Experiment 215: Numerical Modelling in Python

Aim

To demonstrate computer simulation of physical systems by numerically solving differential equations.

Introduction

Many physical systems obey differential equations derivable from the physics of the system. Analytical solutions of these equations are often difficult or impossible to derive except in the simplest cases. Thus, in order to study the behaviour of such systems, we construct models which simulate the behaviour of the system.

Some models are "physical" models, namely scaled versions of the actual system. For example, model aeroplanes are placed in wind tunnels to study the behaviour of real aeroplanes in flight. However, when possible, it is more convenient and flexible to build a mathematical model which can be studied on a computer. This modelling has become so important in almost every branch of physics that the need for larger and more sophisticated simulations is a major driving force behind the evolution of supercomputers.

A simulation consists of the following steps:

- We extract the differential equations describing the physics of the system.
- We construct a model of the system which obeys the same equations as closely as possible.
- We observe the model running to see what it does.

In this experiment, we consider one class of simulation in which the physical system is described by one or more ordinary differential equations with specified initial conditions. We solve these equations by finding functions which satisfy the differential equations and the initial conditions. It is convenient to refer to the independent variable as time t, although this can represent some other quantity in the physical system.

Block diagrams and Integrators

A mathematical model of a differential equation can be represented graphically by a block diagram. These diagrams give you a good idea of the mathematical model's structure, as they show you how its subsystems are connected.

The "integrator" block (Fig. 1) represents an operation which integrates some input function $x_{IN}(t)$ with respect to time, and gives an output. The initial condition x_{IC} is a constant which specifies the output value at the initial time.

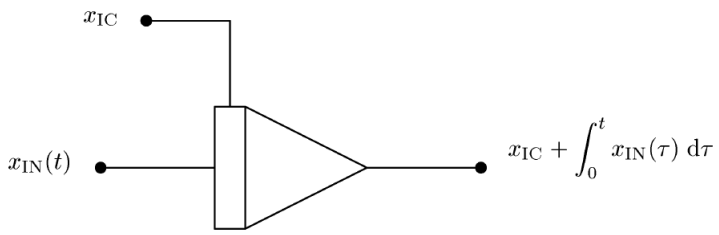


Figure 1: The integrator

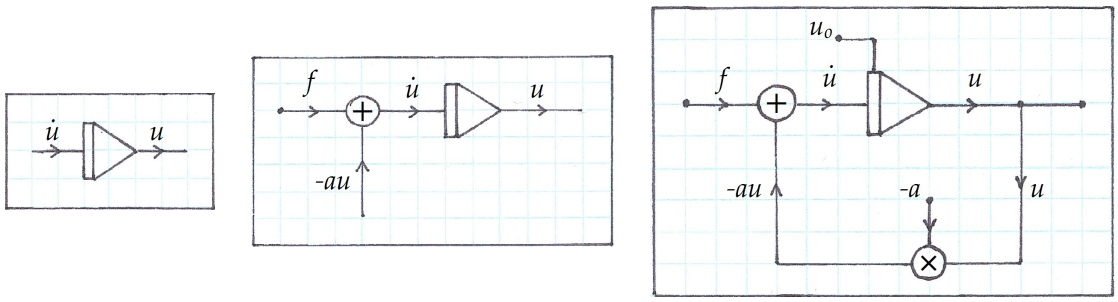
Other useful "elementary blocks" include $\otimes$ and $\oplus$ which multiply and add together two or more inputs respectively.

The following is a systematic method of drawing block diagrams for differential equations.

1. Rearrange the differential equation so that the highest-order derivative of the variable (u, say) is expressed in terms of everything else.
2. If you have a
   (a) first-order time derivative, you need one integrator block for variable u which appears with its time derivative. The time derivative is the integrator input. The variable itself is the output.
   (b) second-order time derivative, connect two integrators in series from left to right. The input to the leftmost integrator is $\ddot{u}(t)$ so that the input for the next integrator is $\dot{u}(t)$. Then the final output is $u(t)$.
   (c) higher-order time derivative, extend the above procedure.

