Chapter 8
Differential equations

If God has made the world a perfect mechanism, he has at least conceded so much to our imperfect intellect that in order to predict little parts of it, we need not solve innumerable differential equations, but can use dice with fair success. Max Born, quoted in H. R. Pagels, The Cosmic Code [40]

Abstract This chapter aims at giving an overview on some of the most used methods to solve ordinary differential equations. Several examples of applications to physical systems are discussed, from the classical pendulum to the physics of Neutron stars.

8.1 Introduction

We may trace the origin of differential equations back to Newton in 16871 and his treatise on the gravitational force and what is known to us as Newton’s second law in dynamics.

Needless to say, differential equations pervade the sciences and are to us the tools by which we attempt to express in a concise mathematical language the laws of motion of nature. We uncover these laws via the dialectics between theories, simulations and experiments, and we use them on a daily basis which spans from applications in engineering or financial engineering to basic research in for example biology, chemistry, mechanics, physics, ecological models or medicine.

We have already met the differential equation for radioactive decay in nuclear physics. Other famous differential equations are Newton’s law of cooling in thermodynamics, the wave equation, Maxwell’s equations in electromagnetism, the heat equation in thermodynamics, Laplace’s equation and Poisson’s equation, Einstein’s field equation in general relativity, Schrödinger equation in quantum mechanics, the Navier-Stokes equations in fluid dynamics, the Lotka-Volterra equation in population dynamics, the Cauchy-Riemann equations in complex analysis and the Black-Scholes equation in finance, just to mention a few. Excellent texts on differential equations and computations are the texts of Eriksson, Estep, Hansbo and Johnson [41], Butcher [42] and Hairer, Nørsett and Wanner [43].

There are five main types of differential equations,

- ordinary differential equations (ODEs), discussed in this chapter for initial value problems only. They contain functions of one independent variable, and derivatives in that variable. The next chapter deals with ODEs and boundary value problems.
- Partial differential equations with functions of multiple independent variables and their partial derivatives, covered in chapter 10.

---

1 Newton had most of the relations for his laws ready 22 years earlier, when according to legend he was contemplating falling apples. However, it took more than two decades before he published his theories, chiefly because he was lacking an essential mathematical tool, differential calculus.
• So-called delay differential equations that involve functions of one dependent variable, derivatives in that variable, and depend on previous states of the dependent variables.
• Stochastic differential equations (SDEs) are differential equations in which one or more of the terms is a stochastic process, thus resulting in a solution which is itself a stochastic process.
• Finally we have so-called differential algebraic equations (DAEs). These are differential equation comprising differential and algebraic terms, given in implicit form.

In this chapter we restrict the attention to ordinary differential equations. We focus on initial value problems and present some of the more commonly used methods for solving such problems numerically. The physical systems which are discussed range from the classical pendulum with non-linear terms to the physics of a neutron star or a white dwarf.

8.2 Ordinary differential equations

In this section we will mainly deal with ordinary differential equations and numerical methods suitable for dealing with them. However, before we proceed, a brief remainder on differential equations may be appropriate.

• The order of the ODE refers to the order of the derivative on the left-hand side in the equation
\[ \frac{dy}{dt} = f(t,y). \]
This equation is of first order and \( f \) is an arbitrary function. A second-order equation goes typically like
\[ \frac{d^2y}{dt^2} = f(t, \frac{dy}{dt}, y). \]
A well-known second-order equation is Newton’s second law
\[ m\frac{d^2x}{dt^2} = -kx, \tag{8.1} \]
where \( k \) is the force constant. ODE depend only on one variable, whereas

• partial differential equations like the time-dependent Schrödinger equation
\[ i\hbar \frac{\partial \psi(x,t)}{\partial t} = \frac{\hbar^2}{2m} \left( \frac{\partial^2 \psi(r,t)}{\partial x^2} + \frac{\partial^2 \psi(r,t)}{\partial y^2} + \frac{\partial^2 \psi(r,t)}{\partial z^2} \right) + V(x)\psi(x,t), \]
may depend on several variables. In certain cases, like the above equation, the wave function can be factorized in functions of the separate variables, so that the Schrödinger equation can be rewritten in terms of sets of ordinary differential equations.

