5. Given the multistep method

$$w_{i+1} = -\frac{3}{2}w_i + 3w_{i-1} - \frac{1}{2}w_{i-2} + 3hf(t_i, w_i), \quad \text{for } i = 2, \ldots, N-1,$$

with starting values w_0, w_1, w_2:

a. Find the local truncation error.

b. Comment on consistency, stability, and convergence.

a). ① $y(t_{i+1}) = y(t_i) + \frac{h}{1!}y'(t_i) + \frac{h^2}{2!}y''(t_i) + \frac{h^3}{3!}y'''(t_i) + \frac{h^4}{4!}y^{(4)}(t_i)$

② $y(t_{i-1}) = y(t_i) + \frac{(-h)}{1!}y'(t_i) + \frac{(-h)^2}{2!}y''(t_i) + \frac{(-h)^3}{3!}y'''(t_i) + \frac{(-h)^4}{4!}y^{(4)}(t_i)$

③ $y(t_{i-2}) = y(t_i) + \frac{(-2h)}{1!}y'(t_i) + \frac{(-2h)^2}{2!}y''(t_i) + \frac{(-2h)^3}{3!}y^{(3)}(t_i) + \frac{(-2h)^4}{4!}y^{(4)}(t_i)$

$y(t_{i+1}) = -\frac{3}{2}y(t_i) + 3y(t_{i-1}) - \frac{1}{2}y(t_{i-2}) + 3h \cdot y'(t_i) \quad i \in [2, 3, 4 \ldots N-1]$

$y(t_{i+1}) + \frac{3}{2}y(t_i) - 3y(t_{i-1}) + \frac{1}{2}y(t_{i-2})$

$= \{y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(t_i) + \frac{h^3}{6}y'''(t_i) + \frac{h^4}{24}y^{(4)}(t_i) + \cdots\}$

$+ (\frac{3}{2}y(t_i)) - \{3y(t_i) - 3hy'(t_i) + \frac{3}{2}h^2y''(t_i) - \frac{h^3}{2}y'''(t_i) + \frac{h^4}{8}y^{(4)}(t_i) + \cdots\}$

$+ \{\frac{1}{2}y(t_i) - hy'(t_i) + h^2y''(t_i) - \frac{2h^3}{3}y'''(t_i) + \frac{1}{3}h^4y^{(4)}(t_i) + \cdots\}$

$= (3hy'(t_i) + \frac{h^4}{4}y^{(4)}(t_i) + \cdots)$

B/C $y(t_{i+1}) = -\frac{3}{2}y(t_i) + 3y(t_{i-1}) - \frac{1}{2}y(t_{i-2}) + 3hy'(t_i) + \frac{h^3}{4}y^{(4)}(\xi_i)$

Divide by h, wolfram gives:

$\Rightarrow \tau_{i+1}(h) \Rightarrow$ Error term $= \boxed{\frac{h^3}{4}y^{(4)}(\xi_i)}$

b) Take $\lim_{h \to 0} \tau_{i+1}(h) = \lim_{h \to 0}\{\frac{h^3}{4}y^{(4)}(\xi_i)\}$

$= \frac{1}{4}y^{(4)}(\xi_i)\lim_{h \to 0}(h^4)$

$= 0 \quad \Rightarrow$ Consistent

Take $m=3$, $a_0 = \frac{-1}{2}$ $a_1 = 3$ $a_2 = -\frac{3}{2}$

$$P(\lambda) = \lambda^3 + \frac{3}{2}\lambda^2 - 3\lambda + \frac{1}{2} = 0$$

$$(\lambda-1)(2\lambda^2+5\lambda-1)=0$$

Obviously it has 3 Roots, Wolfram gives

$\lambda_1=1$ $|\lambda_2|=0.18614$ $|\lambda_3|=2.68614$

These Root does not Satistify Root Condition

So it's unstable, it's divergent by thm 5.24

7. Investigate stability for the difference method

$$w_{i+1} = -4w_i + 5w_{i-1} + 2h[f(t_i, w_i) + 2hf(t_{i-1}, w_{i-1})],$$

for $i = 1, 2, \ldots, N-1$, with starting values w_0, w_1.

$m = 2$, $a_0 = 5$ $a_1 = -4$

$P(\lambda) = \lambda^2 + 4\lambda - 5 = 0$

$\Rightarrow \lambda_1 = 1$, $\lambda_2 = -5$

By def 5.23

This method is Unstable

11. The Backward Euler one-step method is defined by

$$w_{i+1} = w_i + hf(t_{i+1}, w_{i+1}), \quad \text{for } i = 0, \ldots, N-1.$$

Show that $Q(h\lambda) = 1/(1 - h\lambda)$ for the Backward Euler method.

$$\frac{dy}{dt} = \lambda \cdot y$$

$$w_{i+1} = \left(\frac{1}{1-h\cdot\lambda}\right) w_i$$

$$= w_i + h \cdot f(t_{i+1}, w_{i+1})$$

$$w_{i+1} - h\cdot\lambda\, w_{i+1} = w_i$$

$$w_{i+1}(1 - h\cdot\lambda) = w_i$$

$$w_{i+1} = \left(\frac{1}{1-h\cdot\lambda}\right) w_i$$

write $Q(h\lambda) = \frac{1}{1-h\cdot\lambda}$

$$w_{i+1} = Q(h\cdot\lambda)\, w_i$$

hence shown

b. $u_1' = -4u_1 - 2u_2 + \cos t + 4\sin t, \quad u_1(0) = 0;$

$u_2' = 3u_1 + u_2 - 3\sin t, \quad u_2(0) = -1; \quad 0 \le t \le 2; \quad h = 0.1;$

