Caleb Logemann
AER E 546 Fluid Mechanics and Heat Transfer I
Homework 1

#1

(a) How many 'data' points are needed to obtain a third order accurate polynomial approximation? Derive a finite difference formula for $\partial T/\partial x$ that is third order accurate in Δx. Use only the minimum number of points.

Four data points are needed to obtain a third order accurate polynomial approximation, as the taylor series for a function, f with four coefficients is of the following form.

$$f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \frac{1}{6}f'''(x_0)(x - x_0)^3 + O\Big((x - x_0)^4\Big)$$

Note that this polynomial has errors of order $(x - x_0)^4$, so the approximation is third order accurate.

In order to derive a finite difference formula for $\frac{\partial T}{\partial x}$ that is third order accurate I will first find a third order accurate polynomial approximation using four points. The four points I will use will be equally spaces with spacing Δx and the will be labeled x_{-2}, x_{-1}, x_0, x_1 with function values f_{-2}, f_{-1}, f_0, f_1 respectively. The polynomial approximation will solve the following equations for a, b, c, and d.

$$\begin{aligned}
f_{-2} &= a + b(-2\Delta x) + c(-2\Delta x)^2 + d(-2\Delta x)^3 \\
f_{-1} &= a + b(-\Delta x) + c(-\Delta x)^2 + d(-\Delta x)^3 \\
f_0 &= a \\
f_1 &= a + b\Delta x + c(\Delta x)^2 + d(\Delta x)^3
\end{aligned}$$

Clearly $a = f_0$. The 2nd and 4th equations can be added to solve for c. Summing these equations gives

$$\begin{aligned}
f_{-1} + f_1 &= 2f_0 + 2c(\Delta x)^2 \\
c &= \frac{f_{-1} + f_1 - 2f_0}{2(\Delta x)^2}
\end{aligned}$$

Summing the first equation and -2 times the second equation gives

$$\begin{aligned}
f_{-2} - 2f_{-1} &= -f_0 + 2c(\Delta x)^2 + -6d(\Delta x)^3 \\
f_{-2} - 2f_{-1} &= -f_0 + f_{-1} + f_1 - 2f_0 + -6d(\Delta x)^3 \\
f_{-2} - 3f_{-1} + 3f_0 - 1f_1 &= -6d(\Delta x)^3 \\
d &= \frac{f_{-2} - 3f_{-1} + 3f_0 - 1f_1}{-6(\Delta x)^3}
\end{aligned}$$

Plugging all these values into the final equation allows for b to be found.

$$\begin{aligned}
f_1 &= f_0 + b\Delta x + \frac{f_{-1} + f_1 - 2f_0}{2} + \frac{-f_{-2} + 3f_{-1} - 3f_0 + 1f_1}{6} \\
\frac{6f_1}{6} &= \frac{6f_0}{6} + b\Delta x + \frac{3f_{-1} + 3f_1 - 6f_0}{6} + \frac{-f_{-2} + 3f_{-1} - 3f_0 + 1f_1}{6} \\
\frac{f_{-2} - 6f_{-1} + 3f_0 + 2f_1}{6} &= b\Delta x \\
b &= \frac{f_{-2} - 6f_{-1} + 3f_0 + 2f_1}{6\Delta x}
\end{aligned}$$

Now that we have a polynomial approximation of

$$p(x) = a + b(x - x_0) + c(x - x_0)^2 + d(x - x_0)^3$$

where the values of a, b, c, and d were computed above. We can now compute the first derivative of this approximation, which is

$$p'(x) = b + 2c(x - x_0) + 3d(x - x_0)^2.$$

The first derivative at x_0 is thus b, or $p'(x_0) = b$. Therefore a third order approximation of the first derivative at the point x_0 is

$$b = \frac{f_{-2} - 6f_{-1} + 3f_0 + 2f_1}{6\Delta x}$$

(b) Derive the second order accurate centered difference formula for $\dfrac{\partial^2 T}{\partial x^2}$.

First I will derive the second order polynomial approximation for three points centered around x_0. I will label the points x_{-1}, x_0, and x_1 with function values f_{-1}, f_0, and f_1 respectively. The third order accurate polynomial approximation will be of the form

$$a + b(x - x_0) + c(x - x_0)^2$$

Note that the second derivative of this approximation is always $2c$, so the formula for 2 times c will also be the centered finite difference for the second derivative of second order. Finding this approximation amounts to solving the following three equations.

$$\begin{aligned} f_{-1} &= a - b\Delta x + c(\Delta x)^2 \\ f_0 &= a \\ f_1 &= a + b\Delta x + c(\Delta x)^2 \end{aligned}$$

Clearly $a = f_0$. The first and third equations can be summed to find c.

$$\begin{aligned} f_{-1} + f_1 &= 2f_0 + 2c(\Delta x)^2 \\ c &= \frac{f_{-1} - 2f_0 + f_1}{2(\Delta x)^2} \end{aligned}$$

Thus we don't even need to solve for b because the second order central finite difference for the second derivative is

$$\frac{f_{-1} - 2f_0 + f_1}{(\Delta x)^2}$$

#2

(a) The equation for a damped oscillator is

$$\ddot{Y} + \sigma\dot{Y} + \omega^2 Y = 0.$$

Let the non-dimensional frequency be $\omega = 1$. Consider the two damping rates $\sigma = 0.0$ and $\sigma = 0.5$. Solve this by RK2, out to $t = 32$, with the intial conditions $Y(0) = 1$ and $\dot{Y}(0) = 0$. The time-step can be $\Delta t = 32/N$, where N is the number of integration points. Plot solutions with $N = 21, 101, 301$. What is the analytical solution? Compare your numerical solutions to the exact result.