• We distinguish also between linear and non-linear differential equation where e.g.,
\[ \frac{dy}{dt} = g^3(t)y(t), \]
is an example of a linear equation, while
\[ \frac{dy}{dt} = g^3(t)y(t) - g(t)y^2(t), \]
is a non-linear ODE. Another concept which dictates the numerical method chosen for solving an ODE, is that of initial and boundary conditions. To give an example, in our study
of neutron stars below, we will need to solve two coupled first-order differential equations, one for the total mass \(m\) and one for the pressure \(P\) as functions of \(\rho\)

\[
\frac{dm}{dr} = 4\pi r^2 \rho(r)/c^2,
\]

and

\[
\frac{dP}{dr} = -\frac{Gm(r)}{r^2} \rho(r)/c^2.
\]

where \(\rho\) is the mass-energy density. The initial conditions are dictated by the mass being zero at the center of the star, i.e., when \(r = 0\), yielding \(m(r = 0) = 0\). The other condition is that the pressure vanishes at the surface of the star. This means that at the point where we have \(P = 0\) in the solution of the integral equations, we have the total radius \(R\) of the star and the total mass \(m(r = R)\). These two conditions dictate the solution of the equations. Since the differential equations are solved by stepping the radius from \(r = 0\) to \(r = R\), so-called one-step methods (see the next section) or Runge-Kutta methods may yield stable solutions.

In the solution of the Schrödinger equation for a particle in a potential, we may need to apply boundary conditions as well, such as demanding continuity of the wave function and its derivative.

• In many cases it is possible to rewrite a second-order differential equation in terms of two first-order differential equations. Consider again the case of Newton’s second law in Eq. (8.1). If we define the position \(x(t) = y^{(1)}(t)\) and the velocity \(v(t) = y^{(2)}(t)\) as its derivative

\[
\frac{dy^{(1)}(t)}{dt} = \frac{dx(t)}{dt} = y^{(2)}(t),
\]

we can rewrite Newton’s second law as two coupled first-order differential equations

\[
m\frac{dy^{(2)}(t)}{dt} = -kx(t) = -ky^{(1)}(t),
\]

(8.2)

and

\[
\frac{dy^{(1)}(t)}{dt} = y^{(2)}(t).
\]

(8.3)

8.3 Finite difference methods

These methods fall under the general class of one-step methods. The algorithm is rather simple. Suppose we have an initial value for the function \(y(t)\) given by

\[y_0 = y(t = t_0).\]

We are interested in solving a differential equation in a region in space \([a, b]\). We define a step \(h\) by splitting the interval in \(N\) sub intervals, so that we have

\[h = \frac{b - a}{N}.
\]

With this step and the derivative of \(y\) we can construct the next value of the function \(y\) at

\[y_1 = y(t_1 = t_0 + h),\]
and so forth. If the function is rather well-behaved in the domain \([a,b]\), we can use a fixed step size. If not, adaptive steps may be needed. Here we concentrate on fixed-step methods only. Let us try to generalize the above procedure by writing the step \(y_{i+1}\) in terms of the previous step \(y_i\)

\[
y_{i+1} = y(t = t_i + h) = y(t_i) + h \Delta(t_i, y_i(t_i)) + O(h^{p+1}),
\]

where \(O(h^{p+1})\) represents the truncation error. To determine \(\Delta\), we Taylor expand our function \(y\)

\[
y_{i+1} = y(t = t_i + h) = y(t_i) + h \left( y'(t_i) + \cdots + y^{(p)}(t_i) \frac{h^{p-1}}{p!} \right) + O(h^{p+1}), \tag{8.4}
\]

where we will associate the derivatives in the parenthesis with

\[
\Delta(t_i, y_i(t_i)) = \left( y'(t_i) + \cdots + y^{(p)}(t_i) \frac{h^{p-1}}{p!} \right).
\]

