MATLAB in Physics

This series of lectures describes the MATLAB scientific computing environment and how it can be used by physics students.

In this lecture we will cover:

- obtaining MATLAB
- Simple MATLAB operations
- Visualisation using MATLAB
- Data analysis in MATLAB

Contents

Vectors and matrices in MATLAB

The name MATLAB comes from 'MATrix LABoratory' and when the software was first created the main purpose was to perform numerical matrix computations. The software has gained a range of new functionality over the course of the next 25 years but the manipulation of matrices is still at the heart of the system.

There are a number of ways to create vectors and matrices in MATLAB:

```
% Using horizontal concatenation:
rowVector = [1 2 3 4 5]

% Using vertical concatenation:
colVector = [1; 2; 3; 4; 5]

% Using the linspace or logspace function:
linRowVect = linspace(0,1,5)

% Using the colon operator
rowVect2 = 0:5:20

% Conjugate-transposing an existing vector (use .' for transpose without
% taking complex conjugate)
colVect2 = rowVect2'

% Create a matrix similarly:
a = [1 2; 3 4]

% Create a complex matrix
c = [ 1 + 1i  2 + 2i ; 3 + 3i 4 + 4i ]
```

```
rowVector =

     1     2     3     4     5


colVector =

     1
     2
     3
     4
     5


linRowVect =

         0    0.2500    0.5000    0.7500    1.0000


rowVect2 =

     0     5    10    15    20
```

```
colVect2 =

     0
     5
    10
    15
    20


a =

     1     2
     3     4


c =

   1.0000 + 1.0000i   2.0000 + 2.0000i
   3.0000 + 3.0000i   4.0000 + 4.0000i
```

Mathematics in MATLAB

The main purpose of MATLAB is to provide an environment in which numeric operations can be undertaken on matricies and vectors. There are several hundred functions within MATLAB itself, and many more in specific toolboxes.

Some examples are :

```
% Add matricies together
c + c

% Multiply matrices together (using matrix multiply)
a * c

% Multiply matrices together (element-wise) - note the use of the '.'. This
% is a common MATLAB feature where an element-wise operation uses '.' and
% the matrix operation does not
a .* c

% Find the sin of the values in a matrix
sin(a)
```

```
ans =
   2.0000 + 2.0000i   4.0000 + 4.0000i
   6.0000 + 6.0000i   8.0000 + 8.0000i


ans =

   7.0000 + 7.0000i  10.0000 +10.0000i
  15.0000 +15.0000i  22.0000 +22.0000i


ans =

   1.0000 + 1.0000i   4.0000 + 4.0000i
   9.0000 + 9.0000i  16.0000 +16.0000i


ans =

    0.8415    0.9093
    0.1411   -0.7568
```

Using help and documentation in MATLAB

MATLAB comes with extensive help and documentation on how to use all its functions and tutorials on best practices for programming etc. A trivial example is

```
help sin

% For more extensive documentation use the doc command, which brings up an
% interactive browser to view all the documentation. This includes more
% examples and usage, a search facility, and all the tutorials on all
% installed products
doc plot
```

```
 SIN    Sine of argument in radians.
    SIN(X) is the sine of the elements of X.

    See also ASIN, SIND.

    Overloaded methods:
       codistributed/sin
       sym/sin

    Reference page in Help browser
       doc sin

 Overloaded functions or methods (ones with the same name in other directories)
   <a href="matlab:doc simulink/plot">doc simulink/plot</a>
   <a href="matlab:doc curvefit/plot">doc curvefit/plot</a>
   <a href="matlab:doc finance/plot">doc finance/plot</a>
   <a href="matlab:doc fixedpoint/plot">doc fixedpoint/plot</a>
   <a href="matlab:doc ident/plot">doc ident/plot</a>
   <a href="matlab:doc mpc/plot">doc mpc/plot</a>
   <a href="matlab:doc rf/plot">doc rf/plot</a>
   <a href="matlab:doc wavelet/plot">doc wavelet/plot</a>
```