actual solutions $u_1(t) = 2e^{-t} - 2e^{-2t} + \sin t$ and $u_2(t) = -3e^{-t} + 2e^{-2t}$.

```
>> rk4system
i           t_i                       w_1,i                     u_1,i                     w_2,i
0         0.000000000    0.000000000    0.000000000    -1.000000000    -1.000000000    0.000000000
1         0.100000000    0.272041371    0.272046747    -1.077045487    -1.077050748    0.000007522
2         0.200000000    0.495481689    0.495490745    -1.115543329    -1.115552167    0.000012654
3         0.300000000    0.679521862    0.679533376    -1.124820190    -1.124831390    0.000016062
4         0.400000000    0.831387407    0.831400506    -1.112289511    -1.112302210    0.000018244
5         0.500000000    0.956713900    0.956727976    -1.083819502    -1.083833097    0.000019569
6         0.600000000    1.059862687    1.059877322    -1.044032404    -1.044046484    0.000020308
7         0.700000000    1.144179454    1.144194367    -0.996547692    -0.996561983    0.000020655
8         0.800000000    1.212205973    1.212220983    -0.944179530    -0.944193856    0.000020749
9         0.900000000    1.265853461    1.265868453    -0.889096951    -0.889111203    0.000020685
10        1.000000000    1.306544398    1.306559301    -0.832953645    -0.832967757    0.000020524
11        1.100000000    1.335328440    1.335343211    -0.776992999    -0.777006934    0.000020307
12        1.200000000    1.352976992    1.352991603    -0.722132992    -0.722146729    0.000020054
13        1.300000000    1.360060184    1.360074615    -0.669034699    -0.669048223    0.000019777
14        1.400000000    1.357009301    1.357023533    -0.618157470    -0.618170767    0.000019476
15        1.500000000    1.344167161    1.344181170    -0.569803291    -0.569816344    0.000019148
16        1.600000000    1.321828471    1.321842231    -0.524152358    -0.524165146    0.000018785
17        1.700000000    1.290271841    1.290285319    -0.481291536    -0.481304032    0.000018379
18        1.800000000    1.249784809    1.249797962    -0.441237050    -0.441249220    0.000017920
19        1.900000000    1.200682999    1.200695782    -0.403952511    -0.403964314    0.000017398
20        2.000000000    1.143324356    1.143336716    -0.369363183    -0.369374572    0.000016807
>>
```

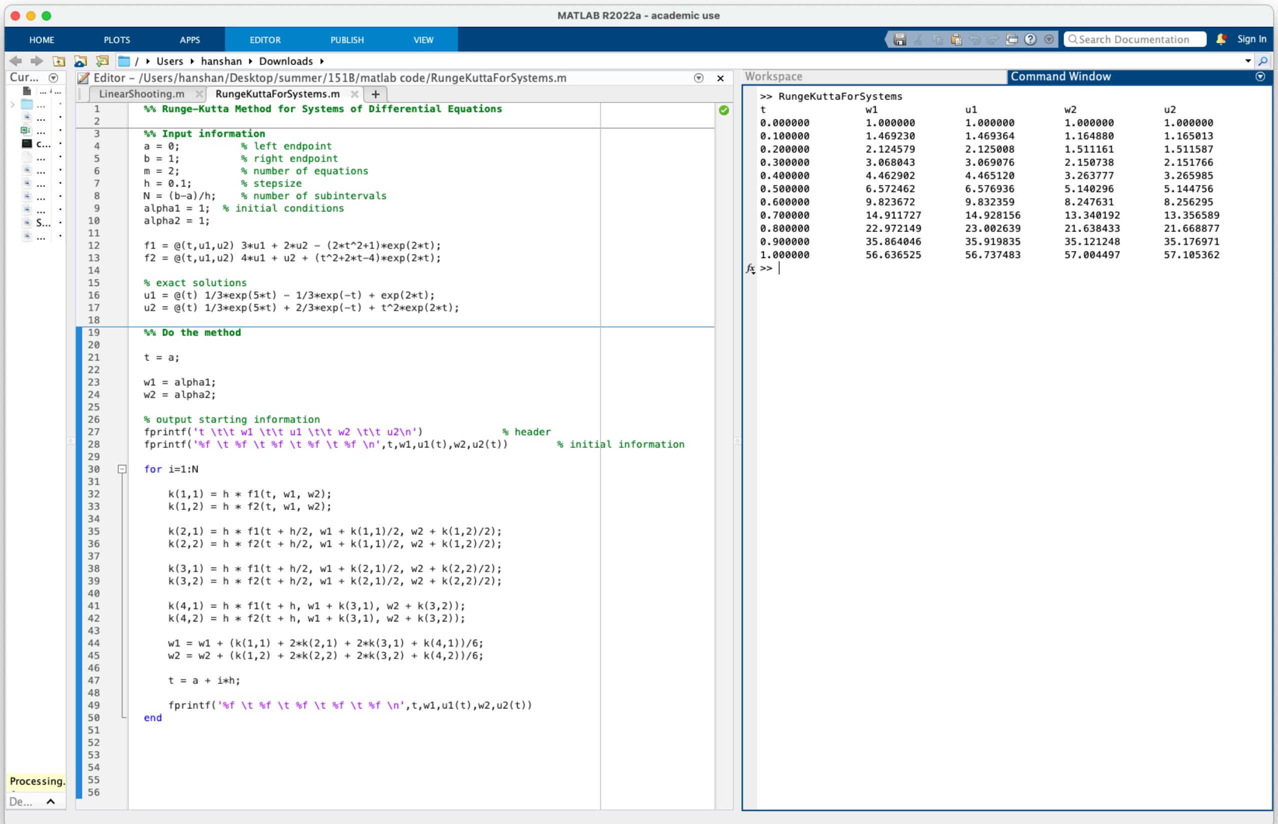
Approx Exact

```
%% Set up the problem parameters
t_start = 0;
t_end = 2;
alpha = [0,-1];
h = 0.1;


%% Compute the approximation, the true solution, and the error
[s,t,N] =  rk4(t_start, t_end,alpha,h);
true_soln = F(t);
err = zeros(N+1,1);
for i = 1:(N+1)
    err(i) = norm(s(i,:) - true_soln(i,:));
end

ts = zeros(100,1);
ts(:,1) = linspace(t_start,t_end).';
S = F(ts);


%% Plot the system in phase space
plot(s(:,1), s(:,2),'b*-');
hold on;
plot(S(:,1), S(:,2));
legend('Approximate Solution', 'True Solution');
xlabel('u_1');
ylabel('u_2');
hold off

%% Print the errors

fprintf('i\t \tt_i \t\t\t w_1,i \t\t\t u_1,i \t\t\t w_2,i \t\t\t y_2,i \t\t\t err_i \n')
for i= 1:(N+1)
    fprintf('%d\t  %.9f\t  %.9f\t  %.9f\t  %.9f\t  %.9f\t  %.9f\t\n', i-1, t(i), s(i,1), true_soln(i,1), s(i,2), true_soln(i,2), err(i));
end

%% The vector valued function of 2 equations defining the system of ODE
function s = f(t,r)
   s(1) = -4*r(1) - 2*r(2) + cos(t) +4*sin(t);
   s(2) = 3*r(1) + r(2) - 3*sin(t);
end

%% The vector valued true solution
function S = F(t)
    S(:,1) = 2*exp(-t) - 2*exp(-2*t) + sin(t);
    S(:,2) = -3*exp(-t) + 2* exp(-2*t);
end


%% The function implementing Runge-Kutta 4 for a system of 2 equations
function [w,t,N] = rk4(t_start,t_end,alpha,h)

    N = int16((t_end - t_start)/h);

    t = zeros(N+1,1);           % Initialize arrays
    w = zeros(N+1,2);

    t(1) = t_start;             % Initial conditions
    w(1,:) = alpha;

    for i = 1:(N)               % Begin looping
        t(i+1) = t(i) + h;

        k1 = h * f(t(i), w(i,:));
        k2 = h * f(t(i) + h/2, w(i,:) + k1/2);
        k3 = h * f(t(i) + h/2, w(i,:) + k2/2);
        k4 = h * f(t(i+1), w(i,:) + k3);

        w(i+1,:) = w(i,:) + (k1 + 2*k2 + 2*k3 + k4)/6;

    end
end
```

