Modelling simple oscillators using numerical methods

Student ID: 83458200
Joseph Kellett

School of Physics and Astronomy
The University of Manchester

A Technical Report

March 2014

Abstract

The best numerical integrator for simulating a simple harmonic oscillator was determined by comparing the results of the methods with the analytical solution for solvable situations. The Verlet method was found to produce the least error for a given step size. The Verlet method was then used to determine the characteristics of an oscillator with mass 5.21kg, spring constant 0.6 Nm^{-1} and varying damping coefficients. This included situations that could not be solved analytically such as certain forced oscillations. The Verlet method was found to agree with several theoretical results and successfully predicted resonance. The relationship between the resonant amplitude and the damping coefficient was explored using the produced results.

1. Introduction

Numerical methods are often used to model systems that cannot be solved analytically. In this case both symplectic and non-symplectic integrators are used to model a damped harmonic oscillator. In general these integrators can be used to solve many different kinds of differential equations to varying degrees of accuracies. Modern day applications of numerical integrators include weather forecasting and solving complex quantum-mechanical wave equations.

Numerical methods in general have been in existence since ancient times however their utility has increased with the age of computing allowing for fast computation of simple calculations.[1]

2. Theory

2.1. The analytical solution to a damped harmonic oscillator

The solution to damped harmonic motion can be derived by substituting Hooke's law and a damping term into Newton's second law.

$$m\ddot{x} = -kx - \gamma\dot{x} \tag{1}$$

where m is the mass of the oscillator, k is the spring constant of the spring, x is the displacement of the oscillator from its mean position, $\ddot{x}$ is the acceleration and γ is the damping coefficient.[2]

Using the trial solution $x(t) = Ae^{\lambda t}$,where A and λ are complex constants, it can be shown that:

$$m\lambda^2 + \gamma\lambda + k = 0 \tag{2}$$

The solution to this auxiliary equation is:

$$\lambda = \frac{-\gamma \pm \sqrt{\gamma^2 - 4mk}}{2m} \tag{3}$$

A is determined by the initial conditions of the oscillator. It should be noted that when $\gamma=0$ the solution simplifies to:

$$x(t) = Ae^{i\omega t} \tag{4}$$

where $\lambda = i\omega = i\sqrt{\frac{k}{m}}$. This is the solution to simple harmonic motion and also provides the natural angular frequency of the oscillator: $\omega_0 = \sqrt{\frac{k}{m}}$ [2]. It should also be noted that when $\gamma^2 - 4mk > 0$, λ is real and the system does not oscillate, fulfilling the definition of heavy damping. Likewise light damping is defined as $\gamma^2 - 4mk < 0$ and critical damping is defined as $\gamma^2 - 4mk = 0$.

The resonant angular frequency of a damped oscillator is given by:

$$\omega = \omega_0 \sqrt{1 - \frac{\gamma^2}{2\omega_0^2}} \tag{5}$$

2.2. The Euler integrator

The Euler method relies on the fact that any curve can be approximated as a series of tangents to the curve. This allows the derivative of the curve to be approximated using finite differences.

$$y'(x) = \frac{y(x+h) - y(x)}{h} + O(h) \tag{6}$$

where *h* is the step size and *O(h)* means the error is of the order of h and higher.

Equation 6[1]. can be used to find the velocity and subsequently the acceleration of a harmonic oscillator. By substituting these solutions into each other it can be shown that:

$$x_{n+1} = x_n + v_n . h + O(h^2) \tag{7}$$

$$v_{n+1} = v_n + a_n . h + O(h^2) \tag{8}$$

The acceleration (*a*) can be found by computing the force acting on the oscillator at that point in time.

2.3. The Improved Euler integrator

The Improved Euler method takes account of the errors of order h^2 to give more accurate values for the displacement and velocity at the expense of becoming slightly more computationally expensive.[1]

$$x_{n+1} = x_n + v_n . h + \frac{h^2}{2} . a_n + O(h^3) \tag{9}$$

The equation for the velocity of the oscillator is the same as that of the Euler method.

2.4. The Euler-Cromer integrator

The Euler-Cromer method is an adaption of the Euler method that calculates the displacement using the velocity found at the end of the step instead of the beginning.

$$x_{n+1} = x_n + v_{n+1} . h + O(h^2) \tag{10}$$

The equation for the velocity of the oscillator is the same as that of the Euler method.

2.5. The Verlet integrator

The Verlet method utilises a central derivative[1]:

$$y'(x) \approx \frac{y(x+h) - y(x-h)}{2\mathrm{h}} \tag{11}$$

When applied to solving the differential equation of the harmonic oscillator it can be shown that:

$$v_n = \frac{x_{n+1} - x_{n-1}}{2\mathrm{h}} + O(h^2) \tag{12}$$

$$a_n = \frac{x_{n+1} + x_{n-1} - 2\mathrm{x}_n}{h^2} + O(h^2) \tag{13}$$

Equation (13) can be rearranged to show that:

$$x_{n+1} = 2\mathrm{x}_n - x_{n-1} + h^2 . a_n + O(h^4) \tag{14}$$

3. Determining the best numerical method to model the oscillator

The Euler, Euler-Cromer, Improved Euler and Verlet numerical methods were compared to the analytical solution in the solvable case of no damping. The analytical solution predicts that the total energy of the system and the amplitude of the oscillations remains constant. The initial displacement of the oscillator was 0m and the initial velocity was -1ms^{-1}.

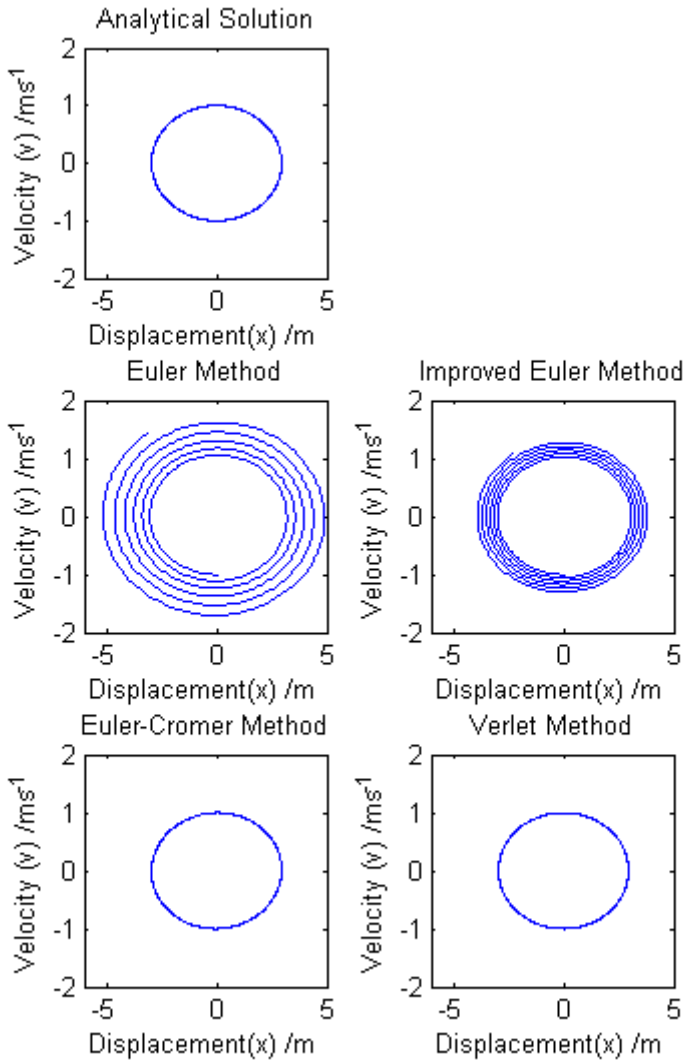

Fig 1. Phase space graphs for all the methods with the numerical solution for comparison. The step size used was 0.1s and the simulation was run for 100s.

