = round(X(end,1),6,'significant')
    yLast = round(X(end,2),6,"significant")
    h = h/2;  % for next while loop
end
x = round(xLast, 5, 'significant')
y = round(yLast, 5, 'significant')
```

7.

(a) Let $y = x'$ $\therefore y' = x'' = t - t^2 x' - 3x = t - t^2 y - 3x$

$\therefore x' = y$

$y' = t - t^2 y - 3x$ $\qquad x(0) = 1,\ y(0) = 2$

(b) Using MATLAB code, adapted from problems #1-5 above. Two types of data display shown, one using "array2table", another using "sprintf".

T = 3×3 table

	t	R_K_h01_x	R_K_h01_y
1	0	1.000000000...	2.000000000...
2	0.50000000...	1.543003281...	0.070750258...
3	1.00000000...	1.147433241...	-1.38850129...

```
ans =
    'time        R-K h=0.1        R-K h=0.1
                    x                y
     0.00         1.00000          2.000000
     0.50         1.54300          0.070750
     1.00         1.14743         -1.388501'
```

Code on next page.

```
clear
% Change fn = @(t,x,y) [f(t,x,y);g(t,x,y)],
%        x(0), y(0)
x0 = 1;
y0 = 2;
% fn = [x' = f(t,x,y), y' = g(t,x,y)]
fn = @(t,x,y) [y, t - t^2*y - 3*x];

% table display times
TableTimes = [0, 0.5, 1.0];
NumRows = length(TableTimes);
%initialize table array, 3 columns:
%  times, x,y for R-K h=0.1
V = zeros(NumRows, 3);

% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% Runge-Kutta Method
h = 0.1;
Col = 2;
% times to compute the X vector values
t = TableTimes(1):h:TableTimes(end);
X = zeros(length(t),2);  % pre-allocate memory
X(1,:) = [x0,y0];  % X0, MATLAB not zero based
Row = 1;
V(Row,Col:Col+1) = X(1,:);
for i = 2:length(t)
    h2 = h/2;    % shorten calcs below
    k1 = fn(t(i-1), X(i-1,1), X(i-1,2));
    z = X(i-1,:) + h2*k1;  % temp holder
    k2 = fn(t(i-1) + h2, z(1), z(2));
    z = X(i-1,:) + h2*k2;
    k3 = fn(t(i-1)+h2, z(1), z(2));
    z = X(i-1,:) + h*k3;
    k4 = fn(t(i-1)+h, z(1), z(2));
    X(i,:) = X(i-1,:) + h*(k1 + 2*k2 + 2*k3 + k4)/6;

    % display results only at specified times
    if abs(t(i) - TableTimes(Row+1)) < 0.00001
        Row = Row + 1;
        V(Row,Col) = X(i,1);
        V(Row,Col+1) = X(i,2);
    end
end

format long
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'R_K_h01_x', ...
                              'R_K_h01_y'}

Header1 = 'time\tR-K h=0.1\tR-K h=0.1 \n';
Header2 = '\t\t     x \t\t     y \n';
stHeader = strcat(Header1,Header2);
stFormatSpec = '%.2f\t %.5f\t %9.6f\n';
stV = sprintf(stFormatSpec, V(1,:), V(2,:), V(3,:));
s = strcat(stHeader,stV);
sprintf(s)
```

8.

Note: the exact values are obtained at $t = 0.1, 0.2, 0.3$. ∴ The calculation is only performed at $t = 0.4$, once using the Adams-Bashforth formula to obtain $[x_4, y_4]$, and then again using $[x_4, y_4]$ to obtain $f_4(0.4, x_4, y_4)$ and $g_4(0.4, x_4, y_4)$ to calculate a corrected $[x_4, y_4]$ using Adams-Moulton.

Using MATLAB, $x_4 = 1.995204$, $y_4 = -0.662442$

```
clear
h = 0.1;
fn = @(x,y) [x - 4*y, -x + y];
x0 = 1;
y0 = 0;
X = zeros(5,2); % x,y values at t = 0, 0.1, 0.2, 0.3, 0.4
X(1,:) = [x0, y0];  % MATLAB not zero based
X(2,:) = [1.12735, -0.111255];  % from exact solution
X(3,:) = [1.32042, -0.250847];
X(4,:) = [1.60021, -0.429696];

% Adams-Bashforth to get [x5,y5]

fn1 = fn(X(1,1), X(1,2));
fn2 = fn(X(2,1), X(2,2));
fn3 = fn(X(3,1), X(3,2));
fn4 = fn(X(4,1), X(4,2));
X(5,:) = X(4,:) + h/24*(55*fn4 - 59*fn3 + 37*fn2 - 9*fn1);

% Adams-Moulton to get corrected X(5,:)

fn5 = fn(X(5,1), X(5,2));
% display all values
X(5,:) = X(4,:) + h/24*(9*fn5 + 19*fn4 - 5*fn3 + fn2)

% just display end value to 6 decimals
x = sprintf('%9.6f', X(5,1))
y = sprintf('%9.6f', X(5,2))
```