We define

\[
y'(t_i) = f(t_i, y_i)
\]

and if we truncate \(\Delta\) at the first derivative, we have

\[
y_{i+1} = y(t_i) + hf(t_i, y_i) + O(h^2), \tag{8.6}
\]

which when complemented with \(t_{i+1} = t_i + h\) forms the algorithm for the well-known Euler method. Note that at every step we make an approximation error of the order of \(O(h^2)\), however the total error is the sum over all steps \(N = (b - a)/h\), yielding thus a global error which goes like \(NO(h^2) \approx O(h)\). To make Euler’s method more precise we can obviously decrease \(h\) (increase \(N\)). However, if we are computing the derivative \(f\) numerically by e.g., the two-steps formula

\[
f_{\Delta h}(x) = \frac{f(x+h) - f(x)}{h} + O(h),
\]

we can enter into roundoff error problems when we subtract two almost equal numbers \(f(x+h) - f(x) \approx 0\). Euler’s method is not recommended for precision calculation, although it is handy to use in order to get a first view how a solution may look like. As an example, consider Newton’s equation rewritten in Eqs. (8.2) and (8.3). We define \(y_0 = y^{(1)}(t = 0)\) an \(v_0 = y^{(2)}(t = 0)\).

The first steps in Newton’s equations are then

\[
y_1^{(1)} = y_0 + hv_0 + O(h^2)
\]

and

\[
y_1^{(2)} = v_0 - hv_0k/m + O(h^2).
\]

The Euler method is asymmetric in time, since it uses information about the derivative at the beginning of the time interval. This means that we evaluate the position at \(y_1^{(1)}\) using the velocity at \(y_0^{(2)} = v_0\). A simple variation is to determine \(y_1^{(1)}\) using the velocity at \(y_{n+1}^{(2)}\), that is (in a slightly more generalized form)

\[
y_{n+1}^{(1)} = y_n^{(1)} + h y_{n+1}^{(2)} + O(h^2)
\]

and

\[
y_{n+1}^{(2)} = y_n^{(2)} + ha_n + O(h^2).
\]

The acceleration \(a_n\) is a function of \(a_n(y_n^{(1)}, y_n^{(2)}, t)\) and needs to be evaluated as well. This is the Euler-Cromer method.

Let us then include the second derivative in our Taylor expansion. We have then
The second derivative can be rewritten as
\[ y'' = f' = \frac{df}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial t} = \frac{\partial f}{\partial y} f \]
and we can rewrite Eq. (8.4) as
\[ y_{i+1} = y(t = t_i + h) = y(t_i) + hf(t_i) + \frac{h^2}{2} \left( \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} f \right) + O(h^3), \]
which has a local approximation error \( O(h^3) \) and a global error \( O(h^2) \). These approximations can be generalized by using the derivative \( f \) to arbitrary order so that we have
\[ y_{i+1} = y(t = t_i + h) = y(t_i) + hf(t_i, y_i) + \cdots + f^{(p-1)}(t_i, y_i) \frac{h^{p-1}}{p!} + O(h^p). \]
These methods, based on higher-order derivatives, are in general not used in numerical computation, since they rely on evaluating derivatives several times. Unless one has analytical expressions for these, the risk of roundoff errors is large.

### 8.3.1 Improvements of Euler’s algorithm, higher-order methods

The most obvious improvements to Euler’s and Euler-Cromer’s algorithms, avoiding in addition the need for computing a second derivative, is the so-called midpoint method. We have then
\[ y_{n+1}^{(1)} = y_n^{(1)} + \frac{h}{2} \left( y_n^{(2)} + y_n^{(2)} \right) + O(h^2) \]
and
\[ y_{n+1}^{(2)} = y_n^{(2)} + ha_n + O(h^3), \]
yielding
\[ y_{n+1}^{(1)} = y_n^{(1)} + hy_n^{(2)} + \frac{h^2}{2} a_n + O(h^3) \]
implies that the local truncation error in the position is now \( O(h^3) \), whereas Euler’s or Euler-Cromer’s methods have a local error of \( O(h^2) \). Thus, the midpoint method yields a global error with second-order accuracy for the position and first-order accuracy for the velocity. However, although these methods yield exact results for constant accelerations, the error increases in general with each time step.