First I will compute the analytical solution to this differential equation. This can be done by finding the characteristic polynomial of the equation, which is

$$r^2 + \sigma r + 1 = 0.$$

Using the quadractic formula, we see that the roots of this polynomial are $r = -\frac{\sigma}{2} \pm \frac{\sqrt{\sigma^2-4}}{2}$. When $\sigma = 0.0$, the roots are $r = \pm i$. In the case of complex roots the general solution will be

$$Y(t) = c_1 \cos(t) + c_2 \sin(t).$$

Using the intial conditions we see that the exact solution is

$$Y(t) = \cos(t).$$

When $\sigma = 0.5$ the roots are $r = -\frac{1}{4} \pm \frac{\sqrt{15}}{4} i$. In this case the general solution is

$$Y(t) = e^{-\frac{1}{4}t}\left(c_1 \cos\left(\frac{\sqrt{15}}{4}t\right) + c_2 \sin\left(\frac{\sqrt{15}}{4}t\right)\right)$$

and the exact solution with boundary conditions is

$$Y(t) = e^{-\frac{1}{4}t} \cos\left(\frac{\sqrt{15}}{4}t\right).$$

Now in order to solve this equation numerically with RK2, we first need to transform this second order differential equation into a system of first order differential equations. To do this let $Z = \dot{Y}$, then the system becomes

$$\begin{aligned} \dot{Y} &= Z \\ \dot{Z} &= -\sigma Z - \omega^2 Y \end{aligned}$$

This is in the form $\dot{x} = RHS(x)$ where

$$\begin{aligned} x &= [Y, Z]^T \\ RHS(x) &= brx_2, -\sigma x_2 - \omega^2 x_1{}^T. \end{aligned}$$

The following is a method for running RK2 given a function to evaluate the RHS.

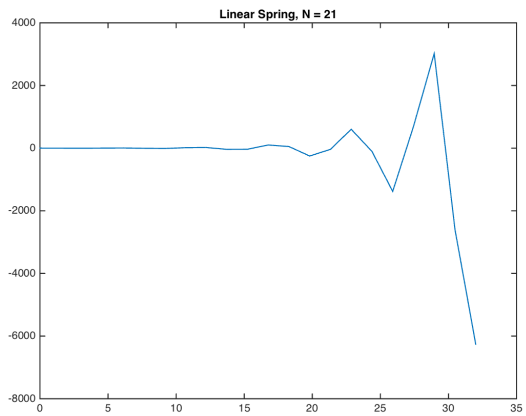
```
function [result] = RK2(RHSFunc, x0, nTimeSteps, tFinal)
    nEquations = length(x0);
    result = zeros(nTimeSteps+1, nEquations);
    result(1,:) = x0;

    deltaT = tFinal/nTimeSteps;

    for i = 1:nTimeSteps
        t = (i-1)*deltaT;
        temp = result(i,:) + 0.5*deltaT*RHSFunc(t, result(i,:));
        result(i+1, :) = result(i,:) + deltaT*RHSFunc(t + 1/2*deltaT, temp);
    end
end
```

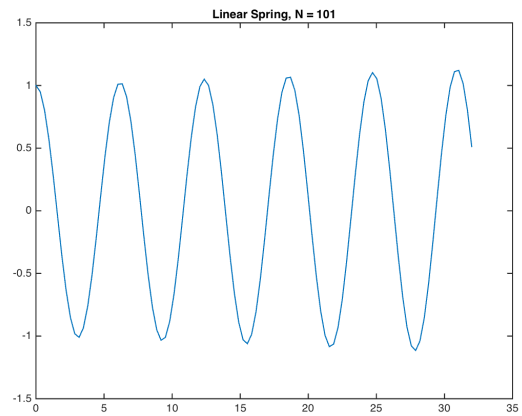
The following script now uses the previous function to run RK2 for the undamped and damped linear spring.