```
X = 5×2
    1.0000         0
    1.1274   -0.1113
    1.3204   -0.2508
    1.6002   -0.4297
    1.9952   -0.6624
```

```
x = ' 1.995204'
```

```
y = '-0.662442'
```

8.6 More on Errors; Stability

Note Title 5/21/2020

1.

(a)

$$y' - y = t - 3, \quad \frac{d}{dt}\left(ye^{-t}\right) = te^{-t} - 3e^{-t}$$

$$\therefore ye^{-t} = -te^{-t} - e^{-t} + 3e^{-t} + C$$

$$\therefore y = -t + 2 + ce^{t}, \quad y(0) = 2 \Rightarrow c = 0$$

$$\therefore y(t) = 2 - t$$

(b)

If $y(0) = 2.001$, then from (a), $y = -t + 2 + ce^{t}$,

$$\therefore 2.001 = 2 + c, \quad c = .001.$$

$$\therefore y(t) = 2 - t + 0.001e^{t}$$

$$\therefore \phi_2(t) - \phi_1(t) = \left[2 - t + 0.001e^{t}\right] - \left[2 - t\right] = 0.001e^{t}$$

At $t = 1$, $\phi_2(1) - \phi_1(1) = 0.001e = 0.00272$

$\lim_{t \to \infty} 0.001 e^t = \infty$

2.

(a)

For $0 \le t \le 1$, $0 \le t^2 \le 1$.

Adding e^t, $e^t \le t^2 + e^t \le 1 + e^t$.

Since $\phi_2' = e^t$, $\phi_1' = 1 + e^t$, $\therefore \phi_2'(t) \le \phi'(t) \le \phi_1'(t)$

Note all 3 functions are non-negative on $[0,1]$.

$$\therefore \int_0^t \phi_2'(s)\,ds \le \int_0^t \phi'(s)\,ds \le \int_0^t \phi_1'(s)\,ds, \quad t \in (0,1].$$

$\therefore \phi_2(t) - \phi_2(0) \leq \phi(t) - \phi(0) \leq \phi_1(t) - \phi_1(0)$

and since $\phi_2(0) = \phi(0) = \phi_1(0) = 0$,

$\underline{\phi_2(t) \leq \phi(t) \leq \phi_1(t)}$ on $[0,1]$.

(6)

For ϕ_2: $y' = e^y \Leftrightarrow -y'e^{-y} = -1$

$\therefore \int \frac{d}{dt}(e^{-y})\,dt = \int -1\,dt$

$\therefore e^{-y} = -t + c$. $y(0) = 0 \Rightarrow 1 = c$

$\therefore e^{-y} = 1 - t$, $e^y = \frac{1}{1-t}$, $\therefore y = \ln\left[\frac{1}{1-t}\right]$

$\therefore \underline{\phi_2(t) = \ln\left[\frac{1}{1-t}\right]}$

For $\phi_1(t)$: $y' = 1 + e^y$ Let $u(t) = e^{y(t)}$

$\therefore u(0) = e^{y(0)} = e^0 = 1$

$\ln(u) = \ln(e^y) = y$. $\therefore y' = \frac{1}{u} \cdot u'$

$\therefore \frac{u'}{u} = 1 + u$, or $\frac{u'}{u^2+u} = 1 \Leftrightarrow \frac{u'}{u^2+u+\frac{1}{4}-\frac{1}{4}} = 1$

$$\therefore \frac{u'}{(u+\frac{1}{2})^2-(\frac{1}{2})^2} = 1$$

$$\therefore \int \frac{u'}{(u+\frac{1}{2})^2-(\frac{1}{2})^2}\,dt = \int dt$$

using $\int \frac{du}{u^2-a^2} = \frac{1}{2a}\ln\left|\frac{u-a}{u+a}\right| + C$

$$\therefore \frac{1}{2(\frac{1}{2})}\ln\left|\frac{u+\frac{1}{2}-\frac{1}{2}}{u+\frac{1}{2}+\frac{1}{2}}\right| = t + C$$

Note $u(t) > 0$ on $0 \le t \le 1$

$$\therefore \ln\left(\frac{u}{u+1}\right) = t + C. \quad u(0) = 1 \Rightarrow -\ln(2) = C$$

$$\therefore \ln\left(\frac{u}{u+1}\right) = t - \ln(2) \Rightarrow \frac{u}{u+1} = e^t e^{-\ln(2)} = \frac{1}{2}e^t$$

$$\therefore 2u = e^t(u+1), \text{ now using } u = e^y$$

$$2e^y = e^t(e^y+1), \quad 2 = e^t(1+e^{-y})$$

$$\therefore 2e^{-t} - 1 = e^{-y}, \quad e^y = \frac{1}{2e^{-t}-1} = \frac{e^t}{2-e^t}$$

$$\therefore$$

$$y = \underline{\phi_1(t)} = \ln\left[\frac{e^t}{2-e^t}\right]$$

From above, $\underline{\phi_2(t)} = \ln\left[\frac{1}{1-t}\right]$

$$\therefore \lim_{t \to 1^-} \phi_2(t) = \lim_{t \to 1^-} \ln\left[\frac{1}{1-t}\right] = +\infty$$

Since $e^{\ln(2)} = 2$,

$$\lim_{t \to \ln(2)^-} \phi_1(t) = \lim_{t \to \ln(2)^-} \left[\frac{e^t}{2 - e^t}\right] = +\infty$$

Since $\phi_2(t) \leq \phi(t) \leq \phi_1(t)$ on $0 \leq t \leq 1$,

$\phi(t) \to \infty$ for some $t : \ln(2) \leq t \leq 1$

(c)

From (b), for $y' = e^y$, $e^{-y} = -t + c$

$\therefore e^{-3.4298} = -(0.9) + C$, $\therefore C = 0.93239$

$\therefore e^{-y} = 0.93239 - t$

$\therefore y = \phi_2(t) = \ln\left(\frac{1}{0.93239 - t}\right)$

For $y' = 1 + e^y$, $\ln\left(\frac{u}{u+1}\right) = t + c$, $u(t) = e^{y(t)}$

$\therefore \frac{u}{u+1} = Ke^t$, $K = e^c$

$$\therefore \quad u = Ke^t(u+1), \quad e^y = Ke^t(e^y+1)$$

$$\therefore \quad K = \frac{e^y}{e^t(e^y+1)}, \quad y(0.9) = 3.4298 \Rightarrow$$

$$K = \frac{e^{3.4298}}{e^{0.9}(e^{3.4298}+1)} = 0.39381$$

$$\therefore \quad e^y = (0.39381)e^t(e^y+1)$$

$$\frac{e^{-t}}{(0.39381)} = 1 + e^{-y}, \quad e^{-y} = \frac{e^{-t} - 0.39381}{0.39381}$$

$$e^y = \frac{0.39381}{e^{-t} - 0.39381} = \frac{(0.39381)e^t}{1-(0.39381)e^t}$$

$$\therefore \quad y = \phi_1(t) = \ln\left[\frac{(0.39381)e^t}{1-(0.39381)e^t}\right]$$

From above,

$$\phi_2(t) = \ln\left[\frac{1}{0.93239 - t}\right]$$

When $t = 0.932$, $\phi_2(t) = 7.84064$

As $t \to 0.93239$, then $\phi_2 \to \infty$

and $\phi_2 \leq \phi$ on $0 \leq t \leq 1$.

$\therefore \phi(t) \to \infty$ as $t \to 0.93239^{-}$.

Also, the denominator in $\phi_1(t)$ is close to 0 when $t \approx 0.932$, so $\phi_1(t)$ is large.

3.

(a)

Table 8.6.3 shows that $h = 0.0166$ is insufficient.

$\therefore$ Start with $h = 0.005$ and proceed using MATLAB.

T = 3×7 table

	t	Exact	h_005	h_001	h_0005	h_00025	h_0001
1	0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.0500	0.0567	0.0510	0.0552	0.0559	0.0563	0.0566
3	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000

The error is $h = 0.0005$ is 0.0008 at $t = 0.05$.

Error at $h = 0.00025$ is 0.0004 at $t = 0.05$

Error at $h = 0.0001$ is 0.0001 at $t = 0.05$

$\therefore$ $h = 0.00025$ seems sufficient.

MATLAB code on next page.