Fig 1. shows that both the Euler and Improved Euler methods diverge from the true value which is inaccurate. Euler-Cromer does not diverge however it is a different shape from the analytical solution. This suggests that the values predicted by Euler-Cromer are less accurate than Verlet's, however the inaccuracy is constant over time.

The energies were calculated for each method from the sum of the kinetic energy and the potential energy of the oscillator predicted by the values of displacement and velocity.

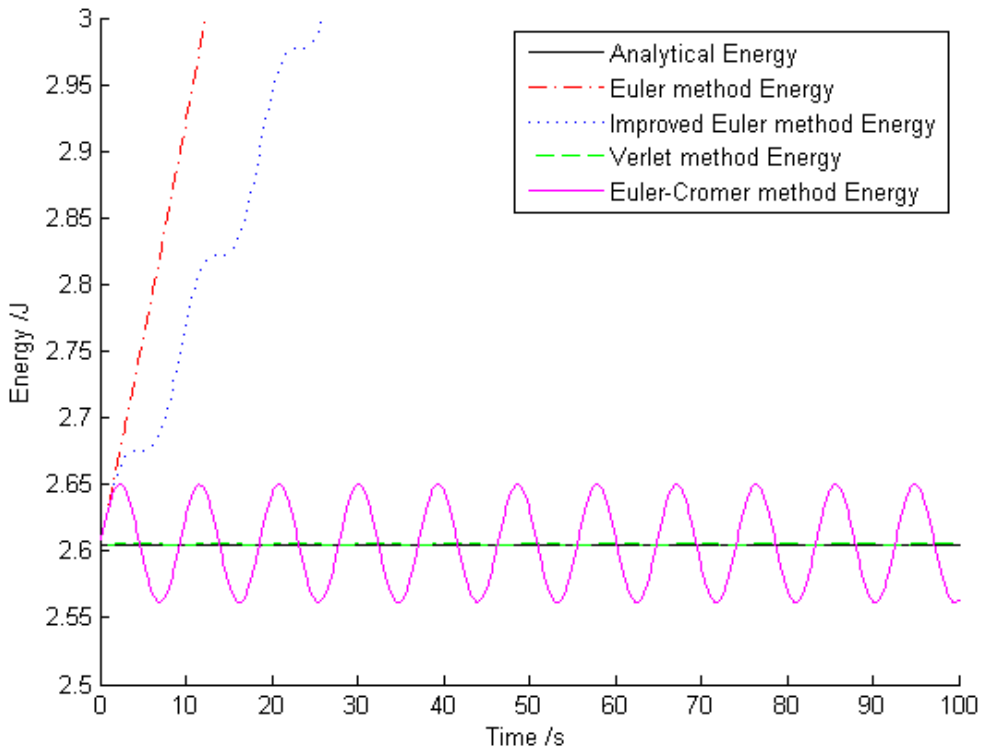

Fig 2. The energy of the oscillator as described by various methods using a step size of 0.1 seconds over 100 seconds. Euler-Cromer oscillates about the true value.

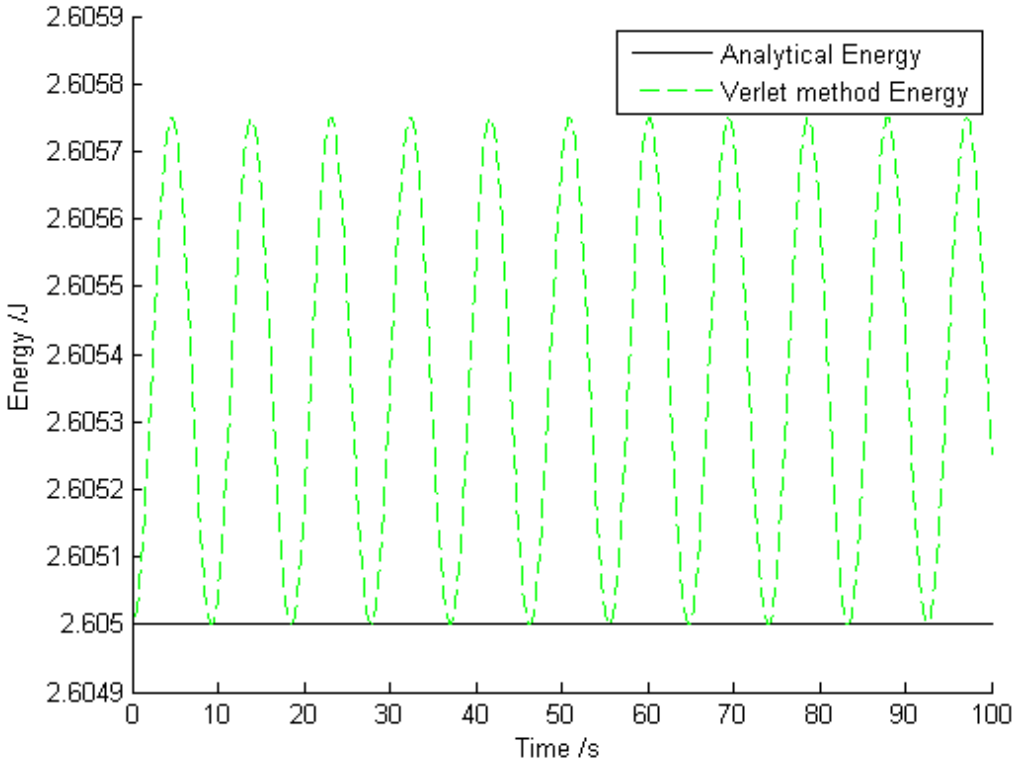

Fig. 3 A magnified version of Fig 2. containing only the Verlet method and analytical solution. The Verlet energy oscillates above the energy predicted by the analytical solution.

Fig 2. and Fig 3. shows that in terms of energy Verlet is the most accurate and that both Verlet and Euler-Cromer are symplectic (conserve energy over large periods of time). Verlet varies above the analytical energy (2.605 Joules) while Euler-Cromer oscillates about the analytical energy.

To quantify the relative accuracies of the different the step size required for an error of 0.1% on the energy value of each method compared with the analytic solution over 100 seconds. The run time for each method was also recorded using the Run and time feature of **Matlab** to account for the fact that step size does not directly correspond to the efficiency of the method.[3]

Numerical Method	Required step size /s	Run time /s
Euler	0.0000867	1.3
Improved Euler	0.000176	0.079
Euler-Cromer	0.00588	0.0092
Verlet	0.186	0.00011

Table 1. The required step size to achieve 0.1% accuracy on the energy value after 100 seconds. The time it took for each method to run with the specified step size is specific to the computer system it was tested on and should only be considered relative to the run times of other methods.

It was concluded that the Verlet method was the best method to use when modelling the oscillator as it while Euler-Cromer oscillates about the true value of energy it does not accurately predict the displacement and velocity terms. In a situation that requires a time averaged energy term over an integer number of oscillations Euler-Cromer would be the better choice as the mean energy of the Verlet method is above the analytical value.

4. Investigating the properties of a damped harmonic oscillator

The Verlet method was used to model the effects of damping for light, critical and damping. For comparison the numerical solution was also calculated over the same period of time and for the same conditions. The damping coefficients for the light damping, critical damping and heavy damping were $\gamma=\frac{1}{2}\sqrt{mk}$, $\gamma=\sqrt{mk}$ and $\gamma=2\sqrt{mk}$ respectively.