   Variables at the outputs are called "state variables" and are usually relabelled x_1, x_2, etc. Their values at any time represent the system's "state".
3. Include the initial conditions. Connect the integrators with other elementary blocks described above, according to the differential equation. This is usually the trickiest part.

Example: Solve the initial value problem $\dot{u}(t) = -au(t) + f(t)$ with $u(0) = u_0$.



The differential equation is first arranged to give an algebraic expression for the derivative of the function and this has to be integrated. The fundamental unit used to carry out the simulations is thus the integrator which is usually drawn as shown in Figure 1. The input is a function of time which is integrated (with respect to time) to give the output. The initial condition is a constant which specifies the output value at the initial time.

In the past, integrators were constructed out of mechanical devices or electronic circuits. By connecting these to other devices which carry out mathematical operations, it is possible to construct a mechanical or electronic analogue of the physical system which obeys the same differential equations. This is the basis of "analogue computers" which were commonly used before we had cheap and powerful digital computers.

To perform integration numerically with a digital computer, we take the value of the function at the initial time and the expression for the derivative, then evaluate the function value a short time later, Δt. This is used as the starting value for a second iteration to find the function value after $2\Delta t$. This process can be repeated until we obtain a set of function values at times $\Delta t, 2\Delta t, 3\Delta t$, etc. This set of numerical values is a "numerical solution" of the differential equation.

We shall use a PYTHON program `scipy.integrate.odeint` to integrate the differential equations. Import the functions in your PYTHON program by calling

```
from scipy.integrate import odeint
```

The `scipy.integrate.odeint` function is then available as `odeint`. This implements a fourth-order Runge-Kutta integrator for a vector first-order differential equation. During the course of the experiment we see how this can be used to solve higher-order differential equations as well as coupled systems of equations. The Appendix gives a full description of the input and output arguments of the function for reference.

In the following, sections involving experimental work are numbered.

Integrating a known function

As a first example, consider integrating a function of time whose algebraic form is known. Suppose that the force in newtons on a 1 kg mass is given by

$$F(t) = 1 + 2\sin(3t)$$

We wish to find the velocity $v(t)$, given that at $t = 0$, the mass is travelling at -1 ms^{-1}.

This problem can be written as an initial value problem for v as

$$\frac{dv}{dt} = 1 + 2\sin(3t) \qquad v(0) = -1$$

The solution may be written as

$$v(t) = -1 + \int_0^t [1 + 2\sin(3t)]\, d\tau$$

This leads to the block diagram shown in Figure 2. In order to use the PYTHON function `odeint` to solve this equation, we need to write a function file which tells us how to find the input to the integrator (i.e., the derivative of v) at time t.

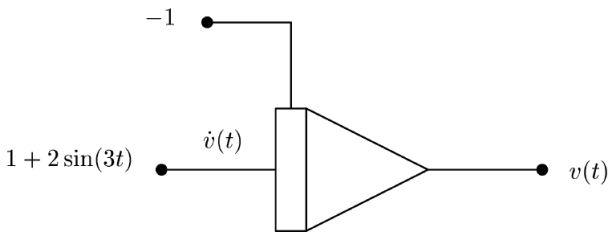


Figure 2: Integrating a known function

(1) Create the following function:

```
def deriv1(v,t):
    return 1.0+2.0*np.sin(3.0*t)
```

Note that this function always has (at least) two input arguments. The first is the independent variable t and the second is the current value of v. In general the derivative $\dot{v}(t)$ depends on both of these variables, but in this case it does not depend on v.

