Introductory numerical physics & Numerically solving the equations for radioactive decay and for oscillatory motion

Department of Physics, University of Surrey module:
Energy, Entropy and Numerical Physics (PHY2063)

1 Introduction to Numerical Physics part of Energy, Entropy and Numerical Physics

This numerical physics course is part of the second-year Energy, Entropy and Numerical Physics module (PHY2063), and is online at the EENP module on SurreyLearn. See there for assignments, deadlines etc.

Very few models in physics can be solved analytically with just pen and paper. Therefore most predictions and modelling done in physics requires numerically solving the equations of the model. This is true for electromagnetism, quantum physics (including nuclear and particle physics), astrophysics, soft matter physics, etc. This course introduces numerical physics through some simple common examples, and discusses general points about numerical physics, in particular the key problem of estimating how accurate the numerical solution is.

It is taught using Fortran, which is as good a language as any for numerical physics; Fortran was designed for doing numerical physics, and number crunching is its strength. Fortran is also a good easy language to get started with. Almost all high performance computing (e.g., solving large PDEs, quantum matrix problems, molecular dynamics, fluid mechanics, etc) is done using Fortran or C. Other languages have other strengths, python would be a bit quicker to use for many data or image analysis problems, for example. The numerical physics skills you will learn using Fortran will be transferable to number crunching applications in other computer languages. And the coding skills are transferable to a huge variety of other tasks people use computers to do.

The course looks at numerically solving ODEs (ordinary differential equations) and PDEs (partial differential equations), and introduces the numerical modelling of random processes, using both a very widely used numerical technique called Monte Carlo (MC), and the use of Bayesian statistics. MC is used a lot in fields from statistical physics, to nuclear and particle physics. Bayesian statistics is used from particle physics to predictive text on mobile phones. A lot of modelling in physics is based on PDEs, from heat and mass flows in statistical physics, a lot of electromagnetism and quantum physics, etc. And the numerical techniques we will learn are also used extensively outside physics, for example to model share prices in stock markets. Increasing numbers of graduates in physics (and related subjects) are going into data science, which uses many models of random processes such as Monte Carlo modelling and Bayesian statistics.

Examples of ODEs are the equations for radioactive decay and for a mass attached to a spring. PDEs are even more common in physics than ODEs, for example, Schrödinger's time-independent equation in three dimensions is a PDE, Maxwell's equations which govern electromagnetism (including light), and

the wave equation, are all PDEs. In many cases (all except very simple cases) we cannot solve them analytically, and so can only solve them numerically.

1.1 Course structure

The autumn semester computing is composed of 3 assignments, Assignments 0, 1 and 2. Assignment 0 is on ODEs, Assignment 1 is on MC and Bayesian statistics, and Assignment 2 is on PDEs.

All assignments are assessed. However, only Assignments 1 and 2 contribute to the module mark for the module. Assignment 0 is for feedback only, i.e., it does not contribute to the final module mark, the mark is for you to assess how well you are doing.

2 Background to numerically solving first and second-order ODEs

We will start with a very simple example, the ODE for the mass of a radioactive isotope remaining after a time t, $M(t)$. If the rate of decay of the isotope is r then the ODE is

$$\frac{\mathrm{d}M(t)}{\mathrm{d}t} = -rM(t)$$

and if the decay rate $r = 0.3\ \mathrm{s}^{-1}$ then

$$\frac{\mathrm{d}M(t)}{\mathrm{d}t} = -0.3M(t)$$

Now as this a first-order ODE we know to obtain a particular solution we need **exactly one** boundary condition (BC). Usually this is the value of the mass of the isotope at time $t = 0$. If for example, $M(t = 0) = 0.1$ kg then the particular solution is

$$M(t) = M(t = 0)\exp(-rt) = 0.1\exp(-0.3t)$$

this is the familiar exponential decay of the amount of remaining isotope.

