Advanced Programming in Engineering
Ordinary Differential Equations (ODE 1)

Problem

Solving ordinary differential equations is an often recurring task in engineering, as in most quantitative sciences. Hence, a wide range of numerical techniques have been developed to calculate approximate solutions to ordinary differential equations (ODE). Several of these methods are briefly described in the hand-outs accompanying the lecture, including a discussion of their strengths and weaknesses. In this exercise, you will program and apply three algorithms to get a feeling of how they perform. It is good practice to try out unfamiliar algorithms on familiar problems, so one can verify the results and assess their accuracy. We here opt for the one-dimensional oscillator.

Background

Consider a body of mass m attached to a spring. As long as the position x relative to the potential minimum is small, the body experiences a potential that is in good approximation quadratic, $U(x) = \frac{1}{2}kx^2$, with k the spring constant. The path $x(t)$ of the body, as a function of time t, obeys the well-known Newtonian equations of motion. These can be posed as one second order ordinary differential equation,

$$\frac{d^2x(t)}{dt^2} = \frac{F(x(t))}{m}, \tag{1}$$

where $F(x) = -dU(x)/dx$ denotes the force on the body. By introducing the velocity $v(t)$ as an auxiliary variable, the above expression can be rewritten as two coupled first order ordinary differential equations,

$$\frac{dx(t)}{dt} = v(t), \tag{2}$$

$$\frac{dv(t)}{dt} = \frac{F(x(t))}{m}. \tag{3}$$

Given the initial conditions $x(0)$ and $v(0)$, the coupled solutions $x(t)$ and $v(t)$ are in principle fully determined.

For the *numerical approximation* of the exact motion, we will make use of two arrays[1] for the calculated positions `x[i]` and velocities `v[i]` of the body at a series of points in time, $t = \mathtt{i}\Delta t$, where Δt denotes the time step and `i` an integer, $\mathtt{0} \leq \mathtt{i} \leq \mathtt{MaxStep}$. The path is calculated, step by step, using a numerical algorithm based on the analytical equations of motion. This requires 'discretization' of these equations; that is, rather than the infinitesimal time steps of differential calculus, we use a finite time step Δt. Numerical algorithms are typically geared to first order ODEs rather than second order ODEs. By discretizing Eqs (2) and (3), we arrive at

$$\mathtt{x[i+1]} = \mathtt{x[i]} + \mathtt{v[i]} * \mathtt{DeltaT} \tag{4}$$

$$\mathtt{v[i+1]} = \mathtt{v[i]} + \mathtt{F(x[i])} * \mathtt{DeltaT/m} \tag{5}$$

This is the first order forward Euler method.

Given: $k = 5\,\mathrm{N/m}$ and $m = 2\,\mathrm{kg}$. You are free to chose all other relevant quantities.

[1] For clarity we will use the notational convention:

- *Italics* for analytical expressions and `typewriter` for (quasi-)program code.
- Numerical quantities always have a value that is equivalent, either exactly or approximately, to that of the matching analytical quantity, e.g. $\mathtt{DeltaT} = \Delta t$ and $\mathtt{x[i]} \approx x(\mathtt{i}\Delta t)$. Note that in some languages arrays start at 1; in those cases, $\mathtt{x[i+1]} \approx x(\mathtt{i}\Delta t)$.
- Functions have their argument between round brackets (...), arrays have their (integer) index between square brackets `[...]`.

Questions

1 Discuss how Eqs (4) and (5) follow from Eqs (2) and (3), and in what limit(s) these discretizations can be applied.

2 The calculation of the trajectory requires a time step Δt and a total simulation length $T = \texttt{MaxStep} * \Delta t$. Make motivated orders of magnitude estimates for both, based on the physical parameters of the problem.

3 It is good practice to think, in advance, of methods to test the correctness of your numerical results.[2] Propose at least one criterion to test the calculated trajectory.

4 Write a program to simulate the oscillator, based on Eqs (4) and (5).

5 Does the program pass the test(s) you proposed in question (3)? Did the test criterion(s) make sense? Explore what happens when you change the time step.

6 Plot the total energy E as a function of time, for various values of Δt. Can you explain our interest in the total energy? Identify the calculated function $E(t)$, and create a clear graph[3] to support your conclusion.

The accuracy of the calculated trajectory improves upon reducing the time step, but this increases the computational costs and does not solve the fundamental flaw of the Euler algorithm. We are fortunate that other algorithms prove much better suited for solving Newton's equations of motion. A workhorse in the field is the leap-frog version of the Verlet algorithm:[4]

$$\texttt{x[i+1]} = \texttt{x[i]} + \texttt{w[i+1]} * \texttt{DeltaT} \quad (6)$$

$$\texttt{w[i+1]} = \texttt{w[i]} + \texttt{F(x[i])} * \texttt{DeltaT/m}. \quad (7)$$

Note the subtle difference with the Euler scheme.