(2) Having written this file, call the integration routine `odeint` from the PYTHON command line as follows:

```
from scipy.integrate import odeint
tout=arange(0,10,0.1)
vout = odeint(deriv1,-1.0,tout)
```

Note that the input arguments to `odeint` are:

- the name of the function which specifies the derivatives, `deriv1`
- the initial value of the dependent variable $v(0)$, -1 ms^{-1},
- the independent function for which you would like the integrated function, the array `tout`.

The output of `odeint` consists of:

- a matrix whose columns are the values of the dependent variables. Here, there is only one dependent variable, so `vout` is a column vector of the same length as `tout`.

A simple analytic solution is

$$v = t - \frac{1 + 2\cos(3t)}{3}$$

(3) Plot the velocity as calculated by the numerical and analytic solutions by entering

```
vanalyt = tout-(1.0+2.0*cos(3.0*tout))/3.0
plot(tout,vout,'x',tout,vanalyt)
```

Print out your graphs and check that they do coincide.

A scalar first-order differential equation

Now consider a slightly different initial value problem,

$$\frac{dv}{dt} = -v + 1 + 2\sin(3t) \qquad v(0) = -1$$

This could model a problem in which there is an external force $F(t)$, plus a frictional force proportional to the velocity (Figure 3).

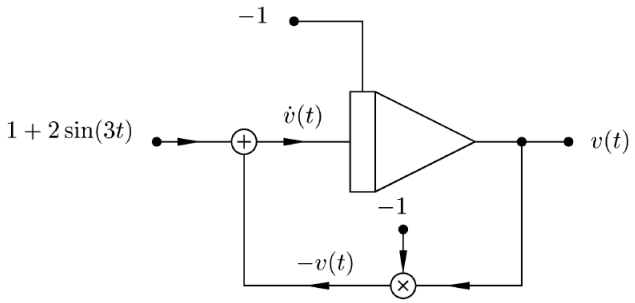


Figure 3: Solving a first-order scalar differential equation

(4) Write the function `deriv2` to calculate the derivative $\dot{v}(t)$ for this problem by modifying `deriv1`.

(5) Use `odeint` to find the solution of the initial value problem from $t = 0$ s to $t = 10$ s using a time step of $h = 0.1$ s. Plot the result of your simulation using crosses together with the analytic solution

$$v = \frac{5 + \sin(3t) - 3\cos(3t) - 7\exp(-t)}{5}$$

Damped simple harmonic motion

The differential equation for damped simple harmonic motion is

$$\frac{d^2y}{dt^2} + B\,\frac{dy}{dt} + C\,y = 0$$

with initial velocity $\dot{y}\,(0)$ and initial position $y\,(0)$. Although this is a second-order differential equation, we can convert it into a form which can be integrated using `odeint`.

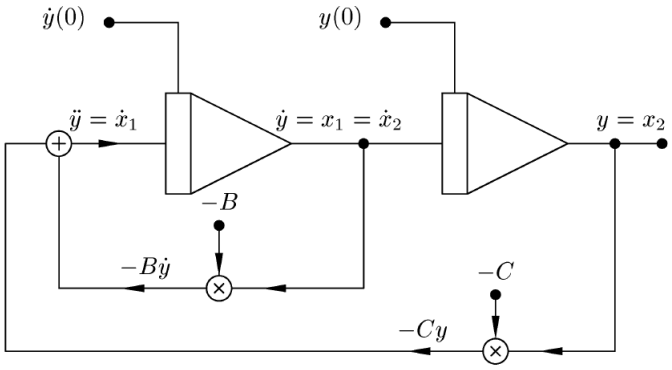


Figure 4: Solving the damped simple harmonic oscillator

In Figure 4, two integrators are connected in series to give the desired output $y\,(t)$. Note that the input to the second integrator is $\dot{y}\,(t)$ and the input to the first $\ddot{y}\,(t)$. By rearranging the differential equation, we have

$$\frac{d^2y}{dt^2} = -B\,\frac{dy}{dt} - C\,y$$

The input to the first integrator is found by combining $y\,(t)$ and $\dot{y}\,(t)$.