One method that avoids this is the so-called half-step method. Here we define
\[ y_{n+1/2}^{(2)} = y_{n-1/2}^{(2)} + ha_n + O(h^2), \]
and
\[ y_{n+1}^{(1)} = y_n^{(1)} + hy_{n+1/2}^{(2)} + O(h^2). \]
Note that this method needs the calculation of \( y_{1/2}^{(2)} \). This is done using for example Euler’s method
\[ y_{1/2}^{(2)} = y_0^{(2)} + \frac{h}{2} a_0 + O(h^2). \]
As this method is numerically stable, it is often used instead of Euler’s method. Another method which one may encounter is the Euler-Richardson method with

\[ y_{n+1}^{(2)} = y_n^{(2)} + ha_{n+1/2} + O(h^2), \]

and

\[ y_{n+1}^{(1)} = y_n^{(1)} + hy_{n+1/2}^{(2)} + O(h^2). \]

### 8.3.2 Verlet and Leapfrog algorithms

Another set of popular algorithms, which are both numerically stable and easy to implement are the Verlet and Leapfrog algorithms. These algorithms are much used in so-called Molecular Dynamics applications, see for example Refs. [44, 45]. Consider again a second-order differential equation like Newton’s second law, whose one-dimensional version reads

\[
m \frac{d^2x}{dt^2} = F(x,t),
\]

which we rewrite in terms of two coupled differential equations

\[
\frac{dx}{dt} = v(x,t) \quad \text{and} \quad \frac{dv}{dt} = F(x,t)/m = a(x,t).
\]

If we now perform a Taylor expansion

\[
x(t+h) = x(t) + hx^{(1)}(t) + \frac{h^2}{2} x^{(2)}(t) + O(h^3).
\]

In our case the second derivative is known via Newton’s second law, namely \( x^{(2)} = a(x,t) \). If we add to the above equation the corresponding Taylor expansion for \( x(t-h) \), we obtain, using the discretized expressions

\[
x_i \pm h = x_{i \pm 1} \quad \text{and} \quad x_i = x(t_i),
\]

\[
x_{i+1} = 2x_i - x_{i-1} + h^2 x_i^{(2)} + O(h^4).
\]

We note that the truncation error goes like \( O(h^4) \) since all the odd terms cancel when we add the two Taylor expansions. We see also that the velocity is not directly included in the equation since the function \( x^{(2)} = a(x,t) \) is supposed to be known. If we need the velocity however, we can compute it using the well-known formula

\[
x_{i}^{(1)} = \frac{x_{i+1} - x_{i-1}}{2h} + O(h^2).
\]

We note that the velocity has a truncation error which goes like \( O(h^2) \). In for example so-called Molecular dynamics calculations, since the acceleration is normally known via Newton’s second law, there is seldomly a need for computing the velocity. The above sets of equations for the position \( x(t) \) and the velocity defines the Verlet formula. The Leapfrog algorithm is also easily derived.

We can rewrite the above Taylor expansion for \( x(t+h) \) as

\[
x(t+h) = x(t) + h \left( x^{(1)}(t) + \frac{h}{2} x^{(2)}(t) \right) + O(h^3).
\]
8.3 Finite difference methods

Noting that
\[ x^{(1)}(t + h/2) = \left( x^{(1)}(t) + \frac{h}{2} x^{(2)}(t) \right) + O(h^2), \]
we obtain
\[ x(t + h) = x(t) + h + x^{(1)}(t + h/2) + O(h^3), \]
which needs to be combined with
\[ x^{(1)}(t + h/2) = x^{(1)}(t - h/2) + hx^{(2)}(t) + O(h^2). \]
Again, there is a lower truncation error in \( h \) for the velocity. Furthermore, the positions and the velocities are evaluated at different time steps. If one needs \( x^{(1)}(t_0) \), this can be computed using
\[ x^{(1)}(t) = \left( x^{(1)}(t \mp h/2) \pm \frac{h}{2} x^{(2)}(t) \right) + O(h^2). \]

