problems, and compare the results to the actual values.

a. $y' = y/t - (y/t)^2, \quad 1 \le t \le 2, \quad y(1) = 1$, with $h = 0.1$; actual solution $y(t) = t/(1 + \ln t)$.

```
>> ModifiedEulerMethod
i       t_i             w_i             y(t_i)          |y(t_i) - w_i|
0       1.000000000     1.000000000     1.000000000     0.000000000
1       1.100000000     1.004132231     1.004281728     0.000149497
2       1.200000000     1.014713674     1.014952314     0.000238640
3       1.300000000     1.029519692     1.029813689     0.000293997
4       1.400000000     1.047204371     1.047533919     0.000329549
5       1.500000000     1.066909315     1.067262354     0.000353039
6       1.600000000     1.088063734     1.088432687     0.000368953
7       1.700000000     1.110275064     1.110655052     0.000379988
8       1.800000000     1.133265741     1.133653557     0.000387816
9       1.900000000     1.156834929     1.157228433     0.000393504
10      2.000000000     1.180834469     1.181232218     0.000397749
>>
```

```matlab
%% Inputs

a = 1;          % left endpoint
b = 2;          % right endpoint
h = 0.1;        % stepsize
N = (b-a)/h;    % the number of steps
alpha = 1;      % initial y value

f = @(t,y) y/t-(y/t)^2;        % as in dy/dt = f(t,y);

%% Modified Euler

t = zeros(1,N+1);       % stores all the t values
w = zeros(1,N+1);       % stores all the approximation values

t(1) = a;
w(1) = alpha;

for i=1:N
    t(i+1) = a + i*h;
    w(i+1) = w(i) + (h/2)*( f(t(i),w(i)) + f(t(i+1), w(i) + h*f(t(i),w(i)) ) );
end
```

```matlab
%% Compute the actual errors, error bound, and print information

error = zeros(1,N+1);
fprintf('i\tt_i\t\tw_i\t\ty(t_i)\t\t|y(t_i) - w_i|\n')

for i=1:N+1
    error(i) = abs( y(t(i)) - w(i) );                  % | y(t_i) - w_i |
    fprintf('%d\t%.9f\t%.9f\t%.9f\t%.9f\n',i-1,t(i),w(i),y(t(i)),error(i))
end
```

b. $y' = 1 + y/t + (y/t)^2, \quad 1 \le t \le 3, \quad y(1) = 0$, with $h = 0.2$; actual solution $y(t) = t\tan(\ln t)$.

```
>> ModifiedEulerMethod
i	t_i		w_i		y(t_i)		|y(t_i) - w_i)|
0	1.000000000	0.000000000	0.000000000	0.000000000
1	1.200000000	0.219444444	0.221242773	0.001798328
2	1.400000000	0.485049474	0.489681664	0.004632189
3	1.600000000	0.804011613	0.812752741	0.008741128
4	1.800000000	1.184855971	1.199438640	0.014582669
5	2.000000000	1.638422893	1.661281756	0.022858863
6	2.200000000	2.178877213	2.213501813	0.034624601
7	2.400000000	2.825065093	2.876551420	0.051486327
8	2.600000000	3.602524716	3.678475331	0.075950614
9	2.800000000	4.546613553	4.658665058	0.112051505
10	3.000000000	5.707569910	5.874099978	0.166530068
>>
```

```
%% Inputs

a = 1;          % left endpoint
b = 3;          % right endpoint
h = 0.2;        % stepsize
N = (b-a)/h;    % the number of steps
alpha = 0;      % initial y value

f = @(t,y) 1+y/t+(y/t)^2;        % as in dy/dt = f(t,y);

%% Modified Euler

t = zeros(1,N+1);       % stores all the t values
w = zeros(1,N+1);       % stores all the approximation values

t(1) = a;
w(1) = alpha;

for i=1:N
    t(i+1) = a + i*h;
    w(i+1) = w(i) + (h/2)*( f(t(i),w(i)) + f(t(i+1), w(i) + h*f(t(i),w(i)) ) );
end

%% Plot the approximation
```

```
%% Compute the actual errors, error bound, and print information

error = zeros(1,N+1);
fprintf('i\tt_i\t\tw_i\t\ty(t_i)\t\t|y(t_i) - w_i)|\n')

for i=1:N+1
    error(i) = abs( y(t(i)) - w(i) );                  % | y(t_i) - w_i |
    fprintf('%d\t%.9f\t%.9f\t%.9f\t%.9f\n',i-1,t(i),w(i),y(t(i)),error(i))
end
```