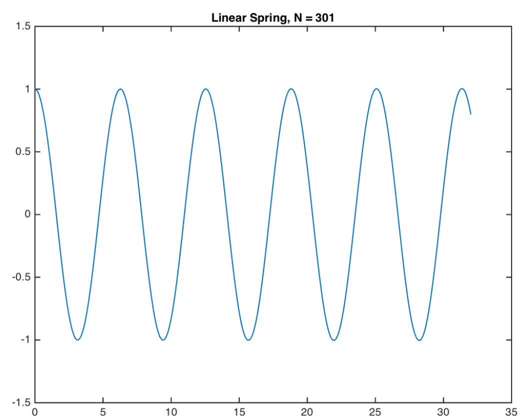
```
% Problem 2a
tFinal = 32;
% initial conditions
x0 = [1, 0];

% damping rate
sigma = 0.0;

RHSFunc = @(t, x) [x(2), -sigma*x(2) - x(1)];

n1 = 21;
sol1 = RK2(RHSFunc, x0, n1, tFinal);
plot(linspace(0, 32, n1+1), sol1(:,1));
title('Linear Spring, N = 21');
saveas(gcf, 'Figures/01_01.png', 'png');
n2 = 101;
sol2 = RK2(RHSFunc, x0, n2, tFinal);
plot(linspace(0, 32, n2+1), sol2(:,1));
title('Linear Spring, N = 101');
saveas(gcf,'Figures/01_02.png', 'png');
n3 = 301;
sol3 = RK2(RHSFunc, x0, n3, tFinal);
plot(linspace(0, 32, n3+1), sol3(:,1));
title('Linear Spring, N = 301');
saveas(gcf,'Figures/01_03.png', 'png');

exactSolFunc = @(t) cos(t);
exactSol = exactSolFunc(linspace(0, 32, 1000));
plot(linspace(0, 32, 1000), exactSol, linspace(0, 32, n1+1), sol1(:,1));
title('Linear Spring');
legend('Exact Solution', 'N = 21');
saveas(gcf,'Figures/01_04.png', 'png');

plot(linspace(0, 32, n2+1), sol2(:,1), 'ro',...
    linspace(0, 32, n3+1), sol3(:,1), 'b+',...
    linspace(0, 32, 1000), exactSol, 'k');
legend('N = 101', 'N = 301', 'Exact Solution');
title('Linear Spring');
saveas(gcf,'Figures/01_05.png', 'png');

sigma = 0.5;

RHSFunc = @(t, x) [x(2), -sigma*x(2) - x(1)];

n1 = 21;
sol1 = RK2(RHSFunc, x0, n1, tFinal);
plot(linspace(0, 32, n1+1), sol1(:,1));
title('Damped Linear Spring, N = 21');
saveas(gcf,'Figures/01_06.png', 'png');
n2 = 101;
sol2 = RK2(RHSFunc, x0, n2, tFinal);
plot(linspace(0, 32, n2+1), sol2(:,1));
title('Damped Linear Spring, N = 101');
saveas(gcf,'Figures/01_07.png', 'png');
n3 = 301;
sol3 = RK2(RHSFunc, x0, n3, tFinal);
plot(linspace(0, 32, n3+1), sol3(:,1));
title('Damped Linear Spring, N = 301');
saveas(gcf,'Figures/01_08.png', 'png');

exactSolFunc = @(t) exp(-sigma/2*t).*cos(sqrt(15)/4 * t);
```

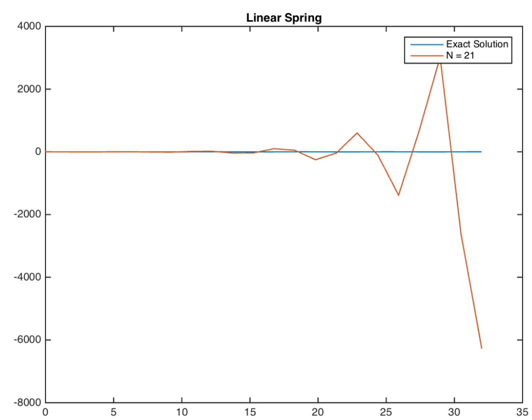
```
exactSol = exactSolFunc(linspace(0, 32, 1000));
plot(linspace(0, 32, 1000), exactSol, linspace(0, 32, n1+1), sol1(:,1));
title('Damped Linear Spring');
legend('Exact Solution', 'N = 21');
saveas(gcf,'Figures/01_09.png', 'png');

plot(linspace(0, 32, n2+1), sol2(:,1), 'ro',...
    linspace(0, 32, n3+1), sol3(:,1), 'b+',...
    linspace(0, 32, 1000), exactSol, 'k');
legend('N = 101', 'N = 301', 'Exact Solution');
title('Damped Linear Spring');
saveas(gcf,'Figures/01_10.png', 'png');
```

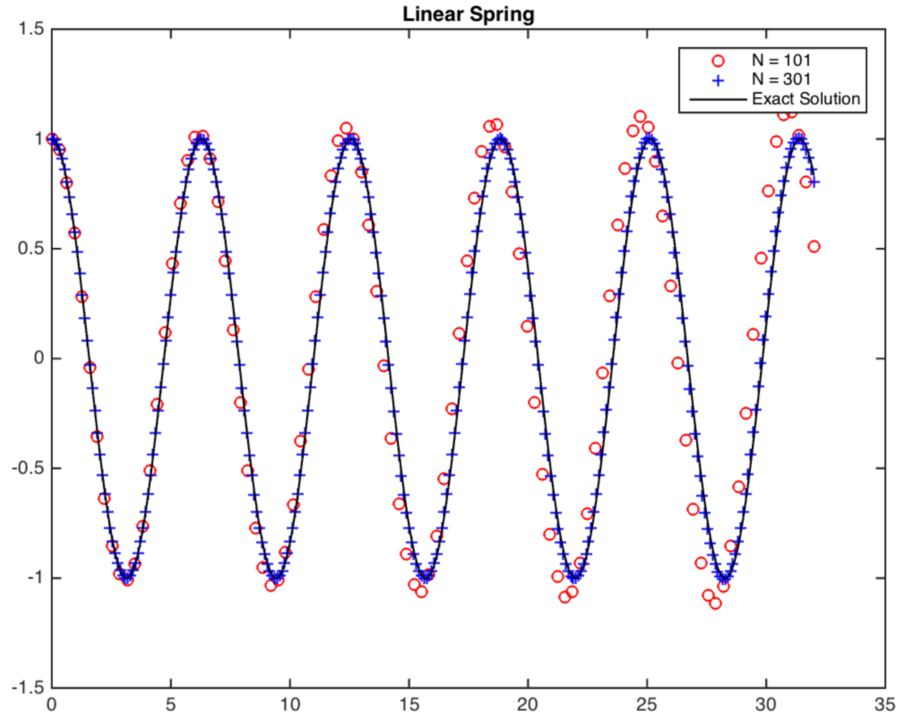
The following images are produced for the undamped linear spring, that is when $\sigma = 0.0$. Note that for $N = 21$, the numerical solution diverges from the exact solution, but for $N = 101$ and $N = 301$, the numerical solution is close to exact solution and gets more accurate as N is increased.



