Numerical integration of differential equations applied to a damped harmonic oscillator

Meirin Evans

9214122

School of Physics and Astronomy

The University of Manchester

Second Year Laboratory Report

Apr 2016

Abstract

Through the application of five different numerical integration methods (Euler, improved Euler, Verlet, Euler-Cromer and RK4) to an analytically solvable damped harmonic oscillator, the RK4 method was found to be closest to the analytic solution. The RK4 method could then be implemented to approximate non-solvable situations such as a sudden force application part-way through a cycle.

1. Introduction

A damped harmonic oscillator, such as a mass, *m*, moving horizontally on a spring with constant *k*, follows Equation 1;

$$ma + bv + kx = F \text{,} \tag{1}$$

where *x* is displacement from equilibrium, *v* is velocity, *a* is acceleration, *b* is a damping term and *F* is a forcing term. This can only be solved in situations where *F*=0, which is unforced, or is forced but has a specific form, like a sinusoid. For simple harmonic motion with *F*=0 and *b*=0 *a* is given by

$$a = -\omega_0^2 x \text{,} \tag{2}$$

where ω_0 is the object's natural frequency, which is

$$\omega_0 = \sqrt{\frac{k}{m}} \text{.} \tag{3}$$

2. Theory

The solution for Equation 1 with *F*=0 is

$$x(t) = Ae^{\frac{-bt}{2m}} \sin(t\sqrt{\frac{k}{m} - (\frac{b}{2m})^2} + \phi) \text{,} \tag{4}$$

where *t* is the time since releasing *m*, *A* is the initial amplitude and ϕ is an initial phase. Differentiating Equation 4 gives

$$v(t) = Ae^{\frac{-bt}{2m}} (\frac{-b}{2m} \sin(t\sqrt{\frac{k}{m} - (\frac{b}{2m})^2} + \phi) + \sqrt{\frac{k}{m} - (\frac{b}{2m})^2} \cos(t\sqrt{\frac{k}{m} - (\frac{b}{2m})^2} + \phi)) \text{.} \tag{5}$$

A in Equations 4 and 5 is

$$A = \frac{\sqrt{x(t=0)^2 + v(t=0)^2}}{\sin^2\phi + \frac{k}{m}\cos^2\phi} \text{,} \tag{6}$$

whilst ϕ is

$$\phi = \tan^{-1}(\frac{x(t=0)}{v(t=0)}\sqrt{\frac{k}{m}}) \text{.} \tag{7}$$

The total energy, *E*, of a damped harmonic oscillator (kinetic plus potential energy) is

$$E = \frac{1}{2}kx^2 + \frac{1}{2}mv^2 \quad (8)$$

When *b* has a value of

$$b = 2\sqrt{mk} \quad (9)$$

m returns to the equilibrium position in the shortest time without oscillation, called critical damping.

When a sinusoidal force, $F_0\sin\omega t$, with ω similar to ω_0 drives a damped harmonic oscillator it reaches steady state oscillations after an initial transient period has passed. Steady state means constant amplitude oscillations. Applying this type of force allows investigation of resonance. *A* as a function of ω is

$$A(\omega) = \frac{F_0}{m\sqrt{(\frac{k}{m} - \omega^2)^2 + \omega^2 \frac{b^2}{m^2}}} \quad (10)$$

The peak of a resonance curve occurs at ω slightly less than ω_0 [1]. The smaller *b* the closer the peak is to ω_0.

3. Analytical method

In a **MATLAB** script five numerical integration methods (Euler, improved Euler, Verlet, Euler-Cromer and RK4) were compared with the analytic solution for the case $m = 1.79$ kg, $k = 3.79$ Nm^{-1}, $b = 0.1$ Nsm^{-1}, $x(t=0) = 0$ m, $v(t=0) = -1$ ms^{-1} [2]. To achieve this *x(t)* and *E(t)* were plotted using Equation 8 for all methods. Equations 2, 3, 6 and 7 were used for the analytic method. The method with the smallest energy residuals relative to the analytic method was taken as the best approximation. This method was then applied to investigate critical damping through Equation 9, steady state oscillations with a sinusoidal driving force, and also application of sudden forces at different points in a cycle. Resonance was studied through Equation 10 using the analytic method.

The damping term and time step used in Equations 4 and 5 were chosen to best illustrate the differences between different methods.

4. Results

As described in the Analytical method, plots of *x(t)* and *E(t)* are shown in Figures 1 and 2 respectively.

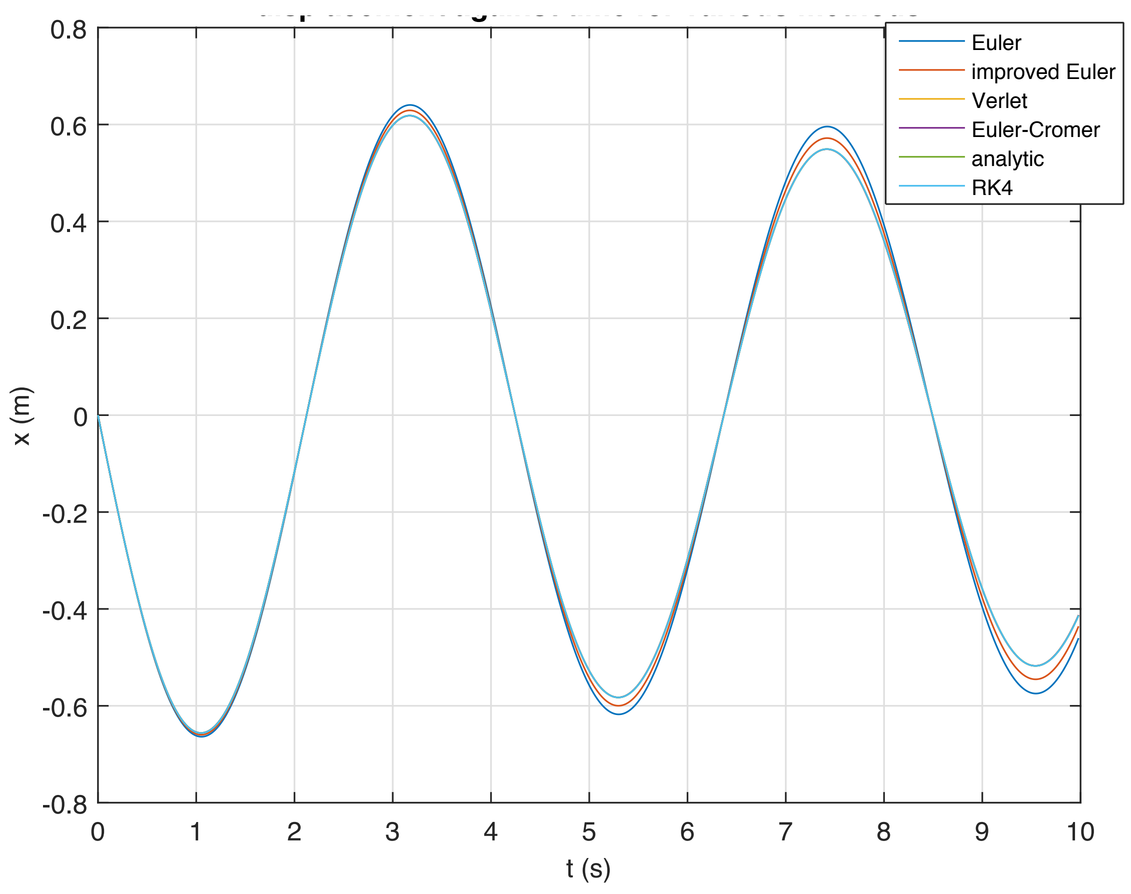

Fig 1. Oscillator displacement in m against time since release in s for Euler, improved Euler, Verlet, Euler-Cromer, RK4 and analytic methods. The Verlet (yellow), Euler-Cromer (purple) and analytic (green) plots cannot be seen since they are very close to the RK4 (cyan) plot.

To avoid confusion with the decrease of energy with time, the residuals for Figure 2 relative to the analytic solution are shown in Figure 3. Justification for the chosen time step of 0.01s is shown in Figure 4. Euler's method was chosen to illustrate this as the least accurate method. A larger time step shows Euler's method adding energy to the system and a smaller time step deviates more from the equilibrium position (also adding artificial energy to the system). Another way to compare methods is to plot the oscillator's phase space, shown in Figure 5, which is simply *x* against *v*. The analytic solution's phase space is an ellipse with decreasing semi-minor and major axes. Figure 6 displays critical damping, with the critically damped oscillator returning to the equilibrium position after about 6s. The oscillator with half-critical damping completes at least one oscillation whereas the oscillator with double-critical damping does not return to equilibrium. A driving frequency of $\omega = 1.5\ \mathrm{s}^{-1}$ in Figure 7 shows that it takes time for forced oscillations to reach steady state, as there is not much difference between the forced and unforced cases to begin with. Thereafter the forced case shows the amplitude increasing until reaching steady state.