7. Repeat Exercise 3 using the Midpoint method.

problems, and compare the results to the actual values.

a. $y' = y/t - (y/t)^2, \quad 1 \le t \le 2, \quad y(1) = 1$, with $h = 0.1$; actual solution $y(t) = t/(1 + \ln t)$.

```
>> MidpointMethod
i       t_i             w_i             y(t_i)          |y(t_i) - w_i)|
0       1.000000000     1.000000000     1.000000000     0.000000000
1       1.100000000     1.004535147     1.004281728     0.000253419
2       1.200000000     1.015325665     1.014952314     0.000373351
3       1.300000000     1.030247009     1.029813689     0.000433320
4       1.400000000     1.047998180     1.047533919     0.000464261
5       1.500000000     1.067742740     1.067262354     0.000480386
6       1.600000000     1.088921367     1.088432687     0.000488680
7       1.700000000     1.111147810     1.110655052     0.000492758
8       1.800000000     1.134148124     1.133653557     0.000494568
9       1.900000000     1.157723623     1.157228433     0.000495190
10      2.000000000     1.181727456     1.181232218     0.000495237
```

```matlab
%% Inputs

  a = 1;            % left endpoint
  b = 2;            % right endpoint
  h = 0.1;          % stepsize
  N = (b-a)/h;      % the number of steps
  alpha = 1;        % initial y value

  f = @(t,y) y/t-(y/t)^2;        % as in dy/dt = f(t,y);

%% Midpoint Method

t = zeros(1,N+1);       % stores all the t values
w = zeros(1,N+1);       % stores all the approximation values

t(1) = a;
w(1) = alpha;

for i=1:N
    t(i+1) = a + i*h;
    w(i+1) = w(i) + +h*f( t(i)+h/2, w(i) + (h/2)*f(t(i),w(i)) );
end
%% Compute the actual errors, error bound, and print information

error = zeros(1,N+1);
fprintf('i\tt_i\t\tw_i\t\ty(t_i)\t\t|y(t_i) - w_i)|\n')

for i=1:N+1
    error(i) = abs( y(t(i)) - w(i) );                % | y(t_i) - w_i |
    fprintf('%d\t%.9f\t%.9f\t%.9f\t%.9f\n',i-1,t(i),w(i),y(t(i)),error(i))
end
```

b. $y' = 1 + y/t + (y/t)^2, \quad 1 \le t \le 3, \quad y(1) = 0$, with $h = 0.2$; actual solution $y(t) = t\tan(\ln t)$.

```
>> MidpointMethod
i       t_i             w_i             y(t_i)          |y(t_i) - w_i)|
0       1.000000000     0.000000000     0.000000000     0.000000000
1       1.200000000     0.219834711     0.221242773     0.001408062
2       1.400000000     0.486177047     0.489681664     0.003504617
3       1.600000000     0.806184892     0.812752741     0.006567849
4       1.800000000     1.188439258     1.199438640     0.010999382
5       2.000000000     1.643888921     1.661281756     0.017392834
6       2.200000000     2.186860893     2.213501813     0.026640921
7       2.400000000     2.836435713     2.876551420     0.040115707
8       2.600000000     3.618492555     3.678475331     0.059982776
9       2.800000000     4.568894384     4.658665058     0.089770675
10      3.000000000     5.738647465     5.874099978     0.135452513
```

```matlab
%% Inputs

a = 1;              % left endpoint
b = 3;              % right endpoint
h = 0.2;            % stepsize
N = (b-a)/h;        % the number of steps
alpha = 0;          % initial y value

f = @(t,y) 1+y/t+(y/t)^2;          % as in dy/dt = f(t,y);
```

```matlab
%% Midpoint Method

t = zeros(1,N+1);       % stores all the t values
w = zeros(1,N+1);       % stores all the approximation values

t(1) = a;
w(1) = alpha;

for i=1:N
    t(i+1) = a + i*h;
    w(i+1) = w(i) + +h*f( t(i)+h/2, w(i) + (h/2)*f(t(i),w(i)) );
end
```

```matlab
%% Compute the actual errors, error bound, and print information

error = zeros(1,N+1);
fprintf('i\tt_i\t\tw_i\t\ty(t_i)\t\t|y(t_i) - w_i)|\n')

for i=1:N+1
    error(i) = abs( y(t(i)) - w(i) );                    % | y(t_i) - w_i |
    fprintf('%d\t%.9f\t%.9f\t%.9f\t%.9f\n',i-1,t(i),w(i),y(t(i)),error(i))
end
```

problems, and compare the results to the actual values.

a. $y' = y/t - (y/t)^2, \quad 1 \le t \le 2, \quad y(1) = 1$, with $h = 0.1$; actual solution $y(t) = t/(1 + \ln t)$.

