Numerical integration methods

The spring and mass

You will get experimental data of a mass-spring system, and write a Python program to solve the equation of motion. You will then use this simulation data to plot a visualization of position vs. time and the phase plot. You will also calculate the energy and plot it. You will have to discuss the output, and eventually improve the code.

This lab contains explicit questions for you to answer labelled with **Q**. However, you still need to maintain a notebook for the lab, make plots and write Python code.

Background knowledge for exercise 4

Python lists, arrays, numpy, pyplot
Physics Harmonic oscillator, Hooke's law. Review these from your first year text.

1 First Session

1.1 Introduction

Physics of springs is based on Hooke's Law

$$F_{\text{spring}} = -ky\,, \tag{1}$$

where y is the vertical displacement, and k is the spring constant measured in N/m. Hooke's law can be used with Newton's second law to derive the equation of motion:

$$\frac{\mathrm{d}^2y}{\mathrm{d}t^2} + \frac{k}{m}y = 0\,. \tag{2}$$

Q1 Show how equation (2) was derived. Point out the main approximations involved (if any).

The most common method used to find numerical solutions to equations of motion is by setting up a pair of *coupled ordinary differential equations*.

Given a mass, m, coordinate $\vec{q}$, momentum $\vec{p}$, and force $\vec{F}$,

$$\frac{\mathrm{d}\vec{p}}{\mathrm{d}t} = \vec{F} \tag{3a}$$

$$\frac{\mathrm{d}\vec{q}}{\mathrm{d}t} = \frac{\vec{p}}{m}\,. \tag{3b}$$

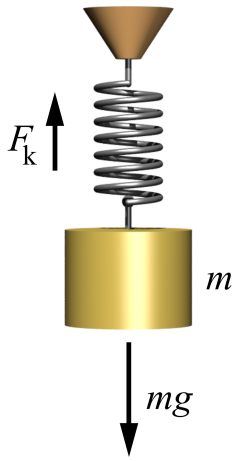

Figure 1: A vertical spring-mass system

The force can be any function of $\vec{p}$ and $\vec{q}$ in general, but in this exercise, we will look at a linear spring: $\vec{F} = \vec{F}_{\text{spring}}$

1.2 Numerical Methods

The solution to equation (2) is well-known, but in general, differential equations can be difficult to solve. In practice, we can calculate approximate solutions by approximating the derivatives as

$$\frac{\mathrm{d}y}{\mathrm{d}t} \approx \frac{1}{\Delta t}\big[y(t+\Delta t) - y(t)\big]\,. \tag{4}$$

Therefore, equations (3) can be approximated with

$$\vec{p}(t+\Delta t) = \vec{p}(t) + \Delta t\, F(\vec{q}(t)) \tag{5a}$$

$$\vec{q}(t+\Delta t) = \vec{q}(t) + \frac{\Delta t}{m}\vec{p(t)}\,. \tag{5b}$$

Equations (5) are *update formulae*. Given initial position, momentum, and starting time $t = t_0$, we can determine a numerical approximation of position and momentum at times $t_0 + \Delta t, 2\Delta t, \ldots$. The numerical solution will approach the actual solution as $\Delta t \to 0$.

This time-stepping scheme is called the "Forward Euler" method, because it uses the derivative at the start of an interval to extrapolate *forward*. We will see other time-stepping schemes in the next session.

The scheme can also be written in a recursive form,

$$\vec{p}_{i+1} = \vec{p}_i + \Delta t\, F(\vec{q}_i) \tag{6a}$$

$$\vec{q}_{i+1} = \vec{q}_i + \frac{\Delta t}{m}\vec{p}_i\,, \tag{6b}$$

where $\vec{p}_i = \vec{p}(t_i)$, $\vec{q}_i = \vec{q}(t_i)$, and $t_i = t_0 + i\Delta t$.

1.3 Numerical methods for the mass-spring system

The equation of motion for the vertical spring-mass system can be written in the coupled form as:

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -\Omega_0^2 y \tag{7a}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = v\,, \tag{7b}$$

where $\Omega_0 = \sqrt{k/m}$ is the angular frequency of oscillation.