We now relabel the outputs of the integrators x_1 and x_2 and regard them as components of a column vector $\mathbf{x}$, the *state vector* of the system. From the block diagram, we see that the scalar second-order differential equation is equivalent to the *vector* first-order equations

$$\frac{dx_1}{dt} = -B\,x_1 - C\,x_2$$
$$\frac{dx_2}{dt} = x_1$$

with initial conditions $x_1\,(0) = \dot{y}\,(0)$ and $x_2\,(0) = y\,(0)$.

(6) Create the function `shm` which calculates the derivatives for the damped simple harmonic oscillator problem from the vector differential equation above.

```
def shm(x,t,B,C):
    return np.array([-B*x[0]-C*x[1], x[0]])
```

Notice that the derivative `dx` is now a *column vector* as is the variable `x`. Another new feature of this example is the additional parameters B and C which are required by this function. Such parameters can be passed to the derivative function by specifying them at the end of the parameter list of `odeint`.

(7) Use the following to obtain solutions for the undamped simple harmonic oscillator starting from rest.

```
tout=np.array(0,2,0.02)
    xout = odeint(shm,[0,10]),tout,args=(0,4.0*pi**2));
```

i.e. a solution is required from $t = 0$ s to $t = 2$ s using a time step of $h = 0.02$ s, initial conditions are $x_1(0) = \dot{y}(0) = 0$ and $x_2(0) = y(0) = 10$, parameters B and C required by the function `shm` have the values 0 and $4\pi^2$ respectively.

(8) Plot both the velocity $\dot{y}(t)$ and the displacement $y(t)$ using:

```
plot(tout,xout)
```

where the first column of `xout` contains the velocity $x_1(t)$ and the second column contains the displacement $x_2(t)$. Confirm that the oscillation frequency, the amplitudes and relative phase of the velocity and displacement functions are as expected.

(9) Rerun the simulation with $B = 0.1$ and $C = 1$ using $t = 0$ to 64 s with $h = 0.2$ s. Use initial condition $\dot{y}(0) = 0$ and $y(0) = 1$. Plot $y(t)$ as a function of time from your simulation. Check the results of your simulation as follows:

The analytic solution to the differential equation is

$$y(t) = A \exp\left(-\frac{t}{\tau}\right) \cos(\omega' t + \delta)$$

Deduce the values of τ and ω' in terms of B and C.

Determine τ and ω from your simulation and compare them with the values you expect. Note that τ is equal to the time required for the oscillation amplitude to fall by a factor $1/e$ and that ω' can be found from the zero crossings.

(10) Rerun the simulation with $C = 1$ and adjust B to give critical damping. Calculate the analytic solution and display this together with your simulation results for initial conditions of your choosing.

Relaxation oscillators

The simulation technique also works for equations which do not possess analytic solutions. For example, consider the following non-linear differential equation

$$\frac{d^2y}{dt^2} + B\,y\frac{dy}{dt} + C\,y = 0.$$

For $B = 0$ this reduces to the equation for undamped simple harmonic motion. For other B values, the equation characterizes a class of systems called "relaxation oscillators". These have solutions that change slowly (relax) for half a cycle, then change rapidly in the second half cycle.

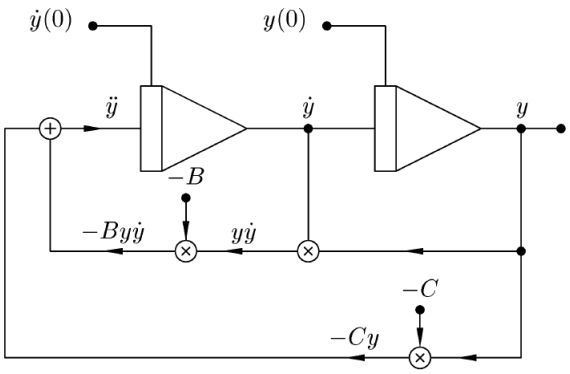


Figure 5: The relaxation oscillator

(11) Confirm that the block diagram for this differential equation is as shown in Figure 5. Write a function file corresponding to this differential equation and run the simulation with $B = 0.5$ and $C = 0.5$ starting from rest and initial displacement $y(0) = 1$. Use $h = 0.2$ s and a final time of 20 s. Plot both the displacement and velocity in time synchronization (use the `subplot(211)` command).