RK4

```
>> RungeKuttaOrder4
i        t_i            w_i            y(t_i)         |y(t_i) - w_i|
0        1.000000000    1.000000000    1.000000000    0.000000000
1        1.100000000    1.004281504    1.004281728    0.000000224
2        1.200000000    1.014952003    1.014952314    0.000000311
3        1.300000000    1.029813343    1.029813689    0.000000346
4        1.400000000    1.047533558    1.047533919    0.000000361
5        1.500000000    1.067261988    1.067262354    0.000000366
6        1.600000000    1.088432319    1.088432687    0.000000368
7        1.700000000    1.110654685    1.110655052    0.000000367
8        1.800000000    1.133653191    1.133653557    0.000000366
9        1.900000000    1.157228069    1.157228433    0.000000364
10       2.000000000    1.181231856    1.181232218    0.000000363
>>
```

```
%% Inputs

a = 1;           % left endpoint
b = 2;           % right endpoint
h = 0.1;         % stepsize
N = (b-a)/h;     % the number of steps
alpha = 1;       % initial y value

f = @(t,y) y/t-(y/t)^2  ;         % as in dy/dt = f(t,y);
```

```
%% Runge Kutta Order 4

t = zeros(1,N+1);       % stores all the t values
w = zeros(1,N+1);       % stores all the approximation values

t(1) = a;
w(1) = alpha;

for i=1:N
    t(i+1) = a + 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) + (1/6)*(k1 + 2*k2 + 2*k3 + k4);
end
```

```
%% Compute the actual errors, error bound, and print information

error = zeros(1,N+1);
fprintf('i\tt_i\t\tw_i\t\ty(t_i)\t\t|y(t_i) - w_i)|\n')

for i=1:N+1
    error(i) = abs( y(t(i)) - w(i) );                   % | y(t_i) - w_i |
    fprintf('%d\t%.9f\t%.9f\t%.9f\t%.9f\n',i-1,t(i),w(i),y(t(i)),error(i))
end
```

b. $y' = 1 + y/t + (y/t)^2, \quad 1 \le t \le 3, \quad y(1) = 0$, with $h = 0.2$; actual solution $y(t) = t\tan(\ln t)$.

```
>> RungeKuttaOrder4
i       t_i             w_i             y(t_i)          |y(t_i) - w_i)|
0       1.000000000     0.000000000     0.000000000     0.000000000
1       1.200000000     0.221245707     0.221242773     0.000002935
2       1.400000000     0.489684166     0.489681664     0.000002503
3       1.600000000     0.812752162     0.812752741     0.000000579
4       1.800000000     1.199432022     1.199438640     0.000006619
5       2.000000000     1.661265115     1.661281756     0.000016640
6       2.200000000     2.213469317     2.213501813     0.000032497
7       2.400000000     2.876494115     2.876551420     0.000057305
8       2.600000000     3.678378996     3.678475331     0.000096335
9       2.800000000     4.658506284     4.658665058     0.000158775
10      3.000000000     5.873838570     5.874099978     0.000261408
```

```
%% Inputs

a = 1;            % left endpoint
b = 3;            % right endpoint
h = 0.2;          % stepsize
N = (b-a)/h;      % the number of steps
alpha = 0;        % initial y value

f = @(t,y) 1+y/t+(y/t)^2  ;        % as in dy/dt = f(t,y);

%% Runge Kutta Order 4

t = zeros(1,N+1);        % stores all the t values
w = zeros(1,N+1);        % stores all the approximation values

t(1) = a;
w(1) = alpha;

for i=1:N
    t(i+1) = a + 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) + (1/6)*(k1 + 2*k2 + 2*k3 + k4);
end

%% Compute the actual errors, error bound, and print information

error = zeros(1,N+1);
fprintf('i\tt_i\t\tw_i\t\ty(t_i)\t\t|y(t_i) - w_i)|\n')

for i=1:N+1
    error(i) = abs( y(t(i)) - w(i) );                    % | y(t_i) - w_i |
    fprintf('%d\t%.9f\t%.9f\t%.9f\t%.9f\n',i-1,t(i),w(i),y(t(i)),error(i))
end
```

29. Show that the Midpoint method and the Modified Euler method give the same approximations to the initial-value problem

$$y' = -y + t + 1, \quad 0 \le t \le 1, \quad y(0) = 1,$$

for any choice of h. Why is this true?