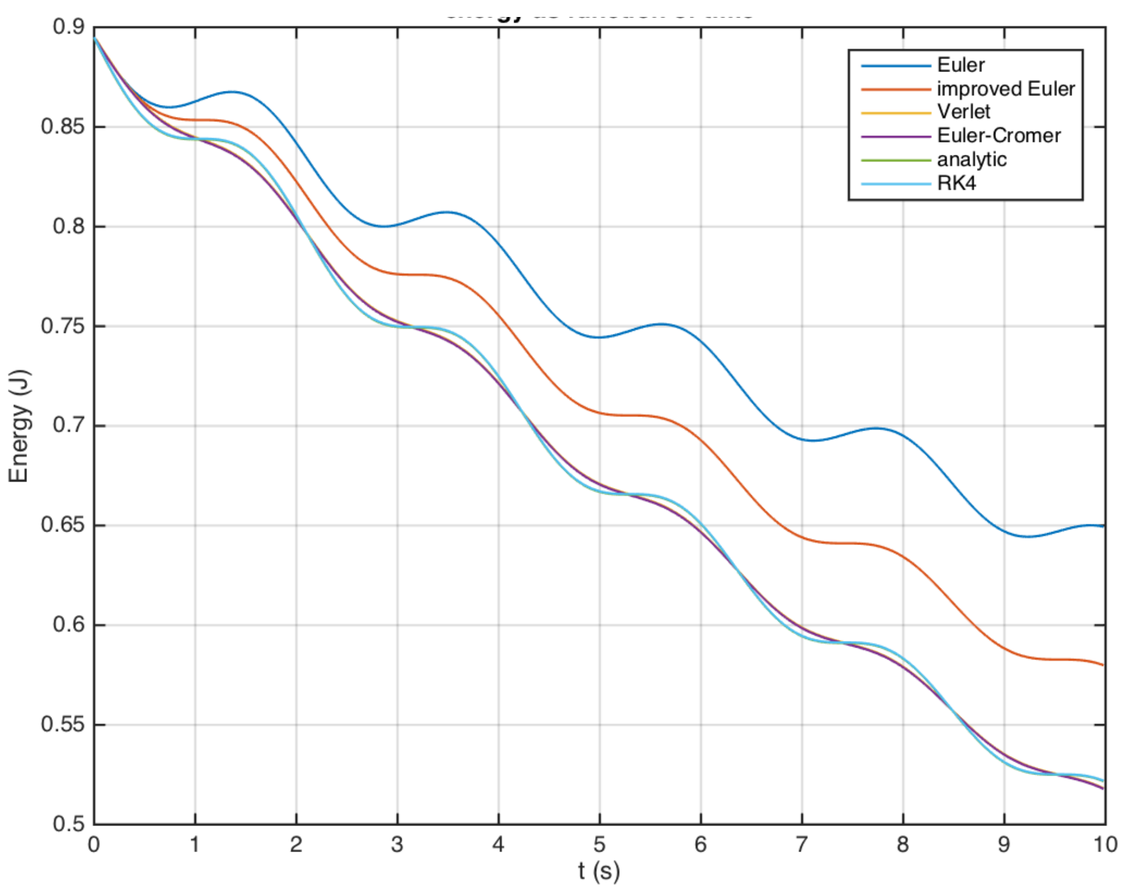

Fig 2. Oscillator energy in J against time since release in s for Euler, improved Euler, Verlet, Euler-Cromer, RK4 and analytic methods. As for Figure 1, the Verlet (yellow), Euler-Cromer (purple) and RK4 (cyan) plots are close together so are not seen separately.

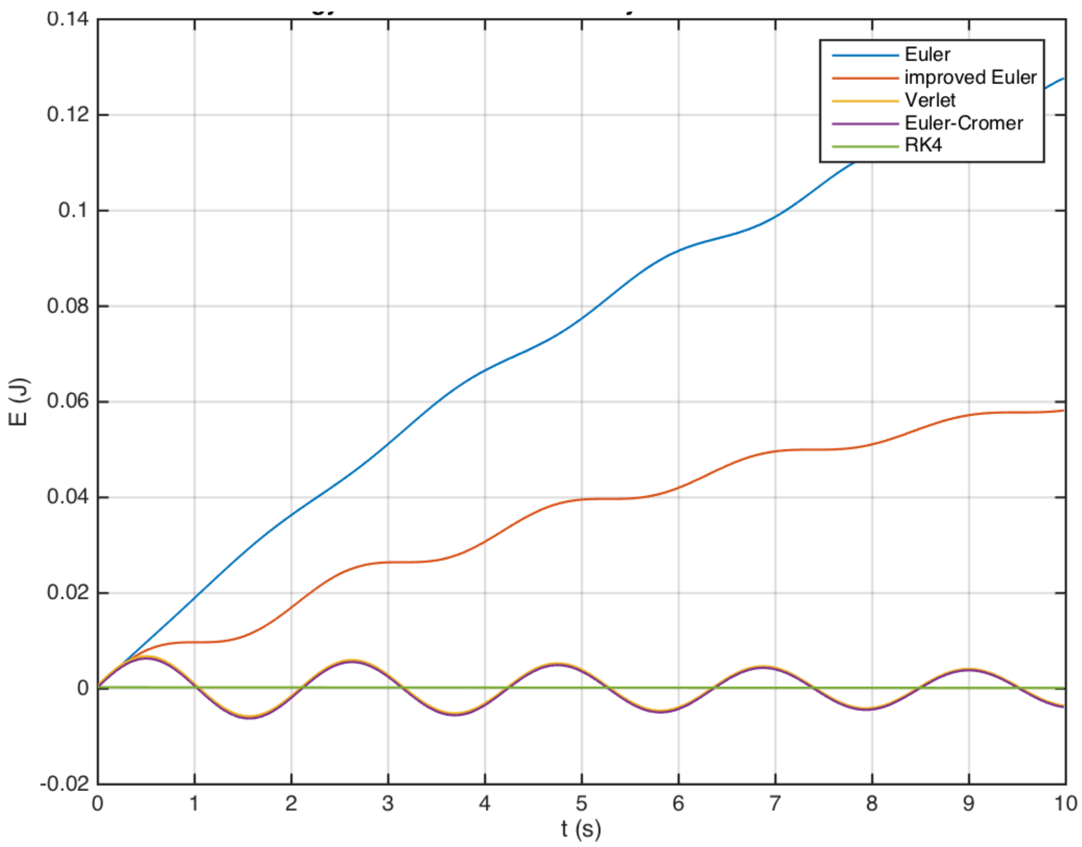

Fig 3. Residuals of oscillator energy in J from Figure 2 for Euler, improved Euler, Verlet, Euler-Cromer and RK4 relative to the analytic solution. As for Figures 1 and 2 the Verlet (yellow), Euler-Cromer (purple) and RK4 (cyan) plots cannot be distinguished since they are close.

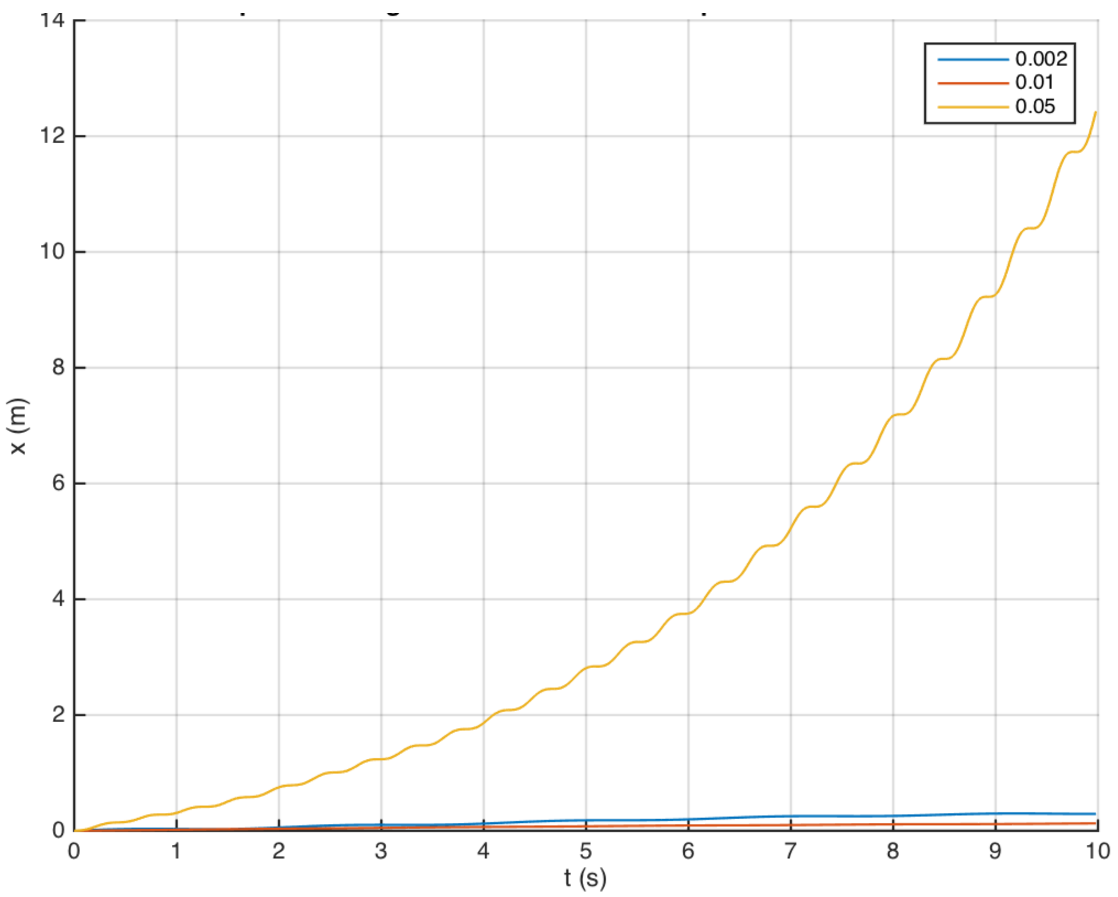

Fig 4. Oscillator displacement in m against time since release in s as Figure 1 but for Euler method only with various time steps.