(12) Repeat the simulation for $B = 5$. It is appropriate to time-steps $h = 0.5$ s and to carry out the simulation until $t = 60$ s.

Compare the solution to that for simple harmonic motion. The displacement should decrease approximately linearly until it reaches some negative value, when it rapidly increases back to the initial value before repeating cyclically. Does the velocity curve agree with this?

The relaxation oscillator equation may be written in the form:

$$\frac{d^2y}{dt^2} + \left(B\,\frac{dy}{dt} + C \right) y = 0$$

If B is small, we may think of the coefficient of y as the square of an "instantaneous angular frequency". If $dy/dt > 0$, this coefficient is larger than C and y changes faster. If $dy/dt < 0$, the coefficient is smaller than C and y changes slower. These correspond to the "fast" and "slow" portions of the observed solutions.

As B becomes large, the solutions become increasingly non-sinusoidal. The (almost) linear negative-going portion of the $B = 5$ solution can be thought of as a section of a long period cosine wave.

The motion of bowed violin strings is of this form although it is not obvious how the differential equation arises from the physics of the violin.

Projectile motion with air resistance

An object moving at high speed experiences air resistance approximately proportional to the square of the speed. Figure 6 shows a projectile travelling at speed v and the forces acting on it.

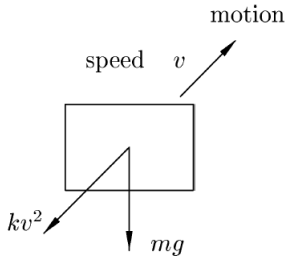


Figure 6: Forces on a projectile

If the horizontal and vertical coordinates of the projectile are x and y, Newton's second law gives:

$$m\frac{d^2x}{dt^2} = -k\,v\frac{dx}{dt}$$
$$m\frac{d^2y}{dt^2} = -m\,g - k\,v\frac{dy}{dt}$$

(13) Draw two block diagrams, one for each differential equation. You should find that four integrators are required and that it is necessary to have some connections between the two block diagrams. Use a state vector with four components to write the function file to simulate the projectile's motion with `odeint`. Choose the parameters so that a projectile with mass 0.01 kg has a terminal velocity of 100 ms^{-1} when falling vertically. Assume $g = 9.8$ ms^{-2}.

(14) Run a simulation with initial speed 600 ms^{-1} and initial angle 45°. These correspond approximately to values for a 0.22 rifle bullet. A suitable value for h is 0.5 s. Simulate for 35 s. Test different initial angles to find the maximum range of the bullet and the angle of elevation. Plot the bullet trajectory for this case. Compare your result with the maximum range expected if air friction is neglected. You may find the following function useful for finding the range starting from vectors of x and y positions.

```
def findrange(x,y):
    i2=np.min(find(y<0.0))
    i1=i2-1
    r = (y[i1]*x[i2]-y[i2]*x[i1])/(y[i1]-y[i2])
    return r
```

(15) Change the time step to $h = 0.1$ s and use trial-and-error to determine the initial angle required for ranges of 50, 100 and 200 m. An accuracy of ± 0.5 m in the range is acceptable. Calculate how far above the target the rifle should be aimed. For each range, prepare a table showing range, initial angle, maximum height of trajectory, distance above the target for initial aim and the final velocity.

Coupled masses

We now consider a coupled system of differential equations which describe the motion of two masses connected by springs, free to slide on a horizontal surface (Figure 7). Suppose the masses are both equal to m and that the two outer springs have spring constants k. The central spring is assumed to have spring constant k_c.