$w_0 = 1$ in Both MidPoint & Modified Euler method

Mid Point Method:

$w_{i+1} = w_i + h \cdot f(t_i + 0.5h, w_i + 0.5h \cdot f(t_i, w_i))$

$= w_i + h \cdot f(t_i + 0.5h, w_i + 0.5h \cdot (-w_i + t_i + 1))$

$= w_i + h(-w_i - 0.5h(-w_i + t_i + 1) + t_i + 0.5h + 1)$

$= w_i(1 - h + 0.5h^2) + t_i(h - 0.5h^2) + h$

Modified Euler's method

$w_{i+1} = w_i + 0.5h(f(t_i, w_i) + f(t_{i+1}, w_i + hf(t_i, w_i)))$

$= w_i + 0.5h(-w_i + t_i + 1 - w_i - h(-w_i + t_i + 1)$

$= w_i(1 - h + 0.5h^2) + t_i(h - 0.5h^2) + h$

3. Use the Runge-Kutta-Fehlberg method with tolerance $TOL = 10^{-6}$, $hmax = 0.5$, and $hmin = 0.05$ to approximate the solutions to the following initial-value problems. Compare the results to the actual values.

a. $y' = y/t - (y/t)^2, \quad 1 \le t \le 4, \quad y(1) = 1;$ actual solution $y(t) = t/(1 + \ln t)$.

```
%% Inputs

a = 1;                  % left endpoint
b = 4;                  % right endpoint
alpha = 1;            % initial y value
tol = 1e-6;             %tolerance
hmax = 0.5;            % maximum step size
hmin = 0.05;            % minimum step size

f = @(t,y) y/t-(y/t)^2;                % as in dy/dt = f(t,y);
y = @(t) t/(1+log(t));     % exact solution
```

```
%% Runge-Kutta-Fehlberg

t = a;
w = alpha;
h = hmax;
FLAG = 1;
N = (b-a)/hmin;
i = 1;
%j = 1;

fprintf('i \t t \t \t y_i = y(t_i) \t w_i \t\t h_i \t\t R \n')
fprintf('%d \t %f \t %f \t %f \n',0,t(1),y(t(1)),w(1))

while(FLAG == 1 && i < N+1) %% j < N+1?
    format long
    K1 = h * f(t, w);
    K2 = h * f(t + h/4, w + K1/4);
    K3 = h * f(t + 3*h/8, w + 3*K1/32 + 9*K2/32);
    K4 = h * f(t + 12*h/13, w + 1932*K1/2197 - 7200*K2/2197 + 7296*K3/2197);
    K5 = h * f(t + h, w + 439*K1/216 - 8*K2 + 3680*K3/513 - 845*K4/4104);
    K6 = h * f(t + h/2, w - 8*K1/27 + 2*K2 - 3544*K3/2565 + 1859*K4/4104 - 11*K5/40);

    R = (1/h)*abs( K1/360 - 128*K3/4275 - 2197*K4/75240 + K5/50 + 2*K6/55 );    % approximates the LTE

    if(R <= tol)
        t = t + h;
        w = w + 25*K1/216 + 1408*K3/2565 + 2197*K4/4104 - K5/5;
        %i = i+1; ???
        % move output stuff here?
    else
        disp("Ooops, might need to adjust the indices....");
    end

    % output stuff -- can move this inside the after line 45
    fprintf('%d \t %f \t %f \t %f \t %f \t %.9f \n',i,t,y(t),w,h,R)

    % choose a new stepsize
    delta = 0.84*(tol/R)^(1/4);
    if(delta <= 0.1)
        h = 0.1*h;
    else
        if(delta >= 4)
            h = 4*h;
        else
            h = delta*h;
        end
    end

    % check step size
    if(h > hmax)
        h = hmax;
    end

    if(t >= b)
        FLAG = 0;
    else
        if(t + h > b)
            h = b - t;
        else
            if(h < hmin)
                FLAG = 0;
                disp("Minimum h exceeded");
            end
        end
    end

    i = i + 1;
    %j = j+1;
end
```

```
>> RungeKuttaFehlberg
i        t              y_i = y(t_i)    w_i             h_i             R
0        1.000000       1.000000        1.000000
Ooops, might need to adjust the indices....
1        1.000000       1.000000        1.000000        0.500000        0.000099294
Ooops, might need to adjust the indices....
2        1.000000       1.000000        1.000000        0.133051        0.000001058
3        1.110195       1.005124        1.005124        0.110195        0.000000521
4        1.219131       1.017521        1.017521        0.108937        0.000000193
5        1.357269       1.039675        1.039675        0.138138        0.000000208
6        1.529011       1.073276        1.073276        0.171742        0.000000192
7        1.747058       1.121395        1.121395        0.218047        0.000000179
8        2.028642       1.188170        1.188170        0.281583        0.000000166
9        2.399435       1.279539        1.279540        0.370793        0.000000152
10       2.898515       1.404184        1.404184        0.499080        0.000000136
11       3.398515       1.528564        1.528564        0.500000        0.000000040
12       3.898515       1.651496        1.651496        0.500000        0.000000014
13       4.000000       1.676239        1.676239        0.101485        0.000000000
```