```
clear
% Change afn = @(t,y) f(t,y), y(0)
% afn = dy/dt = f(t,y)
afn = @(t,y) -100*y + 100*t +1;
y0 = 1;

% table display times
TableTimes = [0, 0.05, 0.1];
NumRows = length(TableTimes);
StepSize = [0.005, 0.001, 0.0005, 0.00025, 0.0001];
%initialize table array
%   # Cols = time + exact + # Euler tries
V = zeros(NumRows, 2 + length(StepSize));

% Populate table times (Col = 1)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,2) = exp(-100*t) + t;  % exact solution
end

Col = 2;
for h = StepSize
    % times to compute the y values
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory
    Col = Col + 1;  % column for the StepSize
    y(1) = y0;
    Row = 1;
    V(Row,Col) = y(1);
    for i =2:length(t)
        % Euler formula
        y(i) = y(i-1) + h*afn(t(i-1), y(i-1));

        % display result only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Exact', ...
                                   'h_005', ...
                                   'h_001', ...
                                   'h_0005', ...
                                   'h_00025', ...
                                   'h_0001'}
```

(5)

$$y_{n+1} = y_n + h f(t_{n+1}, y_{n+1})$$

$$= y_n + h\left[-100 y_{n+1} + 100(t_n + h) + 1\right]$$

$$= y_n - 100h\, y_{n+1} + 100h\, t_n + 100h^2 + h$$

$$\therefore y_{n+1} = \frac{y_n + 100h(t_n + h) + h}{1 + 100h}$$

T = 3×7 table

	t	Exact	h_005	h_001	h_0005	h_00025	h_0001
1	0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	0.0500	0.0567	0.0673	0.0585	0.0576	0.0572	0.0569
3	0.1000	0.1000	0.1003	0.1001	0.1001	0.1001	0.1000

Error at $t = 0.05$ for $h = 0.0005$ is 0.0009

Error at $t = 0.05$ for $h = 0.00025$ is 0.0005

and at $t = 0.10$ for $h = 0.00025$ is 0.0001

$\therefore$ $h = 0.00025$ seems sufficient.

MATLAB code on next page.

```
clear
% Change afn = @(t,y) f(t,y), y(0)
% afn from solving y(n) = y(n-1) + h*f(t(n+1), y(n+1))
afn = @(h,t,y) (y + 100*h*(t + h) + h)/(1 + 100*h);
y0 = 1;

% table display times
TableTimes = [0, 0.05, 0.1];
NumRows = length(TableTimes);
StepSize = [0.005, 0.001, 0.0005, 0.00025, 0.0001];
%initialize table array
%   # Cols = time + exact + # Euler tries
V = zeros(NumRows, 2 + length(StepSize));

% Populate table times (Col = 1)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,2) = exp(-100*t) + t;  % exact solution
end

Col = 2;
for h = StepSize
    % times to compute the y values
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory
    Col = Col + 1;  % column for the StepSize
    y(1) = y0;
    Row = 1;
    V(Row,Col) = y(1);
    for i =2:length(t)
        % Backward Euler formula
        y(i) = afn(h, t(i-1), y(i-1));

        % display result only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Exact', ...
                              'h_005', ...
                              'h_001', ...
                              'h_0005', ...
                              'h_00025', ...
                              'h_0001'}
```

(c)

Using MATLAB,

T = 3×5 table

	t	Exact	h_010	h_005	h_001
1	0	1.0000	1.0000	1.0000	1.0000
2	0.0500	0.0567	0.0574	0.0568	0.0567
3	0.1000	0.1000	0.1001	0.1000	0.1000

Error at $t = 0.05$ for $h = 0.01$ is 0.0007.

Error at $t = 0.05$ for $h = 0.005$ is 0.0001,

at $t = 0.10$ for $h = 0.005$ is < 0.0001

$\therefore$ $h = 0.005$ is sufficient. Code below.