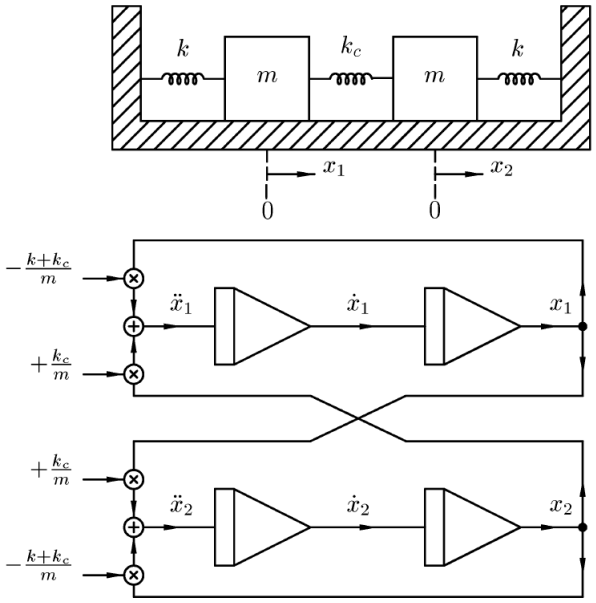


Figure 7: The coupled oscillator

If x_1 and x_2 are the displacements of the masses from equilibrium, the equations of motion are:

$$m\frac{d^2x_1}{dt^2} = -k\,x_1 - k_c\,(x_1 - x_2)$$
$$m\frac{d^2x_2}{dt^2} = -k\,x_2 - k_c\,(x_2 - x_1)$$

(16) Confirm that the block diagram in the lower half of Figure 7 corresponds to the differential equations given above. Write a function file for simulating the system using the parameters $m = 1$ kg, $k = 1$

Nm^{-1}. Carry out a simulation with $k_c = 0.2$ Nm^{-1} starting with the masses at rest and $x_1(0) = 1, x_2(0) = 0$. A suitable value of h is 0.2 s and you should carry out the simulation for $0 < t < 60$ s.

(17) On the same set of axes, plot graphs of the total kinetic energy of each mass and the total potential energy of the springs assuming that they have their natural lengths when the masses are at equilibrium. Confirm that the total energy of the system remains constant during your simulation by plotting the sum on the same set of axes.

Notice how the energy of the system appears to move between the two masses (the left mass has maximum amplitude when the right mass has zero amplitude and vice versa). The motion of either mass appears to be the result of "beats" between two solutions at different frequencies. These "normal mode" frequencies are

$$\omega_s = \sqrt{k} \qquad \text{for the "symmetric mode"}$$
$$\omega_a = \omega_s\sqrt{1 + 2k_c/k} \qquad \text{for the "antisymmetric mode"}$$

Normal modes of a system are those responses which persist unchanged with time. In the above simulation, the system is not in a normal mode since neither mass oscillates sinusoidally.

For a normal mode, both masses execute simple harmonic motion with the same frequency, and amplitudes of both masses are the same. In the symmetric mode, the two oscillations are in phase and the central spring length does not change, so the frequency must be independent of k_c. In the antisymmetric mode, oscillations are exactly in antiphase and the central spring length does change, so its frequency depends on k_c. Either mode can be excited independently by choosing the appropriate initial conditions.

(18) Run the simulation with the initial conditions:

(a) $x_1(0) = 1, x_2(0) = 1$, to excite the symmetric mode

(b) $x_1(0) = 1, x_2(0) = -1$, to excite the antisymmetric mode.

Plot your results and confirm the amplitudes, phases and oscillation frequencies of the masses are as expected.

List of Equipment

1. PC with PYTHON or rather SPYDER installed.

S.M. Tan, May 1993, R. Au-Yeung, 13 May 2013

This version July 21, 2015