b. $y' = 1 + y/t + (y/t)^2, \quad 1 \le t \le 3, \quad y(1) = 0$, with $h = 0.2$; actual solution $y(t) = t\tan(\ln t)$.

```
>> RungeKuttaFehlberg
i         t            y_i = y(t_i)    w_i             h_i             R
0         1.000000     0.000000        0.000000
summer time sadness
1         1.000000     0.000000        0.000000        0.500000        0.000070341
2         1.145027     0.156023        0.156024        0.145027        0.000000143
summer time sadness
3         1.145027     0.156023        0.156024        0.198058        0.000002162
4         1.282230     0.325497        0.325497        0.137204        0.000000455
5         1.422575     0.523264        0.523264        0.140345        0.000000775
6         1.548224     0.723412        0.723412        0.125649        0.000000654
7         1.665576     0.932028        0.932029        0.117352        0.000000605
8         1.777365     1.152147        1.152148        0.111790        0.000000585
9         1.884723     1.385123        1.385123        0.107357        0.000000577
10        1.988200     1.631694        1.631695        0.103477        0.000000573
11        2.088100     1.892329        1.892330        0.099900        0.000000572
12        2.184602     2.167350        2.167351        0.096503        0.000000572
13        2.277824     2.456993        2.456995        0.093222        0.000000572
14        2.367849     2.761440        2.761442        0.090025        0.000000574
15        2.454744     3.080835        3.080837        0.086895        0.000000575
16        2.538569     3.415295        3.415298        0.083825        0.000000576
17        2.619383     3.764919        3.764922        0.080814        0.000000578
18        2.697246     4.129790        4.129794        0.077863        0.000000579
19        2.772221     4.509983        4.509987        0.074974        0.000000580
20        2.844373     4.905561        4.905566        0.072152        0.000000582
21        2.913773     5.316583        5.316588        0.069400        0.000000583
22        2.980493     5.743101        5.743107        0.066720        0.000000584
23        3.000000     5.874100        5.874106        0.019507        0.000000005
```

```matlab
%% Inputs

a = 1;                  % left endpoint
b = 3;                  % right endpoint
alpha = 0;            % initial y value
tol = 1e-6;             %tolerance
hmax = 0.5;            % maximum step size
hmin = 0.05;            % minimum step size

f = @(t,y) 1+(y/t)+(y/t)^2;              % as in dy/dt = f(t,y);
y = @(t) t*tan(log(t));     % exact solution
```

```matlab
%% Runge-Kutta-Fehlberg

t = a;
w = alpha;
h = hmax;
FLAG = 1;
N = (b-a)/hmin;
i = 1;
%j = 1;

fprintf('i \t t \t \t y_i = y(t_i) \t w_i \t\t h_i \t\t R \n')
fprintf('%d \t %f \t %f \t %f \n',0,t(1),y(t(1)),w(1))

while(FLAG == 1 && i < N+1) %% j < N+1?
    format long
    K1 = h * f(t, w);
    K2 = h * f(t + h/4, w + K1/4);
    K3 = h * f(t + 3*h/8, w + 3*K1/32 + 9*K2/32);
    K4 = h * f(t + 12*h/13, w + 1932*K1/2197 - 7200*K2/2197 + 7296*K3/2197);
    K5 = h * f(t + h, w + 439*K1/216 - 8*K2 + 3680*K3/513 - 845*K4/4104);
    K6 = h * f(t + h/2, w - 8*K1/27 + 2*K2 - 3544*K3/2565 + 1859*K4/4104 - 11*K5/40);

    R = (1/h)*abs( K1/360 - 128*K3/4275 - 2197*K4/75240 + K5/50 + 2*K6/55 );    % approximates the LTE

    if(R <= tol)
        t = t + h;
        w = w + 25*K1/216 + 1408*K3/2565 + 2197*K4/4104 - K5/5;
        %i = i+1; ???
        % move output stuff here?
    else
        disp("summer time sadness")%"Ooops, might need to adjust the indices....");
    end

    % output stuff -- can move this inside the after line 45
    fprintf('%d \t %f \t %f \t %f \t %f \t %.9f \n',i,t,y(t),w,h,R)

    % choose a new stepsize
    delta = 0.84*(tol/R)^(1/4);
    if(delta <= 0.1)
        h = 0.1*h;
    else
        if(delta >= 4)
            h = 4*h;
        else
            h = delta*h;
        end
    end

    if(t >= b)
        FLAG = 0;
    else
        if(t + h > b)
            h = b - t;
        else
            if(h < hmin)
                FLAG = 0;
                disp("Minimum h exceeded");
            end
        end
    end

    i = i + 1;
    %j = j+1;
end
```