Example: visualising sound signals

MATLAB comes with a range of example data sets that can be used to investigate the functionality of the tool.

```
% The last few lines of 'help audiovideo'
% Example audio data (MAT files).
%     chirp         - Frequency sweeps          (1.6 sec, 8192 Hz)
%     gong          - Gong                      (5.1 sec, 8192 Hz)
%     handel        - Hallelujah chorus         (8.9 sec, 8192 Hz)
%     laughter      - Laughter from a crowd     (6.4 sec, 8192 Hz)
%     splat         - Chirp followed by a splat (1.2 sec, 8192 Hz)
%     train         - Train whistle             (1.5 sec, 8192 Hz)

% Load in a sound file saved as a .mat file
load handel;
```

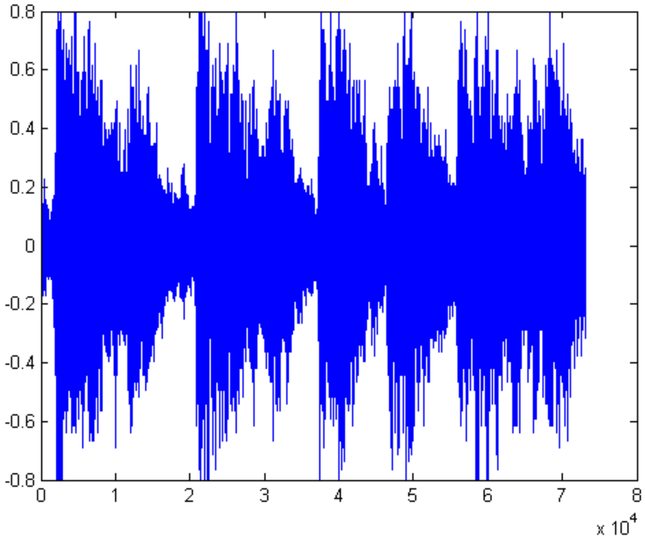
Play the sound

```
sound(y)
```

Visualise the waveform

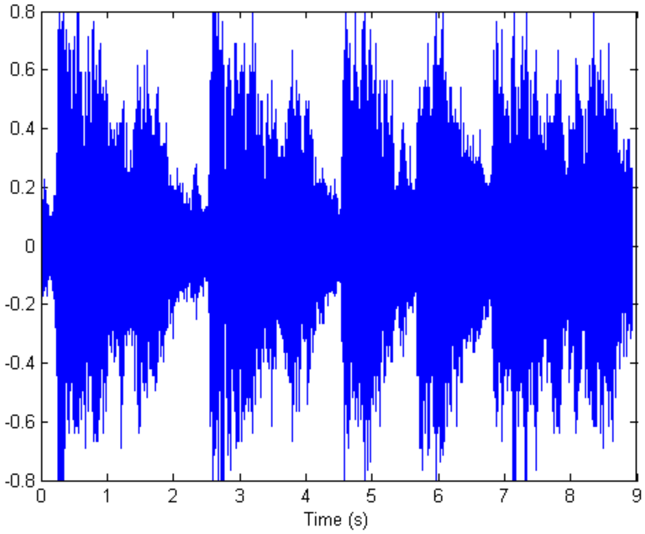
Look at the waveform versus sample number

```
plot(y)
```



Plot the data and label the axes

```
% The x-axis is the time, create a time vector
ts = 1/Fs;  % Sample time is inverse of sample frequency
t = ts*(1:numel(y))' - ts;
plot(t,y)
xlabel('Time (s)')
```