a. $u_1' = 3u_1 + 2u_2 - (2t^2+1)e^{2t}, \quad u_1(0) = 1;$

$u_2' = 4u_1 + u_2 + (t^2 + 2t - 4)e^{2t}, \quad u_2(0) = 1; \quad 0 \le t \le 1; \quad h = 0.2;$

actual solutions $u_1(t) = \frac{1}{3}e^{5t} - \frac{1}{3}e^{-t} + e^{2t}$ and $u_2(t) = \frac{1}{3}e^{5t} + \frac{2}{3}e^{-t} + t^2e^{2t}$



```matlab
%% Runge-Kutta Method for Systems of Differential Equations

%% Input information
a = 0;          % left endpoint
b = 1;          % right endpoint
m = 2;          % number of equations
h = 0.1;        % stepsize
N = (b-a)/h;    % number of subintervals
alpha1 = 1;  % initial conditions
alpha2 = 1;

f1 = @(t,u1,u2) 3*u1 + 2*u2 - (2*t^2+1)*exp(2*t);
f2 = @(t,u1,u2) 4*u1 + u2 + (t^2+2*t-4)*exp(2*t);

% exact solutions
u1 = @(t) 1/3*exp(5*t) - 1/3*exp(-t) + exp(2*t);
u2 = @(t) 1/3*exp(5*t) + 2/3*exp(-t) + t^2*exp(2*t);

%% Do the method

t = a;

w1 = alpha1;
w2 = alpha2;

% output starting information
fprintf('t \t\t w1 \t\t u1 \t\t w2 \t\t u2\n')            % header
fprintf('%f \t %f \t %f \t %f \t %f \n',t,w1,u1(t),w2,u2(t))        % initial information

for i=1:N

    k(1,1) = h * f1(t, w1, w2);
    k(1,2) = h * f2(t, w1, w2);

    k(2,1) = h * f1(t + h/2, w1 + k(1,1)/2, w2 + k(1,2)/2);
    k(2,2) = h * f2(t + h/2, w1 + k(1,1)/2, w2 + k(1,2)/2);

    k(3,1) = h * f1(t + h/2, w1 + k(2,1)/2, w2 + k(2,2)/2);
    k(3,2) = h * f2(t + h/2, w1 + k(2,1)/2, w2 + k(2,2)/2);

    k(4,1) = h * f1(t + h, w1 + k(3,1), w2 + k(3,2));
    k(4,2) = h * f2(t + h, w1 + k(3,1), w2 + k(3,2));

    w1 = w1 + (k(1,1) + 2*k(2,1) + 2*k(3,1) + k(4,1))/6;
    w2 = w2 + (k(1,2) + 2*k(2,2) + 2*k(3,2) + k(4,2))/6;

    t = a + i*h;

    fprintf('%f \t %f \t %f \t %f \t %f \n',t,w1,u1(t),w2,u2(t))
end
```

```
>> RungeKuttaForSystems
```

t	w1	u1	w2	u2
0.000000	1.000000	1.000000	1.000000	1.000000
0.100000	1.469230	1.469364	1.164880	1.165013
0.200000	2.124579	2.125008	1.511161	1.511587
0.300000	3.068043	3.069076	2.150738	2.151766
0.400000	4.462902	4.465120	3.263777	3.265985
0.500000	6.572462	6.576936	5.140296	5.144756
0.600000	9.823672	9.832359	8.247631	8.256295
0.700000	14.911727	14.928156	13.340192	13.356589
0.800000	22.972149	23.002639	21.638433	21.668877
0.900000	35.864046	35.919835	35.121248	35.176971
1.000000	56.636525	56.737483	57.004497	57.105362