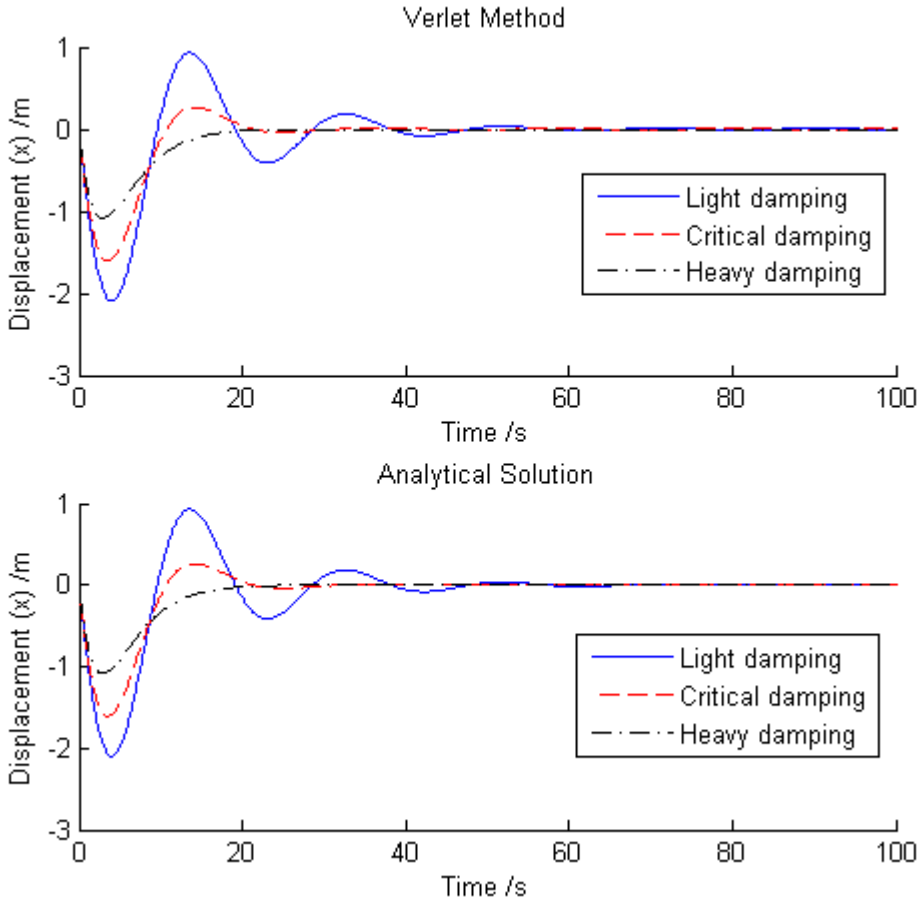

Fig 4. The effects of damping on the oscillator. The results predicted by the Verlet method and the analytical solution are the same within a small margin of error.

These results conform with theory as described in section 2.1. It also confirms that the Verlet continues to be accurate when the damping term is non-zero.

5. Investigating the characteristics of a forced oscillator and resonance

The Verlet method was used to investigate the properties of a non-damped oscillator when exposed to various types of force. The majority of the time this is not solvable analytically.

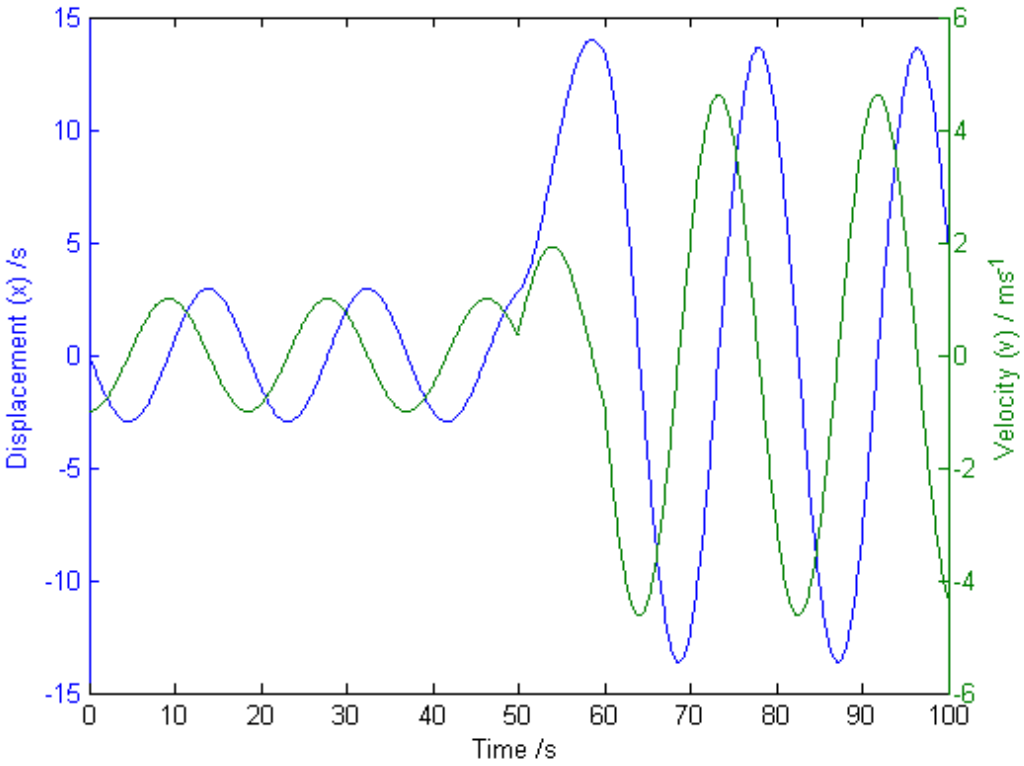

Fig 5. The effect of a step force of 5N for 5s starting at 50s on the displacement of the oscillator.

The motion exhibited in Fig 5. can be explained by considering the sum of the forces acting on the mass of the oscillator. The external driving force increases the velocity past its original amplitude. The acceleration at 50s is constant as the external force dwarfs the force due to the spring, however as the displacement increases the force from the spring begins to curb the acceleration. After the force ceases at 55s simple harmonic motion is resumed albeit with larger amplitudes. Again this is to be expected as the external force has transferred energy to the system.