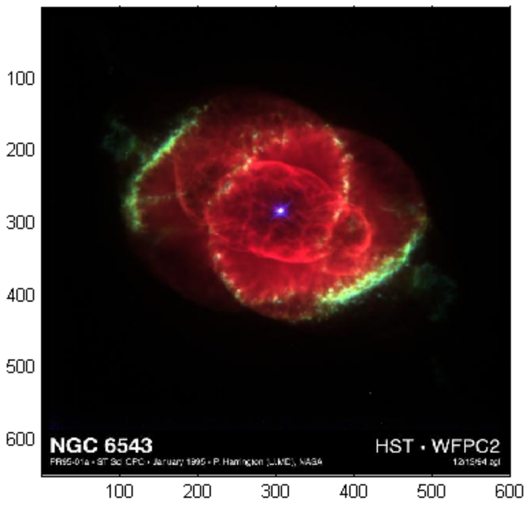


Use array manipulation to reverse the signal

```
yRev = flipud(y);
sound(yRev)
```

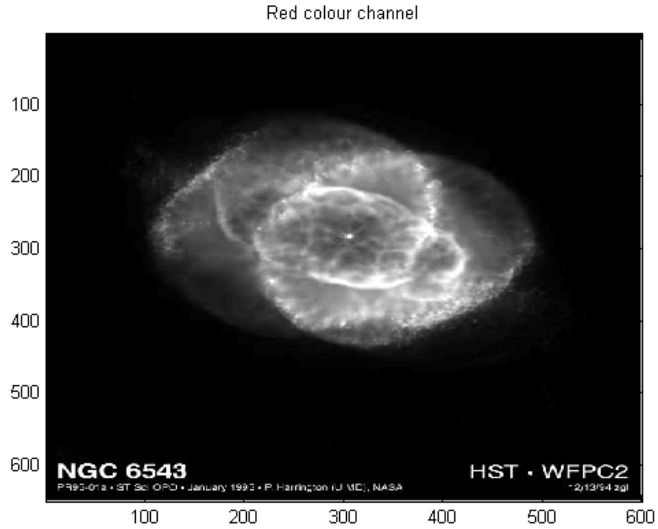
Example: manipulating images using matrix indexing

```
im = imread('ngc6543a.jpg');
image(im); axis square
```



Extract the red-green-blue channels

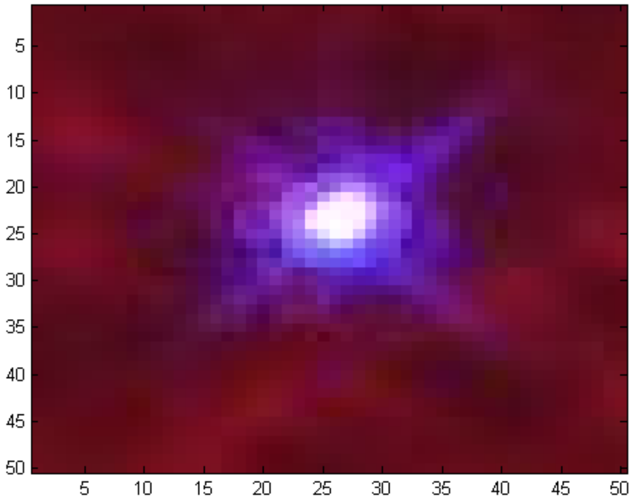
```
imRed = im(:,:,1);
imGreen = im(:,:,2);
imBlue = im(:,:,3);

imagesc(imRed);
title('Red colour channel')
colormap(gray);
```



Red colour channel

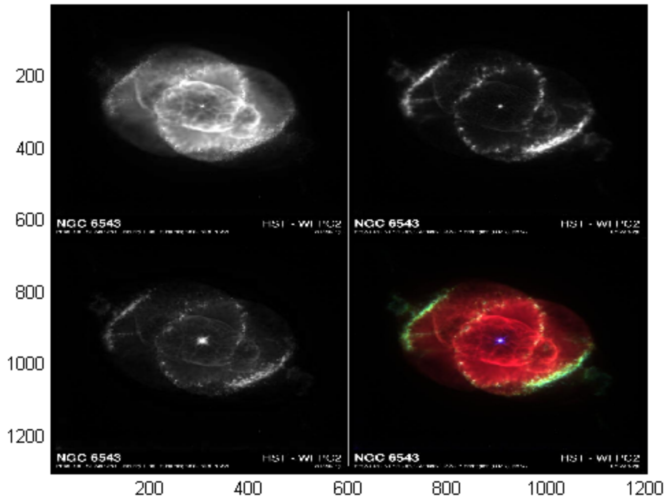
Extract the centre of the image

```
offsetX = 280;
offsetY = 260;
centreStar = im(offsetY + (1:50),offsetX + (1:50),:);
imagesc(centreStar);
```