11.1

3. Use the Linear Shooting method to approximate the solution to the following boundary-value problems.

a. $y'' = -3y' + 2y + 2x + 3, \quad 0 \le x \le 1,\ y(0) = 2,\ y(1) = 1;$ use $h = 0.1$.

b. $y'' = -4x^{-1}y' - 2x^{-2}y + 2x^{-2}\ln x, \quad 1 \le x \le 2,\ y(1) = -\frac{1}{2},\ y(2) = \ln 2;$ use $h = 0.05$.

1

```python
#Input
import numpy as np
a = 0
b = 1
alpha = 2
beta = 1
N = 10

def p(x):
    return -3
def q(x):
    return 2
def r(x):
    return  2*x+3
```

2

```python
h = (b-a)/N
u2=np.zeros(N+2)
u1=np.zeros(N+2)
v1=np.zeros(N+2)
v2=np.zeros(N+2)
u1[0]= alpha
u2[0] = 0
v1[0] = 0
v2[0] = 1


print(h)
```

3

```python
for i in range(0,N+1):
    x = a + i * h
    k11 = h * u2[i]
    k12 = h * ( p(x) * u2[i] + q(x) * u1[i] + r(x) )

    k21 = h * ( u2[i] + k12/2 )
    k22 = h * ( p(x + h/2) * ( u2[i] + k12/2 ) + q(x + h/2) * ( u1[i] + k11/2 ) + r(x + h/2) )

    k31 = h * ( u2[i] + k22/2 )
    k32 = h * ( p(x + h/2) * ( u2[i] + k22/2 ) + q(x + h/2) * ( u1[i] + k21/2 ) + r(x + h/2) )

    k41 = h * ( u2[i] + k32 )
    k42 = h * ( p(x + h) * ( u2[i] + k32 ) + q(x + h)*( u1[i] + k31 ) + r(x + h) )

    u1[i+1] = u1[i] + ( k11 + 2*k21 + 2*k31 + k41 )/6
    u2[i+1] = u2[i] + ( k12 + 2*k22 + 2*k32 + k42 )/6

    k_prime11 = h * v2[i]
    k_prime12 = h * ( p(x) * v2[i] + q(x) * v1[i] )

    k_prime21 = h * ( v2[i] + k_prime12/2 )
    k_prime22 = h * ( p(x + h/2) * ( v2[i] + k_prime12/2 ) + q(x + h/2) * ( v1[i] + k_prime11/2 ) )

    k_prime31 = h * ( v2[i] + k_prime22/2 )
    k_prime32 = h * ( p(x + h/2) * ( v2[i] + k_prime22/2 ) + q(x + h/2) * ( v1[i] + k_prime21/2 ) )

    k_prime41 = h * ( v2[i] + k_prime32 )
    k_prime42 = h * ( p(x + h) * ( v2[i] + k_prime32 ) + q(x + h)*( v1[i] + k_prime31) )

    v1[i+1] = v1[i] + (k_prime11 + 2*k_prime21 + 2*k_prime31 + k_prime41)/6
    v2[i+1] = v2[i] + (k_prime12 + 2*k_prime22 + 2*k_prime32 + k_prime42)/6

    print("x_i \t u1_i \t v1_i \n{:.2f}   {:.5f}   {:.5f}".format(x,u1[i],v1[i]))
w1=np.zeros(N+2)
w2=np.zeros(N+2)
```

4

```python
w1[0] = alpha
w2[0] = (beta - u1[-2])/(v1[-2])
#print(a,w1[0],w2[0])
print("---------------")
#import pandas as pd
for i in range(0,N+1):
    x = a + i*h
    W1 = u1[i] + w2[0]*v1[i]
    W2 = u2[i] + w2[0]*v2[i]
    print('x_i \t   W1 \t  W2 \n{:.2f}    {:.5f}    {:.5f}'.format(x ,W1,W2))
```

```
x_i     u1_i      v1_i
0.00    2.00000   0.00000
x_i     u1_i      v1_i
0.10    2.03213   0.08667
x_i     u1_i      v1_i
0.20    2.11883   0.15238
x_i     u1_i      v1_i
0.30    2.24919   0.20370
x_i     u1_i      v1_i
0.40    2.41589   0.24525
x_i     u1_i      v1_i
0.50    2.61412   0.28027
x_i     u1_i      v1_i
0.60    2.84087   0.31107
```

1

```
x_i     u1_i      v1_i
0.70    3.09441   0.33927
x_i     u1_i      v1_i
0.80    3.37391   0.36604
x_i     u1_i      v1_i
0.90    3.67922   0.39220
x_i     u1_i      v1_i
1.00    4.01065   0.41837
```

2

```
x_i     W1        W2
0.00    2.00000   -7.19616
x_i     W1        W2
0.10    1.40843   -4.77471
x_i     W1        W2
0.20    1.02226   -3.04606
x_i     W1        W2
0.30    0.78332   -1.80076
x_i     W1        W2
0.40    0.65104   -0.89203
x_i     W1        W2
0.50    0.59723   -0.21691
x_i     W1        W2
0.60    0.60235   0.29682
```

3

```
x_i     W1        W2
0.70    0.65295   0.69986
x_i     W1        W2
0.80    0.73986   1.02788
x_i     W1        W2
0.90    0.85689   1.30599
x_i     W1        W2
1.00    1.00000   1.55194
```