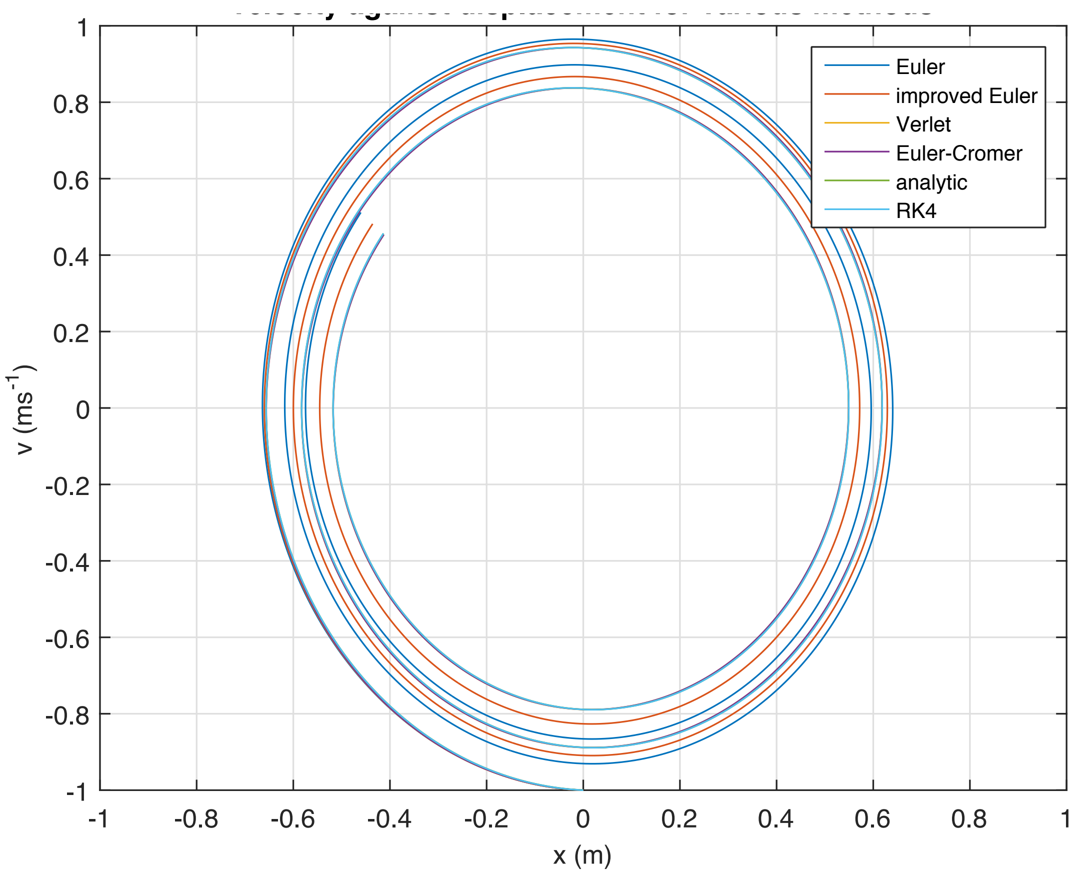

Fig 5. Phase space plots (oscillator displacement in m against velocity in ms^{-1} for Euler, improved Euler, Verlet, Euler-Cromer, RK4 and analytic methods. As for Figures 1, 2 and 3 the Verlet (yellow), Euler-Cromer (purple), analytic (green) and RK4 (cyan) plots are close so are mostly seen as one.

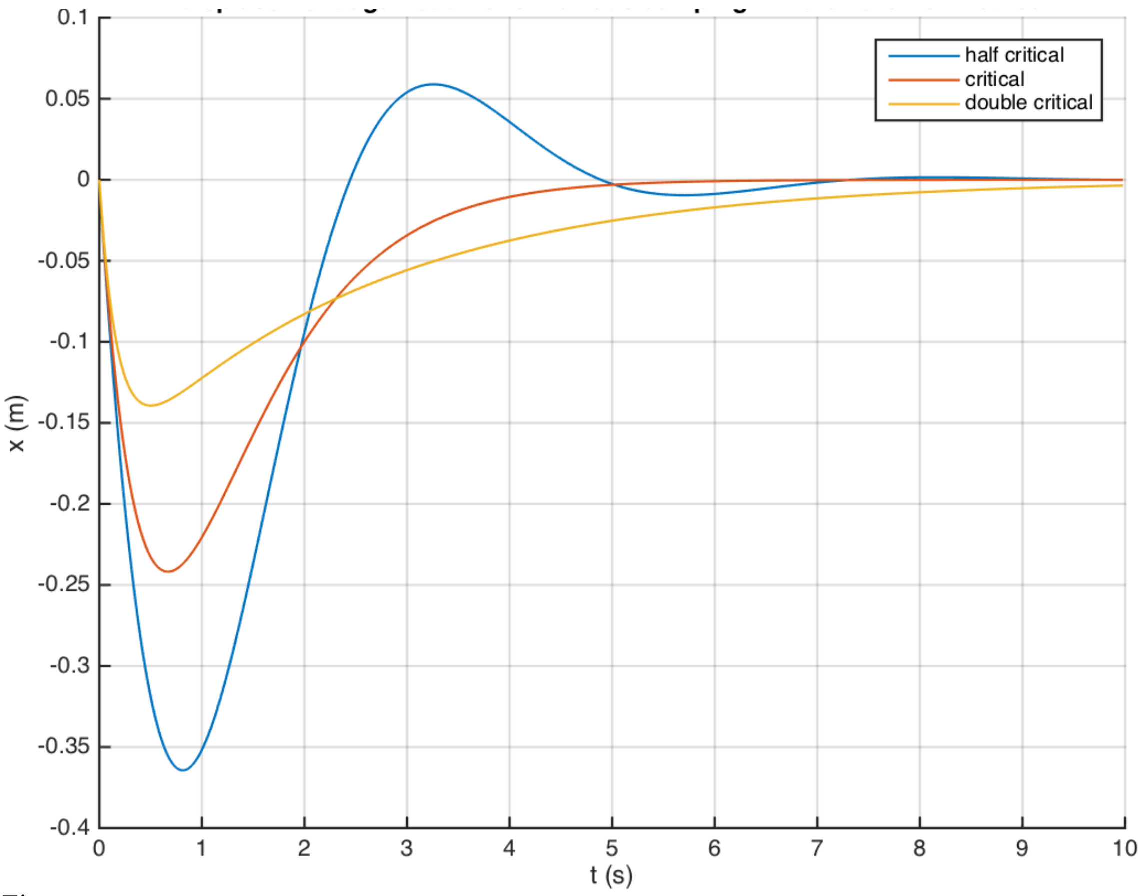

Fig 6. Plot of oscillator displacement in m against time since release in s with critical, half-critical and double-critical damping for the RK4 method. $x = 0$ is the equilibrium position.