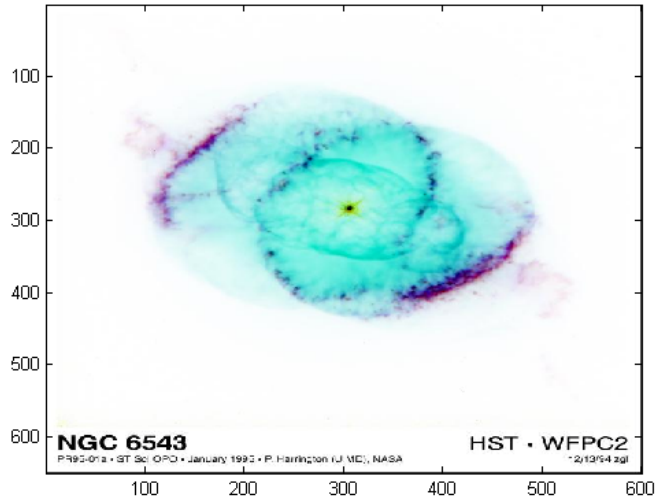
Combine images into a composite

```
compositeImage = imagesc([imRed(:,:,[1 1 1]), imGreen(:,:,[1 1 1]);...
    imBlue(:,:,[1 1 1]), im]);
```



Using matrix addition to process images

```
whiteImage = 255*ones(size(im),'uint8');
blackImage = zeros(size(im),'uint8');
inverseImage = whiteImage - im;

% Show the inverse image
imagesc(inverseImage);
```



CASE STUDY: Visualising the behaviour of a pendulum

Let us now look at using the MATLAB techniques that we've covered so far to visualise a simple physical system.

A main use of scientific computing environments such as MATLAB is to visualise the behaviour of physical systems, in order to build up physical intuition and discover general properties of the solutions.

The system that we are examining here is the simple harmonic osciallator. This system is governed by the equation of motion $\ddot{x} = -\omega_0^2 x$, having solution $x(t) = x_0 \cos(\omega_0 t)$

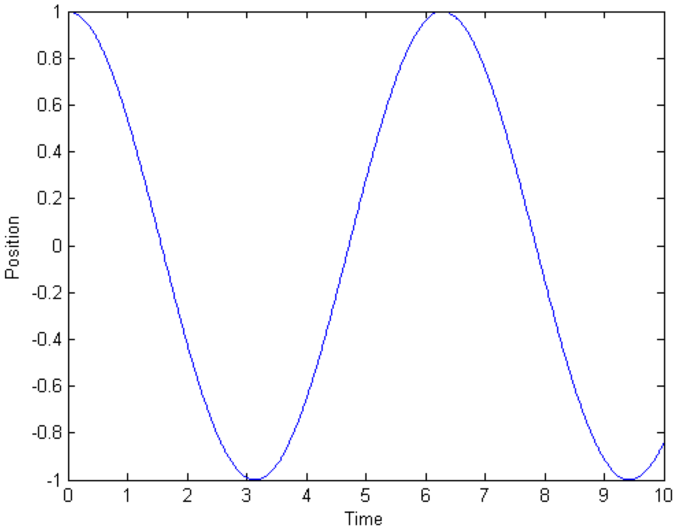
We want to visualise these solutions to discover the physical behaviour