The initial conditions can be handled in similar ways and the inaccuracy which arises between \( x^{(1)}(0) \) and \( x^{(1)}(h/2) \) is normally ignored. Summarizing, the popular Leapfrog algorithm implies the evaluation of position and velocity at different time steps. The final algorithm is given by the following steps
\[ x^{(1)}(t + h/2) = x^{(1)}(t - h/2) + hx^{(2)}(t) + O(h^2), \]
which is used in
\[ x(t + h) = x(t) + h + x^{(1)}(t + h/2) + O(h^3), \]
and finally
\[ x^{(1)}(t + h) = x^{(1)}(t + h/2) + \frac{h}{2} x^{(2)}(t + h) + O(h^3), \]

8.3.3 Predictor-Corrector methods

Consider again the first-order differential equation
\[ \frac{dy}{dt} = f(t, y), \]
which solved with Euler’s algorithm results in the following algorithm
\[ y_{i+1} \approx y(t_i) + hf(t_i, y_i) \]
with \( t_{i+1} = t_i + h \). This means geometrically that we compute the slope at \( y_i \) and use it to predict \( y_{i+1} \) at a later time \( t_{i+1} \). We introduce \( k_1 = f(t_i, y_i) \) and rewrite our prediction for \( y_{i+1} \) as
\[ y_{i+1} \approx y(t_i) + hk_1. \]
We can then use the prediction \( y_{i+1} \) to compute a new slope at \( t_{i+1} \) by defining \( k_2 = f(t_{i+1}, y_{i+1}) \). We define the new value of \( y_{i+1} \) by taking the average of the two slopes, resulting in
\[ y_{i+1} \approx y(t_i) + \frac{h}{2} (k_1 + k_2). \]
The algorithm is very simple, namely
1. Compute the slope at \( t_i \), that is define the quantity \( k_1 = f(t_i, y_i) \).
2. Make a prediction for the solution by computing \( y_{i+1} \approx y(t_i) + hk_1 \) by Euler’s method.
3. Use the prediction \( y_{i+1} \) to compute a new slope at \( t_{i+1} \) defining the quantity \( k_2 = f(t_{i+1}, y_{i+1}) \).
4. Correct the value of \( y_{i+1} \) by taking the average of the two slopes yielding \( y_{i+1} \approx y(t_i) + \frac{h}{2}(k_1 + k_2) \).

It can be shown [24] that this procedure results in a mathematical truncation which goes like \( O(h^3) \), to be contrasted with Euler’s method which runs as \( O(h) \). One additional function evaluation yields a better error estimate.

This simple algorithm conveys the philosophy of a large class of methods called predictor-corrector methods, see chapter 15 of Ref. [36] for additional algorithms. A simple extension is obviously to use Simpson’s method to approximate the integral
\[
y_{i+1} = y_i + \int_{t_i}^{t_{i+1}} f(t, y) dt,
\]
when we solve the differential equation by successive integrations. The next section deals with a particular class of efficient methods for solving ordinary differential equations, namely various Runge-Kutta methods.

### 8.4 More on finite difference methods, Runge-Kutta methods

Runge-Kutta (RK) methods are based on Taylor expansion formulae, but yield in general better algorithms for solutions of an ODE. The basic philosophy is that it provides an intermediate step in the computation of \( y_{i+1} \).

To see this, consider first the following definitions
\[
\frac{dy}{dt} = f(t, y),
\]
and
\[
y(t) = \int f(t, y) dt,
\]
and
\[
y_{i+1} = y_i + \int_{t_i}^{t_{i+1}} f(t, y) dt.
\]

To demonstrate the philosophy behind RK methods, let us consider the second-order RK method, RK2. The first approximation consists in Taylor expanding \( f(t, y) \) around the center of the integration interval \( t_i \) to \( t_{i+1} \), i.e., at \( t_i + h/2 \), \( h \) being the step. Using the midpoint formula for an integral, defining \( y(t_i + h/2) = y_{i+1/2} \) and \( t_i + h/2 = t_{i+1/2} \), we obtain
\[
\int_{t_i}^{t_{i+1}} f(t, y) dt \approx hf(t_{i+1/2}, y_{i+1/2}) + O(h^3).
\]