a. $y' = y/t - (y/t)^2, \quad 1 \le t \le 2, \quad y(1) = 1,$ with $h = 0.1$; actual solution $y(t) = \dfrac{t}{1 + \ln t}$.

3 Step Adams - Bashforth method

```
%% Inputs

a = 1;          % left endpoint
b = 2;          % right endpoint
h = 0.1;        % stepsize
N = (b-a)/h;    % the number of steps
alpha = 1;      % initial y value

f = @(t,y) (y/t)-(y/t)^2;        % as in dy/dt = f(t,y);

%% Adams-Bashforth 3-Step

t = zeros(1,N+1);       % stores all the t values
w = zeros(1,N+1);       % stores all the approximation values

t(1) = a;
w(1) = alpha;

% run Runge-Kutta for two steps to get w(2), w(3)
for i=1:2
    t(i+1) = a + 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) + (1/6)*(k1 + 2*k2 + 2*k3 + k4);
end

for i=3:N
    t(i+1) = a + i*h;
    w(i+1) = w(i) + (h/12)*(23*f(t(i),w(i)) -16*f(t(i-1),w(i-1)) + 5*f(t(i-2),w(i-2)));
end

%% Compute the actual errors, error bound, and print information

error = zeros(1,N+1);
fprintf('i\tt_i\t\tw_i\t\ty(t_i)\t\t|y(t_i) - w_i)|\n')

for i=1:N+1
    error(i) = abs( y(t(i)) - w(i) );                % | y(t_i) - w_i |
    fprintf('%d\t%.9f\t%.9f\t%.9f\t%.9f\n',i-1,t(i),w(i),y(t(i)),error(i))
end
```

```
>> AdamsBashforth3Step
```

i	t_i	w_i	y(t_i)	|y(t_i) - w_i)|
0	1.000000000	1.000000000	1.000000000	0.000000000
1	1.100000000	1.004281504	1.004281728	0.000000224
2	1.200000000	1.014952003	1.014952314	0.000000311
3	1.300000000	1.029357854	1.029813689	0.000455835
4	1.400000000	1.046873038	1.047533919	0.000660882
5	1.500000000	1.066478787	1.067262354	0.000783567
6	1.600000000	1.087583699	1.088432687	0.000848988
7	1.700000000	1.109769135	1.110655052	0.000885918
8	1.800000000	1.132746544	1.133653557	0.000907013
9	1.900000000	1.156309151	1.157228433	0.000919282
10	2.000000000	1.180305680	1.181232218	0.000926538

```
>> AdamsBashforth4Step
i       t_i             w_i             y(t_i)          |y(t_i) - w_i)|
0       1.000000000     1.000000000     1.000000000     0.000000000
1       1.100000000     1.004281504     1.004281728     0.000000224
2       1.200000000     1.014952003     1.014952314     0.000000311
3       1.300000000     1.029813343     1.029813689     0.000000346
4       1.400000000     1.047727830     1.047533919     0.000193911
5       1.500000000     1.067536218     1.067262354     0.000273864
6       1.600000000     1.088756654     1.088432687     0.000323967
7       1.700000000     1.110999394     1.110655052     0.000344342
8       1.800000000     1.134009322     1.133653557     0.000355765
9       1.900000000     1.157589883     1.157228433     0.000361450
10      2.000000000     1.181596676     1.181232218     0.000364457
>>
```