Visualising a system

Define the time range

```
t = linspace(0,10,1000)';

% Calculate the solution function
x = cos(t);

% Plot the solution
plot(t,x)
```

```
xlabel('Time')
ylabel('Position')
```



Other forms of visualisation

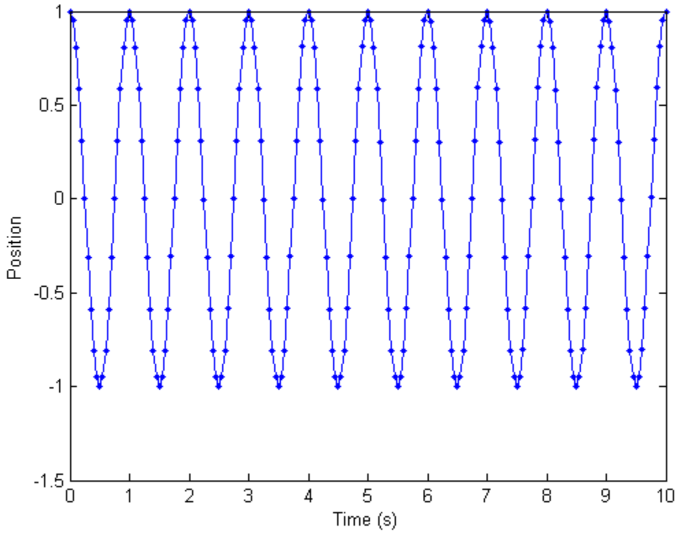
A position versus time plot is the form of visualisation with which you are probably most familiar, however there are other aspects of the solution that can be discovered by using different visualisation techniques.

```
% Perform a simulation of the SHM motion
x0 = 1;  % Starting position
k = (2*pi)^2;  % Spring stiffness
m = 1;  % Mass of object

res = shmSimulation(x0, k, m);
```

Time series plot

This shows the position versus time

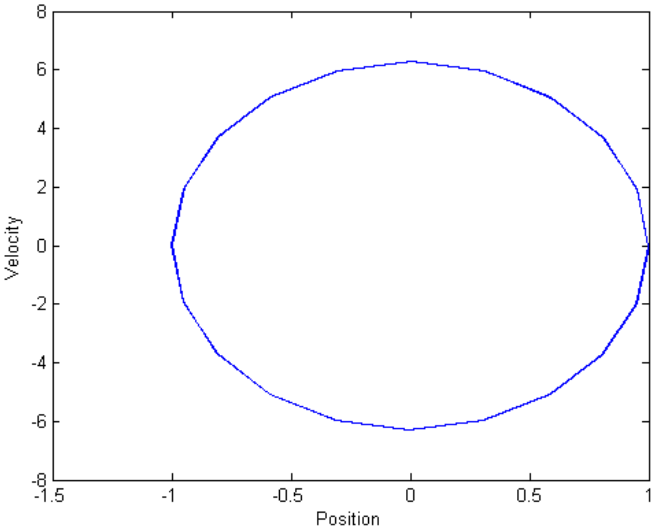
```
plotTimeSeries(res)
```



Phase space plot

This shows the position of the mass versus its velocity. What does this plot tell you about the energy in the system?

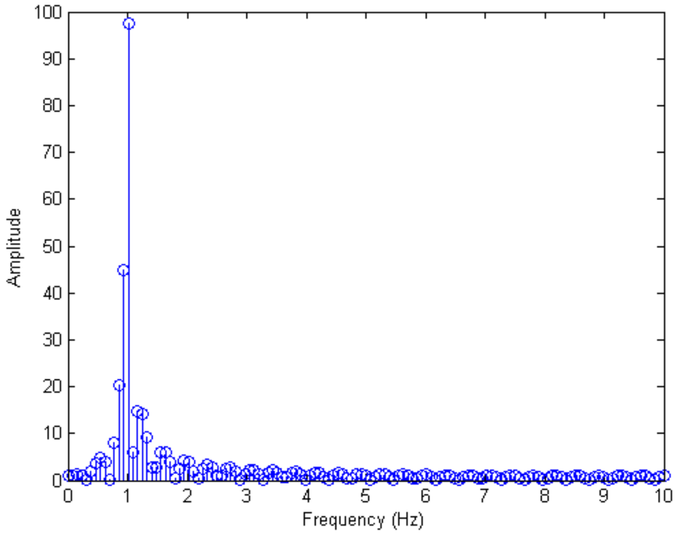
```
plot(res.Position,res.Velocity,'b-')
xlabel('Position')
ylabel('Velocity')
```