7 Code this algorithm[5] and explore the behaviour of the total energy. One may assume in the energy calculation, and only there, that $v(\texttt{i}\Delta t) \approx \texttt{w[i]}$. Compare your results against those obtained with the Euler scheme. What is the influence of the time step on the performance of the leap-frog scheme?

8 Inspection of Eq. (6) reveals that $\texttt{w[i+1]}$ equals the average velocity in step `i+1`, and hence $\texttt{w[i+1]} \approx v((\texttt{i} + \frac{1}{2})\Delta t)$. The velocity at $t = \texttt{i}\Delta t$ is therefore more accurately calculated as $v(\texttt{i}\Delta t) \approx (\texttt{w[i+1]} + \texttt{w[i]})/2$. Note that the integration algorithm, Eq. (6) and (7), and hence the calculated trajectory, remains unchanged; we merely change the *interpretation* of the contents of the `w` array. Implement the revised calculation of the kinetic energy at time t. How does this change affect the calculated total energy?

9 Add up the Taylor expansions $x((i+1)\Delta t) = x(i\Delta t) + \ldots$ and $x((i-1)\Delta t) = x(i\Delta t) + \ldots$, to arrive at the original algorithm by Verlet (which was not leap-frog).

   a. What conclusion on numerical differentiation can be deduced by comparing this addition against Eq. (1)?

[2] A code that fails the test is clearly incorrect. Note that, depending on the test, your program might pass the test but still be at fault...

[3] Unlike curved lines, straight lines are easy to recognize.

[4] We use `w` to emphasize that the velocity `w[i]` differs from the velocity `v[i]` in the Euler algorithm.

[5] That is, just the two lines for `x` and `w`. There is no `v` in these two lines, so there should be no `v` in your implementation either.

b. Show that the two versions of Verlet's algorithm are equivalent, by solving `w[i]` from Eq. (6) and inserting the result twice into Eq. (7).

c. Extract, by comparing the summed Taylor expansions with Verlet's algorithm, the order of the local truncation error of this algorithm.

The so-called Runge-Kutta algorithms solve an ODE, like $dy/dt = f(y,t)$, by calculating y and f at several points within a time step Δt. The idea is that the extra function evaluations per time step increase the accuracy of the calculation for a given Δt. Reversely, the gain may also be used to increase the time step at a given accuracy.

10 Simulate the harmonic oscillator using matlab's standard solver `ode45`. (You may want to consult the provided brief examples on how to invoke this solver.) Plot the total energy again as a function of time. What strikes you in the time array returned by this algorithm?

11 Which of the three algorithms performs 'best'? Properties to consider include accuracy, computational costs and long-time stability.

Suppose the harmonic oscillator is also subjected to friction, with friction parameter γ, and to a driving force, with amplitude f and frequency ω. This will turn the equation of motion into

$$m\frac{d^2x(t)}{dt^2} = F(x(t)) - \gamma\frac{dx(t)}{dt} + f\cos(\omega t). \tag{8}$$

12 How can we include the friction force in the `ode45` algorithm?
And how can we include the friction force in the leap-frog algorithm? Note that `w[i]` is the mid-step velocity $v((i-\frac{1}{2})\Delta t)$, while Eq. (7) is based on the integer step forces. What consequence does this have for your integration scheme?

13 Include the friction force in the simulation method of your choice. Set the value of the friction parameter such that your oscillator is almost stationary after 10 to 20 oscillations. Explain the shape of the resulting total energy plot $E(t)$.

14 How can we introduce the driving force into the `ode45` and leap-frog algorithms?

15 Simulate the behaviour of the oscillator for several values of f and ω, in combination with the above selected γ. We are interested in the periodic motion of the oscillator after the start-up effects have vanished. Describe what you observe.

Advanced questions (for grade > 8)

a In the presence of friction and a periodic driving force, the amplitude A of the oscillator will become enslaved to the driving force. Use your simulation algorithm to make a plot of $\log(A/f)$ against $\log(\omega)$ for driving frequencies within two orders of magnitude of the eigenfrequency of the oscillator. Compare your result against the theoretical solution of this problem.

b A diatomic molecule (like N_2 or O_2) can be modelled as two particles connected by a harmonic spring, with spring constant k and equilibrium length l. Simulate the three-dimensional motion of this molecule, in the absence of friction and driving forces. Propose and apply new tests, in addition to energy conservation, to validate the correctness of your algorithm.