```
clear
% Change afn = @(t,y) f(t,y), y(0)
% afn = dy/dt = f(t,y)
afn = @(t,y) -100*y + 100*t +1;
y0 = 1;

% table display times
TableTimes = [0, 0.05, 0.1];
NumRows = length(TableTimes);
StepSize = [0.010, 0.005, 0.001];
%initialize table array
%   # Cols = time + exact + # Euler tries
V = zeros(NumRows, 2 + length(StepSize));

% Populate table times (Col = 1)
for i = 1:NumRows
    t = TableTimes(i);
    V(i,1) = t;
    V(i,2) = exp(-100*t) + t;  % exact solution
end
```

```
Col = 2;
for h = StepSize
    h2 = h/2;    % shorten calcs below
    % times to compute the y values
    t = TableTimes(1):h:TableTimes(end);
    y = zeros(length(t),1);  % pre-allocate memory
    Col = Col + 1;  % column for the StepSize
    y(1) = y0;
    Row = 1;
    V(Row,Col) = y(1);
    for i =2:length(t)
        % Runge-Kutta method
        h2 = h/2;    % shorten calcs below
        k1 = afn(t(i-1), y(i-1));
        k2 = afn(t(i-1) + h2, y(i-1) + h2*k1);
        k3 = afn(t(i-1) + h2, y(i-1) + h2*k2);
        k4 = afn(t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;

        % display result only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'Exact', ...
                              'h_010', ...
                              'h_005', ...
                              'h_001'}
```

4.

(a)

Try Laplace transform. $\mathcal{L}\{t^2\} = \frac{2}{s^3}$, $\mathcal{L}\{t\} = \frac{1}{s^2}$

$$\mathcal{L}\{y'\} = s\mathcal{L}\{y\} - y(0) = s\mathcal{L}\{y\} - 4$$

$$\therefore\ s\mathcal{L}\{y\} - 4 = -10\mathcal{L}\{y\} + 2.5\left(\frac{2}{s^3}\right) + 0.5\left(\frac{1}{s^2}\right)$$

$$\mathcal{L}\{y\} = \frac{5}{(s+10)s^3} + \frac{1}{2(s+10)s^2} + \frac{4}{s+10}$$

$$= \frac{10}{2(s+10)s^3} + \frac{s}{2(s+10)s^3} + \frac{4}{s+10}$$

$$= \frac{1}{2}\frac{1}{s^3} + \frac{4}{s+10}$$

$$\mathcal{L}^{-1}\left(\frac{1}{s^3}\right) = \frac{1}{2}t^2,\quad \mathcal{L}^{-1}\left(\frac{1}{s+10}\right) = e^{-10t}$$

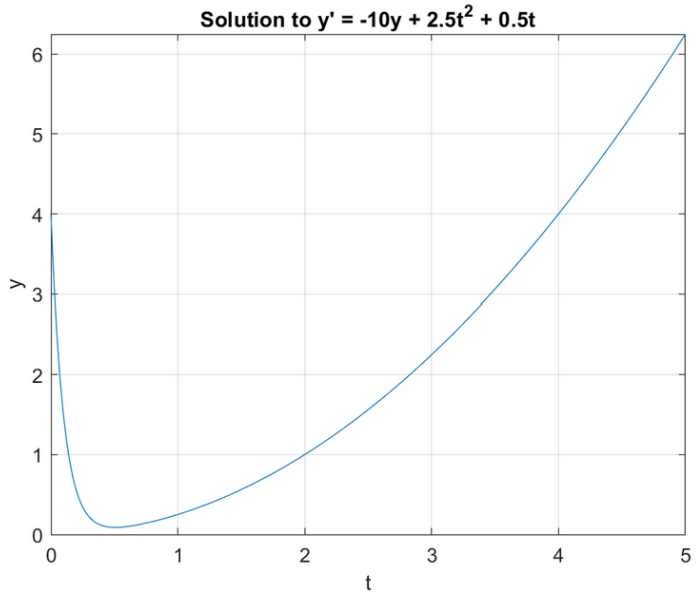
$$\therefore\ y(t) = 4e^{-10t} + \frac{1}{4}t^2$$

Using MATLAB, and checking above

```
clear
syms y(t)
eqn = diff(y,t) == -10*y + 2.5*t^2 + 0.5*t;
cond = y(0) == 4;
y = dsolve(eqn,cond)
fplot(y,[0,5])
grid on
xlabel 't', ylabel 'y'
title 'Solution to y'' = -10y + 2.5t^2 + 0.5t'
```

y =

$$4\,e^{-10t} + \frac{t^2}{4}$$



(6)

Using MATLAB (R2020a), several tables are produced of various step sizes (h values). Code on next page. Accuracy starts to fail around h = 0.19 and is way off at h = 0.25.

T = 9×3 table

	t	Exact	h =0.1
1	0	4.0000	4.0000
2	0.6000	0.0999	0.0875
3	1.2000	0.3600	0.3575
4	1.8000	0.8100	0.8075
5	2.5000	1.5625	1.5600
6	3.1000	2.4025	2.4000
7	3.7000	3.4225	3.4200
8	4.3000	4.6225	4.6200
9	5.0000	6.2500	6.2475

T = 9×3 table

	t	Exact	h =0.18
1	0	4.0000	4.0000
2	0.5400	0.0910	-1.9819
3	1.0800	0.2917	1.3369
4	1.8000	0.8100	1.2355
5	2.3400	1.3689	1.1443
6	2.8800	2.0736	2.1818
7	3.6000	3.2400	3.2817
8	4.1400	4.2849	4.2568
9	4.8600	5.9049	5.8907

```
clear
% Change afn = @(t,y) f(t,y), y(0)
% afn = dy/dt = f(t,y)
afn = @(t,y) -10*y + 2.5*t^2 + 0.5*t;
y0 = 4;