Frequency spectrum

Find the frequencies present in the motion

```
plotFrequencySpectrum(1/(res.Time(2) - res.Time(1)),res.Position)
```



Experimental design and real data

At its core, physics is an experimental science. What distinguishes one mathematical model from another as a being 'physical' is determined by which model agrees with the way the universe behaves. Usually more than one model will be required to cover the entire range of parameters, for example Newtonian mechanics being replaced with special relativity as speeds approach the speed of light.

Most physics experiments consist of a number of tunable parameters that can be used to change the behaviour of the system, and a number of output properties that can be measured. A key skill in experimental design lies in determining which parameters and properties are important, and how to measure them.

In the next example we create simulated data that represents the sort of data we would expect to take if we measured the position of an oscillating pendulum at various points in its motion. The input parameters are the intial angle of the pendulum, the mass of the weight at the end of the pendulum and the length of the pendulum. What other parameters have we ignored? (Hint: shape)

The output property we are measuring is the position of the pendulum (or the angle that the pendulum makes with the vertical). What other properties could we measure? How would we measure them?

The experiment that we are simulating consists of a pendulum and a webcam, with the webcam taking a picture at regular intervals of the pendulum position. Experimental noise comes into play due to the uncertainty in the postion measurement and the uncertainty in the timing of the webcam snapshots.

(An automated experiment like such as this is nice to have - in practice you will usually need to take the measurements manually as undergraduates.)

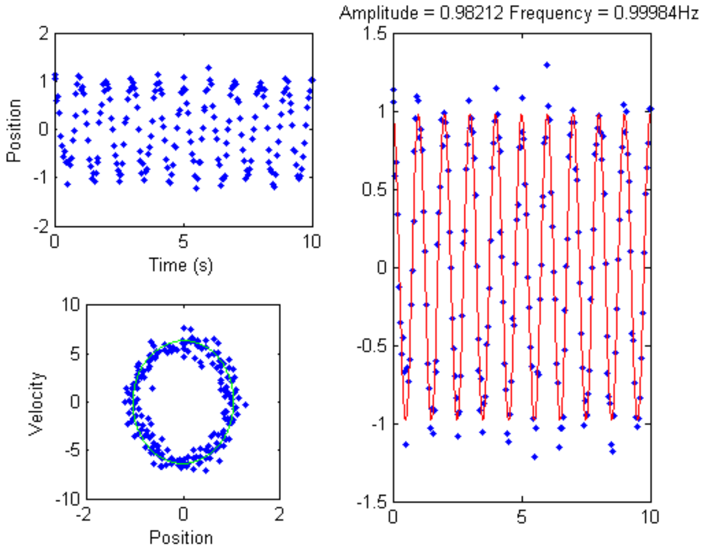
Data import - noisy data

```
% Use shmDataSimulated and/or pendulumDataSimulated to produce some data
x0=1;
springConstant = (2*pi)^2;
noiseFactor = 0.1;
shmDataSimulated(x0,springConstant,noiseFactor);

x0=1;
pendulumLength=9.8/(2*pi)^2;
noiseFactor = 0.1;
pendulumDataSimulated(x0,pendulumLength,noiseFactor);

% Open the data in Excel if desired

% Use importdata to examine the Excel spreadsheet.

% Use the variable editor to input the data.
```

Analyse the data

This plots the data and phase space, then fits to the SHM model.

```
analyseExperimentalData('shmData.xls');
```