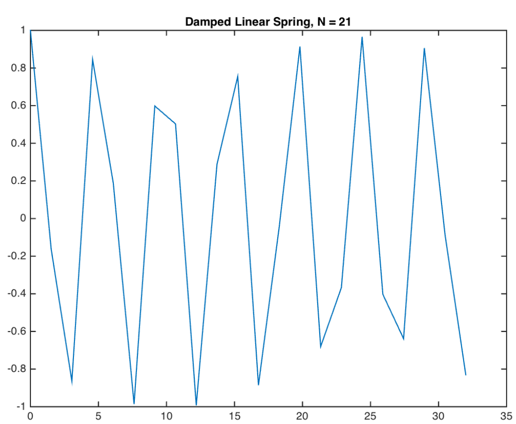


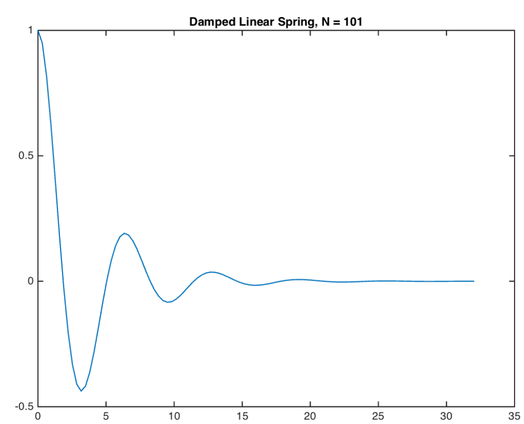


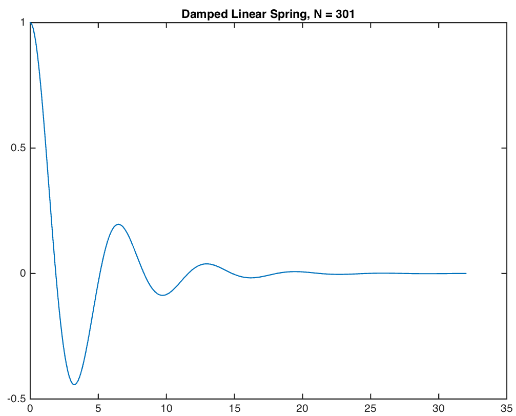


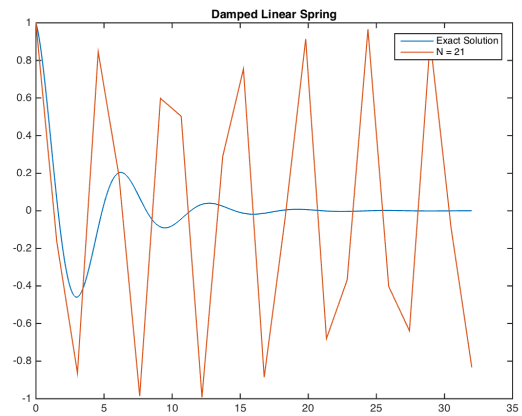


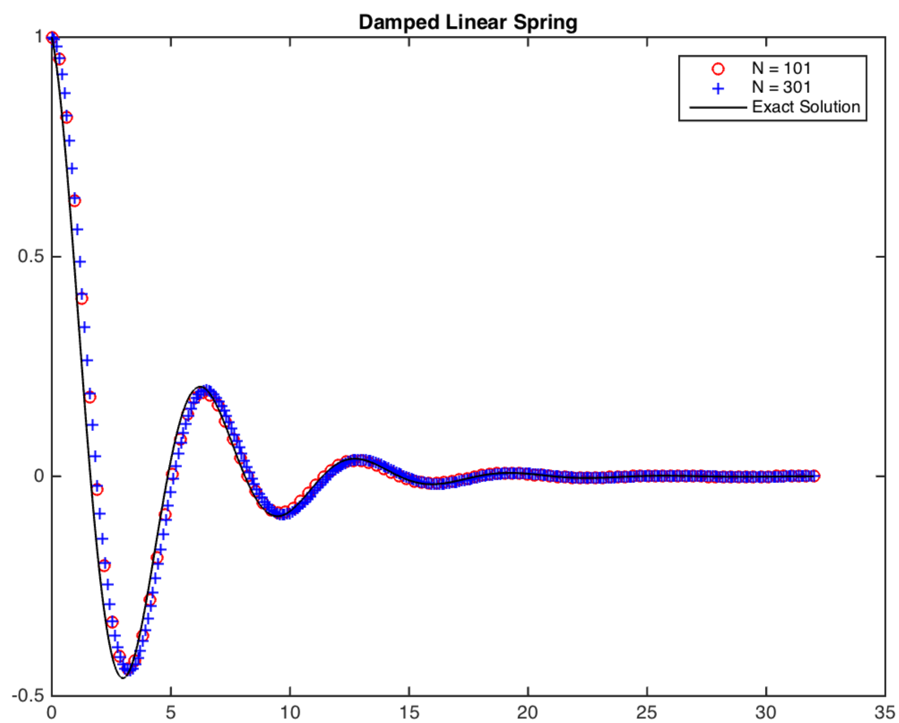
For the damped case, i.e. when $\sigma = 0.5$, the following images are produced. Note that the numerical solution for $N = 21$ doesn't grow rapidly and diverge, but is doesn't accurately represent the exact solution. Again as N is increased the accuracy of the numerical solution increases.