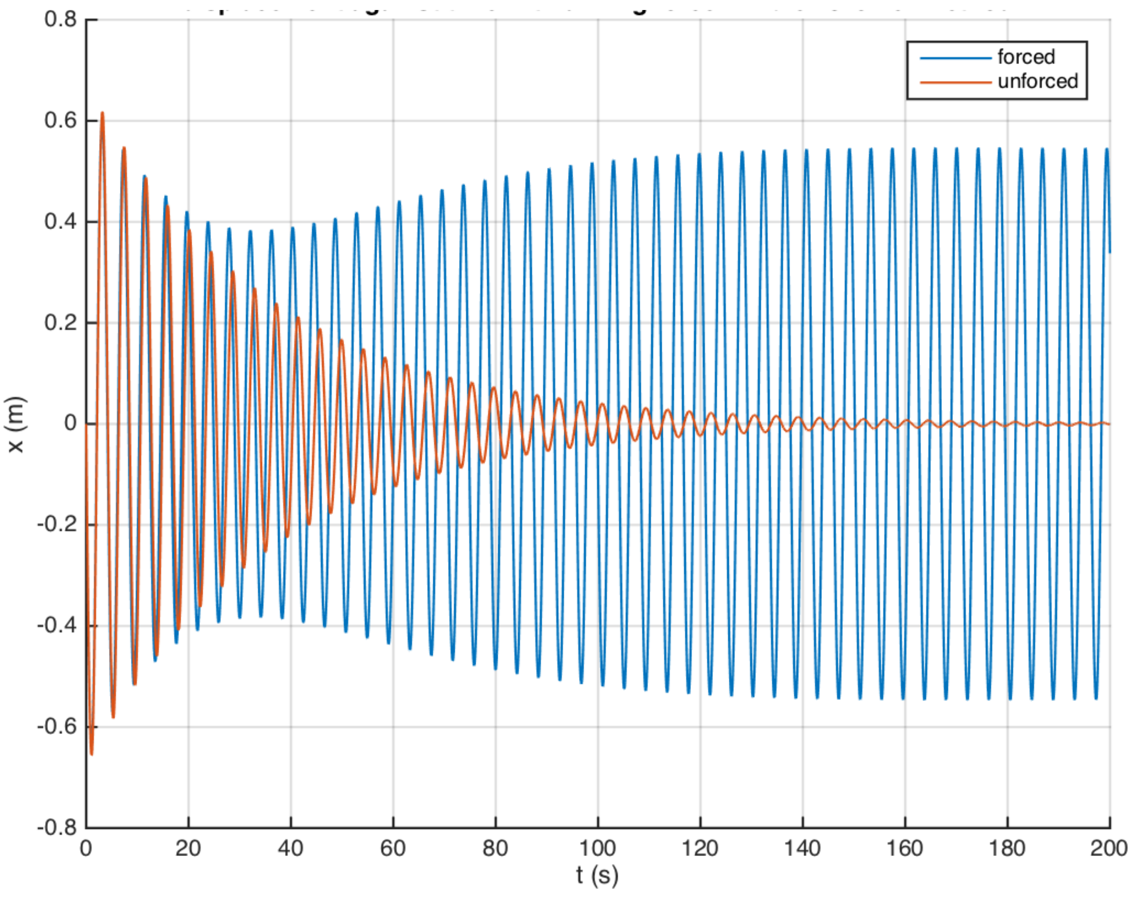

Fig 7. Oscillator displacement in m against time since release in s for forced (blue) and unforced (red) oscillations using the RK4 method. The plots are close together until about 20s (transient period). Steady state oscillations are seen after about 140s, by this time the unforced oscillations have almost died away.

Figure 8 shows the effect of a sudden force application during oscillation. In all Figure 8 subplots a constant force of 1N was applied after the times where two plots are distinguishable. These cases have no analytic solution therefore can only be investigated using numerical integration.

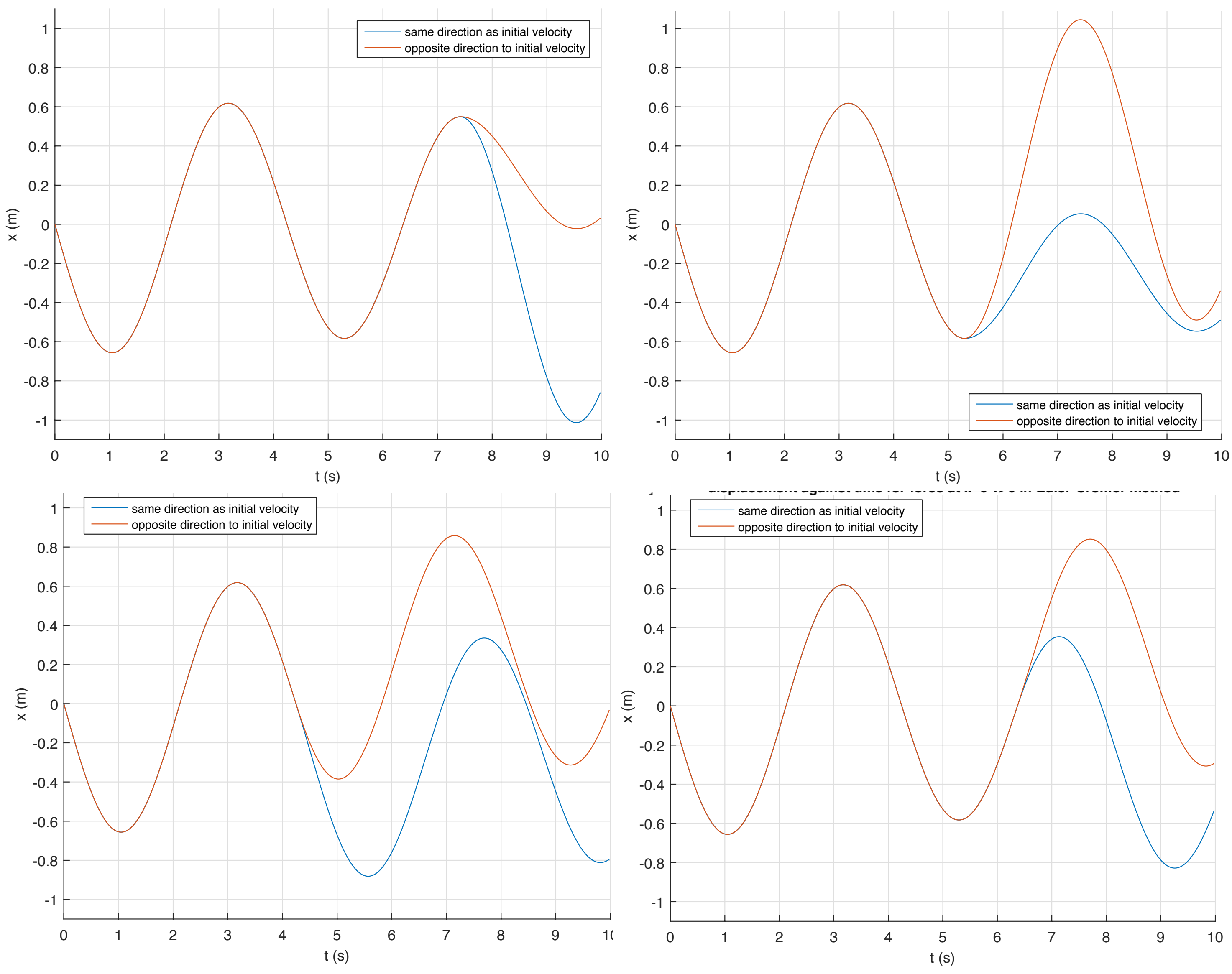

Fig 8. Oscillator displacement in m against time since release in s with application of a sudden (then continued) force at different parts throughout an oscillation cycle using the RK4 method. Top left shows application at a peak, top right at a trough, bottom left for application when x = 0 m and $v < 0$ ms^{-1}, bottom right for x = 0 m and $v > 0$ ms^{-1}. Blue curves are for a force direction being the same as $v(t=0)$ whilst red are for a force oppositely directed from $v(t=0)$.

Figure 9 shows resonance curves for oscillators of different natural frequency. k/m in Equation 10 is replaced with $k/m + 0.1^2 - 0.2(k/m)^{0.5}$ for the red curve and $k/m + 0.1^2 + 0.2(k/m)^{0.5}$ for the yellow curve. Figure 9 shows the curves moving to lower frequency and the peak height increasing for lower natural frequency.