4

b)

```python
import numpy as np
a = 1
b = 2
alpha = -1/2
beta = np.log(2)
N = 20

def p(x):
    return -4/x
def q(x):
    return -2/x**2
def r(x):
    return  (2/x**2)*np.log(x)
```

1

```python
h = (b-a)/N
u2=np.zeros(N+2)
u1=np.zeros(N+2)
v1=np.zeros(N+2)
v2=np.zeros(N+2)
u1[0]= alpha
u2[0] = 0
v1[0] = 0
v2[0] = 1


print(h)
```

2

```python
for i in range(0,N+1):
    x = a + i * h
    k11 = h * u2[i]
    k12 = h * ( p(x) * u2[i] + q(x) * u1[i] + r(x) )

    k21 = h * ( u2[i] + k12/2 )
    k22 = h * ( p(x + h/2) * ( u2[i] + k12/2 ) + q(x + h/2) * ( u1[i] + k11/2 ) + r(x + h/2) )

    k31 = h * ( u2[i] + k22/2 )
    k32 = h * ( p(x + h/2) * ( u2[i] + k22/2 ) + q(x + h/2) * ( u1[i] + k21/2 ) + r(x + h/2) )

    k41 = h * ( u2[i] + k32 )
    k42 = h * ( p(x + h) * ( u2[i] + k32 ) + q(x + h)*( u1[i] + k31 ) + r(x + h) )

    u1[i+1] = u1[i] + ( k11 + 2*k21 + 2*k31 + k41 )/6
    u2[i+1] = u2[i] + ( k12 + 2*k22 + 2*k32 + k42 )/6

    k_prime11 = h * v2[i]
    k_prime12 = h * ( p(x) * v2[i] + q(x) * v1[i] )

    k_prime21 = h * ( v2[i] + k_prime12/2 )
    k_prime22 = h * ( p(x + h/2) * ( v2[i] + k_prime12/2 ) + q(x + h/2) * ( v1[i] + k_prime11/2 ) )

    k_prime31 = h * ( v2[i] + k_prime22/2 )
    k_prime32 = h * ( p(x + h/2) * ( v2[i] + k_prime22/2 ) + q(x + h/2) * ( v1[i] + k_prime21/2 ) )

    k_prime41 = h * ( v2[i] + k_prime32 )
    k_prime42 = h * ( p(x + h) * ( v2[i] + k_prime32 ) + q(x + h)*( v1[i] + k_prime31) )

    v1[i+1] = v1[i] + (k_prime11 + 2*k_prime21 + 2*k_prime31 + k_prime41)/6
    v2[i+1] = v2[i] + (k_prime12 + 2*k_prime22 + 2*k_prime32 + k_prime42)/6

    print("x_i \t u1_i \t v1_i \n{:.2f}   {:.5f}   {:.5f}".format(x,u1[i],v1[i]))
w1=np.zeros(N+2)
w2=np.zeros(N+2)
```

3

```python
w1[0] = alpha
w2[0] = (beta - u1[-2])/(v1[-2])
#print(a,w1[0],w2[0])
print("---------------")
#import pandas as pd
for i in range(0,N+1):
    x = a + i*h
    W1 = u1[i] + w2[0]*v1[i]
    W2 = u2[i] + w2[0]*v2[i]
    print('x_i \t   W1 \t  W2 \n{:.2f}   {:.5f}   {:.5f}'.format(x ,W1,W2))
```

4

```
x_i      u1_i     v1_i
1.00   -0.50000   0.00000
x_i      u1_i     v1_i
1.05   -0.49883   0.04535
x_i      u1_i     v1_i
1.10   -0.49560   0.08264
x_i      u1_i     v1_i
1.15   -0.49067   0.11342
x_i      u1_i     v1_i
1.20   -0.48434   0.13889
x_i      u1_i     v1_i
1.25   -0.47686   0.16000
x_i      u1_i     v1_i
1.30   -0.46840   0.17751
```

1

```
x_i      u1_i     v1_i
1.35   -0.45915   0.19204
x_i      u1_i     v1_i
1.40   -0.44924   0.20408
x_i      u1_i     v1_i
1.45   -0.43878   0.21403
x_i      u1_i     v1_i
1.50   -0.42787   0.22222
x_i      u1_i     v1_i
1.55   -0.41658   0.22893
x_i      u1_i     v1_i
1.60   -0.40500   0.23437
x_i      u1_i     v1_i
1.65   -0.39316   0.23875
```

2

```
x_i      u1_i     v1_i
1.70   -0.38114   0.24221
x_i      u1_i     v1_i
1.75   -0.36896   0.24490
x_i      u1_i     v1_i
1.80   -0.35666   0.24691
x_i      u1_i     v1_i
1.85   -0.34427   0.24836
x_i      u1_i     v1_i
1.90   -0.33183   0.24931
x_i      u1_i     v1_i
1.95   -0.31935   0.24984
x_i      u1_i     v1_i
2.00   -0.30685   0.25000
```

3

```
x_i        W1       W2
1.00   -0.50000   4.00001
x_i        W1       W2
1.05   -0.31742   3.32794
x_i        W1       W2
1.10   -0.16502   2.78739
x_i        W1       W2
1.15   -0.03699   2.34899
x_i        W1       W2
1.20    0.07121   1.99075
x_i        W1       W2
1.25    0.16314   1.69601
x_i        W1       W2
1.30    0.24165   1.45199
```

4

```
x_i       W1       W2
1.35   0.30902   1.24880
x_i       W1       W2
1.40   0.36708   1.07873
x_i       W1       W2
1.45   0.41734   0.93568
x_i       W1       W2
1.50   0.46102   0.81482
x_i       W1       W2
1.55   0.49913   0.71230
x_i       W1       W2
1.60   0.53250   0.62501
x_i       W1       W2
1.65   0.56184   0.55041
```