(b) The equation for a nonlinear spring (without damping) is

$$\ddot{Y} + Y - BY^3 = 0.$$

Solve by RK2 out to $t = 32$ with the intial conditions $Y(0) = 1$ and $\dot{Y}(0) = 0$. Plot $Y(t)$ for $B = 0.2, 0.6, 0.9, 0.999$. Chose N large enough to get an accurate solution; that will depend on the value of B.

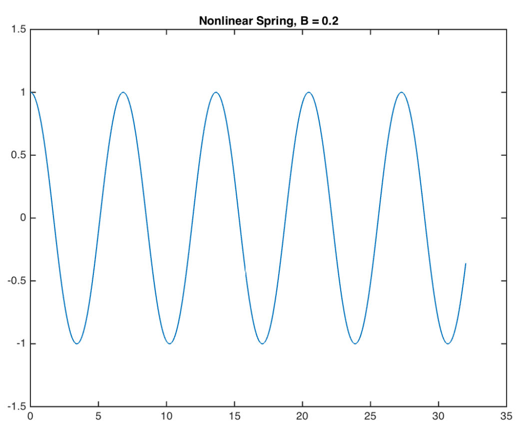
First in order to apply RK2 to this problem we must turn this second order ODE into a system of first order ODEs. In order to accomplish this let $\dot{Y} = Z$, then we have the following system

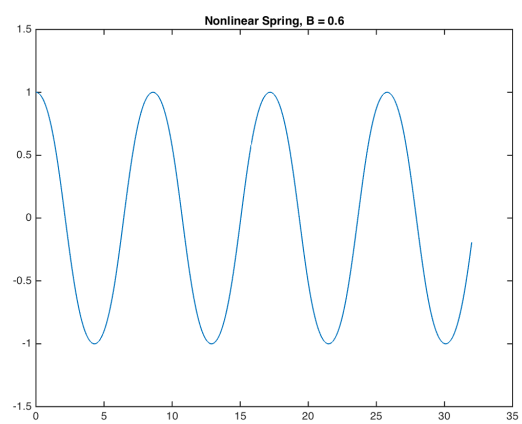
$$\dot{Y} = Z$$
$$\dot{Z} = -Y - BY^3$$

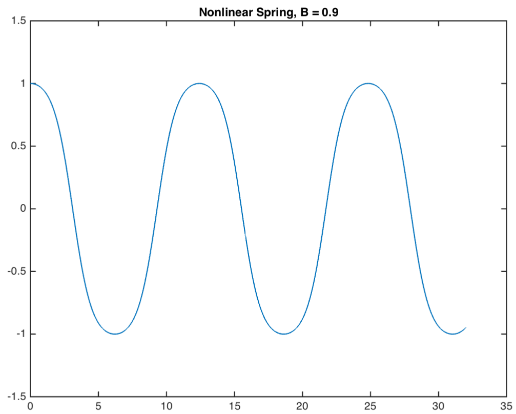
The following script now uses the same RK2 method shown in part (a), but with the RHS given above.

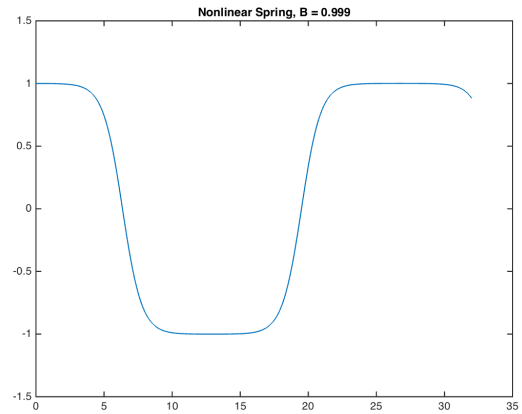
```
% Problem 2b
tFinal = 32;
x0 = [1,0];
B = 0.2;
RHSFunc = @(t, x) [x(2), B*x(1)^3 - x(1)];
n = 1000;
sol = RK2(RHSFunc, x0, n, tFinal);
plot(linspace(0,32,n+1), sol(:,1));
title('Nonlinear Spring, B = 0.2')
saveas(gcf, 'Figures/01_11.png', 'png');

B = 0.6;
RHSFunc = @(t, x) [x(2), B*x(1)^3 - x(1)];
n = 1000;
sol = RK2(RHSFunc, x0, n, tFinal);
plot(linspace(0,32,n+1), sol(:,1));
title('Nonlinear Spring, B = 0.6')
saveas(gcf, 'Figures/01_12.png', 'png');

B = 0.9;
RHSFunc = @(t, x) [x(2), B*x(1)^3 - x(1)];
n = 1000;
sol = RK2(RHSFunc, x0, n, tFinal);
plot(linspace(0,32,n+1), sol(:,1));
title('Nonlinear Spring, B = 0.9')
saveas(gcf, 'Figures/01_13.png', 'png');

B = 0.999;
RHSFunc = @(t, x) [x(2), B*x(1)^3 - x(1)];
n = 3000;
sol = RK2(RHSFunc, x0, n, tFinal);
plot(linspace(0,32,n+1), sol(:,1));
title('Nonlinear Spring, B = 0.999')
saveas(gcf, 'Figures/01_14.png', 'png');
```

In this script I used $N = 1000$ for $B = 0.2, 0.6, 0.9$ and $N = 3000$ for $B = .999$. B is controlling the nonlinearity of the spring and the larger that contribution is the smaller the timestep needs to be. The following images are produced.