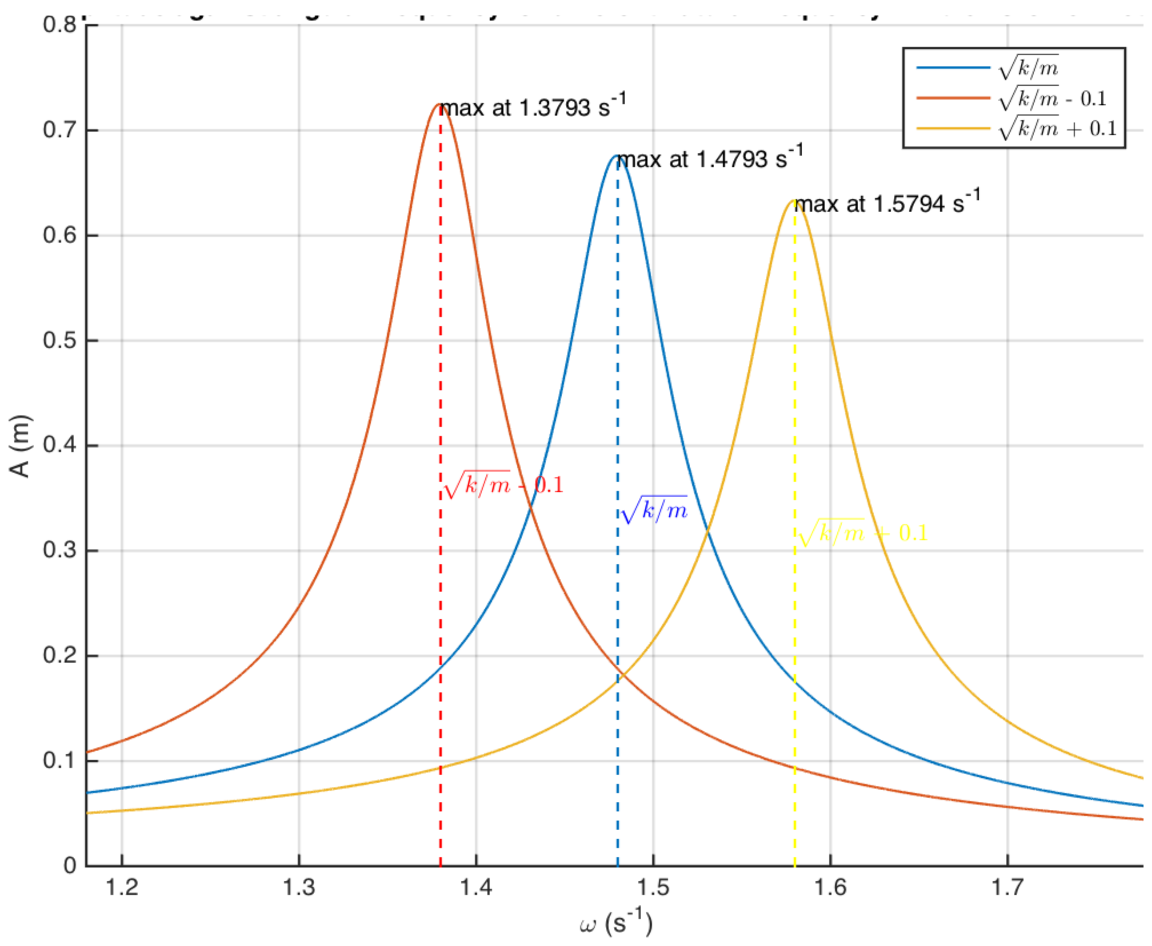

Fig 9. Resonance curves (oscillator amplitude in m against angular frequency in s^{-1}) for various natural frequencies. The vertical lines label the natural frequency of that curve and the maxima of each curve are labelled. The lines look as if they are pointing to the maxima but they are actually slightly separated (by too small a distance to be seen).

5. Discussion

Running the simulation from Figure 5 for an infinite time would show the analytic solution's phase space spiralling towards the origin. Equally, an infinite time in Figure 6 would show the oscillator with double critical damping not returning to equilibrium. Study of Figure 7 shows that peaks for forced and unforced oscillations do not align after the transient period, which is expected as the forced oscillator oscillates at ω in steady state. Figure 9 suggests the natural frequencies are the same as the peak frequencies, which would contradict the discussion in the Theory. However, they have differences of $(5.4 \pm 0.6) \times 10^{-4}$ s^{-1}, $(5.1 \pm 0.6) \times 10^{-4}$ s^{-1} and $(4.8 \pm 0.6) \times 10^{-4}$ s^{-1} respectively for the red, blue and yellow curves. These error values were obtained from dividing the range in ω by the number of values in the ω array. This shows that the value of *b* used was relatively small and corresponded to light damping.

Figures 1, 2, 3 and 5 confirm that the Verlet and Euler-Cromer methods are good approximations, but RK4 is better. Using Figure 3, Verlet's method's maximum deviation from the analytic solution was (0.80 ± 0.23) % whilst it was (0.75 ± 0.23) % for Euler-Cromer's method and (0.03564158 ± 0.00000003) % for RK4, which is why RK4 was used in Figures 6-9. These errors were obtained using the **MATLAB** std function on the deviation arrays.

6. Summary

It was found that five different numerical integration methods could be ranked according to increasing ability in replicating the analytic solution to a damped harmonic oscillator. The order was Euler, improved Euler, Verlet, Euler-Cromer then RK4.

References

[1] King, G., *Vibrations and Waves*, Wiley, First Edition, 2009, page 56.

[2] MATLAB, Version 5, The Math Works Inc, Natick, Mass 01760.

The number of words in this document is 1531.

This document was last saved on 25/4/2016 at 16:55.