StepSize = [0.1, 0.18, 0.19, 0.20, 0.25];

for h = StepSize
    % times to compute the y values
    th = 0:h:5;
    NumTimes = length(th);

    % create just 9 table row display times
    N = 9;
    % an array of N integer pointers,
    %  to be used as th(idx())
    idx = floor(linspace(1, length(th), N));

    %initialize table array
    %   # Cols = time + exact + Euler method
    V = zeros(N, 3);
    % Populate table times (Col = 1),
    % exact solution (Col = 2)
    for Row =1:N
        t = th(idx(Row));  % time for Row
        V(Row,1) = t;
        % exact solution
        V(Row,2) = 4*exp(-10*t) + (t^2)/4;
    end

    y = zeros(NumTimes,1);  % pre-allocate memory
    y(1) = y0;
    for i = 2:NumTimes  % Euler formula
        y(i) = y(i-1) + h*afn(th(i-1), y(i-1));
    end
    % populate table with Euler values in column 3
    for Row = 1:N
        V(Row,3) = y(idx(Row));
    end

    % Display table
    st = strcat('h = ', num2str(h));
    T = array2table(V);
    T.Properties.VariableNames = {'t', 'Exact', st}
end
```

T = 9×3 table

	t	Exact	h =0.19
1	0	4.0000	4.0000
2	0.5700	0.0946	-2.8430
3	1.1400	0.3249	2.4484
4	1.7100	0.7310	-0.8252
5	2.4700	1.5252	0.5025
6	3.0400	2.3104	3.0477
7	3.6100	3.2580	2.7123
8	4.1800	4.3681	4.7577
9	4.9400	6.1009	6.3549

T = 9×3 table

	t	Exact	h =0.2
1	0	4.0000	4.0000
2	0.6000	0.0999	-3.9200
3	1.2000	0.3600	4.3600
4	1.8000	0.8100	-3.2000
5	2.4000	1.4400	5.4400
6	3.0000	2.2500	-1.7600
7	3.6000	3.2400	7.2400
8	4.2000	4.4100	0.4000
9	5.0000	6.2500	2.2400

T = 9×3 table

	t	Exact	h =0.25
1	0	4.0000	4
2	0.5000	0.0895	9.0703
3	1.2500	0.3906	-30.0381
4	1.7500	0.7656	-67.6912
5	2.5000	1.5625	232.5768
6	3.0000	2.2500	522.0400
7	3.7500	3.5156	-1.7508e+03
8	4.2500	4.5156	-3.9427e+03
9	5.0000	6.2500	1.3328e+04

(c)

Using MATLAB (R2020a to use flexibility of T.Properties.VariableNames), several tables are

displayed with different h values.

```
clear
% Change afn = @(t,y) f(t,y), y(0)
% afn = dy/dt = f(t,y)
afn = @(t,y) -10*y + 2.5*t^2 + 0.5*t;
y0 = 4;

StepSize = [0.1, 0.20, 0.25, 0.30, 0.35];
% create just 9 table row display times
N = 9;

for h = StepSize
    % times to compute the y values
    th = 0:h:5;
    NumTimes = length(th);

    % an array of N integer pointers,
    %  to be used as th(idx())
    idx = floor(linspace(1, length(th), N));

    %initialize table array
    %   # Cols = time + exact + Euler method
    V = zeros(N, 3);
    % Populate table times (Col = 1),
    % exact solution (Col = 2)
    for Row =1:N
        t = th(idx(Row));  % time for Row
        V(Row,1) = t;
        % exact solution
        V(Row,2) = 4*exp(-10*t) + (t^2)/4;
    end

    % Runge-Kutta method
    y = zeros(NumTimes,1);  % pre-allocate memory
    y(1) = y0;
    h2 = h/2;  % shorten calcs below
    for i = 2:NumTimes
        k1 = afn(th(i-1), y(i-1));
        k2 = afn(th(i-1) + h2, y(i-1) + h2*k1);
        k3 = afn(th(i-1) + h2, y(i-1) + h2*k2);
        k4 = afn(th(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;
    end

    % populate table with R-K values in column 3
    for Row = 1:N
        V(Row,3) = y(idx(Row));
    end