This is just one example, ODEs are common in physics, e.g., the time-independent one-dimensional Schrödinger equation is a second order ODE

$$-\frac{\hbar^2}{2m}\frac{\mathrm{d}^2\psi(x)}{\mathrm{d}x^2} + V(x)\psi(x) = E\psi(x)$$

which we need to solve for the wavefunction $\psi(x)$ and energy E. Here $V(x)$ is the potential energy function. Another ODE is the equation for a mass m attached to a spring of spring constant k

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -kx$$

where we want to solve for the position as a function of time, $x(t)$.

Taylor Series Revision To get the formula for the Euler method we will need the Taylor series expansion expression for the value of $M(t)$ at the point $t_f = t_s + h$ in terms of M and its derivatives at the nearby point $t = t_s$. Nearby means that the distance h between the two points is small. You did the Taylor series in first year maths. The series expansion for the point $t = t_s$ is

$$M(t_f) = M(t_s + h) = M(t_s) + h\left(\frac{\mathrm{d}M}{\mathrm{d}t}\right)_{t_s} + \frac{1}{2}h^2\left(\frac{\mathrm{d}^2M}{\mathrm{d}t^2}\right)_{t_s} + \cdots$$

where the subscripts t_s indicate that the derivatives are evaluated at the point $t = t_s$.

We can also write this as

$$M_f = M_s + hM'_s + \frac{1}{2}h^2M''_s + \cdots$$

where I use the standard notation of a dash to indicate the first derivative, and two dashes to indicate the second derivative. Here $M_f = M(t_f)$, $M_s = M(t_s)$, $M'_s = M'(t_s)$ and $M''_s = M''(t_s)$. If h is small we can drop all but the first two terms

$$M_f \simeq M_s + hM'_s$$

Note that $M(t)$ denotes a function M of the variable t but M_s and M'_s are not functions they are just numbers, the values of the functions M and M' at the point t_s.

3 Solving a single first-order ODE on a computer: Euler method

We will introduce the simple Euler method for numerically integrating an ODE, and then also introduce the more accurate and slightly more complex modified Euler method. For all methods of solution it is important to remember that for a first-order ODE the solution of a first order ordinary differential equation requires a single boundary condition to be specified (this is usually an initial condition). The specification of the value of the required function at a single point is thus necessary to generate the required solution.

So we would like to solve the first-order ODE

$$\frac{\mathrm{d}M(t)}{\mathrm{d}t} = -rM(t) \qquad r = 0.3 \text{ s}^{-1}$$

with the BC $M(t = 0) = 0.1$, and over some range of values of t, say from $t = 0$ to $t = 30$ s.

The above ODE is just an example of a first order ODE, in general the derivative is some function of the variable, here t. The key point to take from the above equation is that at any point along the t axis the equation does not tell us what we ultimately want to know, the value of the function $M(t)$, but it does tell us the slope of the function, i.e., how fast it is changing.

3.1 The Euler method

The Euler method works by dividing the range between $t = 0$ and $t = 30$ s into many small intervals each of length h: this is called the step length. For example, if $h = 0.1$, then Euler's method of solving an ODE starts from $t = 0$ and integrates from there to $t = 30$ s, in $30/0.1 = 300$ little steps, each 0.1 in size.. The method works one step of length h at a time.

We start at the point $t = 0$. At that point we use the known value of $M(t = 0)$ to calculate the value of the function a distance h away along the time axis, i.e., we calculate $M(t = h)$. Then we use the now known value of $M(t = h)$ to calculate the slope at $t = h$, and so calculate $M(t = 2h)$, and by repeating this we integrate along the t axis.

3.1.1 Single Euler step

Euler's algorithm for a single step starts with a point along the t-axis, call it t_s where we know the value of the function $M(t_s) = M_s$ and its derivative, $M'(t_s) = M'_s$. Note that as M' is a known function of t and M (this is the ODE) whenever we know t and M we can calculate M'. We need the expression we got from truncating the Taylor series in the box above. This is

$$M_f \simeq M_s + hM'_s$$

this tells us how to calculate the value of the function $M(t)$ at the end of the step, M_f, in terms of the value of the function at the start of the step, M_s *and* its derivative at the start of the step, M'_s. You can start writing a program to solve an ODE, by writing one to do a single step like this.