Appendix

```
% ————————————————————————————————————————————
% ————————————————————————————————————————————
% project 2
% Numerical integration of differential equations - damped harmonic oscillator
% Meirin Evans, Apr 16
% ------------------------------------------------------------------------------------------------

% ------------------------------------------------------------------------------------------------
% clear workspace & figure windows
clear;
close all;
% ------------------------------------------------------------------------------------------------

% ————————————————————————————————————————————
% decide on parameters
prompt = 'Use m = 1.79 kg, k = 3.92 N per m, b = 0.1 Ns per m, x(t=0) = 0 m, v(t=0) =
                    -1 m per s? (y/n) ‘;                    % ask user if they want to input
                                                             their own values
if input(prompt, 's') == ’n'                                 % if they want to input own values
   prompt = 'Please enter m in kg ';                         % ask for m
   m = input(prompt);                                        % user inputted m
   prompt = 'Please enter k in N per m ‘;                    % ask for k
   k = input(prompt);                                        % user inputted k
   prompt = 'Please enter b in Ns per m ';                   % ask for b
   b = input(prompt);                                        % user inputted b
   prompt = 'Please enter x(t=0) in m ‘;                     % ask for x(t=0)
   xinitial = input(prompt);                                 % user inputted x(t=0)
   prompt = 'Please enter v(t=0) in m per s ‘;               % ask for v(t=0)
   vinitial = input(prompt);                                 % user inputted v(t=0)
else                                                         % if they’re happy to use set values
   m = 1.79;                                                 % m from class list
   k = 3.92;                                                 % k from class list
   b = 0.1;                                                  % chosen b
   xinitial = 0;                                             % suggested x(t=0)
   vinitial = -1;                                            % suggested v(t=0)
end                                                          % end if statement
h = 0.01;                                                    % step size in t
simtime = 10;                                                % total run time
steps = simtime/h;                                           % number of steps
t = 0:h:simtime;                                             % create t array
t = t(1:length(t)-2);                                        % cut last 2 elements of t array
% ————————————————————————————————————————————
% ——————————————————————————————————————————————
% solutions for varying methods
[x1, v1, E1] = doEuler(xinitial, vinitial, m, k, b, zeros(size(t)), h, steps);
                                                    % call Euler function
[x2, v2, E2] = improvedEuler(xinitial, vinitial, m, k, b, zeros(size(t)), h, steps);
                                                    % call improved Euler method
[x3, v3, E3] = Verlet(xinitial, vinitial, x2, v2, m, k, b, zeros(size(t)), h, steps);
                                                    % call Verlet  function
[x4, v4, E4] = EulerCromer(xinitial, vinitial, m, k, b, zeros(size(t)), h, steps);
                                                    % call Euler-Cromer function
[x5, v5, E5] = analytic(xinitial, vinitial, t, m, k, b, steps); % call analytic function
[x6, v6, E6] = RK4(xinitial, vinitial, t, m, k, b, zeros(size(t)), h, steps);
                                                    % call RK4 method
% ——————————————————————————————————————————————

% -----------------------------------------------------------------------------------------------
% plot solutions for comparison
fig1 = figure;                                      % new figure window
plot(t, x1); grid                                   % plot Euler  method
xlabel('t (s)’, ‘FontSize’, 12);                    % x label
ylabel('x (m)', ‘FontSize’, 12);                    % y label
set(gca, ‘FontSize’, 12);                           % axes ticks font size
hold on;                                            % plots on same figure
plot(t, x2);                                        % plot improved Euler method
plot(t, x3);                                        % plot Verlet method
plot(t, x4);                                        % plot Euler-Cromer method
plot(t, x5);                                        % plot analytic method
plot(t, x6);                                        % plot RK4 method
g = legend(‘Euler', 'improved Euler', 'Verlet', 'Euler-Cromer', ‘analytic’, ‘RK4’);
                                                    % legend
set(g, ‘FontSize’, 10);                             % legend font size
g.Position = [0.82, 0.81, 0.05, 0.05];              % legend position
hold off;                                           % turn hold off
print('Fig1 method comparison', '-depsc');          % save figure
fig2 = figure;                                      % new figure window
plot(t, E1-E5); grid                                % plot Euler method
xlabel('t (s)', ‘FontSize’, 12);                    % x label
ylabel('E (J)', ‘FontSize’, 12);                    % y label
set(gca, ‘FontSize’, 12);                           % axes ticks font size
hold on;                                            % plots on same figure
plot(t, E2-E5);                                     % plot improved Euler method
plot(t, E3-E5);                                     % plot Verlet method
plot(t, E4-E5);                                     % plot Euler- Cromer method
plot(t, E6-E5);                                     % plot RK4 method
g = legend('Euler', 'improved Euler', 'Verlet', ‘Euler-Cromer’, ‘RK4’); % legend
set(g, ‘FontSize’, 10);                             % legend font size
```

```
hold off;                                      % turn hold off
print('Fig2 method energy residuals', '-depsc');   % save figure
% ------------------------------------------------------------------------------------------------------

% ------------------------------------------------------------------------------------------------------
% plot solutions to show effect of smallness of step size
[x7, v7, E7] = doEuler(xinitial, vinitial, m, k, b, zeros(size(t)), h/5, steps);
                                       % Euler function for smaller step size
[x8, v8, E8] = doEuler(xinitial, vinitial, m, k, b, zeros(size(t)), h*5, steps);
                                       % Euler function for larger step size
fig3 = figure;                                          % new figure window
xlabel('t (s)', 'FontSize', 12);                        % x label
ylabel('x (m)', 'FontSize', 12);                        % y label
set(gca, 'FontSize', 12);                               % axes ticks font size
hold on;                                                % plots on same figure
plot(t, E7-E5); grid                                    % plot smaller step size
plot(t, E1-E5);                                         % plot initial step size
plot(t, E8-E5);                                         % plot larger step size
g = legend(num2str(h/5), num2str(h), num2str(h*5));     % legend
set(g, 'FontSize', 10);                                 % legend font size
hold off;                                               % turn hold off
print('Fig3 step size comparison', '-depsc');           % save figure
% ------------------------------------------------------------------------------------------------------

% ------------------------------------------------------------------------------------------------------
% write data to file
fid = fopen('displacement.txt', 'w');                   % open file to write to
exisw = exist('displacement.txt');                      % 2 if file exists
for a = 1:length(x1);                                   % read across all data
  fprintf(fid, '%17.15f\t%17.15f\t%17.15f\t%17.15f\t%17.15f%17.15f\n', [x1(a),
             x2(a), x3(a), x4(a), x5(a), x6(a)]');   % write formatted data to file
end                                                     % end for
fclose(fid);                                            % close file
exisr = exist('displacement.txt');                      % 2 if file exists
% ------------------------------------------------------------------------------------------------------

% ------------------------------------------------------------------------------------------------------
% determine number of simulation steps for accuracy
for j = 1:length(E4);                          % loop over all data
  accuracyEC(j) = (E4(j)-E5(j))/E5(j);         % deviation from analytic energy
  accuracyV(j) = (E3(j)-E5(j))/E5(j);          % deviation from analytic energy
  accuracyRK(j) = (E6(j)-E5(j))/E5(j);         % deviation from analytic energy
end                                            % end for loop
minaccuracyEC = max(abs(accuracyEC));          % min accuracy
minaccuracyV = max(abs(accuracyV));            % min accuracy
minaccuracyRK = max(abs(accuracyRK));          % min accuracy
```

```
sigmaminaccuracyEC = std(abs(accuracyEC));   % error on min accuracy
sigmaminaccuracyV = std(abs(accuracyV));     % error on min accuracy
sigmaminaccuracyRK = std(abs(accuracyRK));   % error on min accuracy
% -------------------------------------------------------------------------------------

% -------------------------------------------------------------------------------------
% read in data generated from simulation
fid = fopen('displacement.txt', ‘r');        % read in written data
array = fscanf(fid, '%f', [6, inf]);         % scan across whole file
fclose(fid);                                 % close file
x1dat = array(1, :);                         % 1st column
x2dat = array(2, :);                         % 2nd column
x3dat = array(3, :);                         % 3rd column
x4dat = array(4, :);                         % 4th column
x5dat = array(5, :);                         % 5th column
x6dat = array(6, :);                         % 6th column
% -------------------------------------------------------------------------------------

% -------------------------------------------------------------------------------------
% plot phase space of oscillator & energy as function of time
fig4 = figure;                               % new figure window
plot(x1dat, v1); grid                        % plot Euler method
xlabel('x (m)', ‘FontSize’, 12);             % x label
ylabel('v (ms^-^1)', ‘FontSize’, 12);        % y label
set(gca, ‘FontSize’, 12);                    % axes ticks font size
xlim([-abs(vinitial) abs(vinitial)];         % clearly show ellipse
hold on;                                     % plots on same figure
plot(x2dat, v2);                             % plot improved Euler method
plot(x3dat, v3);                             % plot Verlet method
plot(x4dat, v4);                             % plot Euler-Cromer method
plot(x5dat, v5);                             % plot analytic method
plot(x6dat, v6);                             % plot RK4 method
g = legend('Euler', 'improved Euler', 'Verlet', 'Euler-Cromer', ‘analytic’, ‘RK4’);
                                             % legend
set(g, ‘FontSize’, 10);                      % legend font size
hold off;                                    % turn hold off
print('Fig4 phase space', '-depsc');         % save figure

fig5 = figure;                               % new figure window
plot(t, E1); grid                            % plot Euler method
xlabel('t (s)', ‘FontSize’, 12);             % x label
ylabel('Energy (J)', ‘FontSize’, 12);        % y label
set(gca, ‘FontSize’, 12);                    % axes ticks font size
hold on;                                     % plots on same figure
plot(t, E2);                                 % plot improved Euler method
plot(t, E3);                                 % plot Verlet method
plot(t, E4);                                % plot Euler- Cromer method