This means in turn that we have
\[
y_{i+1} = y_i + hf(t_{i+1/2}, y_{i+1/2}) + O(h^3).
\]
However, we do not know the value of \( y_{i+1/2} \). Here comes thus the next approximation, namely, we use Euler’s method to approximate \( y_{i+1/2} \). We have then
\[
y_{i+1/2} = y_i + \frac{h}{2} \frac{dy}{dt} = y(t_i) + \frac{h}{2} f(t_i, y_i).
\]

This means that we can define the following algorithm for the second-order Runge-Kutta method, RK2.
\[
k_1 = hf(t_i, y_i),
\]
\[
k_2 = hf(t_{i+1/2}, y_i + k_1/2),
\]
with the final value
\[
y_{i+1} \approx y_i + k_2 + O(h^2).
\]

The difference between the previous one-step methods is that we now need an intermediate step in our evaluation, namely \( t_i + h/2 = t_{i+1/2} \) where we evaluate the derivative \( f \). This involves more operations, but the gain is a better stability in the solution. The fourth-order Runge-Kutta, RK4, which we will employ in the solution of various differential equations below, is easily derived. The steps are as follows. We start again with the equation
\[
y_{i+1} = y_i + \int_{t_i}^{t_{i+1}} f(t, y)dt,
\]

but instead of approximating the integral with the midpoint rule, we use now Simpson’s rule at \( t_i + h/2 \), \( h \) being the step. Using Simpson’s formula for an integral, defining \( y(t_i + h/2) = y_{i+1/2} \) and \( t_i + h/2 = t_{i+1/2} \), we obtain
\[
\int_{t_i}^{t_{i+1}} f(t, y)dt \approx \frac{h}{6} \left[ f(t_i, y_i) + 4f(t_{i+1/2}, y_{i+1/2}) + f(t_{i+1}, y_{i+1}) \right] + O(h^2).
\]

This means in turn that we have
\[
y_{i+1} = y_i + \frac{h}{6} \left[ f(t_i, y_i) + 4f(t_{i+1/2}, y_{i+1/2}) + f(t_{i+1}, y_{i+1}) \right] + O(h^2).
\]

However, we do not know the values of \( y_{i+1/2} \) and \( y_{i+1} \). The fourth-order Runge-Kutta method splits the midpoint evaluations in two steps, that is we have
\[
y_{i+1} \approx y_i + \frac{h}{6} \left[ f(t_i, y_i) + 2f(t_{i+1/2}, y_{i+1/2}) + 2f(t_{i+1/2}, y_{i+1/2}) + f(t_{i+1}, y_{i+1}) \right],
\]
since we want to approximate the slope at \( y_{i+1/2} \) in two steps. The first two function evaluations are as for the second order Runge-Kutta method. The algorithm is as follows

1. We compute first
   \[
k_1 = hf(t_i, y_i),
   \]
   which is nothing but the slope at \( t_i \). If we stop here we have Euler’s method.
2. Then we compute the slope at the midpoint using Euler’s method to predict \( y_{i+1/2} \), as in the second-order Runge-Kutta method. This leads to the computation of
   \[
k_2 = hf(t_i + h/2, y_i + k_1/2).
   \]
3. The improved slope at the midpoint is used to further improve the slope of \( y_{i+1/2} \) by computing
\[ k_3 = hf(t_i + h/2, y_i + k_2/2). \]

4. With the latter slope we can in turn predict the value of \( y_{i+1} \) via the computation of

\[ k_4 = hf(t_i + h, y_i + k_3). \]

5. The final algorithm becomes then

\[ y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \]

Thus, the algorithm consists in first calculating \( k_1 \) with \( t_i, y_i \) and \( f \) as inputs. Thereafter, we increase the step size by \( h/2 \) and calculate \( k_2 \), then \( k_3 \) and finally \( k_4 \). With this caveat, we can then obtain the new value for the variable \( y \). It results in four function evaluations, but the accuracy is increased by two orders compared with the second-order Runge-Kutta method. The fourth order Runge-Kutta method has a global truncation error which goes like \( O(h^4) \).