The initial conditions will be $y_0 = A$ (the amplitude measured in experiment), and $v_0 = 0$ m/s. Using (7), the numerical approximation can be written as

$$y_{i+1} = y_i + \Delta t\, v_i \tag{8a}$$

$$v_{i+1} = v_i - \Delta t\, \Omega_0^2 y_i \tag{8b}$$

for $i = 0, 1, 2, \ldots$. This is the update scheme you will use in your first program.

Q2 Show the derivation of equations (8) using the equation of motion (2) and the definition of the Forward Euler method (5).

1.4 The experiment

You should have a mass hanging from a spring. Underneath the mass is a motion sensor, which is plugged into a data acquisition device (DAQ). The output of the DAQ is analyzed by a LabView application (`~\Desktop\2nd Year lab Files\MotionSensor.vi`)

The goal of the lab exercise is to determing the period of oscillation of the bob, T (and therefore the frequency Ω_0). This will be used in our later analysis.

Do the following:

1. Measure the mass of the bob.
2. Open `MotionSensor.vi`.
3. Switch to the "Setup/Callibrate" panel, and change the samples/second to 100.
4. Switch to the "Collect" panel. Position the bob at about 20 cm above the motion sensor.
5. Start the bob oscillation by either raising or lowering it from equilibrium and letting go; a few centimeters should be enough.

6. Click "start detector" the motion sensor should start buzzing now. Click "Collect Data" to begin the data collection.
7. Let the bob oscillate for about 10 seconds, then stop the detector.
8. *If the motion sensor is not aligned well, the data may be poor. As long as the graph shows a fairly regular motion, it should be possible to measure the period.*
9. Switch to the "Analyze" panel. Adjust the yellow cursors to the begining and end of 5 to 10 oscillations.
10. *You may need to change the bounds of the plot to get a closer view. You do this by highlighting the last value on either scale and replacing it by your value.*
11. Read the time between cursors, and divide by the number of oscillations observed to get the period.
12. Record the period.
13. Open File/Position Data Set. Save the data file on your memory stick.
14. Plot Distance vs. Time with labelled axes. *Ignore the "Dist. Error" for now.*. Calculate the period from your plot.

Q3 What is the value of the spring constant, k?

1.5 Python programming

Scripts for simulations typically follow the same pattern

1. define constants/parameters
2. set the initial conditions
3. use the numerical approximation (8) to step forward in time
4. loop until the final time
5. plot the graph and interpret the result

Additional steps that may be useful include

- wrapping it in a function for repeated simulations,
- saving the data for later analysis.

The outline for our specific program is,

1. import required modules (eg. `pylab`, or `numpy` and `matplotlib.pyplot`)
2. define constants Δt and Ω_0.
3. calculate the time values, t_i for the plot using $\Delta t = 0.01$ s, $t_0 = 0$ s, $T = 10.0$ s.
4. preallocate the array (with the `zeros()` function)
5. set initial conditions
6. write the time-stepping loop
7. plot the data (*position vs. time*) with labelled axes

Write and run the program. Save the program and its output on your memory stick.

1.6 Programming - Analysis

Q4 What do you see happening in the plot? Is this what you expect? Can you interpret the graph?

There are other graphs that are useful for analysis:

1. 'velocity vs. time' (time on horizontal axis, velocity on vertical axis)
2. 'velocity vs. position' (position on horizontal axis, velocity on vertical axis)

The latter is called a phase-plot.
Create these plots with labelled axes and save them.

1.6.1 Energy

Energy (and other conserved quantities) are very useful for analyzing dynamical systems. For the spring-mass system, the mechanical energy is conserved,

$$E_{\text{tot}} = K(\dot{y}) + U(y)\,, \tag{9}$$

where K is kinetic energy ($K = \frac{1}{2}mv^2$) and U is potential energy (both elastic and gravitational). The energy expression we will use is

$$E_{\text{tot}} = \frac{1}{2}mv^2 + \frac{1}{2}ky^2\,. \tag{10}$$

The term for gavitational potential energy can be omitted by wisely choosing the reference for potential energy. *You should be able to prove this statement.*

- Modify your program to calculate the energy at each step. Note that the energy is not zero at $t = 0$. The mass (m) and spring constant (k) will need to be added to your code.
- Plot 'Energy vs. time'

Q5 What does the energy plot suggest? Does this explain the strange appearance of the *velocity vs. time* and *velocity vs. position* plots?