#3 Repeat the linear spring computation (ex. 2.a) with AB2. What does the solution for $\sigma = 0.0$ tell you about the stability of AB2?

I implemented the following function to run AB2 method for any RHS function.

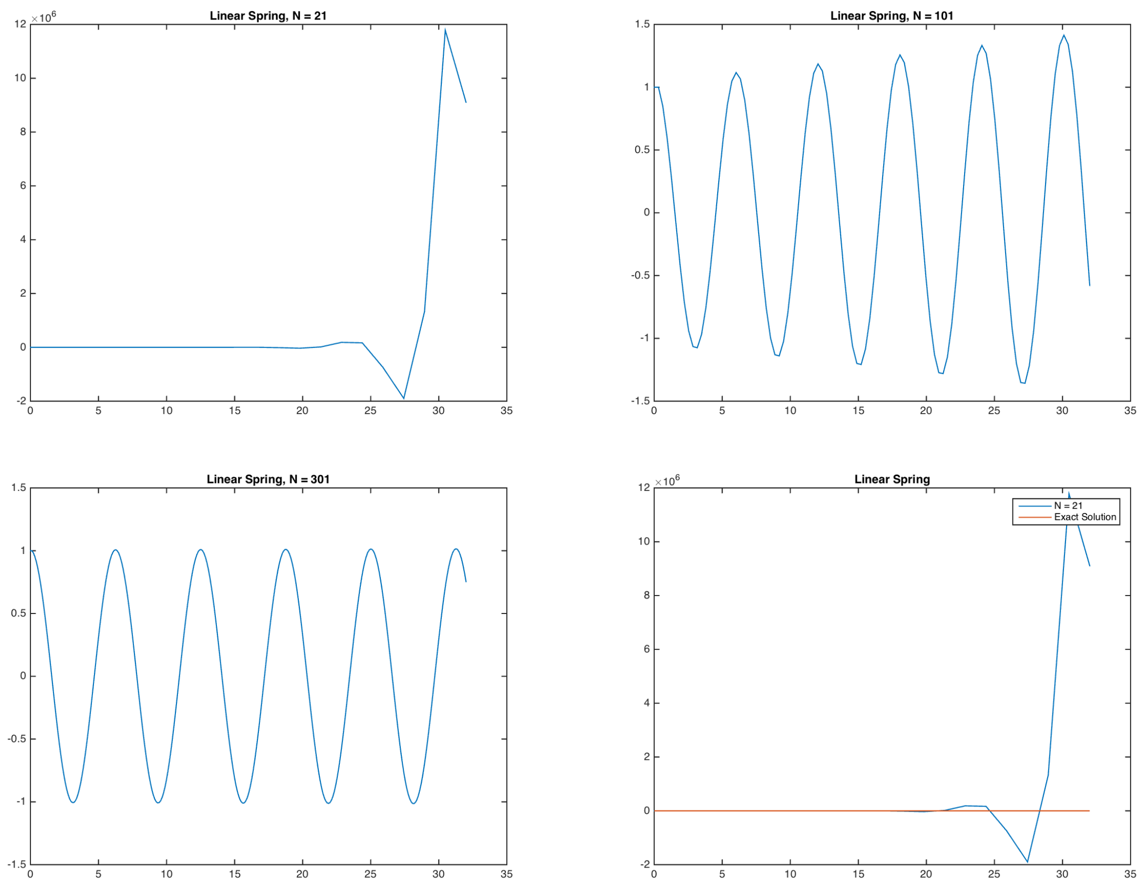
```
function [result] = AB2(RHSFunc, x0, nTimeSteps, tFinal)
    nEquations = length(x0);
    result = zeros(nTimeSteps+1, nEquations);
    result(1,:) = x0;
    deltaT = tFinal/nTimeSteps;

    % take first step with explicit Euler method
    rhsOld = RHSFunc(0, result(1,:));
    result(2,:) = x0 + deltaT*rhsOld;

    for i = 2:nTimeSteps
        t = (i-1)*deltaT;
        rhsNew = RHSFunc(t, result(i,:));
        result(i+1, :) = result(i,:) + deltaT*(1.5*rhsNew - 0.5*rhsOld);
        rhsOld = rhsNew;
    end
end
```