plot(t, E5);                                % plot analytic method
plot(t, E6);                                % plot RK4 method
g = legend('Euler', 'improved Euler', 'Verlet', 'Euler-Cromer', ‘analytic’, ‘RK4’);
                                            % legend
hold off;                                   % turn hold off
print('Fig5 energy', ‘-depsc');             % save figure
% -------------------------------------------------------------------------------------------------------------

disp(‘RK4 method is best’);           % tell user RK4 is best

% -------------------------------------------------------------------------------------------------------------
% plot damping term for best method
[x9] = RK4(xinitial, vinitial, t,m, k, sqrt(k*m), zeros(size(t)), h, steps);
                                                        % half critical damping
[x10] = RK4(xinitial, vinitial, t, m, k, 2*sqrt(k*m), zeros(size(t)), h, steps);
                                                        % critical damping
[x11] = RK4(xinitial, vinitial, t, m, k, 4*sqrt(k*m), zeros(size(t)), h, steps);
                                                        % double critical
fig6 = figure;                                          % new figure window
xlabel('t (s)', ‘FontSize’, 12);                        % x label
ylabel('x (m)', ‘FontSize’, 12);                        % y label
set(gca, ‘FontSize’, 12);                               % axes ticks font size
hold on;                                                % plots on same figure
plot(t, x9); grid                                       % plot smaller step size
plot(t, x10);                                           % plot initial step size
plot(t, x11);                                           % plot larger step size
g = legend('half critical', 'critical', 'double critical’);  % legend
set(g, ‘FontSize’, 10);                                 % legend font size
hold off;                                               % turn hold off
print('Fig6 damping term', '-depsc');                   % save figure
% ———————————————————————————————————————————————————————

% -------------------------------------------------------------------------------------------------------------
% effect of sin external force for best method
simtime2 = simtime*20;                                  % total run time
steps2 = simtime2/h;                                    % number of steps
t2 = 0:h:simtime2;                                      % new t array
t2 = t2(1:length(t2)-2);                                % cut last 2 elements of t2
F0 = 0.1;                                               % force magnitude
[x12] = RK4(xinitial, vinitial, t2, m, k, b, F0*sin(1.5*t2), h, steps2);
                                                        % call RK4 function
[x13] = RK4(xinitial, vinitial, t2, m, k, b, zeros(size(t2)), h, steps2);
                                                        % call RK4 function
fig7 = figure;                                          % new figure window
xlabel('t (s)', ‘FontSize’, 12);                        % x label
```

```
ylabel('x (m)', ‘FontSize’, 12);                          % y label
set(gca, ‘FontSize’, 12);                                 % axes ticks font size
hold on;                                                  % plots on same grid
plot(t2, x12); grid                                       % forced
plot(t2, x13);                                            % unforced
g = legend('forced', ‘unforced');                         % label plots
set(g, ‘FontSize’, 10);                                   % legend font size
hold off;                                                 % hold off
print('Fig7 driving force', '-depsc');                    % save figure
% -----------------------------------------------------------------------------------------------------

% -----------------------------------------------------------------------------------------------------
% amplitude of oscillation as function of frequency for best method
omega = linspace(sqrt(k/m)-0.3, sqrt(k/m)+0.3, 10000);          % omega array
A = F0./(m*sqrt((k/m - omega.^2).^2 + (omega.^2)*(b/m)^2));
                                                    % amplitude as function of omega
Alow = F0./(m*sqrt((k/m + 0.1^2 -0.2*sqrt(k/m) - omega.^2).^2 + (omega.^2)*(b/
                                                          m)^2)); % smaller omega0
Ahigh = F0./(m*sqrt((k/m + 0.1^2 +0.2*sqrt(k/m) - omega.^2).^2 + (omega.^2)*(b/
                                                          m)^2)); % higher omega0
fig8 = figure;                                            % new figure window
xlabel('\omega (s^-^1)', ‘FontSize’, 12);                 % x label
ylabel('A (m)', ‘FontSize’, 12);                          % y label
set(gca, ‘FontSize’, 12);                                 % axes ticks font size
xlim([min(omega) max(omega)]);                            % set x axis limits
hold on;                                                  % plots on same grid
plot(omega, A); grid                                      % plot figure
plot(omega, Alow);                                        % plot low omega0
plot(omega, Ahigh);                                       % plot high omega0
g = legend('$$\sqrt{k/m}$$', '$$\sqrt{k/m}$$ - 0.1', '$$\sqrt{k/m}$$ + 0.1’);
                                                          % label plots
set(g, 'Interpreter', ‘latex’, ‘FontSize’, 12);           % use sqrt in legend
[maxA, maxw] = findpeaks(A, omega);                       % find max
[maxAlow, maxwlow] = findpeaks(Alow, omega);              % find low omega0 max
[maxAhigh, maxwhigh] = findpeaks(Ahigh, omega);           % find high omega0 max
maxwstr = num2str(maxw);                                  % change maxw to string
maxwlowstr = num2str(maxwlow);                  % change maxwlow to string
maxwhighstr = num2str(maxwhigh);                % change maxwhigh to string
text(maxw, maxA, sprintf('max at %s s^-^1', maxwstr), ‘FontSize’, 11);
                                                % label maxima
text(maxwlow, maxAlow, sprintf('max at %s s^-^1', maxwlowstr), ‘FontSize’, 11);
                                                % label maxima
text(maxwhigh, maxAhigh, sprintf('max at %s s^-^1', maxwhighstr), ‘FontSize’,
11);                                            % label maxima
line([sqrt(k/m) sqrt(k/m)], [0 F0/(b*sqrt(k/m))], 'LineStyle', ‘--'); % line at omega0
line([sqrt(k/m)-0.1 sqrt(k/m)-0.1], [0 F0/(b*sqrt(k/m + 0.1^2 -0.2*sqrt(k/m)))],
          'LineStyle', '--', 'Color', 'r');                  % line at omega0 - 0.1
line([sqrt(k/m)+0.1 sqrt(k/m)+0.1], [0 F0/(b*sqrt(k/m + 0.1^2 +0.2*sqrt(k/m)))],
          'LineStyle', '--', 'Color', 'y');                  % line at omega0 + 0.1
text(sqrt(k/m), maxA/2, '$$\sqrt{k/m}$$', 'Interpreter', 'latex', 'Color', ‘b’, ‘Font
                                        Size’, 11);       % label omega0
text(sqrt(k/m)-0.1, maxAlow/2, '$$\sqrt{k/m}$$ - 0.1', 'Interpreter', 'latex', 'Color',
                                                 'r', ‘FontSize’, 11);             % label omega0 - 0.1
text(sqrt(k/m)+0.1, maxAhigh/2, '$$\sqrt{k/m}$$ + 0.1', 'Interpreter', 'latex', 'Color',
                                                 ‘y', ‘FontSize’, 11);             % label omega0 + 0.1
hold off;                                             % turn hold off
deltaomega = sqrt(k/m) - maxw;               % difference between max and omega0
deltaomegalow = sqrt(k/m) - 0.1 - maxwlow;
                                                      % difference between max & omega0low
deltaomegahigh = sqrt(k/m) + 0.1 - maxwhigh;
                                                      % difference between max & omega0high
print('Fig8 resonance curve', '-depsc');      % save figure
% ---------------------------------------------------------------------------------------------------------

% ---------------------------------------------------------------------------------------------------------
% change parameters to pre set values
m = 1.79;                                    % pre set m
k = 3.92;                                    % pre set k
b = 0.1;                                     % pre set b
xinitial = 0;                                % pre set x(t=0)
vinitial = -1;                               % pre set v(t=0)
disp('Figures 9-12 are for m = 1.79 kg, k = 3.92 N per m, b = 0.1 Ns per m, x(t=0) = 0
          m, v(t=0) = -1 m per s’);       % tell user pre set values now being used
% —————————————————————————————————————————————————————

% ---------------------------------------------------------------------------------------------------------
% effect of sudden application of external force for best method
% force at peak
[x14, v14] = RK4(xinitial, vinitial, t, m, k, b, zeros(1, 743), h, 743);
                                                      % call RK4 function
[x15] = RK4(x14(end), v14(end), t, m, k, b, -ones(1, int64(steps) - 743 + 2), h,
                              steps - 743 + 2);         % call RK4 function
x15 = x15(2:length(x15));                     % cut 1st element of x15
x16 = cat(2, x14, x15);                       % put x14 & x15 together
[x17] = RK4(x14(end), v14(end), t, m, k, b, ones(1, int64(steps) - 743 + 2), h,
                              steps - 743 + 2);         % call RK4 function
x17 = x17(2:length(x17));                     % cut 1st element of x17
x18 = cat(2, x14, x17);                       % put x14 & x17 together
fig9 = figure;                               % new figure window
xlabel('t (s)', ‘FontSize’, 12);             % x label
ylabel('x (m)', ‘FontSize’, 12);             % y label
```

```
set(gca, 'FontSize', 12);                    % axes ticks font size
ylim([-1.1 1.1]);                            % set y limits
hold on;                                     % plots on same figure
plot(t, x16); grid                           % same direction as v(t=0)
plot(t, x18);                                % opposite direction to v(t=0)
g = legend('same direction as initial velocity', 'opposite direction to initial velocity');
                                             % label plots
set(g, 'FontSize', 10);                      % legend font size
hold off;                                    % hold off
print('Fig9 sudden force peak', '-depsc');  % save figure

% force at trough
[x19, v19] = RK4(xinitial, vinitial, t, m, k, b, zeros(1, 531), h, 531);
                                                     % call RK4 function
[x20] = RK4(x19(end), v19(end), t, m, k, b, -ones(1, int64(steps) - 531 + 2), h,
                        steps - 531 + 2);            % call RK4 function
x20 = x20(2:length(x20));                            % cut 1st element of x20
x21 = cat(2, x19, x20);                              % put x19 & x20 together
[x22] = RK4(x19(end), v19(end), t, m, k, b, ones(1, int64(steps) - 531 + 2), h,
                        steps - 531 + 2);            % call RK4 function
x22 = x22(2:length(x22));                            % cut 1st element of x22
x23 = cat(2, x19, x22);                              % put x19 & x22 together
fig10 = figure;                                      % new figure window
xlabel('t (s)', 'FontSize', 12);                     % x label
ylabel('x (m)', 'FontSize', 12);                     % y label
set(gca, 'FontSize', 12);                            % axes ticks font size
ylim([-1.1 1.1]);                                    % set y limits
hold on;                                             % plots on same grid
plot(t, x21); grid                                   % same direction as v(t=0)
plot(t, x23);                                        % opposite direction to v(t=0)
g = legend('same direction as initial velocity', 'opposite direction to initial velocity');
                                                     % label plots
g.Location = 'best';                                 % best location for legend
set(g, 'FontSize', 10);                              % legend font size
hold off;                                            % hold off
print('Fig10 sudden force trough', '-depsc');        % save figure