In terms of what we need to calculate in a single step, it looks like

$$\begin{aligned} M'_s &= -rM_s \\ t_f &= t_s + h \\ M_f &= M_s + hM'_s \end{aligned}$$

or we go from maths type notation with superscripts and subscripts to writing our variables names as we do in code

$$\begin{aligned} \texttt{massdash\_s} &= -\texttt{r} * \texttt{mass\_s} \\ \texttt{t\_f} &= \texttt{t\_s} + \texttt{h} \\ \texttt{mass\_f} &= \texttt{mass\_s} + \texttt{h} * \texttt{massdash\_s} \end{aligned}$$

where I have picked sensible variable names. You should always use sensible variable names as this makes debugging much easier. With poor variable names, if you leave a program for a few days, it can be difficult for you to remember how it works. These lines implement one step, and at the end of this step, t_f (`t_f`) and M_f (`mass_f`) are the values of t a distance h along the t-axis, and our estimate for the M value there, respectively.

3.1.2 Obtaining the complete function (array)

To obtain the complete function $M(t)$ from $t = 0$ to $t = 30$ s, at the discrete set of points $t = 0, h, 2h, 3h, \ldots 30$, we need to appreciate that once we discretise the time axis into a grid of spacing h, the function M becomes an array

$$\{M(0), M(h), M(2h), \ldots, M(30.0)\} \rightarrow \{\texttt{mass(0)}, \texttt{mass(1)}, \texttt{mass(2)}, \ldots, \texttt{mass(300)}\}$$

where on the left hand side we have the function of time t, and on the right hand side we have the array with its elements denoted by i, which is an integer. This is if $h = 0.1$ s, and so we need array elements 0 to 300 to have enough elements to go from $t = 0$ to $t = 30$ s in steps of 0.1.

Then with time and mass both arrays one step looks like

$$\begin{aligned} \texttt{massdash} &= -\texttt{r} * \texttt{mass(i)} \\ \texttt{t(i+1)} &= \texttt{t(i)} + \texttt{h} \\ \texttt{mass(i+1)} &= \texttt{mass(i)} + \texttt{h} * \texttt{massdash} \end{aligned}$$

in the first line we calculate the derivative (which we don't need to be an array), then in the next two lines we calculate the next values of the arrays `t` and `mass`.

Task 1

Write a FORTRAN program to evaluate the solution of the first-order ordinary differential equation for radioactive decay at a rate $r = 0.3$ s^{-1}

$$\frac{\mathrm{d}M(t)}{\mathrm{d}t} = -0.3M(t)$$

with the single BC $M(t = 0) = 0.1$ kg. Obtain the mass as a function of time from $t = 0$ s to $t = 30$ s, in steps of 0.1 s (300 steps). Note that here we know the solution, it is

$$M(t) = M(t = 0)\exp(-rt) = 0.1\exp(-0.3t)$$

When you write any program, it is always best to at first get it to calculate something you know already, so you can compare solution to what you know it should be to check and debug the program. Then later you and modify the program you know works to do something new.

When you have the exact solution this is not only helpful to debug the program, you can use to see how accurate the solution is, for example as you vary h. As $h \rightarrow 0$ then the solution is exact but then you calculate an infinite number of points, which is not possible. So for $h > 0$ the array you calculate is approximate. Basically all numerical physics is approximate, at least to some degree.

Task 1A: estimating the accurary of a numerical solution

You should write the difference between the array $M(i)$ and the exact solution, as a function of time, to a file, and plot it to see how big is the error. You should then decrease h (remembering to increase the size of the array). How big is the largest difference between your numerical solution, and the exact solution, for 300 steps and $h = 0.1$ s? How big is the largest difference when you reduce the number of steps to 100 (and so $h = 0.3$ s)?

You should find that the error varies approximately linearly with h. So the smaller is h the more accurate is the solution, but of course the slower the program runs. This is not such a big problem for ODEs, but it is much more serious for PDEs.