Q6 For a spring-mass system, the phase plot should be an ellipse. Explain why, using energy conservation. Give an explanation for the appearance of your phase plot.

Q7 (bonus) Determine the leading error in our numerical method. Hint: use a Taylor expansion of $y(t+\Delta t)$ and find the terms we ignored in equation (5).

1.7 More on numerical integration schemes

If you properly implemented the numerical approximation in Equation (8), you should have found that the simulation was bad: the amplitude increased, and energy was not conserved. The reason for the failure was the integration method. We will discuss other numerical integrators, and implement a symplectic scheme, which is suitable for conservative systems.

In the first program, we used the most primitive numerical integration method called the *Forward Euler* or *Explicit Euler* method,

$$\begin{aligned} y[i+1] &= y[i] + \Delta t\, v[i] \\ v[i+1] &= v[i] - \Delta t\, \Omega_0^2 y[i]\,. \end{aligned}$$

Forward Euler extrapolates the position through an interval Δt using only the (approximate) derivative at the begining of the interval. *It can be proved [1] that Forward Euler is unstable, which means that the amplitude and total energy increases in time.* In phase space, the increase in energy looks like a spiral outward. **The accuracy of the Forward Euler method can be improved by decreasing the time step, but the energy will always increase.**

A similar method to Forward Euler is *Backward Euler* or *Implicit Euler*. This method uses the derivative at the end of the interval (making an implicit scheme) to calculate the next point,

$$y[i+1] = y[i] + \Delta t\, v[i+1] \tag{11a}$$

$$v[i+1] = v[i] - \Delta t\, \Omega^2 y[i+1]\,. \tag{11b}$$

Backward Euler is absolutely stable, which means that longer timesteps can be used (especially for "stiff" or rapidly changing ODEs). However, it *artificially dissipates* energy.

A simple remedy is to combine the Forward and Backward Euler methods to create the *Euler-Cromer* or *Symplectic Euler Method*,

$$y[i+1] = y[i] + \Delta t\, v[i] \tag{12a}$$

$$v[i+1] = v[i] - \Delta t\, \frac{k}{m} y[i+1]\,. \tag{12b}$$

The result is an explicit method, which *conserves* approximate energy. Do the following:

- Insert the new code lines for the symplectic method (12) into your previous program, and plot the "Energy vs. time" and phase plots.
  You may want to simulate using both methods, and plot both results on the same axes.
- Compare and discuss the plots from the Forward Euler and Symplectic Euler Methods. Use the same time step for both methods in the comparison.

Q8 How do the energy and phase plots change using the symplectic method?

Keep in mind for the future that it is essential to use a method that conserves energy (e.g. symplectic methods) when simulating conservative problems.

Your submission for this exercise should include

- All plots marked above (position, energy, and phase space plots for both Forward and Symplectic Euler Methods)
- answers to questions 1-8,
- the final version of your program,
- values from your experiment (period, amplitude, frequency, and spring constant).

2 Second session

In the first part of this exercise, we analyzed the equation of motion of a spring-mass system and wrote Python code to numerically integrate the equation. In this session, we will add a physical damping term, and *qualitatively* compare the numerical simulation to the experiment.

2.1 Adding a damping term

In the real world, mechanical energy cannot be perfectly conserved. So far, the equations of motion you have used model the system *without damping* (physical dissipation). In general, the damping force exerted on a body depends on the velocity $\vec{v}$ of the body. Drag is one source of damping caused by velocity of a body relative to the surrounding medium (air or water) commonly modelled as quadratic drag,

$$\vec{F}_{\mathrm{d}} = -\frac{1}{2} C \rho A |\vec{v}| \vec{v}\,, \tag{13}$$

where

C is the drag coefficient (dimensionless),
ρ is the density of the medium (kg/m^3),
A is the cross-sectional area perpendicular to the flow (m^2), and
$\vec{v}$ is the velocity of the body relative to the medium (m/s).

