[**DOING PHYSICS WITH MATLAB**](http://www.physics.usyd.edu.au/teach_res/mp/mphome.htm)

**DYNAMICS OF OSCILLATING AND CHAOTIC SYSTEMS**

# 

Ian Cooper

School of Physics, University of Sydney

ian.cooper@sydney.edu.au

[**DOWNLOAD DIRECTORY FOR MATLAB SCRIPTS**](http://www.physics.usyd.edu.au/teach_res/mp/mscripts)

**maths\_ft\_01.m**

mscript used to calculate the Fourier transform, the power spectral density and the inverse Fourier transform functions by the direct integration of the Fourier integrals using Simpson’s rule. A wide variety of functions, sound files and data files (eg ecg) can be investigated. All parameters can be changed within the mscript.

**simpson1d.m**

mscript for computing the value of a definite integral using Simpson’s rule. An odd number of grid points must be used to accurately calculate the integral.

**wav\_S1000\_1008.wav**

Data sound file to find the Fourier transform of a beat signal.

**Train.wav**

Data sound file for a train whistle.

**ecg.mat**

Raw data for the recording an ecg.

*Arguably the most broad-based evolution in the world view of science in the twentieth century will be associated with chaotic dynamics.*

S.N. Rasband Chaotic Dynamics of nonlinear Systems*.*

**Introduction**

Equations of motion for a variety of oscillating and chaotic [1D] systems are solved numerically using the **Runga-Kutta Method**. Graphical outputs are used to display the results of the simulations: displacement *x* vs time *t;* velocity *v* vs time *t* and **phase space** plots of displacement *x* vs velocity *v.*

We will consider the dynamics of simple linear oscillating systems then more complicated nonlinear systems that exhibit chaotic behaviour. A system that exhibits chaotic behaviour is not one where the motion is random. When the trajectory of a dynamical system is calculated again and again with the same initial conditions, you will always get the same result. The unpredictability – the chaotic motion – is a result of the fact that even small differences in the initial conditions are amplified when solving the equation of motion to give enormously different results, and so, it becomes impossible to make predictions.

**Equation of Motion**

The [1D] motion of a system is governed by Newton’s Second law which can be expressed as

(1) 

where  is the resultant force acting upon the mass *m* of the system.

The Runga-Kutta Method used to solve the equation of motion is outlined below. Given a set of initial conditions [ displacement *x*(0) and *v*(0) ], the displacement and velocity are computed at successive time step :



(2a) 

(2b) 

where the Runga-Kutta coefficients are given by

(3) 

**Matlab**

The script mec\_chaos\_01.m is used to solve a number an equation of motion for a variety of systems using the Runge-Kutta method. For different models and changing parameters must be done within the script.

INPUT Section

Values for the model parameters, time scale, limits for the plots and the initial conditions. The phase plot can be saved as an animated gif f\_gif = 1 (default value is f\_gif = 0 and the animation is not saved). The name of the file is stored by the variable ag\_name.

% INPUTS [default values] =====================

% mass of system [1]

m = 1;

% spring constant (global variable) [pi^2]

k = pi^2;

% damping constant [1]

b = 0;

% Time domain: nT must be an ODD number [501 0 8]

nT = 501;

tMin = 0;

tMax = 8;

t = linspace(tMin,tMax,nT);

h = t(2) - t(1);

% Initialise arrays and set initial conditions [10 0]

x = zeros(nT,1);

v = zeros(nT,1);

x(1) = -10;

v(1) = 0;

% Set plot limits - will need to change for different models

% Time plots [-10 10] [-40 40]

xLimits = [-10 10];

yLimits = [-40 40];

CALCULATION Section

The displacement and velocity are calculated at successive time steps by calling a function to evaluate the resultant force and the Runge-Kutta coefficients.

% Runga-Kutta Solution of differential equation

for c = 1 : nT-1

[k1, k2, k3, k4] = coeff(t(c),h,x(c),v(c));

x(c+1) = x(c) + h\*(v(c) + (k1 + k2 +k3)/6);

v(c+1) = v(c) + (k1 + 2\*k2 + 2\*k3 + k4)/6;

end

From the values of the displacement, the acceleration, kinetic energy, potential energy and total energy are calculated.

% Acceleration a

a = gradient(v)/h;

% Force

F = m.\*a;

% Kinetic Energy K

K = (0.5\*m) .\* v.^2;

% Potential energy U

dU = zeros(nT,1);

dU(1) = 0.5\*F(1)\*(x(2)-x(1));

for c = 2:nT-1

dU(c+1) = dU(c) + 0.5\*(F(c)+F(c+1))\*(x(c+1)-x(c));

end

U = max(K)-dU;

% Total Energy

E = K + U;

The values of the potential energy and the total energy are relative quantities and only changes in values are significant. The scale for the potential energy and total energy are set by setting .