5

```
x_i       W1       W2
1.70   0.58772   0.48647
x_i       W1       W2
1.75   0.61064   0.43149
x_i       W1       W2
1.80   0.63100   0.38409
x_i       W1       W2
1.85   0.64915   0.34312
x_i       W1       W2
1.90   0.66540   0.30763
x_i       W1       W2
1.95   0.67999   0.27681
x_i       W1       W2
2.00   0.69315   0.25000
```

6

a. $y'' = -e^{-2y}, \quad 1 \le x \le 2, y(1) = 0, y(2) = \ln 2$; use $N = 10$; actual solution $y(x) = \ln x$.

```
%% Input Information
a = 1;              % left endpoint
b = 2;              % right endpoint
alpha = 0;        % boundary condition at left endpoint
beta = log(2);       % boundary condition at right endpoint
N = 10;             % number of subintervals
tol = 1e-4;         % tolerance
M = 10;             % maximum number of iterations

f = @(x,y,y_prime) -exp(-2*y);
partialf_partialy = @(x,y,y_prime) 2*exp(-2*y);
partialf_partialy_prime = @(x,y,y_prime) 0;
```

```
>> NonLinearShootingWithNewtonsMethod
x          w1       w2
1.000000          0.000000          1.000002
1.100000          0.095310          0.909092
1.200000          0.182321          0.833334
1.300000          0.262363          0.769231
1.400000          0.336471          0.714285
1.500000          0.405464          0.666666
1.600000          0.470002          0.624999
1.700000          0.530627          0.588235
1.800000          0.587785          0.555555
1.900000          0.641852          0.526315
2.000000          0.693145          0.499999
The procedure is complete for j = 4
>>
```

Approx

```
%% Do the method

h = (b-a)/N;
j = 1;
TK = (beta - alpha)/(b-a);

fprintf('x \t w1 \t w2 \n')

while(j <= M)
    w(1,1) = alpha;
    w(2,1) = TK;
    u1 = 0;
    u2 = 1;

    for i=2:N+1
        x = a + (i-2)*h;

        k(1,1) = h * w(2,i-1);
        k(1,2) = h * f( x, w(1,i-1), w(2,i-1) );

        k(2,1) = h * ( w(2,i-1) + k(1,2)/2 );
        k(2,2) = h * f( x + h/2, w(1,i-1) + k(1,1)/2, w(2,i-1) + k(1,2)/2 );

        k(3,1) = h * ( w(2,i-1) + k(2,2)/2 );
        k(3,2) = h * f( x + h/2, w(1,i-1) + k(2,1)/2, w(2,i-1) + k(2,2)/2 );

        k(4,1) = h * ( w(2,i-1) + k(3,2) );
        k(4,2) = h * f( x + h, w(1,i-1) + k(3,1), w(2,i-1) + k(3,2) );

        w(1,i) = w(1,i-1) + ( k(1,1) + 2*k(2,1) + 2*k(3,1) + k(4,1) )/6;
        w(2,i) = w(2,i-1) + ( k(1,2) + 2*k(2,2) + 2*k(3,2) + k(4,2) )/6;

        k_prime(1,1) = h * u2;
        k_prime(1,2) = h * ( partialf_partialy( x, w(1,i-1), w(2,i-1) ) * u1 ...
            + partialf_partialy_prime( x, w(1,i-1), w(2,i-1) )*u2 );

        k_prime(2,1) = h * ( u2 + k_prime(1,2)/2 );
        k_prime(2,2) = h * ( partialf_partialy( x + h/2, w(1,i-1), w(2,i-1) ) * ( u1 + k_prime(1,1)/2 ) ...
            + partialf_partialy_prime( x + h/2, w(1,i-1), w(2,i-1) ) * ( u2 + k_prime(1,2)/2 ) );


        k_prime(2,1) = h * ( u2 + k_prime(1,2)/2 );
        k_prime(2,2) = h * ( partialf_partialy( x + h/2, w(1,i-1), w(2,i-1) ) * ( u1 + k_prime(1,1)/2 ) ...
            + partialf_partialy_prime( x + h/2, w(1,i-1), w(2,i-1) ) * ( u2 + k_prime(1,2)/2 ) );

        k_prime(3,1) = h * ( u2 + k_prime(2,2)/2 );
        k_prime(3,2) = h * ( partialf_partialy( x + h/2, w(1,i-1), w(2,i-1) ) * ( u1 + k_prime(2,1)/2 ) ...
            + partialf_partialy_prime( x + h/2, w(1,i-1), w(2,i-1) ) * ( u2 + k_prime(2,2)/2 ) );

        k_prime(4,1) = h * ( u2 + k_prime(3,2) );
        k_prime(4,2) = h * ( partialf_partialy( x + h, w(1,i-1), w(2,i-1) ) * ( u1 + k_prime(3,1) ) ...
            + partialf_partialy_prime( x + h, w(1,i-1), w(2,i-1) ) * ( u2 + k_prime(3,2) ) );

        u1 = u1 + ( k_prime(1,1) + 2*k_prime(2,1) + 2*k_prime(3,1) + k_prime(4,1) )/6;
        u2 = u2 + ( k_prime(1,2) + 2*k_prime(2,2) + 2*k_prime(3,2) + k_prime(4,2) )/6;
    end

    if(abs(w(1,N+1) - beta) <= tol)
        for i = 1:N+1
            x = a + (i-1) * h;
            fprintf('%f \t %f \t %f \n',x,w(1,i),w(2,i))
        end

        fprintf('The procedure is complete for j = %d \n',j)
        break;
    end

    TK = TK - ( w(1,N+1) - beta )/u1;

    j = j+1;
end
```