    % Display table
    st = strcat('R-K h = ', num2str(h));
    T = array2table(V);
    T.Properties.VariableNames = {'t', 'Exact', st}
end
```

T = 9×3 table

	t	Exact	R-K h =0.1
1	0	4.0000	4.0000
2	0.6000	0.0999	0.1012
3	1.2000	0.3600	0.3601
4	1.8000	0.8100	0.8101
5	2.5000	1.5625	1.5626
6	3.1000	2.4025	2.4026
7	3.7000	3.4225	3.4226
8	4.3000	4.6225	4.6226
9	5.0000	6.2500	6.2501

T = 9×3 table

	t	Exact	R-K h =0.2
1	0	4.0000	4.0000
2	0.6000	0.0999	0.2406
3	1.2000	0.3600	0.3680
4	1.8000	0.8100	0.8127
5	2.4000	1.4400	1.4425
6	3.0000	2.2500	2.2525
7	3.6000	3.2400	3.2425
8	4.2000	4.4100	4.4125
9	5.0000	6.2500	6.2525

T = 9×3 table

	t	Exact	R-K h =0.25
1	0	4.0000	4.0000
2	0.5000	0.0895	1.7528
3	1.2500	0.3906	0.8620
4	1.7500	0.7656	0.9722
5	2.5000	1.5625	1.6293
6	3.0000	2.2500	2.2865
7	3.7500	3.5156	3.5361
8	4.2500	4.5156	4.5326
9	5.0000	6.2500	6.2652

T = 9×3 table

	t	Exact	R-K h =0.3
1	0	4.0000	4.0000
2	0.6000	0.0999	7.6826
3	1.2000	0.3600	14.7447
4	1.8000	0.8100	28.0362
5	2.4000	1.4400	52.9446
6	3.0000	2.2500	99.6560
7	3.6000	3.2400	187.4282
8	4.2000	4.4100	352.6709
9	4.8000	5.7600	664.2207

T = 9×3 table

	t	Exact	R-K h =0.35
1	0	4.0000	4
2	0.3500	0.1514	10.9851
3	1.0500	0.2757	82.1260
4	1.7500	0.7656	611.6820
5	2.4500	1.5006	4.5606e+03
6	2.8000	1.9600	1.2456e+04
7	3.5000	3.0625	9.2946e+04
8	4.2000	4.4100	6.9359e+05
9	4.9000	6.0025	5.1760e+06

The results show stability for $h \leq 0.25$, and instability for $h \geq 0.30$. Also, for accuracy at times < 0.6, a step size of ≈ 0.1 is needed. Two tables demonstrating this are shown below, altering the above code using $N = 50$ (but only displaying 9 rows below) and using $h = 0.05$, $h = 0.10$.

T = 50×3 table

	t	Exact	R-K h =0.05
1	0	4.0000	4.0000
2	0.1000	1.4740	1.4752
3	0.2000	0.5513	0.5522
4	0.3000	0.2216	0.2221
5	0.4000	0.1133	0.1135
6	0.5000	0.0895	0.0896
7	0.6000	0.0999	0.1000
8	0.7000	0.1261	0.1262
9	0.8000	0.1613	0.1614

T = 50×3 table

	t	Exact	R-K h =0.1
1	0	4.0000	4.0000
2	0.1000	1.4740	1.5026
3	0.2000	0.5513	0.5726
4	0.3000	0.2216	0.2335
5	0.4000	0.1133	0.1192
6	0.5000	0.0895	0.0922
7	0.6000	0.0999	0.1012
8	0.7000	0.1261	0.1268
9	0.8000	0.1613	0.1616

(d)

$$y_{n+1} = y_n + h f(t_{n+1}, y_{n+1})$$

$$= y_n + h\left[-10 y_{n+1} + 2.5(t_n + h)^2 + 0.5(t_n + h)\right]$$

$$\therefore y_{n+1}(1+10h) = y_n + 2.5t_n^2 h + 5t_n h^2 + 2.5h^3 + 0.5t_n h + 0.5h^2$$

$$\therefore y_{n+1} = \frac{y_n + 2.5h^3 + (5t_n + 0.5)h^2 + (2.5t_n^2 + 0.5t_n)h}{1+10h}$$

For MATLAB coding, let $d = 1+10h$,

$K_1 = 2.5h^3$, $K_2 = (5t_n + 0.5)h^2$, $K_3 = (2.5t_n^2 + 0.5t_n)h$

afn = @(yn, K1, K2, K3, d) (yn + K1 + K2 + K3)/d

where afn is an autonymous function.

Note also $t_n = nh$.

Note that to find error precisely at $t = 5.0$, need a step size, h, so that an integer multiple of h yields 5.0. For example, for $h = 0.3$, $16(0.3) = 4.8$, $17(0.3) = 5.1$. So a step size of $h = 0.3$ does not "land" on $t = 5.0$.

$\therefore$ if n = # steps, choose $h = \frac{5.0}{n}$.

Thus, search for an h value with the

fewest steps (n) s.t. the error is ≤ 0.01.

From above, R-K method with $h = 0.25$ was fairly close to an error $\simeq 0.01$.

$h = 0.25 \Rightarrow n = \frac{5.0}{0.25} = 20$ steps. 10 steps means $h = \frac{5.0}{10} = 0.5$.

$\therefore$ Try step sizes $h = \frac{5.0}{10}, \frac{5.0}{11}, \ldots, \frac{5.0}{19}, \frac{5.0}{20}$ and see which among those has error ≤ 0.01, and choose the one with the fewest # of steps. Use MATLAB, displaying just the result at $t = 5.0$, and its error. The error is computed from $|\phi(5.0) - y_n|$, where $\phi(5.0)$ is the exact value, and y_n is computed using the Backward Euler formula. Note that if n is the # of steps to "land" on 5.0 using the Backward Euler

method, step #1 in MATLAB is $y(0) = 4$, the initial value. MATLAB is not zero based, so that $y(n)$ in MATLAB code is y_{n-1} in the above formula, so that y_n, computed using $y(n)$ in MATLAB, is the predicted value at $t = 5.0$ after n MATLAB steps. The y_n contains the global error.

The results are:

```
phi = 6.2500

ans = 'n = 20  h = 0.2500  yn = 6.2563  err = 0.0063'
ans = 'n = 19  h = 0.2632  yn = 6.2566  err = 0.0066'
ans = 'n = 18  h = 0.2778  yn = 6.2569  err = 0.0069'
ans = 'n = 17  h = 0.2941  yn = 6.2574  err = 0.0074'
ans = 'n = 16  h = 0.3125  yn = 6.2578  err = 0.0078'
ans = 'n = 15  h = 0.3333  yn = 6.2583  err = 0.0083'
ans = 'n = 14  h = 0.3571  yn = 6.2589  err = 0.0089'
ans = 'n = 13  h = 0.3846  yn = 6.2596  err = 0.0096'   <--
ans = 'n = 12  h = 0.4167  yn = 6.2604  err = 0.0104'
ans = 'n = 11  h = 0.4545  yn = 6.2614  err = 0.0114'
ans = 'n = 10  h = 0.5000  yn = 6.2625  err = 0.0125'
```

Thus, $h \approx \underline{0.3846} = \frac{5.0}{13}$ is the largest step size, with the fewest # of steps, yielding an error < 0.01. Code on next page.