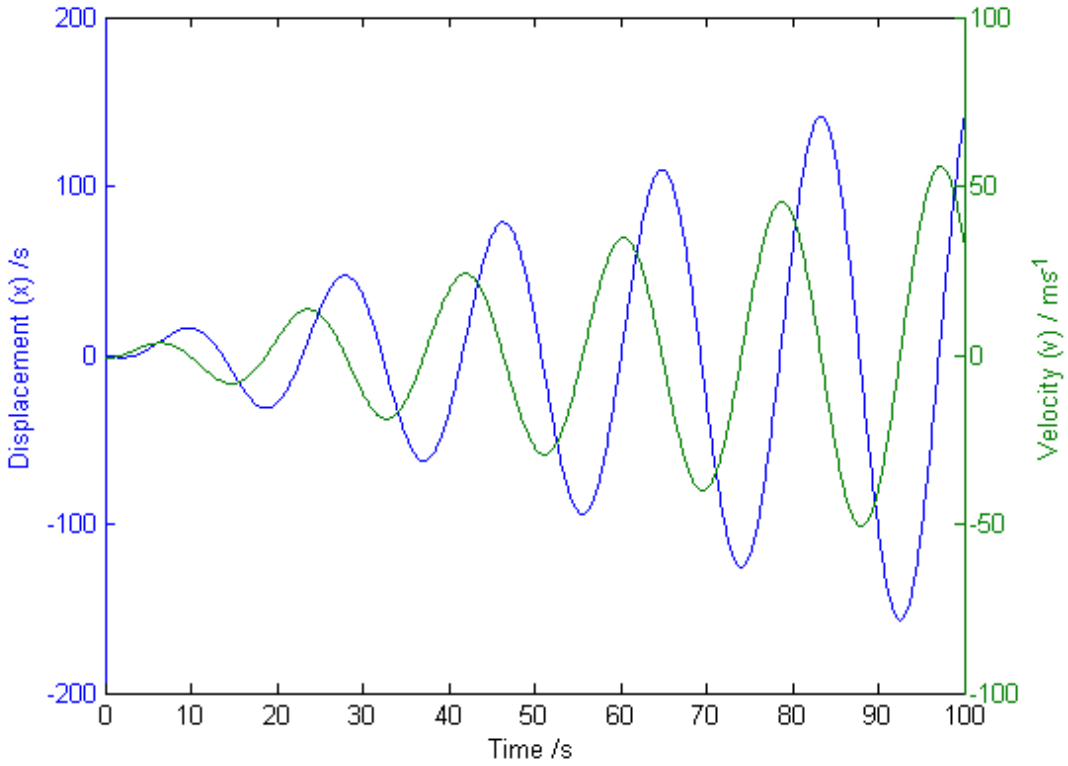

Fig 6. The effect of a sinusoidal force of amplitude 6N oscillating at $\omega = \omega_0$, the resonant frequency for an non-damped oscillator.

The motion exhibited Fig 6. demonstrates how a non-damped will continue to increase with time. The amplitudes of displacement will tend to infinity as time tends to infinity, as shown in Fig 9.

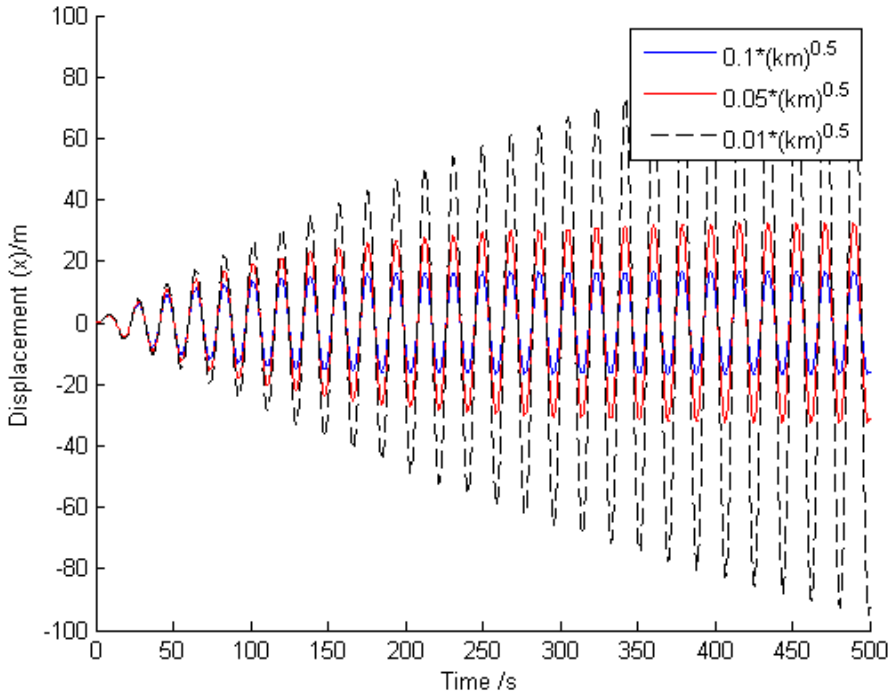

Fig 7. The effect of a sinusoidal force of 1N oscillating at $\omega = \omega_0$ on a system with three different values of damping coefficient.

Fig 7. suggests that given enough time a damped oscillator that is forced at the resonant frequency will reach a maximum amplitude. This infers that this occurs when the power from the external force added to the system and the power lost due to the damping term are equal.

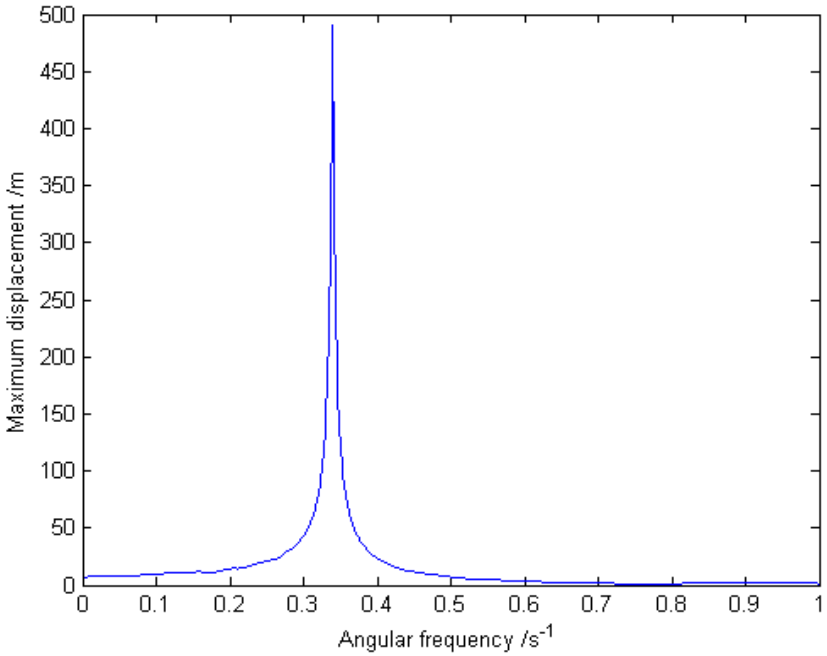

Fig 8. The resonance curve for a forced oscillator with peak force of 3N and damping coefficient of 1% the critical damping value ($\sqrt{k \times m}$). The oscillator was started at rest to prevent extra transient motion. The resonant angular frequency was found to be at $0.339 s^{-1}$.

The resonant frequency found for this curve was agrees with the value calculated from equation (5) to the degree of precision that it was calculated to.