FUNCTION Section

The code for the Runge-Kutta coefficients

function [k1, k2, k3, k4] = coeff(t,h,x,v)

k1 = h\*fn(t,x,v);

k2 = h\*fn(t+h/2, x+h\*v/2, v+k1/2);

k3 = h\*fn(t+h/2, x+h\*v/2+h\*k1/4 ,v+k2/2);

k4 = h\*fn(t+h, x+h\*v+h\*k2/2, v+k3);

end

The code for various functions (comment unwanted code)

function y = fn(t,x,v)

global k m b

% Simple Harmonic Motion

% y = (- k \* x) / m;

% Damped Harmonic Motion

y = (- k \* x - b \* v)/m;

end

**Simulations**

**Simple Harmonic Motion**

The equation of motion for a system of mass *m* executing simple harmonic motion is



where *k* is the spring constant and the period *T,* frequency *f* and angular frequency off the oscillation are  
 



Model parameters and theoretical values:

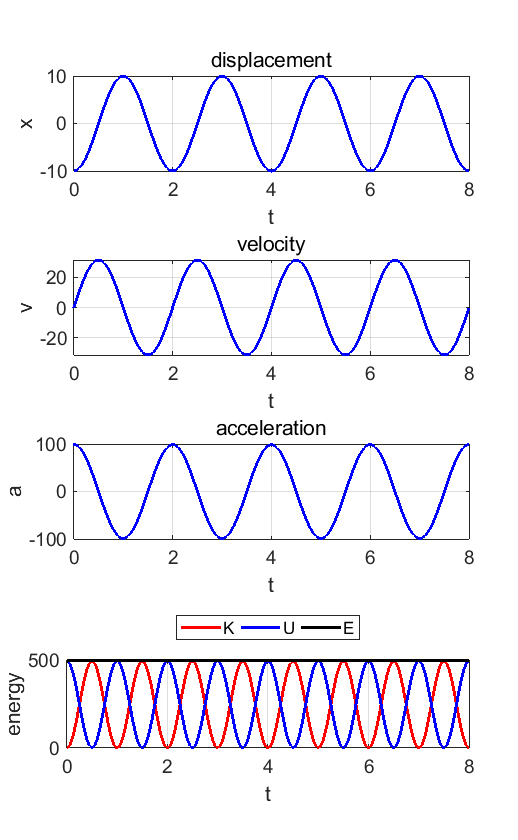
mass  spring constant 

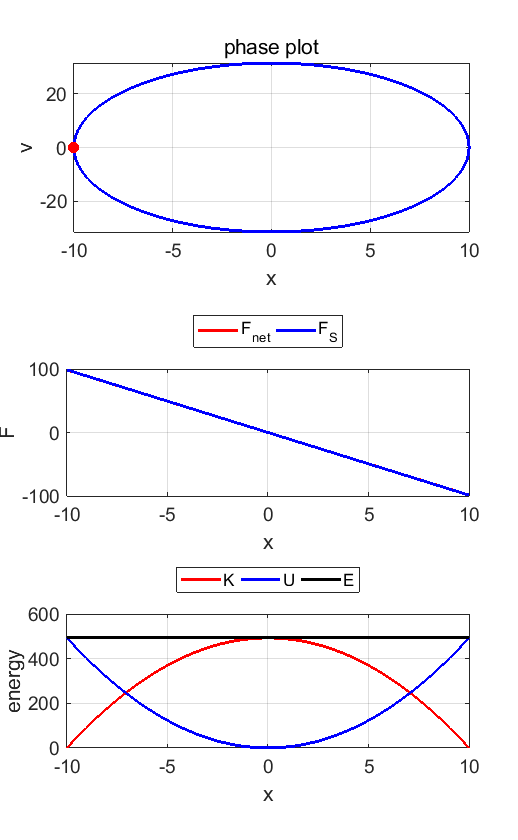
Initial conditions: 

Theoretical values:



There is excellent agreement between all theoretical predictions and the numerical computed values as shown in the graphical output.





In the phase plot, the red dot location corresponds to the initial conditions. The phase plot trajectory evolves in a clockwise sense with time.

View Phase Plot animation

**Damped harmonic motion**

We can model the mass/spring oscillator with viscous damping where energy is dissipated by friction such that the damping (resistive) force is proportional to the velocity.

The equation of motion for the damped mass/spring system is given by



where *b* is the damping constant.

Result for simulation with



are shown below.