```
clear
% backward Euler formula for y' = -10y + 2.5t^2 + 0.5t
afn = @(y, k1, k2, k3, d) (y + k1 + k2 + k3)/d;
y0 = 4;  % initial value
phi = 4*exp(-50) + 5.0^2/4  % exact value at t = 5.0
for h = [5/20, 5/19, 5/18, 5/17, 5/16, ...
         5/15, 5/14, 5/13, 5/12, 5/11, 5/10]
    n = round(5/h);    % make an integer step
    y = zeros(n,1);  % pre-allocate memory
    y(1) = y0;
    d = 1 + 10*h;  % divisor in formula
    k1 = 2.5*h^3;
    for i = 2:n
        k2 = (5*(i-1)*h + 0.5)*h^2;
        k3 = (2.5*((i-1)*h)^2 + 0.5*(i-1)*h)*h;
        y(i) = afn(y(i-1), k1, k2, k3, d);
    end
    err = abs(phi - y(n));
    sprintf('n = %2.0f  h = %6.4f  yn = %6.4f  err = %6.4f',n,h,y(n),err)
end
```

5.

(a) Using Laplace transform, $\mathcal{L}\{y'\} = s\mathcal{L}\{y\} - y(0) = s\mathcal{L}\{y\}$

$\therefore s\mathcal{L}\{y\} - \lambda\mathcal{L}\{y\} = \frac{1}{s} - \frac{\lambda}{s^2} = \frac{s-\lambda}{s^2}$

$\mathcal{L}\{y\} = \frac{1}{s-\lambda} \cdot \frac{s-\lambda}{s^2} = \frac{1}{s^2}$. $\mathcal{L}^{-1}\left\{\frac{1}{s^2}\right\} = t$

$\therefore y(t) = t$

(b) Using MATLAB, results and code next page.

T = 6×5 table

	t	k_1	k_10	k_20	k_50
1	0	0	0	0	0
2	0.2000	0.2000	0.2000	0.2000	0.2000
3	0.4000	0.4000	0.4000	0.4000	0.4000
4	0.6000	0.6000	0.6000	0.6000	0.6000
5	0.8000	0.8000	0.8000	0.8000	0.7905
6	1.0000	1.0000	1.0000	1.0000	-207.0715

```
clear
% afn = dy/dt = f(k,t,y)
afn = @(k,t,y) k*y + 1 - k*t;
y0 = 0;    % initial condition
h = 0.01;  % step size
h2 = h/2;  % shorten R-K calcs below
lambda = [1, 10, 20, 50];  % various values

% table display times
TableTimes = [0, 0.2, 0.4, 0.6, 0.8, 1.0];
NumRows = length(TableTimes);

% times to compute the y values
t = TableTimes(1):h:TableTimes(end);
% pre-allocate memory, y holds R-K values
y = zeros(length(t),1);

%initialize table array
%   # Cols = time + # lambda values
V = zeros(NumRows, 1 + length(lambda));

% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end
```

```
% populate rest of table values
Col = 1;   % initialize
y(1) = y0;
for k = lambda
    Col = Col + 1;  % set display col for each k
    Row = 1;
    V(Row,Col) = y(1);
    for i =2:length(t)
        % Runge-Kutta method
        k1 = afn(k, t(i-1), y(i-1));
        k2 = afn(k, t(i-1) + h2, y(i-1) + h2*k1);
        k3 = afn(k, t(i-1) + h2, y(i-1) + h2*k2);
        k4 = afn(k, t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;

        % display result only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'k_1', ...
                              'k_10', ...
                              'k_20', ...
                              'k_50'}
```

(c)

For $y(t) = t$, the method gives exact values except for large λ values (e.g., $\lambda = 50$). To explain this, look at the component $y(i) = y(i-1) + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)$. If

truncation error is the main culprit, decreasing the size of h should yield better consistency between large and small λ. The table below shows the results with h = 0.001.

T = 6×5 table

	t	k_1	k_10	k_20	k_50
1	0	0	0	0	0
2	0.2000	0.2000	0.2000	0.2000	0.2000
3	0.4000	0.4000	0.4000	0.4000	0.4000
4	0.6000	0.6000	0.6000	0.6000	0.6000
5	0.8000	0.8000	0.8000	0.8000	0.8000
6	1.0000	1.0000	1.0000	1.0000	1.0000

Thus truncation errors must add up with larger λ. Even with h = 0.01, taking the table out to t = 2.0 shows the errors add up even for λ = 20.

T = 6×5 table

	t	k_1	k_10	k_20	k_50
1	0	0	0	0	0
2	0.2000	0.2000	0.2000	0.2000	0.2000
3	0.4000	0.4000	0.4000	0.4000	0.4000
4	0.6000	0.6000	0.6000	0.6000	0.6000
5	0.8000	0.8000	0.8000	0.8000	0.7905
6	2.0000	2.0000	2.0000	1.1127	-1.0604e+24

With $h = 0.0001$, round off errors start to become important. The table below shows worse results with $h = 0.0001$ compared to $h = 0.01$.

T = 6×5 table

	t	k_1	k_10	k_20	k_50
1	0	0	0	0	0
2	0.2000	0.2000	0.2000	0.2000	0.2000
3	0.4000	0.4000	0.4000	0.4000	0.4000
4	0.6000	0.6000	0.6000	0.6000	0.5994
5	0.8000	0.8000	0.8000	0.8000	-13.2236
6	1.0000	1.0000	1.0000	1.0000	-3.0889e+05

$h = 0.0001$

The R-K method involves many multiplications, and $h = 0.0001$ means 10,000 step. That's a lot of multiplications for round-off error to add up, and a larger λ magnifies this, as λx is a bigger number than x.

6.

(a)

Using Laplace transform, $\mathcal{L}\{y'\} = s\mathcal{L}\{y\} - y(0) = s\mathcal{L}\{y\}$

$$s\mathcal{L}\{y\} - \lambda\mathcal{L}\{y\} = 2\left(\frac{1}{s^2}\right) - \lambda\left(\frac{2}{s^3}\right) = \frac{2s - 2\lambda}{s^3}$$

$$\therefore \mathcal{L}\{y\} = \frac{1}{s-\lambda} \cdot \frac{2(s-\lambda)}{s^3} = \frac{2}{s^3} \quad \mathcal{L}^{-1}\left\{\frac{2}{s^3}\right\} = t^2$$

$$\therefore \underline{y(t) = t^2}$$

(b)

Using MATLAB, same code as in #5, except

afn = @(λ,t,y) λy + 2t - λt^2

T = 6×5 table

	t	k_1	k_10	k_20	k_50
1	0	0	0	0	0
2	0.2000	0.0400	0.0400	0.0400	0.0312
3	0.4000	0.1600	0.1600	0.1598	-193.3485
4	0.6000	0.3600	0.3600	0.3477	-4.2477e+06
5	0.8000	0.6400	0.6399	-0.0288	-9.3239e+10
6	1.0000	1.0000	0.9996	-35.5143	-2.0467e+15

Code on next page.

h = 0.01

For $y(t) = t^2$, the values for $\lambda = 1$ are accurate.

They become less accurate as λ increases.