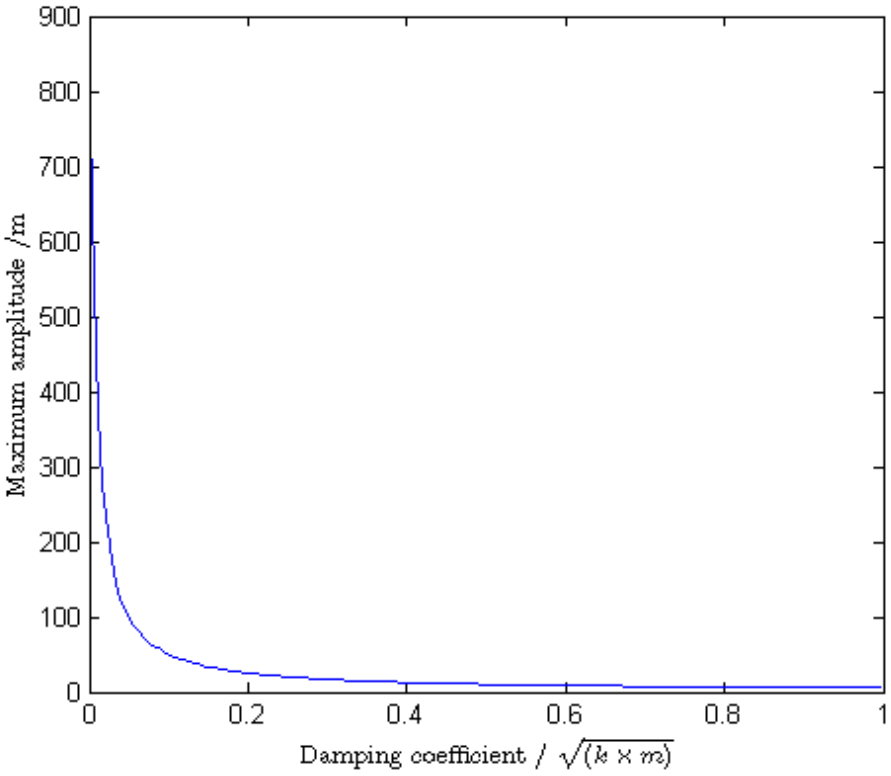

Fig 9. The maximum amplitude of the oscillator at its resonant frequency for varying coefficients of damping (γ). The x axes is in units of the critical damping coefficient.

Fig 9. was produced by simulating the forced damped harmonic oscillator for 5000s. This is several order of magnitudes higher than the time required for most of the terms in Fig 7. to plateau. It can therefore be assumed that for values of γ above 10% the critical damping value Fig 9 is highly accurate however for values far below that the accuracy is uncertain. The simulation time was limited to 5000s to prevent excessive run times when simulating the oscillator.

Further investigation is required to quantify the relationship between the damping coefficient and the maximum amplitude of a resonant oscillator.

5. Errors in numerical methods

Numerical methods have multiple sources of error. These come from the nature of using computers to store and process numbers and the fact that the step size for each method is finite.

The step size used to model the oscillator in sections 4 to 5 was 0.01s unless stated otherwise. This was chosen as it produced a relatively high accuracy when using the Verlet method yet does not take excessive amounts of processing time. The effect of the step size (h) on the error of the numerical methods varies with the different types, as shown by Table 1. These results are to be expected as the error on the Euler and Euler-Cromer methods is of order h^2, Improve Euler is of the order h^3 and the error on the Verlet method is of the order h^4 showing that large step size has less effect on the Verlet method's errors.

A computer will exhibit rounding errors on values with higher precision than can be stored. This can be mitigated by using a higher precision when storing values, in this investigation the precision of the significand of a number has precision of about 16 decimal places.[3] When the proportional errors on the Verlet energy are of the order of 0.01% (affecting the 4^{th} decimal place of the significand) for the step size of 0.01, it can be seen that the rounding errors can usually be consider negligible.

The exception to this is when a large number is added or subtracted to a small number, which can cause the small number to be rounded off. This is significant when the large number is later removed from the sum. To counter this every effort was taken to avoid such calculations when writing the scripts to model the oscillator.

6. Conclusion

This investigation has shown of the four numerical integrators tested, the Verlet method fitted the requirements of high accuracies for displacement, velocity and energy for small computing times.

The Verlet agreed with the analytical solution for all solvable situations and was used to calculate the resonant frequency of the forced damped oscillator and found it to be the same as the predicted value within the precision of the value.

References

[1] Hamming, R.W. (1987) *Numerical Methods for Scientists and Engineers,* Dover Books

[2] King, G.C. (2009) *Vibrations and Waves,* Wiley

[3] **MATLAB** User Documentation, The Math Works Inc. Available at: www.mathworks.co.uk/help/matlab/ (Date accessed: 31/03/2014)

Appendix

Note: There are two scripts recorded here. SHMsimulator allows the user to input initial conditions to find the displacement, velocity and energy of the oscillator as modelled by the 4 different numerical described in this report. Project2ReportScript is the script used to produce the data and graphs used in this report. The custom functions used in both these scripts are also recorded