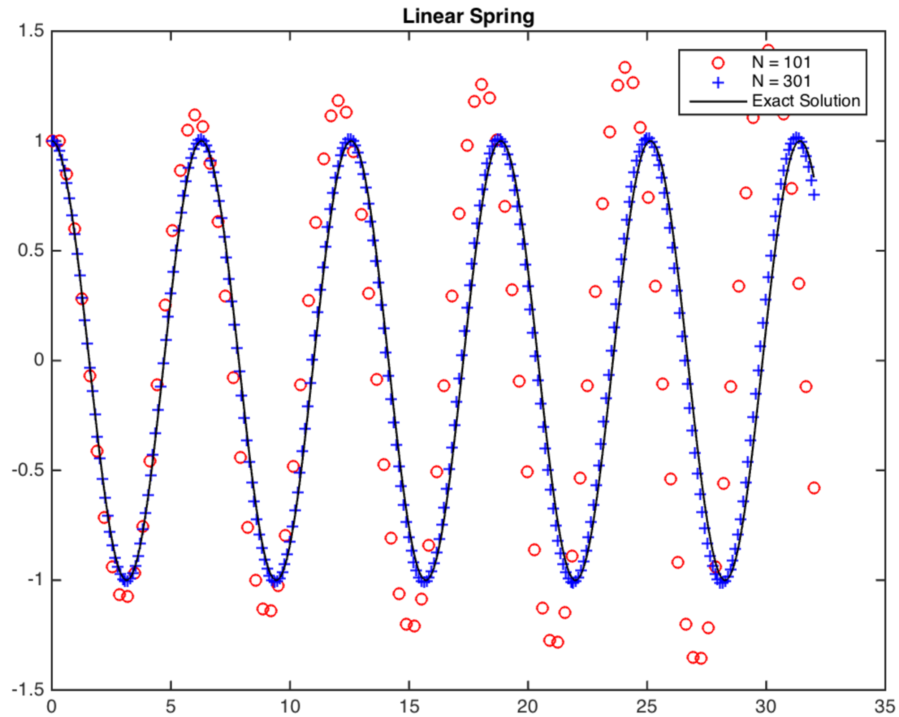
The following script now repeats exercise 2.a with this function instead of RK2.

```
% Problem 3
tFinal = 32;
% initial conditions
x0 = [1, 0];

% damping rate
sigma = 0.0;

RHSFunc = @(t, x) [x(2), -sigma*x(2) - x(1)];

n1 = 21;
sol1 = AB2(RHSFunc, x0, n1, tFinal);
plot(linspace(0, 32, n1+1), sol1(:,1));
title('Linear Spring, N = 21');
saveas(gcf, 'Figures/01_15.png', 'png');
n2 = 101;
sol2 = AB2(RHSFunc, x0, n2, tFinal);
```

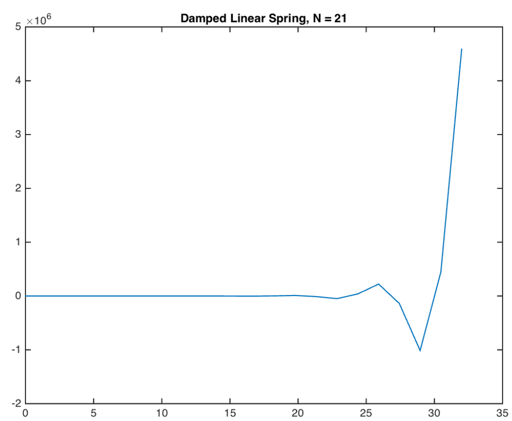
```
plot(linspace(0, 32, n2+1), sol2(:,1));
title('Linear Spring, N = 101');
saveas(gcf,'Figures/01_16.png', 'png');
n3 = 301;
sol3 = AB2(RHSFunc, x0, n3, tFinal);
plot(linspace(0, 32, n3+1), sol3(:,1));
title('Linear Spring, N = 301');
saveas(gcf,'Figures/01_17.png', 'png');

exactSolFunc = @(t) cos(t);
exactSol = exactSolFunc(linspace(0, 32, 1000));
plot(linspace(0, 32, n1+1), sol1(:,1),...
linspace(0, 32, 1000), exactSol);
title('Linear Spring');
legend('N = 21', 'Exact Solution');
saveas(gcf,'Figures/01_18.png', 'png');

plot(linspace(0, 32, n2+1), sol2(:,1), 'ro',...
    linspace(0, 32, n3+1), sol3(:,1), 'b+',...
    linspace(0, 32, 1000), exactSol, 'k');
legend('N = 101', 'N = 301', 'Exact Solution');
title('Linear Spring');
saveas(gcf,'Figures/01_19.png', 'png');

sigma = 0.5;

RHSFunc = @(t, x) [x(2), -sigma*x(2) - x(1)];

n1 = 21;
sol1 = AB2(RHSFunc, x0, n1, tFinal);
plot(linspace(0, 32, n1+1), sol1(:,1));
title('Damped Linear Spring, N = 21');
saveas(gcf,'Figures/01_20.png', 'png');
n2 = 101;
sol2 = AB2(RHSFunc, x0, n2, tFinal);
plot(linspace(0, 32, n2+1), sol2(:,1));
title('Damped Linear Spring, N = 101');
saveas(gcf,'Figures/01_21.png', 'png');
n3 = 301;
sol3 = AB2(RHSFunc, x0, n3, tFinal);
plot(linspace(0, 32, n3+1), sol3(:,1));
title('Damped Linear Spring, N = 301');
saveas(gcf,'Figures/01_22.png', 'png');


exactSolFunc = @(t) exp(-sigma/2*t).*cos(sqrt(15)/4 * t);
exactSol = exactSolFunc(linspace(0, 32, 1000));
plot(linspace(0, 32, n1+1), sol1(:,1), 'ro',...
    linspace(0, 32, 1000), exactSol, 'k');
title('Damped Linear Spring');
legend('N = 21', 'Exact Solution');
saveas(gcf,'Figures/01_23.png', 'png');

plot(linspace(0, 32, n2+1), sol2(:,1), 'b+',...
    linspace(0, 32, n3+1), sol3(:,1), 'ro',...
    linspace(0, 32, 1000), exactSol, 'k');
title('Damped Linear Spring');
legend('N = 101', 'N = 301', 'Exact Solution');
saveas(gcf,'Figures/01_24.png', 'png');
```