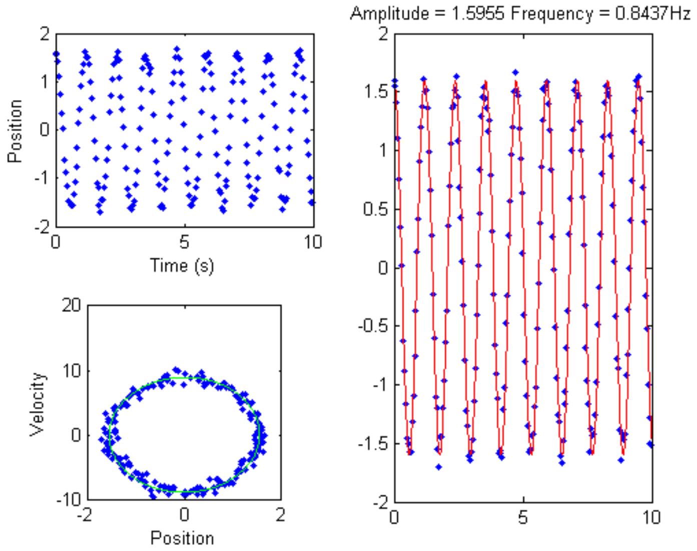
The large-angle pendulum

As the initial angle of the pendulum from the vertical becomes large we could expect some deviations from the behaviour predicted by the simple harmonic oscillator model (why?).

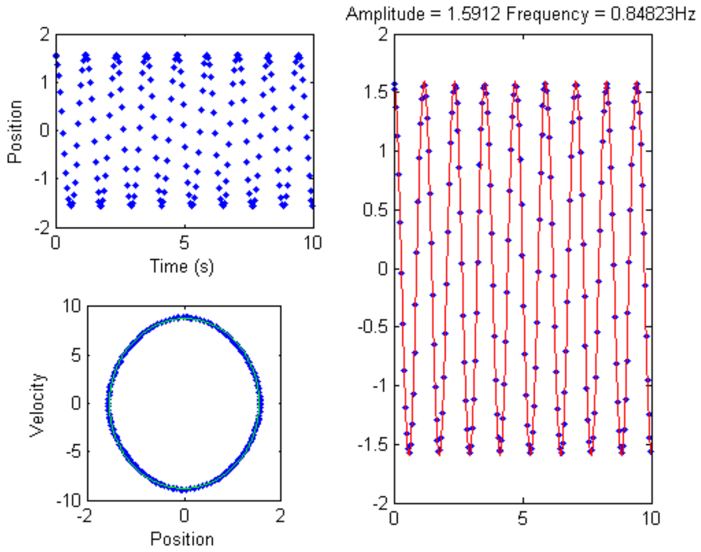
Starting with the pendulum horizontal, how much of this behaviour is observable in the presence of noise?

```
x0 = pi/2; % Radians from vertical
pendulumLength = 9.8/(2*pi)^2;
noiseFactor = 0.05; % 5% noise in measurements
outFileNameNoisy = 'noisyPendulum.xls';
pendulumDataSimulated(x0,pendulumLength,noiseFactor,outFileNameNoisy);

noiseFactor = 0.0001; % 0.01% noise
outFileNameAccurate = 'accuratePendulum.xls';
pendulumDataSimulated(x0,pendulumLength,noiseFactor,outFileNameAccurate);

figure('Name','Noisy experiment');
analyseExperimentalData(outFileNameNoisy);

figure('Name','Accurate experiment');
analyseExperimentalData(outFileNameAccurate);
```

```
%#ok<*NOPTS>
```



Amplitude = 1.5955 Frequency = 0.8437Hz
Position
Time (s)
Velocity
Position
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
5
10
20
10
-10
-2



Amplitude = 1.5912 Frequency = 0.84823Hz
Position
Time (s)
Velocity
Position
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
5
10
-5
-10