SHMsimulator

```
%--------------------------------------------------------------------------
%SHMSIMULATOR Plots the energy of a simple harmonic oscillator using 4
%different numerical methods and an analytical solution.
%The user is given the opportunity to create a new set of SHO solutions
%with these 5 methods, or can skip and use data from created either
%manually or from a previous session.
%The energies predicted from these 5 different methods are then calculated
%and plotted.
% -------------------------------------------------------------------------
% Joseph Kellett
% University of Manchester
% March 2014
% -------------------------------------------------------------------------

close all;
clear all;
format long;

%Greet the user and ask for initial conditions.
fprintf('Welcome to SHMsimulator.')
skip=input('Do you wish to find the energies for a previous oscillator?  y/n
');
if skip==n %User wants to create files for a new oscillator....
    fprintf('You will be asked to enter the values of the \n');
    fprintf('to the following differential equation describing a damped
harmonic oscillator \n');
    fprintf('\n m*a+d*v+k*x=0 \n');
    fprintf('\n where m is the mass of the oscillator, k the spring
constant, \n');
    fprintf('d the damping coefficient, a the acceleration, v the velocity
and \n');
    fprintf('and x the diplacement.\n\n');
    m=input('Please enter the mass of the oscillator, m, in kilograms.  ');
    k=input('Please enter the value of the spring constant, k.   ');
    d=input('Please enter the damping coefficient of the system, d.  ');
    x_0=input('Please enter the initial diplacement of the oscillator in
metres.  ');
    v_0=input('Please enter the initial velocity of the oscillator. ');
    T=input('How long would you like the oscillator to run in seconds?  ');
    h=input('What step size do you wish to run the simulator at?  ');

    fprintf('The program will now plot phase diagrams and energy plots
for \n');
    fprintf('for the Euler, Improved Euler, Euler-Cromer and Verlet
methods');

%--------------------------------------------------------------------------
%Calculating and plotting displacements and velocities
%--------------------------------------------------------------------------

    [ x, v ,t ] = Analytical(x_0,v_0,k,m,d,h,T);%Analytical solution
    [ eux, euv, eut ] = Euler(x_0,v_0,k,m,d,h,T); %Solution from Euler
```

```
    [ iex, iev, iet ] = ImprovedEuler(x_0,v_0,k,m,d,h,T); %Solution from
IEuler
    [ vex, vev, vet ] = Verlet(x_0,v_0,k,m,d,h,T); %Solution from Verlet
    [ ecx, ecv, ect ] = EulerCromer(x_0,v_0,k,m,d,h,T); %Solution from E-
Cromer

    figure; %Creates new figure to be displayed to user.
    subplot(3,2,1) %Splits figure into 3 by 2 subplots.
    plot(real(x),real(v)); %Plots phase diagram using real parts of
analytical solution.
    title('Analytical Solution');
    xlabel('Displacement(x) /m'); ylabel('Velocity (v) /ms^{-1}');

    %For the numerical data...
    subplot(3,2,3)
    plot(eux,euv);
    title('Euler Method');
    xlabel('Displacement(x) /m'); ylabel('Velocity (v) /ms^{-1}');

    subplot(3,2,4)
    plot(iex,iev);
    title('Improved Euler Method');
    xlabel('Displacement(x) /m'); ylabel('Velocity (v) /ms^{-1}');

    subplot(3,2,5)
    plot(ecx,ecv);
    title('Euler-Cromer Method');
    xlabel('Displacement(x) /m'); ylabel('Velocity (v) /ms^{-1}');

    subplot(3,2,6)
    plot(vex,vev);
    title('Verlet Method');
    xlabel('Displacement(x) /m'); ylabel('Velocity (v) /ms^{-1}');

    fprintf('Saving data to files...'); %Inform user data being saved.

%------------------------------------------------------------------------
%Writing files
%------------------------------------------------------------------------

    %Writes a new csv file to export the analytical data to.
    file_IDa=fopen('Analytical.csv', 'w');
    if file_IDa==-1 %Checks that file created correctly
       error('Could not create file!');
    end

    fprintf(file_IDa,'Time,Displacment,Velocity\n');%Writes headers

    %Writes analytical data double precision format.
    for i=1:length(t);
        fprintf(file_IDa,'%f,%f,%f\n',t(i),x(i),v(i));
    end
    fclose(file_IDa);

    %For the rest of the data...
    file_IDe=fopen('Euler.csv', 'w');
    if file_IDe==-1
       error('Could not create file!');
    end
    fprintf(file_IDe,'Time,Displacment,Velocity\n');
    for i=1:length(eut);
        fprintf(file_IDe,'%f,%f,%f\n',eut(i),eux(i),euv(i));
    end
    fclose(file_IDe);

    file_IDi=fopen('ImprovedEuler.csv', 'w');
    if file_IDi==-1
       error('Could not create file!');
```

```
    end
    fprintf(file_IDi,'Time,Displacment,Velocity\n');
    for i=1:length(iet);
        fprintf(file_IDi,'%f,%f,%f\n',iet(i),iex(i),iev(i));
    end
    fclose(file_IDi);

    file_IDc=fopen('EulerCromer.csv', 'w');
    if file_IDc==-1
       error('Could not create file!');
    end
    fprintf(file_IDc,'Time,Displacment,Velocity\n');
    for i=1:length(ect);
        fprintf(file_IDc,'%f,%f,%f\n',ect(i),eux(i),ecv(i));
    end
    fclose(file_IDc);

    file_IDv=fopen('Verlet.csv', 'w');
    if file_IDv==-1
       error('Could not create file!');
    end
    fprintf(file_IDv,'Time,Displacment,Velocity\n');
    for i=1:length(vet);
        fprintf(file_IDv,'%f,%f,%f\n',vet(i),vex(i),vev(i));
    end
    fclose(file_IDv);
    fprintf('Save successful!\n');

      file_IDv=fopen('Verlet.csv', 'w');
    if file_IDv==-1
       error('Could not create file!');
    end
    fprintf(file_IDv,'Time,Displacment,Velocity\n');
    fprintf(file_IDv,'%f,%f,%f\n',m,k,c);
    fclose(file_IDv);
    fprintf('Save successful!\n');

%-------------------------------------------------------------------------
%Reading files
%-------------------------------------------------------------------------

elseif skip==y %if user wants to read data from previous session
    fprintf=('Reading data...\n');

    %Opens analytical file
    file_id=fopen('Analytical.csv');
    if file_id==-1 %Checks that file opened correctly
       error('Could not open file!');
    end
    %Reads file missing the header
    data=textscan(file_id,'%f %f %f','Delimiter', ',', 'HeaderLines', 1);
    fclose(file_id); %close file
    t=data{1}; x=data{2}; v=data{3}; %Splits up the data

    %Repeat for rest of the files...
    file_id=fopen('Euler.csv');
    if file_id==-1
       error('Could not open file!');
    end
    data=textscan(file_id,'%f %f %f','Delimiter', ',', 'HeaderLines', 1);
    fclose(file_id);
    eut=data{1}; eux=data{2}; euv=data{3};

    file_id=fopen('ImprovedEuler.csv');
    if file_id==-1
       error('Could not open file!');
    end
    data=textscan(file_id,'%f %f %f','Delimiter', ',', 'HeaderLines', 1);
```

```
    fclose(file_id);
    iet=data{1}; iex=data{2}; iev=data{3};

    file_id=fopen('EulerCromer.csv');
    if file_id==-1
       error('Could not open file!');
    end
    data=textscan(file_id,'%f %f %f','Delimiter', ',', 'HeaderLines', 1);
    fclose(file_id);
    ect=data{1}; ecx=data{2}; ecv=data{3};

    file_id=fopen('Verlet.csv');
    if file_id==-1
       error('Could not open file!');
    end
    data=textscan(file_id,'%f %f %f','Delimiter', ',', 'HeaderLines', 1);
    fclose(file_id);
    vet=data{1}; vex=data{2}; vev=data{3};

    file_id=fopen('Coefficients.csv');
    if file_id==-1
       error('Could not open file!');
    end
    data=textscan(file_id,'%f %f %f','Delimiter', ',', 'HeaderLines', 1);
    fclose(file_id);
    m=data{1}; k=data{2}; c=data{3};

end

%------------------------------------------------------------------------
%Plotting energies
%------------------------------------------------------------------------
figure; %New graph
hold on %Plot lines on the same graphs
plot(t,Energy(x, v, k, m),'k') %Plot analytical energy against time
plot(eut,Energy(eux, euv, k, m),'r-.')
plot(iet,Energy(iex, iev, k, m),'b:')
plot(vet,Energy(vex, vev, k, m),'g--')
plot(ect,Energy(ecx, ecv, k, m),'m')
%legend and axes labels:
legend('Analytical Energy','Euler method Energy','Improved Euler method
Energy','Verlet method Energy','Euler-Cromer method Energy');
xlabel('Time /s');
ylabel('Energy /J');

fprintf('Thank you for using SHMsimulator. Any data saved can be found in
the\n')
fprintf('same folder as this script\n Exiting...\n')
```