% force when x=0 & v<0
[x24, v24] = RK4(xinitial, vinitial, t, m, k, b, zeros(1, 426), h, 426);
                                             % call RK4 function
[x25] = RK4(x24(end), v24(end), t, m, k, b, -ones(1, int64(steps) - 426 + 2), h,
                        steps - 426 + 2);    % call RK4 function
x25 = x25(2:length(x25));                    % cut 1st element from x25
x26 = cat(2, x24, x25);                      % put x24 & x25 together
[x27] = RK4(x24(end), v24(end), t, m, k, b, ones(1, int64(steps) - 426 + 2), h,
                        steps - 426 + 2);    % call RK4 function
```

```
x27 = x27(2:length(x27));                           % cut 1st element of x27
x28 = cat(2, x24, x27);                             % put x24 & x27 together
fig11 = figure;                                     % new figure window
xlabel('t (s)', ‘FontSize’, 12);                    % x label
ylabel('x (m)', ‘FontSize’, 12);                    % y label
set(gca, ‘FontSize’, 12);                           % axes ticks font size
ylim([-1.1 1.1]);                                   % set y limits
hold on;                                            % plots on same grid
plot(t, x26); grid                                  % same direction as v(t=0)
plot(t, x28);                                       % opposite direction to v(t=0)
g = legend('same direction as initial velocity', 'opposite direction to initial velocity’);
                                                    % label plots
g.Location = ‘best’;                                % best location for legend
set(g, ‘FontSize’, 10);                             % legend font size
hold off;                                           % turn hold off
print('Fig11 sudden force -ve 0', '-depsc');        % save figure

% force at x=0 & v>0
[x29, v29] = RK4(xinitial, vinitial, t, m, k, b, zeros(1, 638), h, 638);
                                                    % call RK4 function
[x30] = RK4(x29(end), v29(end), t, m, k, b, -ones(1, int64(steps) - 638 + 2), h,
                        steps - 638 + 2);           % call RK4 function
x30 = x30(2:length(x30));                           % cut 1st element of x30
x31 = cat(2, x29, x30);                             % put x29 & x30 together
[x32] = RK4(x29(end), v29(end), m, k, b, ones(1, int64(steps) - 638 + 2), h,
                        steps - 638 + 2);           % call RK4 function
x32 = x32(2:length(x32));                           % cut 1st element of x32
x33 = cat(2, x29, x32);                             % put x29 & x32 together
fig12 = figure;                                     % new figure window
xlabel('t (s)', ‘FontSize’, 12);                    % x label
ylabel('x (m)', ‘FontSize’, 12);                    % y label
set(gca, ‘FontSize’, 12);                           % axes ticks font size
ylim([-1.1 1.1]);                                   % set y limits
hold on;                                            % plots on same grid
plot(t, x31); grid                                  % same direction as v(t=0)
plot(t, x33);                                       % opposite direction to v(t=0)
g = legend('same direction as initial velocity', 'opposite direction to initial velocity’);
                                                    % label plots
g.Location = ‘best’;                                % best location for legend
set(g, ‘FontSize’, 10);                             % legend font size
hold off;                                           % turn hold off
print('Fig12 sudden force +ve 0', '-depsc');        % save figure
% ————————————————————————————————————————————————————————————————
% ————————————————————————————————————————————————————————————————
% ———————————————————————————————————————————————
% ———————————————————————————————————————————————
function [ x, v, E ] = analytic( xinitial, vinitial, t, m, k, b, steps )
%analytic calculates solutions for analytic method
%   calculates motion for analytic method
   if vinitial <= 0                                         % -ve v
      phi = atan(sqrt(k/m)*xinitial/vinitial);              % phi
   else                                                     % +ve v
      phi = atan(sqrt(k/m)*xinitial/vinitial) + pi;         % pi phase change
   end                                                      % end if
   A = sqrt((xinitial^2 + vinitial^2)/(sin(phi)^2 + (k*cos(phi)^2)/m));    % amplitude
   for n = 1:steps-1;                                       % start of for loop
      x(n) = -A*exp(-b*t(n)/(2*m))*sin(t(n)*sqrt(k/m - (b/(2*m))^2) - phi);
                                                            % new displacement
      v(n) = -A*exp(-b*t(n)/(2*m))*(-(b/(2*m))*sin(t(n)*sqrt(k/m - (b/(2*m))^2) - phi)
                 + sqrt(k/m - (b/(2*m))^2)*cos(t(n)*sqrt(k/m - (b/(2*m))^2) - phi));
                                                            % new velocity
      E(n) = 0.5*m*v(n)^2 + 0.5*k*x(n)^2;                   % new energy
   end                                                      % end of for loop
end                                                         % end of function
% ———————————————————————————————————————————————

% ———————————————————————————————————————————————
function [ x, v, E ] = doEuler( xinitial, vinitial, m, k, b, F, h, steps )
%Euler calculates solutions for Euler method
%   Takes initial displacement, velocity & time & calculates motion for Euler method
   x(1) = xinitial;                              % 1st displacement term in loop
   v(1) = vinitial;                              % 1st velocity term in loop
   for n = 1:steps - 1;                          % start of for loop
      a(n) = -k*x(n)/m - b*v(n)/m + F(n)/m;      % acceleration
      E(n) = 0.5*m*v(n)^2 + 0.5*k*x(n)^2;        % energy
      x(n+1) = x(n) + v(n)*h;                    % new displacement
      v(n+1) = v(n) + a(n)*h;                    % new velocity
   end                                           % end of for loop
   x = x(1:length(x)-1);                         % cut last element of x array
   v = v(1:length(v)-1);                         % cut last element of v array
end                                              % end of function
% ———————————————————————————————————————————————

% ———————————————————————————————————————————————
function [ x, v, E ] = improvedEuler( xinitial, vinitial, m, k, b, F, h, steps )
%improvedEuler calculates solutions for improved Euler method
%   Takes initial displacement, velocity & time & calculates motion for improved
       Euler method
   x(1) = xinitial;                              % 1st displacement term in loop
   v(1) = vinitial;                              % 1st velocity term in loop
```

```
    for n = 1:steps - 1;                         % start of for loop
        a(n) = -k*x(n)/m - b*v(n)/m + F(n)/m;    % acceleration
        E(n) = 0.5*m*v(n)^2 + 0.5*k*x(n)^2;      % energy
        x(n+1) = x(n) + v(n)*h + 0.5*h*h*a(n);   % new displacement
        v(n+1) = v(n) + a(n)*h;                  % new velocity
    end                                          % end of for loop
    x = x(1:length(x)-1);                        % cut last element of x array
    v = v(1:length(v)-1);                        % cut last element of v array
end                                              % end of function
% ---------------------------------------------------------------------------

% ---------------------------------------------------------------------------
function [ x, v, E ] = Verlet( xinitial, vinitial, x2, v2, m, k, b, F, h, steps )
%Verlet calculates solutions for Verlet method
%   Takes initial displacement & time & calculates motion for Verlet method
    x(1) = xinitial;                             % 1st displacement term in loop
    v(1) = vinitial;                             % 1st velocity term in loop
    x(2) = x2(2);                                % 2nd displacement term in loop
    v(2) = v2(2);                                % 2nd velocity term in loop
    E(1) = 0.5*m*v(1)^2 + 0.5*k*x(1)^2;          % 1st energy term in loop
    for n = 2:steps - 1;                         % start of for loop
        a(n) = -k*x(n)/m - b*v(n)/m + F(n)/m;    % acceleration
        E(n) = 0.5*m*v(n)^2 + 0.5*k*x(n)^2;      % energy
        x(n+1) = 2*x(n) - x(n-1) + h*h*a(n);     % new displacement
        v(n+1) = (x(n+1)-x(n))/(h);              % new velocity
    end                                          % end of for loop
    x = x(1:length(x)-1);                        % cut last element of x array
    v = v(1:length(v)-1);                        % cut last element of v array
end                                              % end of function
% ---------------------------------------------------------------------------

% ---------------------------------------------------------------------------
function [ x, v, E ] = EulerCromer( xinitial, vinitial, m, k, b, F, h, steps )
%EulerCromer calculates solutions for Euler-Cromer method
%   Takes initial displacement, velocity & time & calculates motion for Euler-
        Cromer method
    x(1) = xinitial;                             % 1st displacement term in loop
    v(1) = vinitial;                             % 1st velocity term in loop
    for n = 1:steps - 1;                         % start of for loop
        a(n) = -k*x(n)/m - b*v(n)/m + F(n)/m;    % acceleration
        E(n) = 0.5*m*v(n)^2 + 0.5*k*x(n)^2;      % energy
        v(n+1) = v(n) + a(n)*h;                  % new velocity
        x(n+1) = x(n) + v(n+1)*h;                % new displacement
    end                                          % end of for loop
    x = x(1:length(x)-1);                        % cut last element of x array
    v = v(1:length(v)-1);                        % cut last element of v array
```

```
end                                                    % end of function
% ----------------------------------------------------------------------

% ----------------------------------------------------------------------
function [ x, v, E ] = RK4( xinitial, vinitial, t, m, k, b, F, h, steps )
%RK4 calculates solutions for RK4 method
%   Takes initial displacement & time & calculates motion for RK4 method
   x(1) = xinitial;                                    % 1st displacement term in loop
   v(1) = vinitial;                                    % 1st velocity term in loop
   for n = 1:steps - 1;                                % start of for loop
      a(n) = -k*x(n)/m - b*v(n)/m + F(n)/m;            % acceleration
      E(n) = 0.5*m*v(n)^2 + 0.5*k*x(n)^2;              % energy
      y = [x(n), v(n)];                                % array of x & v
      dy = @deriv;                                     % use derive function
      k1=dy(y, t(n), k, m, b, F(n));                   % 1st calculation point
      k2=dy(y+(h/2)*k1 ,t(n)+h/2, k, m, b, F(n));      % 2nd calculation point
      k3=dy(y+(h/2)*k2, t(n)+h/2, k, m, b, F(n));      % 3rd calculation point
      k4=dy(y+h*k3, t(n)+h, k, m, b, F(n));            % 4th calculation point
      y = y + h*(k1+2*k2+2*k3+k4)/6;                   % new y
      x = [x y(1)];                                    % x is 1st element of y
      v = [v y(2)];                                    % v is 2nd element of y
   end                                                 % end of for loop
   x = x(1:length(x)-1);                               % cut last element of x array
   v = v(1:length(v)-1);                               % cut last element of v array
end                                                    % end of function
% ----------------------------------------------------------------------

% ----------------------------------------------------------------------
function [ retval ] = deriv( y , t, k, m, b, F)
%deriv function to be used in RK4 method
%   derivative of x is v & derivative of v is a
   retval=[y(:,2), -k*y(:,1)/m - b*y(:,2)/m + F/m];          % derivative of y array
end                                                          % end function
% ----------------------------------------------------------------------
% ----------------------------------------------------------------------
```