4 step

Adams - Bashforth method

```
%% Inputs

a = 1;          % left endpoint
b = 2;          % right endpoint
h = 0.1;        % stepsize
N = (b-a)/h;    % the number of steps
alpha = 1;      % initial y value

f = @(t,y) (y/t)-(y/t)^2;        % as in dy/dt = f(t,y);

%% Adams-Bashforth 4-Step

t = zeros(1,N+1);       % stores all the t values
w = zeros(1,N+1);       % stores all the approximation values

t(1) = a;
w(1) = alpha;

% need w_0, w_1, w_2, w_3

% run Runge-Kutta for two steps to get w(2), w(3), w(4)
for i=1:3
    t(i+1) = a + 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) + (1/6)*(k1 + 2*k2 + 2*k3 + k4);
end

for i=4:N
    t(i+1) = a + i*h;
    w(i+1) = w(i) + (h/24)*(55*f(t(i),w(i)) -59*f(t(i-1),w(i-1)) + 37*f(t(i-2),w(i-2)) -9*f(t(i-3),w(i-3)));
end
```

```
%% Compute the actual errors, error bound, and print information

error = zeros(1,N+1);
fprintf('i\tt_i\t\tw_i\t\ty(t_i)\t\t|y(t_i) - w_i)|\n')

for i=1:N+1
    error(i) = abs( y(t(i)) - w(i) );                  % | y(t_i) - w_i |
    fprintf('%d\t%.9f\t%.9f\t%.9f\t%.9f\n',i-1,t(i),w(i),y(t(i)),error(i))
end
```

b. $y' = 1 + y/t + (y/t)^2, \quad 1 \leq t \leq 3, \quad y(1) = 0,$ with $h = 0.2$; actual solution $y(t) = t \tan(\ln t)$.

```
>> AdamsBashforth3Step
i       t_i             w_i             y(t_i)          |y(t_i) - w_i|
0       1.000000000     0.000000000     0.000000000     0.000000000
1       1.200000000     0.221245707     0.221242773     0.000002935
2       1.400000000     0.489684166     0.489681664     0.000002503
3       1.600000000     0.812431708     0.812752741     0.000321033
4       1.800000000     1.198210986     1.199438640     0.001227654
5       2.000000000     1.658431349     1.661281756     0.002850406
6       2.200000000     2.207998700     2.213501813     0.005503113
7       2.400000000     2.866767174     2.876551420     0.009784246
8       2.600000000     3.661748384     3.678475331     0.016726947
9       2.800000000     4.630527549     4.658665058     0.028137510
10      3.000000000     5.826800808     5.874099978     0.047299170
>>
```

```
%% Inputs

a = 1;          % left endpoint
b = 3;          % right endpoint
h = 0.2;        % stepsize
N = (b-a)/h;    % the number of steps
alpha = 0;      % initial y value

f = @(t,y) 1+(y/t)+(y/t)^2;        % as in dy/dt = f(t,y);

%% Adams-Bashforth 3-Step

t = zeros(1,N+1);       % stores all the t values
w = zeros(1,N+1);       % stores all the approximation values

t(1) = a;
w(1) = alpha;

% run Runge-Kutta for two steps to get w(2), w(3)
for i=1:2
    t(i+1) = a + 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) + (1/6)*(k1 + 2*k2 + 2*k3 + k4);
end

for i=3:N
    t(i+1) = a + i*h;
    w(i+1) = w(i) + (h/12)*(23*f(t(i),w(i)) -16*f(t(i-1),w(i-1)) + 5*f(t(i-2),w(i-2)));
end
```

3 Step

```
%% Compute the actual errors, error bound, and print information

error = zeros(1,N+1);
fprintf('i\tt_i\t\tw_i\t\ty(t_i)\t\t|y(t_i) - w_i|\n')

for i=1:N+1
    error(i) = abs( y(t(i)) - w(i) );                 % | y(t_i) - w_i |
    fprintf('%d\t%.9f\t%.9f\t%.9f\t%.9f\n',i-1,t(i),w(i),y(t(i)),error(i))
end
```