Once you have the error for your chosen value of h, it is good idea to write out the estimated accuracy of the solution to screen (just as you would quote error bars in a value you determine in an experiment), for example write to the screen

```
solution for mass estimated to be accurate to 3 decimal places for timestep  = 0.01
```

Note that the error on some points will be bigger than on other points, the error you write to the terminal should be the largest error in any point.

You should also adjust the formatting of writing numbers such as `mass`, `t`, etc to the file, to be around 1 decimal place more than the accuracy. So if the error is estimated to be in the 4th decimal place then a good format would be `f9.5`, which writes numbers to 5 decimal places. As the accuracy will be greater for some points than for others, you will typically want to write numbers with a number of decimal places large enough to be larger than the accuracy of most or all points. However, this will vary a little from one problem to another so some judgement is often needed here.

You can use a program such as 'gnuplot' to plot the data out. Whenever you calculate a function $y(x)$ you should always write it to a file and plot it out, so you can look at it and check that it makes sense. Here we know what the answer is and so can check your answer and code.

gnuplot can be accessed from a terminal by just typing 'gnuplot'. In gnuplot to plot a function type ' pl'filename' ' to plot out a file called 'filename' where this file contains two columns, the first with the x values and the second with the y values. The program has built in help and there are many guides and tutorials that can easily be found on the web using Google. gnuplot will also plot functions so you can compare calculated functions with known functions using gnuplot.

So for example, if you call the file 'mass_of_time.dat ' and the first two columns are the time then mass, then in gnuplot if you type ' pl'mass_of_time.dat', 0.1*exp(-0.3*x) ' this should plot both your solution and the exact result. So you can compare the two and check your program. Note that gnuplot always uses x for the x-axis coordinate.

4 The modified Euler method

The modified Euler method improves on the Euler method by using the fact that a better approximation to the solution $M_f = M(t_f)$ at $t_f = t_s + h$ will be obtained by using the gradient of the function $M(t)$ at the mid-point, $t_m = t_s + h/2$, instead of that at the beginning, t_s. The modified Euler method estimates the solution at the midpoint t_m as

$$\begin{aligned} t_m &= t_s + (h/2) \\ M_m &= M_s + (h/2)M'(t_s) \end{aligned}$$

and then the derivative at the midpoint, M'_m, is obtained using the values of t and of M at the midpoint, t_m and M_m, respectively. The improved estimate for the value of the function at the endpoint $M_f = M(t_f)$ is then

$$M_f = M(t_f) \approx M_s + hM'_m \ .$$

So this method requires additional steps, but is more accurate at a given value of h than the standard Euler method, and so is often preferred.

So one Modified Euler step has two parts. The first is evaluating the new function values at the midpoint

$$\begin{aligned} \texttt{massdash} &= -\texttt{r} * \texttt{mass(i)} \\ \texttt{mass\_mid} &= \texttt{mass(i)} + 0.5 * \texttt{h} * \texttt{massdash} \\ \texttt{massdash\_mid} &= -\texttt{r} * \texttt{mass\_mid} \end{aligned}$$

then using these values at the midpoint we work on the values at the end of the step

$$\begin{aligned} \texttt{t(i+1)} &= \texttt{t(i)} + \texttt{h} \\ \texttt{mass(i+1)} &= \texttt{mass(i)} + \texttt{h} * \texttt{massdash\_mid} \end{aligned}$$

Task 2

Write a FORTRAN program to evaluate the solution of the first-order ordinary differential equation in Task 1, but using the modified Euler method. You should again compute the errors from the exact solution. You will find that at the same value of h the error is smaller for the modified Euler method than for the Euler method.