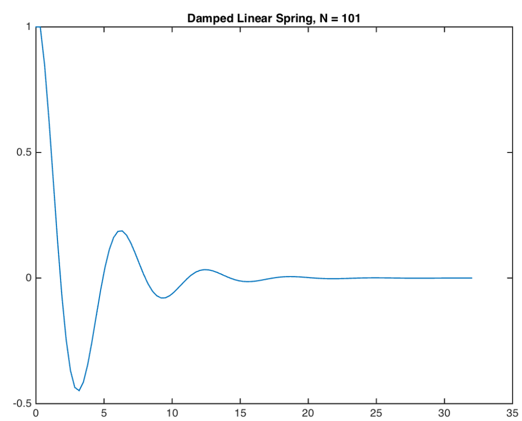
The following images were produced in the undamped case. Note that for $N = 21$ the solution diverges rapidly, much more rapidly than for RK2. This shows that adams-bashforth has a smaller area of stability as the time step needs to be much smaller in order to get an accurate solution. However for $N = 101$ and $N = 301$ the solutions are relatively accurate if less so than for RK2.

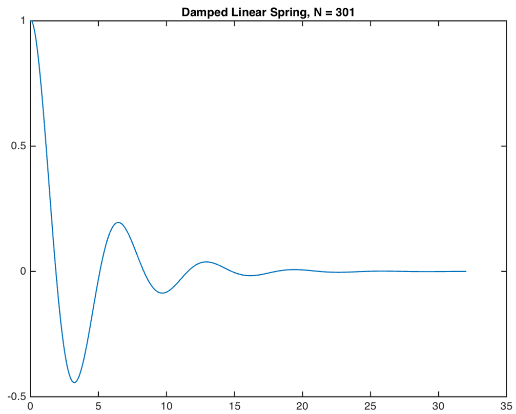


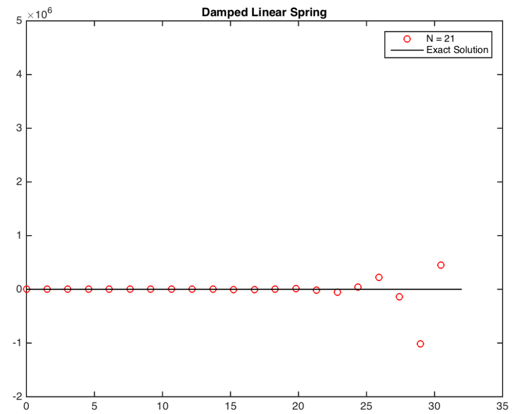


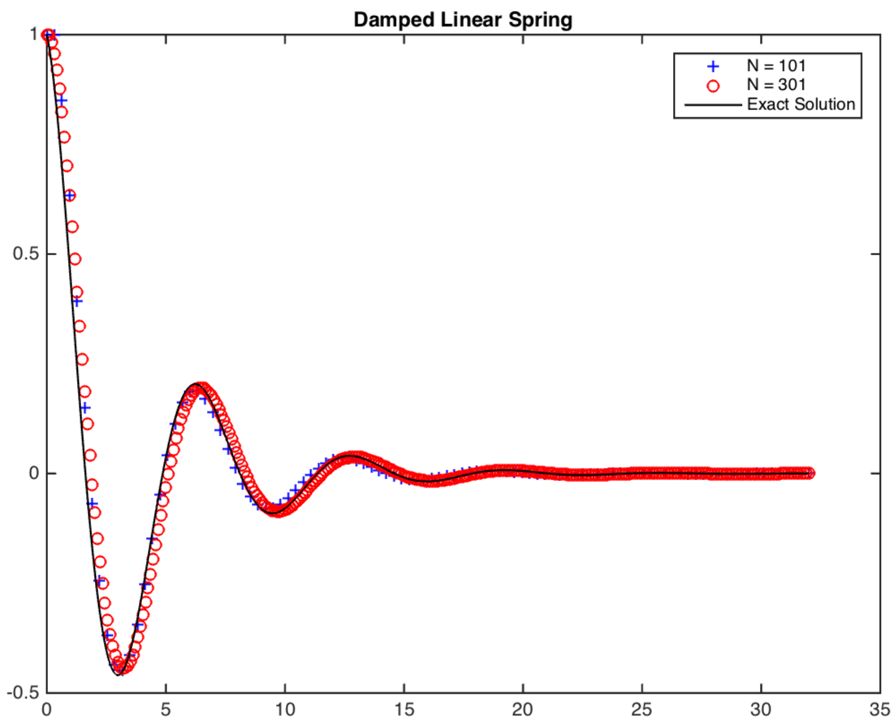
The following images were produced for the damped case Note that the solution diverges for $N = 21$.






Damped Linear Spring, N = 301
1
0.5
0
-0.5
0 5 10 15 20 25 30 35


Damped Linear Spring
×10^6
N = 21
Exact Solution
5
4
3
2
1
0
-1
-2
0 5 10 15 20 25 30 35


Damped Linear Spring
N = 101
N = 301
Exact Solution
1
0.5
0
-0.5
0 5 10 15 20 25 30 35