Project2ReportScript

```
%------------------------------------------------------------------------
% PROJECT2REPORTSCRIPT The script used to generate the graphs and data used
%in Joseph Kellett's project 2 report. Includes resonance and force
%oscillations that is not in SHMsimulator.
% ------------------------------------------------------------------------
% Joseph Kellett
% University of Manchester
% March 2014
% ------------------------------------------------------------------------
close all;
clear all;
format long;

x_0=0;v_0=-1; %Initial conditions.
k=0.6;m=5.21; d=0.0;%Parameters of oscillator
h=0.1; %Step size
```

```
T=100; %Time frame

%-------------------------------------------------------------------------
%Calculating displacement and velocities with all methods for a fixed step
%size, with the same conditions.
%-------------------------------------------------------------------------
[ x, v ,t ] = Analytical(x_0,v_0,k,m,d,h,T); %Analytical solution
[ eux, euv, eut ] = Euler(x_0,v_0,k,m,d,h,T); %Solution from Euler
[ iex, iev, iet ] = ImprovedEuler(x_0,v_0,k,m,d,h,T); %Solution from
%improved Euler
[ vex, vev, vet ] = Verlet(x_0,v_0,k,m,d,h,T); %Solution from Verlet
[ ecx, ecv, ect ] = EulerCromer(x_0,v_0,k,m,d,h,T); %Solution from the Euler
%Cromer method

%-------------------------------------------------------------------------
%Plot phase space diagrams of x and v
%-------------------------------------------------------------------------
figure; %New figure
subplot(3,2,1) %Splits figure into 3 by 2 subplots.
plot(real(x),real(v)); %Plots phase diagram using real parts of analytical
solution.
%Create titles and axes labels:
title('Analytical Solution');
xlabel('Displacement(x) /m'); ylabel('Velocity (v) /ms^{-1}');

subplot(3,2,3)
plot(eux,euv);
title('Euler Method');
xlabel('Displacement(x) /m'); ylabel('Velocity (v) /ms^{-1}');

subplot(3,2,4)
plot(iex,iev);
title('Improved Euler Method');
xlabel('Displacement(x) /m'); ylabel('Velocity (v) /ms^{-1}');

subplot(3,2,5)
plot(ecx,ecv);
title('Euler-Cromer Method');
xlabel('Displacement(x) /m'); ylabel('Velocity (v) /ms^{-1}');

subplot(3,2,6)
plot(vex,vev);
title('Verlet Method');
xlabel('Displacement(x) /m'); ylabel('Velocity (v) /ms^{-1}');

%-------------------------------------------------------------------------
%Energy of different methods
%-------------------------------------------------------------------------
%Same initial conditions
figure;
hold on %So lines are plotted on the same figure
plot(t,Energy(x, v, k, m),'k') %Energy predicted by analytical solution
against t
plot(eut,Energy(eux, euv, k, m),'r-.')
plot(iet,Energy(iex, iev, k, m),'b:')
plot(vet,Energy(vex, vev, k, m),'g--')
plot(ect,Energy(ecx, ecv, k, m),'m')
legend('Analytical Energy','Euler method Energy','Improved Euler method
Energy','Verlet method Energy','Euler-Cromer method Energy');
xlabel('Time /s');
ylabel('Energy /J');

figure;
hold on
plot(t,Energy(x, v, k, m),'k')
plot(vet,Energy(vex, vev, k, m),'g--')
legend('Analytical Energy','Verlet method Energy')
xlabel('Time /s');
```

```
ylabel('Energy /J');

%-------------------------------------------------------------------------
%For a defined error of 0.1% error in energy after 100 seconds run time,
find step size.
%-------------------------------------------------------------------------
h=input('Enter a value for step size h:  ');
x_0=0;v_0=-1; %Initial conditions.
T=100; %Time frame

[ x, v , ~ ] = Analytical(x_0,v_0,k,m,d,h,T);
[ eux, euv, eut ] = Euler(x_0,v_0,k,m,d,h,T); %Solution from Euler
[ iex, iev, iet ] = ImprovedEuler(x_0,v_0,k,m,d,h,T); %Solution from
[ vex, vev, ~ ] = Verlet(x_0,v_0,k,m,d,h,T); %Solution from Verlet
[ ecx, ecv, ~ ] = EulerCromer(x_0,v_0,k,m,d,h,T);
eulererror= max(abs((Energy(eux, euv, k, m) -Energy(x, v, k, m))./Energy(x,
v, k, m)));
improvedeulererror= max(abs((Energy(iex, iev, k, m) -Energy(x, v, k,
m))./Energy(x, v, k, m)));
verleterror= max(abs((Energy(vex, vev, k, m) -Energy(x, v, k, m))./Energy(x,
v, k, m)));
cromererror= max(abs((Energy(ecx, ecv, k, m) -Energy(x, v, k, m))./Energy(x,
v, k, m)));

%-------------------------------------------------------------------------
%Graph of light, critical and heavy damping.
%-------------------------------------------------------------------------
d=0.5*sqrt(k*m); %Light damping, half of critical value
d1=sqrt(k*m); %At critical value
d2=2*sqrt(k*m); %Heavy damping; twice critical value
[ lx, ~, lt ] = Verlet(x_0,v_0,k,m,d,h,T); %Verlet prediction for x, light
damping
[ cx, ~, ct ] = Verlet(x_0,v_0,k,m,d1,h,T);
[ hx, ~, ht ] = Verlet(x_0,v_0,k,m,d2,h,T);
figure
subplot(2,1,1);
hold on
plot(lt, lx);
plot(ct, cx, 'r--');
plot(ht, hx, 'k-.');
legend('Light damping', 'Critical damping', 'Heavy damping')

%To show analytical is the same.
subplot(2,1,2)
hold on
[ lx, lv, lt ] = Analytical(x_0,v_0,k,m,d,h,T);
[ cx, cv, ct ] = Analytical(x_0,v_0,k,m,d1,h,T);
[ hx, hv, ht ] = Analytical(x_0,v_0,k,m,d2,h,T);
plot(lt, real(lx));
plot(ct, real(cx), 'r--');
plot(ht, real(hx), 'k-.');
legend('Light damping', 'Critical damping', 'Heavy damping.')

%-------------------------------------------------------------------------
%Effect of 10s step force on undamped oscillator
%-------------------------------------------------------------------------
x_0=0;v_0=-1;d=0;T=100;
t=0:h:T;
F(length(t))=0;  %Set all force to 0
F(floor(length(t)/2):floor(length(t)/2)+10/h)= 5;  %Force of 5N for 10s.
[ fx, v ]=ForcedVerlet( x_0, v_0, k, m, d, h, T, F );
figure
[haxes,~,~] = plotyy(t,fx,t,v); %Graph with 2 y axis, v and x.
xlabel('Time /s');
ylabel(haxes(1),'Displacement (x) /s') % label left y-axis
ylabel(haxes(2),'Velocity (v) / ms^{-1}') % label right y-axis

%-------------------------------------------------------------------------
```

```
%Effect of force at resonant frequency on an undamped oscillator
%-------------------------------------------------------------------------
x_0=0;v_0=-1;k=0;m=5.21;d=0;T=100; w=0.34; p=0;
t=0:h:T;
F=6*sin(w*t+p);

[ fx, v ]=ForcedVerlet( x_0, v_0, k, m, d, h, T, F );
figure
[haxes,hline1,hline2] = plotyy(t,fx,t,v);
xlabel('Time /s');
ylabel(haxes(1),'Displacement (x) /s')
ylabel(haxes(2),'Velocity (v) / ms^{-1}')

%-------------------------------------------------------------------------
%Damped forced harmonic motion with varying damping terms
%-------------------------------------------------------------------------
figure
hold on
h=0.01; x_0=0; v_0=0; d=0.1*(k*m)^0.5; T=500;
t=0:h:T;
F=1*sin(w*t+p);
[ fx, ~ ]=ForcedVerlet( x_0, v_0, k, m, d, h, T, F );
plot(t,fx);
d=0.05*(k*m)^0.5;
[ fx, ~ ]=ForcedVerlet( x_0, v_0, k, m, d, h, T, F );
plot(t,fx,'r');
d=0.01*(k*m)^0.5;
[ fx, ~ ]=ForcedVerlet( x_0, v_0, k, m, d, h, T, F );
plot(t,fx,'k--');
xlabel('Time /s')
ylabel('Displacement (x)/m')
legend('0.1*(km)^{0.5}','0.05*(km)^{0.5}','0.01*(km)^{0.5}');

%-------------------------------------------------------------------------
%Plotting a resonance curve for an oscillator damped at 0.1*the critical
%value.
%-------------------------------------------------------------------------
h=0.01; x_0=0; v_0=0; k=0.6; m=5.21; d=0.1*(k*m)^0.5; T=1000; p=0;
t=0:h:T;
j=1;
maxx(1/0.01)=0; freq(1/0.01)=0; %Predefine for increased speed
for w=0.01:0.001:1 %Varying w by 0.01.
    freq(j)=w;
    F=3*sin(w*t+p);
    [ fx, ~ ]=ForcedVerlet( x_0, v_0, k, m, d, h, T, F );
    maxx(j)=max(fx);
    j=j+1;
end
figure
plot(freq,maxx);
xlabel('Angular frequency /s^{-1}');
ylabel('Maximum displacement /m');

%-------------------------------------------------------------------------
%Plotting graph of max amplitude against damping coefficient.
%-------------------------------------------------------------------------
h=0.1; %Larger step size to so it doesn't take too long
x_0=0; v_0=0; T=1000; w=0.34; p=0;
t=0:h:T;
i=1;
maxx(1/0.01)=0;  maxmaxx(floor((k*m)^0.5/0.01))=0;%Predefine for increased
speed
freq=0.01:0.01:1;
damp=(0.0001:0.01:(k*m)^0.5)/(k*m)^0.5;

for d=0.0001:0.01:(k*m)^0.5 %Varying d by 0.01. A value of 0 would result in
    %inaccurate results
    j=1;
```

```
    %Find the resonant amplitude for this value of d.
    for w=0.01:0.001:1 %Varying w by 0.001. Start at 0.01 so it has time to
        %make an oscillation before simulation time has run out
        freq(j)=w;
        F=3*sin(w*t+p);
        [ fx, ~ ]=ForcedVerlet( x_0, v_0, k, m, d, h, T, F );
        maxx(j)=max(fx(end-200:end));%Only find the max of the last 200
values
        j=j+1;
    end
    maxmaxx(i)=max(maxx(end-30)); %Only find the max of the last 30 values
    i=i+1;
end
figure
plot(damp,maxmaxx);
%LaTex interpreter to give a square root symbol.
ylabel('Maximum amplitude /m','Interpreter','LaTex')
xlabel('Damping coefficient / $\sqrt{(k \times m)}$','Interpreter','LaTex');
```