```
clear
% afn = dy/dt = f(k,t,y)
afn = @(k,t,y) k*y + 2*t - k*t^2;
y0 = 0;    % initial condition
h = 0.01;  % step size
h2 = h/2;  % shorten R-K calcs below
lambda = [1, 10, 20, 50];  % various values

% table display times
TableTimes = [0, 0.2, 0.4, 0.6, 0.8, 1.0];
NumRows = length(TableTimes);

% times to compute the y values
t = TableTimes(1):h:TableTimes(end);
% pre-allocate memory, y holds R-K values
y = zeros(length(t),1);

%initialize table array
%   # Cols = time + # lambda values
V = zeros(NumRows, 1 + length(lambda));

% Populate table times (Col = 1)
for i = 1:NumRows
    V(i,1) = TableTimes(i);
end

% populate rest of table values
Col = 1;   % initialize
y(1) = y0;
for k = lambda
    Col = Col + 1;  % set display col for each k
    Row = 1;
    V(Row,Col) = y(1);
    for i =2:length(t)
        % Runge-Kutta method
        k1 = afn(k, t(i-1), y(i-1));
        k2 = afn(k, t(i-1) + h2, y(i-1) + h2*k1);
        k3 = afn(k, t(i-1) + h2, y(i-1) + h2*k2);
        k4 = afn(k, t(i-1) + h, y(i-1) + h*k3);
        y(i) = y(i-1) + h*(k1 + 2*k2 + 2*k3 + k4)/6;

        % display result only at specified times
        if abs(t(i) - TableTimes(Row+1)) < 0.00001
            Row = Row + 1;
            V(Row,Col) = y(i);
        end
    end
end

format short
T = array2table(V);
% label the table columms and display the table
T.Properties.VariableNames = {'t', 'k_1', ...
                              'k_10', ...
                              'k_20', ...
                              'k_50'}
```

(c)

As in #5, the truncation errors get magnified as λ is larger.

For $h = 0.001$, the results are:

T = 6×5 table

	t	k_1	k_10	k_20	k_50
1	0	0	0	0	0
2	0.2000	0.0400	0.0400	0.0400	0.0400
3	0.4000	0.1600	0.1600	0.1600	0.1354
4	0.6000	0.3600	0.3600	0.3600	-542.4277
5	0.8000	0.6400	0.6400	0.6399	-1.1956e+07
6	1.0000	1.0000	1.0000	0.9960	-2.6334e+11

Thus, error don't become apparent till $\lambda = 20$.

Using MATLAB for this problem, y_n, the calculated value based on the prior value y, using the R-K algorithm, $y_n \simeq \frac{h^4(y-t^2)}{24}\lambda^4 + \text{lesser terms}$

Thus, for small differences in computation between y and t^2, for a fixed h, as λ increases, y_n gets large with each step. The code for this is on the next page.

```
clear
syms k t y h
h2 = h/2;
afn = @(k, t, y) k*y + 2*t - k*t^2;
k1 = afn(k, t, y);
k2 = afn(k, t + h2, y + h2*k1);
k3 = afn(k, t + h2, y + h2*k2);
k4 = afn(k, t + h, y + h*k3);
yn = y + h*(k1 + 2*k2 + 2*k3 + k4)/6;
collect(yn,k)
```

ans =

$$\frac{h^4\,(y-t^2)}{24}\,k^4+\frac{h\left(\frac{h^2\,(y-t^2)}{2}+\frac{h^2\sigma_1}{2}\right)}{6}\,k^3+\frac{h\left(h\sigma_1+h\left(y+\frac{h\,(h+2t)}{2}-\sigma_2\right)+h\,(y-t^2)\right)}{6}\,k^2+\frac{h\,(6y+2ht+2h\,(h+2t)-(h+t)^2-t^2-4\sigma_2)}{6}\,k+y+\frac{h\,(6h+12t)}{6}$$

where

$$\sigma_1 = y+ht-\sigma_2$$

$$\sigma_2 = \left(\frac{h}{2}+t\right)^2$$

The above shows, at one step, the R-K calculation for y_n based on y_{n-1}, h, t_{n-1}, and λ (λ is k in the MATLAB code). As λ gets large, λ^4 is much larger than λ^3, λ^2, λ, so the λ^4 term is most important. Even for small h, and if y is close to t^2, because of λ^4 factor, y_n can become very large.