b. $y'' = y' \cos x - y \ln y, \quad 0 \le x \le \frac{\pi}{2},\ y(0) = 1,\ y\left(\frac{\pi}{2}\right) = e$; use $N = 10$; actual solution $y(x) = e^{\sin x}$.

```
%% Input Information
a = 0;           % left endpoint
b = pi/2;           % right endpoint
alpha = 1;      % boundary condition at left endpoint
beta = exp(1);     % boundary condition at right endpoint
N = 10;          % number of subintervals
tol = 1e-4;      % tolerance
M = 10;          % maximum number of iterations

f = @(x,y,y_prime) y_prime *cos(x)-y*log(y);
partialf_partialy = @(x,y,y_prime) -y_prime * sin(y)-log(y)-1;
partialf_partialy_prime = @(x,y,y_prime) 1/y^2+y_prime * sin(y);
```

```
%% Do the method

h = (b-a)/N;
j = 1;
TK = (beta - alpha)/(b-a);

fprintf('x \t w1 \t w2 \n')

while(j <= M)
    w(1,1) = alpha;
    w(2,1) = TK;
    u1 = 0;
    u2 = 1;

    for i=2:N+1
        x = a + (i-2)*h;

        k(1,1) = h * w(2,i-1);
        k(1,2) = h * f( x, w(1,i-1), w(2,i-1) );

        k(2,1) = h * ( w(2,i-1) + k(1,2)/2 );
        k(2,2) = h * f( x + h/2, w(1,i-1) + k(1,1)/2, w(2,i-1) + k(1,2)/2 );

        k(3,1) = h * ( w(2,i-1) + k(2,2)/2 );
        k(3,2) = h * f( x + h/2, w(1,i-1) + k(2,1)/2, w(2,i-1) + k(2,2)/2 );

        k(4,1) = h * ( w(2,i-1) + k(3,2) );
        k(4,2) = h * f( x + h, w(1,i-1) + k(3,1), w(2,i-1) + k(3,2) );

        w(1,i) = w(1,i-1) + ( k(1,1) + 2*k(2,1) + 2*k(3,1) + k(4,1) )/6;
        w(2,i) = w(2,i-1) + ( k(1,2) + 2*k(2,2) + 2*k(3,2) + k(4,2) )/6;

        k_prime(1,1) = h * u2;
        k_prime(1,2) = h * ( partialf_partialy( x, w(1,i-1), w(2,i-1) ) * u1 ...
            + partialf_partialy_prime( x, w(1,i-1), w(2,i-1) )*u2 );

        k_prime(2,1) = h * ( u2 + k_prime(1,2)/2 );
        k_prime(2,2) = h * ( partialf_partialy( x + h/2, w(1,i-1), w(2,i-1) ) * ( u1 + k_prime(1,1)/2 ) ...
            + partialf_partialy_prime( x + h/2, w(1,i-1), w(2,i-1) ) * ( u2 + k_prime(1,2)/2 ) );


        k_prime(2,1) = h * ( u2 + k_prime(1,2)/2 );
        k_prime(2,2) = h * ( partialf_partialy( x + h/2, w(1,i-1), w(2,i-1) ) * ( u1 + k_prime(1,1)/2 ) ...
            + partialf_partialy_prime( x + h/2, w(1,i-1), w(2,i-1) ) * ( u2 + k_prime(1,2)/2 ) );

        k_prime(3,1) = h * ( u2 + k_prime(2,2)/2 );
        k_prime(3,2) = h * ( partialf_partialy( x + h/2, w(1,i-1), w(2,i-1) ) * ( u1 + k_prime(2,1)/2 ) ...
            + partialf_partialy_prime( x + h/2, w(1,i-1), w(2,i-1) ) * ( u2 + k_prime(2,2)/2 ) );

        k_prime(4,1) = h * ( u2 + k_prime(3,2) );
        k_prime(4,2) = h * ( partialf_partialy( x + h, w(1,i-1), w(2,i-1) ) * ( u1 + k_prime(3,1) ) ...
            + partialf_partialy_prime( x + h, w(1,i-1), w(2,i-1) ) * ( u2 + k_prime(3,2) ) );

        u1 = u1 + ( k_prime(1,1) + 2*k_prime(2,1) + 2*k_prime(3,1) + k_prime(4,1) )/6;
        u2 = u2 + ( k_prime(1,2) + 2*k_prime(2,2) + 2*k_prime(3,2) + k_prime(4,2) )/6;
    end

    if(abs(w(1,N+1) - beta) <= tol)
        for i = 1:N+1
            x = a + (i-1) * h;
            fprintf('%f \t %f \t %f \n',x,w(1,i),w(2,i))
        end

        fprintf('The procedure is complete for j = %d \n',j)
        break;
    end

    TK = TK - ( w(1,N+1) - beta )/u1;

    j = j+1;
end
```

```
>> NonlinearShootingWithNewtonsMethod
x         w1        w2
0.000000        1.000000        1.000088
0.157080        1.169350        1.155040
0.314159        1.362119        1.295534
0.471239        1.574635        1.403085
0.628319        1.800067        1.456350
0.785398        2.028199        1.434207
0.942478        2.245793        1.320085
1.099557        2.437681        1.106710
1.256637        2.588543        0.799915
1.413717        2.685118        0.420042
1.570796        2.718376        -0.000017
The procedure is complete for j = 7
```

Approx

a. $y'' = -3y' + 2y + 2x + 3, \quad 0 \le x \le 1, y(0) = 2, y(1) = 1$; use $h = 0.1$.