Euler

```
function [ x, v, t ] = Euler( x_0, v_0, k, m, d, h, T )
%EULER Uses the Euler method to find displacement (x) and
%velocity (v) for given values of t.
%   x_0 is defined as the initial condition, v_0 the initial 1D velocity, h
%   the step size used in the Euler method and T is the length of time the
%   simulation is run for, d is the damping constant.

n = T/h; %Number of iterations required

v=zeros(n,1); %Define now to prevent resizing.
x=zeros(n,1);
t=zeros(n,1);
a=zeros(n,1);

v(1)=v_0; %Set initial conditions
x(1)=x_0;
t(1)=0;
for j=1:n
    a(j)=-k*x(j)/m-d*v(j)/m;
    v(j+1)=v(j)+h*a(j);
    x(j+1)=x(j)+h*v(j);
    t(j+1)=h*j;
end
```

Improved Euler

```
function [ x, v, t ] = ImprovedEuler( x_0, v_0, k, m, d, h, T )
%IMPROVEDEULER Uses the Improved Euler method to find displacement (x) and
%velocity (v) for given values of t.
%   x_0 is defined as the initial condition, v_0 the initial 1D velocity, h
%   the step size used in the Euler method and T is the length of time the
%   simulation is run for, d is the damping constant.

n = T/h; %Number of interactions

v=zeros(n,1); %Define now to prevent resizing.
x=zeros(n,1);
t=zeros(n,1);
a=zeros(n,1);

v(1)=v_0; %Initial values
x(1)=x_0;
t(1)=0;
```

```
for j=1:n
    a(j)=-k*x(j)/m-d*v(j)/m;
    v(j+1)=v(j)+h*a(j);
    x(j+1)=x(j)+h*v(j)+h*h*a(j)/2;
    t(j+1)=h*j;
end
end
```

Euler-Cromer

```
function [ x, v, t ] = EulerCromer( x_0, v_0, k, m, d, h, T )
%EULERCROMER Uses the Euler-Cromer method to find x(t) for a SHO with spring
constant k
%and mass m.
%   x_0 is defined as the initial condition, v_0 the initial 1D velocity, h
%   the step size used in the Euler method and T is the length of time the
%   simulation is run for, d is the damping constant.

n = T/h; %Number of iterations

v=zeros(n,1); %Define now to prevent resizing.
x=zeros(n,1);
t=zeros(n,1);
a=zeros(n,1);

v(1)=v_0; %Initial conditions
x(1)=x_0;
t(1)=0;
for j=1:n
    a(j)=-k*x(j)/m-d*v(j)/m;
    v(j+1)=v(j)+h*a(j);
    x(j+1)=x(j)+h*v(j+1);
    t(j+1)=h*j;
end
```

Verlet

```
function [ x, v, t ] = Verlet( x_0, v_0, k, m, d, h, T )
%VERLET Numerically estimates the position and speed of a harmonic
%oscillator using the Verlet method using initial condtions, simulation
%time and step size.

n = T/h;

v=zeros(n,1); %Define now to prevent resizing.
x=zeros(n,1);
t=zeros(n,1);

v(1)=v_0; %Set inital conditions
x(1)=x_0;
t(1)=0;

%Use improved Euler to find first terms required for Verlet
[x_1, ~, ~]=ImprovedEuler(x_0, v_0, k, m, d, h, h );
x(2)=x_1(2);
t(2)=h;

D=2*m+d*h;
B=(d*h-2*m)/D;
A=2*(2*m-k*h*h)/D;

for j=2:n+1
    x(j+1)=A*x(j)+B*x(j-1);
    v(j)=(x(j+1)-x(j-1))/(2*h);
    t(j+1)=h*j;
```

```
end

x=x(1:end-1);%Shorten outputs to n+1 values
t=t(1:end-1);
end
```

Forced Improved Euler

```
function [ x, v ] = ForcedImprovedEuler( x_0, v_0, k, m, d, h, T, F)
%FORCEDIMPROVEDEULER Uses IE method including a term for external force
%   Similar inputs to ImprovedEuler. F is a vector of length n. This
%   function is used to start the ForcedVerlet function

n = T/h;

v=zeros(n,1);
x=zeros(n,1);
a=zeros(n,1);
v(1)=v_0;
x(1)=x_0;

for j=1:n
    a(j)=-k*x(j)/m-d*v(j)/m+F(j)/m;
    v(j+1)=v(j)+h*a(j);
    x(j+1)=x(j)+h*v(j)+h*h*a(j)/2;
end
end
```

Forced Improved Verlet

```
function [ x, v ] = ForcedVerlet( x_0, v_0, k, m, d, h, T, F )
%FORCEDVERLET. Uses an adapated version of the Verlet method that includes
%a force term. The force input (F) must be of a vector of length n.

n = T/h;

v=zeros(n,1); %Define now to prevent resizing.
x=zeros(n,1);
v(1)=v_0;
x(1)=x_0;

%Start the function with FIE.
[x_1, ~]=ForcedImprovedEuler(x_0, v_0, k, m, d, h, h, F );
x(2)=x_1(2);

D=2*m+d*h;
B=(d*h-2*m)/D;
A=2*(2*m-k*h*h)/D;

for j=2:n+1
    x(j+1)=A*x(j)+B*x(j-1)+2*(h)^2/D *F(j);
    v(j)=(x(j+1)-x(j-1))/(2*h);
end
x=x(1:end-1);%Shorten vector so it is of length n+1
end
```

Analytical

```
function [ x, v, t ] = Analytical( x_0, v_0, k, m, d, h, T  )
%ANALYTICAL Analytically solves a un-forced damped harmonic oscillator
%   Uses the exponential trial solution of the damped ODE to solve the
%   oscillator in the unforced case.
n=T/h;
```

```
%Predefine and input the first values as described by the initial condition
v=zeros(n,1); v(1)=v_0;
x=zeros(n,1); x(1)=x_0;
t=zeros(n,1);
a=(-d+sqrt(d*d-4*m*k))/(2*m); %This is generally complex

for j=1:n
    t(j+1)=j*h;
    x(j+1)=((v_0/a)+x_0)*exp(a*t(j+1));
    v(j+1)=v_0*exp(a*t(j+1));
end
end
```