The direction of the damping force is always opposite to the direction of velocity. The drag coefficient is not a universal constant: it depends on viscosity of the medium, shape of the body, and roughness of its surface. It also depends on the velocity through the Reynolds number.

The *Reynolds number* Re, is a very useful dimensionless quantity used to characterize flows in fluid mechanics. In our case, it will characterize the dependence of the drag coefficient on velocity. The Reynolds number is the ratio of the inertial force of the medium to the viscous force:

$$\mathrm{Re} = \frac{\rho v l}{\eta} \tag{14}$$

where

ρ is the density of the medium (kg/m^3),
v is the velocity of the body relative to the medium (m/s),
l is the characteristic length in the direction of flow (m), and
η is the dynamic viscosity (N·s/m^2).

If the flow is *laminar*, the Reynolds number takes small values (Re < 2300) and the drag coefficient is inversely proportional to the velocity. This makes the drag force directly proportional to velocity:

$$\vec{F}_{\mathrm{d}} = -\gamma \vec{v}\,, \tag{15}$$

where γ is the damping coefficient.

When the flow is turbulent, the Reynolds number is large (Re > 4000) the drag coefficient is approximately constant and the drag force is dependent on the square of the velocity.

Q9 What is the Reynolds number for the spring-mass system? *Hint: this may depend on the amplitude of the oscillation.*

We expect that the motion and speed of a typical mass-spring system in air correspond to small Reynolds numbers. Therefore, the equation of motion of our system would be written as:

$$\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} + \gamma \frac{\mathrm{d}y}{\mathrm{d}t} + \Omega_0^2 y = 0\,. \tag{16}$$

2.2 Repeat the experiment

The effects of damping were likely not seen in the last session, because the damping for the bob is quite small. To make damping more observable, we can attach a disk to the bob to increase drag.

- Attach the damping disk to the bob.Weigh the new system.
- Repeat the experiment as outlined in Section 1.4 for a longer period of time so that you can see the decay – about 2 minutes with the fin attached. **Note: because the experiment setup has changed, you should measure the frequency and mass again this session.**
- Using the cursors, take 5-6 readings of amplitude. Assuming an exponential decay of the oscillation envelope, estimate the decay constant γ ($\gamma/2$ is the inverse of time for which amplitude falls to $1/e$ of the initial value). *You do not have to use Python to do this calculation. Please verify that the exponent in position decay is $\gamma/2$ while the exponent in energy decay is γ.*
- Save the position vs. time plot.
- Export the data in case you need it later.

2.3 The new Python program

The coupled equations we need to formulate for our new Python program are

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -\Omega_0^2 y - \gamma v \tag{17a}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = v \tag{17b}$$

Modify your program from part 1.3 to include drag with (17) and plot 'Position vs. time' and 'energy vs. time'. Use the decay coefficient determined experimentally.

Q10 Discuss the plot. What effect does damping has on the energy?

2.4 Compare with experimental data

The practice of Physics often includes these steps:

- come up with a (mathematical) model for a phenomenon,
- make predictions with that model,
- compare those predictions against experiment.

The comparisons can be qualitative (e.g. do both the predicted and measured amplitudes decay with time?), or quantitative (e.g. what is the difference in period between the prediction and experiment?). For now, we will just do a qualitative comparison. Fitting this complex function with `curve_fit` is very hard and requires a carefully designed fitting routine to find the correct answer.

Q11 Compare the experimental data with the output of your program. What do you think makes them different?

Your submission for this exercise should include:

- plots for position and energy of the damped system,
- a plot of the experimental data,
- answers to questions 9-11,
- the final version of your program,
- values from your experiment (period, frequency, decay rate, and spring constant).

References

[1] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes in C*, volume 2. Cambridge university press Cambridge, 1996.

Prepared by John Ladan and Ruxandra Serbanescu, 2016. Edited by Christopher Lee, 2017.