5 $F = ma$ as an example of a second-order differential equation

Here we will study the second-order ODE we get from a mass m attached to a spring of spring constant k

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -\frac{k}{m}x$$

where we want to solve for the position as a function of time, $x(t)$. This is done by splitting it into two first-order ODEs, using

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v$$

for $v(t)$ the velocity of the particle. Then we have two coupled first-order ODEs

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v \qquad \frac{\mathrm{d}v}{\mathrm{d}t} = -\frac{k}{m}x$$

which we solve simultaneously for $x(t)$ and $v(t)$. To do this we need the values of k and m, as examples we take $k = 5$ N/m and $m = 1$ kg. Then the two ODEs are

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v \qquad \frac{\mathrm{d}v}{\mathrm{d}t} = -5x$$

As we have two first-order ODEs we also need two BCs. Here these are the initial positions and velocities, e.g., $v(t = 0) = 0$ m/s and $x(t = 0) = 0.1$ m.

We start with the Euler method for integrating two coupled ODEs simultaneously. The code for one Euler step is

$$\begin{aligned}
\texttt{dxdt} &= \texttt{v} \\
\texttt{dvdt} &= -5 * \texttt{x} \\
\texttt{t(i+1)} &= \texttt{t(i)} + \texttt{h} \\
\texttt{x(i+1)} &= \texttt{x(i)} + \texttt{h} * \texttt{dxdt} \\
\texttt{v(i+1)} &= \texttt{v(i)} + \texttt{h} * \texttt{dvdt}
\end{aligned}$$

Iterating this integrates the particle's position and velocity in time.

The modified Euler method will require evaluating the values of both derivatives at the midpoint (i.e., you will need additional variables `dxdt_mid` and `dvdt_mid` and will need to obtain x and v at the mid-point to evaluate these two derivatives). See Task 2 and above for how the modified Euler method works.

Task 3

Solve the two coupled ODEs

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v \qquad \frac{\mathrm{d}v}{\mathrm{d}t} = -5x$$

with the two BCs that are the initial positions and velocities: $v(t = 0) = 0$ m/s and $x(t = 0) = 0.1$ m. Solve from $t = 0$ to $t = 10$ s. Use 1000 steps. First use the Euler method, then after you have checked that works, use the modified Euler method. If you compare the two solutions at the same step size you

will see that the solution using the modified Euler method is a lot more accurate. The exact solution of these ODEs with these BCs is known, it is

$$x(t) = 0.1\cos(\sqrt{5}t)$$

Show that the solution is $x(t) = 0.1\cos(\sqrt{5}t)$, by writing out to a file the array you obtain for x by numerically solving these ODEs, as well as the x values given by this function.

Then plot both your calculated x values and the exact x values on the same plot. If your program is working correctly and you have chosen a small enough step length the two curves will be almost on top of each other. You can quantify this by in addition plotting the difference between the exact solution and your numerical solution. How big is the largest difference between your numerical solution, and the exact solution, for 1000 steps and $h = 0.01$ s? How big is the largest difference when you reduce the number of steps to 100 (and so $h = 0.1$ s)?

Assignment 0

Write a Fortran program to evaluate the solution of the two ODEs:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v \qquad \frac{\mathrm{d}v}{\mathrm{d}t} = -5x + 0.5\cos(2t)$$

These equations are for a particle of mass m attached to a spring of spring constant k, with $k/m = 5$ N/(m kg) [these units are equivalent to $1/\mathrm{s}^2$], and subject to an additional periodically oscillating force of magnitude $0.5 \times m \times \cos(2t)$ N [1].

To solve these equations, you need two BCs. Use the initial positions and velocities: $v(t = 0) = 2$ m/s and $x(t = 0) = 0$ m. Use the modified Euler method. Integrate the above two coupled ODEs and hence obtain the position of the particle between the times $t = 0$ and $t = 50$ s. Use $10,001$ points, and so a time step of $50/10^4 = 5 \times 10^{-3}$s.

The program should write to a file values of t, x and v (i.e., three columns, whose numbers need to be formatted). If you write a comment line to the file, start this line with # so that a plotting program (eg gnuplot) knows that this line is a comment not a line of data. The program should also **write the name of this file to the screen**.

R.P. Sear (2007-2017), J.S. Al-Khalili & J.A. Tostevin (pre-2007)

[1] Note that the natural frequency of oscillation of the particle is $\sqrt{5} \simeq 2.23$ so if you change the angular frequency of the applied force from 2 to 2.23 then you should see oscillations that grow larger