4 Step

```
>> AdamsBashforth4Step
```

i	t_i	w_i	y(t_i)	|y(t_i) - w_i)|
0	1.000000000	0.000000000	0.000000000	0.000000000
1	1.200000000	0.221245707	0.221242773	0.000002935
2	1.400000000	0.489684166	0.489681664	0.000002503
3	1.600000000	0.812752162	0.812752741	0.000000579
4	1.800000000	1.199042213	1.199438640	0.000396427
5	2.000000000	1.660305996	1.661281756	0.000975759
6	2.200000000	2.211744785	2.213501813	0.001757029
7	2.400000000	2.873531978	2.876551420	0.003019442
8	2.600000000	3.673326649	3.678475331	0.005148682
9	2.800000000	4.649893698	4.658665058	0.008771360
10	3.000000000	5.858994384	5.874099978	0.015105594

```
%% Inputs

a = 1;          % left endpoint
b = 3;          % right endpoint
h = 0.2;        % stepsize
N = (b-a)/h;    % the number of steps
alpha = 0;      % initial y value

f = @(t,y) 1+(y/t)+(y/t)^2;        % as in dy/dt = f(t,y);

%% Adams-Bashforth 4-Step

t = zeros(1,N+1);       % stores all the t values
w = zeros(1,N+1);       % stores all the approximation values

t(1) = a;
w(1) = alpha;

% need w_0, w_1, w_2, w_3

% run Runge-Kutta for two steps to get w(2), w(3), w(4)
for i=1:3
    t(i+1) = a + 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) + (1/6)*(k1 + 2*k2 + 2*k3 + k4);
end

for i=4:N
    t(i+1) = a + i*h;
    w(i+1) = w(i) + (h/24)*(55*f(t(i),w(i)) -59*f(t(i-1),w(i-1)) + 37*f(t(i-2),w(i-2)) -9*f(t(i-3),w(i-3)));
end
```

1. To prove Theorem 5.20, part (i), show that the hypotheses imply that there exists a constant $K > 0$ such that

5.10

$$|u_i - v_i| \le K|u_0 - v_0|, \quad \text{for each } 1 \le i \le N,$$

whenever $\{u_i\}_{i=1}^N$ and $\{v_i\}_{i=1}^N$ satisfy the difference equation $w_{i+1} = w_i + h\phi(t_i, w_i, h)$.

Theorem 5.20 Suppose the initial-value problem

$$y' = f(t, y), \quad a \le t \le b, \quad y(a) = \alpha,$$

is approximated by a one-step difference method in the form

$$w_0 = \alpha,$$
$$w_{i+1} = w_i + h\phi(t_i, w_i, h).$$

Suppose also that a number $h_0 > 0$ exists and that $\phi(t, w, h)$ is continuous and satisfies a Lipschitz condition in the variable w with Lipschitz constant L on the set

$$D = \{(t, w, h) \mid a \le t \le b \text{ and } -\infty < w < \infty, 0 \le h \le h_0\}.$$

Then

(i) The method is stable;

(ii) The difference method is convergent if and only if it is consistent, which is equivalent to

$$\phi(t, y, 0) = f(t, y), \quad \text{for all } a \le t \le b;$$

(iii) If a function τ exists and, for each $i = 1, 2, \ldots, N$, the local truncation error $\tau_i(h)$ satisfies $|\tau_i(h)| \le \tau(h)$ whenever $0 \le h \le h_0$, then

$$|y(t_i) - w_i| \le \frac{\tau(h)}{L} e^{L(t_i - a)}.$$

$|u_i - v_i| \le k|u_0 - v_0|$ is of the form :

$|f(t, y_1) - f(t, y_2)| \le L|y_1 - y_2|$ $\Leftarrow$ L's condition

$u_{i+1} = u_i + h\phi(t_i, u_i, h)$

$v_{i+1} = v_i + h\phi(t_i, v_i, h)$

$\Rightarrow u_{i+1} - v_{i+1} = u_{i+1} - v_{i+1} + h(\phi(t_i, u_i, h) - \phi(t_i, v_i, h))$

$$\begin{aligned}|u_{i+1} - v_{i+1}| &= |u_{i+1} - v_{i+1} + h(\phi(t_i, u_i, h) - \phi(t_i, v_i, h))| \\ &\le |u_i - v_i| + h|\phi(t_i, u_i, h) - \phi(t_i, v_i, h)| \\ &\le |u_i - v_i| + hL|u_i - v_i| \\ &= (1 + hL)|u_i - v_i| \\ &= (1 + hL)^{n+1}|u_0 - v_0|\end{aligned}$$

$\Rightarrow k := (1 + hL)^{n+1}$

$\Rightarrow |u_i - v_i| \le k|u_0 - v_0|$

$\square$