11.3

```python
import numpy as np
import pandas as pd

def dif(aa,bb,alpha,beta,n):

    a = np.zeros([n+2])  # 2 cuz we need w_0 and w_n+1
    b = np.zeros([n+2])
    c = np.zeros([n+2])
    d = np.zeros([n+2])

    # n number of x points
    h = (bb-aa)/(n+1)
    #print(h)
    x = aa + h
    a[1] = 2.0 + (h**2)*q(x)
    b[1] = -1.0 + (h/2)*p(x)
    d[1] = -(h**2)*r(x) + (1 + (h/2)*p(x))*alpha

    for i in range(2,n):
        x = aa + i*h
        a[i] = 2.0 + (h**2)*q(x)
        b[i] = -1.0 + (h/2)*p(x)
        c[i] = -1.0 - (h/2)*p(x)
        d[i] = -(h**2)*r(x)

    x = bb-h
    a[n] = 2.0 + (h**2)*q(x)
    c[n] = -1.0 - (h/2)*p(x)
    d[n] = -(h**2)*r(x) + (1.0 - (h/2)*p(x))*beta

    l = np.zeros([n+2])
    u = np.zeros([n+2])
    z = np.zeros([n+2])

    # Crout algorithm
    l[1] = a[1]
    u[1] = b[1]/a[1]
    z[1] = d[1]/l[1]

    for i in range(2,n):
        l[i] = a[i]-c[i]*u[i-1]
        u[i] = b[i]/l[i]
        z[i] = (d[i] - c[i]*z[i-1])/l[i]


    l[n] = a[n] - c[n]*u[n-1]
    z[n] = (d[n] -c[n]*z[n-1])/l[n]

    w = np.zeros([n+2])

    w[0] = alpha
    w[n+1] = beta
    w[n] = z[n]

    for i in range(n-1,0,-1):
        w[i] = z[i] - u[i]*w[i+1]
    return w
```

```python
def p(x):
    return -3

def q(x):
    return 2

def r(x):
    return 2*x + 3

def main():
    a = 0.0 #left bound
    b = 1.0 #right bound
    alpha = 2.0 #left outcome
    beta = 1.0 #right outcome
    n = 9 #stepsize?

    w = dif(a,b,alpha,beta,n)

    x = np.linspace(a,b,n+2) # add x_0 and x_n+1
    df= df = pd.DataFrame({'x_i' : x, 'w_i' : w})
    print(df)

main()
```

	x_i	w_i
0	0.0	2.000000
1	0.1	1.405352
2	0.2	1.018097
3	0.3	0.779135
4	0.4	0.647367
5	0.5	0.594274
6	0.6	0.600150
7	0.7	0.651452
8	0.8	0.738961
9	0.9	0.856494
10	1.0	1.000000

APProx

b. $y'' = -4x^{-1}y' + 2x^{-2}y - 2x^{-2}\ln x, \quad 1 \le x \le 2,\ y(1) = -\frac{1}{2},\ y(2) = \ln 2$; use $h = 0.05$.

```python
import numpy as np
import pandas as pd

def dif(aa,bb,alpha,beta,n):

    a = np.zeros([n+2])  # 2 cuz we need w_0 and w_n+1
    b = np.zeros([n+2])
    c = np.zeros([n+2])
    d = np.zeros([n+2])

    # n number of x points
    h = (bb-aa)/(n+1)
    #print(h)
    x = aa + h
    a[1] = 2.0 + (h**2)*q(x)
    b[1] = -1.0 + (h/2)*p(x)
    d[1] = -(h**2)*r(x) + (1 + (h/2)*p(x))*alpha

    for i in range(2,n):
        x = aa + i*h
        a[i] = 2.0 + (h**2)*q(x)
        b[i] = -1.0 + (h/2)*p(x)
        c[i] = -1.0 - (h/2)*p(x)
        d[i] = -(h**2)*r(x)

    x = bb-h
    a[n] = 2.0 + (h**2)*q(x)
    c[n] = -1.0 - (h/2)*p(x)
    d[n] = -(h**2)*r(x) + (1.0 - (h/2)*p(x))*beta

    l = np.zeros([n+2])
    u = np.zeros([n+2])
    z = np.zeros([n+2])
    # Crout algorithm
    l[1] = a[1]
    u[1] = b[1]/a[1]
    z[1] = d[1]/l[1]

    for i in range(2,n):
        l[i] = a[i]-c[i]*u[i-1]
        u[i] = b[i]/l[i]
        z[i] = (d[i] - c[i]*z[i-1])/l[i]


    l[n] = a[n] - c[n]*u[n-1]
    z[n] = (d[n] -c[n]*z[n-1])/l[n]

    w = np.zeros([n+2])

    w[0] = alpha
    w[n+1] = beta
    w[n] = z[n]

    for i in range(n-1,0,-1):
        w[i] = z[i] - u[i]*w[i+1]
    return w
```

	x_i	w_i
0	1.00	-0.500000
1	1.05	-0.311147
2	1.10	-0.156628
3	1.15	-0.028817
4	1.20	0.077958
5	1.25	0.167972
6	1.30	0.244487
7	1.35	0.310022
8	1.40	0.366543
9	1.45	0.415599
10	1.50	0.458424
11	1.55	0.496006
12	1.60	0.529145
13	1.65	0.558494
14	1.70	0.584590
15	1.75	0.607873
16	1.80	0.628715
17	1.85	0.647422
18	1.90	0.664255
19	1.95	0.679434
20	2.00	0.693147

Approx

```python
def p(x):
    return -4/x

def q(x):
    return 2/x**2

def r(x):
    return (-2/x**2)*np.log(x)

def main():
    a = 1.0 #left bound
    b = 2.0 #right bound
    alpha = -0.5 #left outcome
    beta = np.log(2) #right outcome
    n = 19 #stepsize?

    w = dif(a,b,alpha,beta,n)

    x = np.linspace(a,b,n+2) # add x_0 and x_n+1
    df= df = pd.DataFrame({'x_i' : x, 'w_i' : w})
    print(df)

main()
```