Fig. 8.1 gives a geometrical interpretation of the fourth-order Runge-Kutta method.

\[ y \]

\[ t_i \quad t_i + h/2 \quad t_i + h \quad t \]

**Fig. 8.1** Geometrical interpretation of the fourth-order Runge-Kutta method. The derivative is evaluated at four points, once at the initial point, twice at the trial midpoint and once at the trial endpoint. These four derivatives constitute one Runge-Kutta step resulting in the final value for \( y_{i+1} = y_i + 1/6(k_1 + 2k_2 + 2k_3 + k_4) \).
In case the function to integrate varies slowly or fast in different integration domains, adaptive methods are normally used. One strategy is always to decrease the step size. As we have seen earlier, this leads to more computations and may eventually even lead to the loss of numerical precision. An alternative is to use higher-order Runge-Kutta methods for example. However, this leads again to more cycles, furthermore, there is no guarantee that higher-order leads to an improved error, see for example the discussions in Ref. [42].

Assume the exact result is $\tilde{y}$ and that we are using a Runge-Kutta method of order $M$. Suppose we run two calculations, one with a step length $h$ (which we will label $y_1$) and one with step length $h/2$ (labelled $y_2$). The exact solution in terms of $y_1$ is

$$\tilde{y} = y_1 + Ch^{M+1} + O(h^{M+2}),$$

where $C$ is some constant and

$$\tilde{y} = y_2 + 2C(h/2)^{M+1} + O(h^{M+2}).$$

Note that we need to perform two calculations in the last equation, one for each interval defined by $h/2$. Calculate two halves in the last equation. The difference between the two solutions is then

$$|y_1 - y_2| = Ch^{M+1}(1 - \frac{1}{2^M}),$$

from which we can define the constant $C$ as

$$C = \frac{|y_1 - y_2|}{(1 - \frac{1}{2^M})h^{M+1}}.$$ (8.14)

We rewrite then the exact solution in terms of a quantity $\varepsilon$

$$\tilde{y} = y_2 + \varepsilon + O((h)^{M+2}),$$

with

$$\varepsilon = \frac{|y_1 - y_2|}{2^M - 1}.$$ 

If we employ our fourth-order Runge-Kutta scheme, we have

$$\tilde{y} = y_2 + \varepsilon + O(h^6),$$

with

$$\varepsilon = \frac{|y_1 - y_2|}{15}.$$ 

The estimate is one order higher than the original Runge-Kutta method to fourth order. But this method is normally rather inefficient since it requires a lot of computations. We solve typically the equation three times at each time step. However, we can compare the estimate $\varepsilon$ with some by us given accuracy $\xi$ say for example $\xi = 10^{-8}$. We can then ask the following question: what is, with a given $y_j$ and $t_j$, the largest possible step size $\tilde{h}$ that leads to an error below $\xi$? We want

$$Ch^{M+1} \leq \xi,$$

which leads to, using Eq. (8.14),

$$\left(\frac{\tilde{h}}{h}\right)^{M+1} \frac{|y_1 - y_2|}{(1 - \frac{1}{2^M})} \leq \xi,$$
meaning that we can define this optimal step length as

\[ \tilde{h} = h \left( \frac{\xi}{\varepsilon} \right)^{1/(M+1)} \].

Using this equation, we can design the following algorithm:

- If the two answers are close, use the current value for the step length \( h \).
- If \( \varepsilon > \xi \) we need to decrease the step size in the next time step.
- If \( \varepsilon < \xi \) we need to increase the step size in the next time step.

At each step, two different approximations for the solution are made and compared. If the two answers are in close agreement, the approximation is accepted. If the two answers do not agree to a specified accuracy, the step size is reduced. If the answers agree to more significant digits than required, the step size is increased. Even though this algorithm is rather simple to implement, it requires unnecessarily many computations.

It is possible to reduce the number of operations by combining Runge-Kutta algorithms of different orders. A much used algorithm is the so-called Runge-Kutta-Fehlberg algorithm which uses a combination of fourth and fifth order Runge-Kutta methods, normally abbreviated to RKF45. Without going into much details, the philosophy of such methods consists in evaluating the function \( f \) such that the function values can be used for both the fourth order and the fifth order method, avoiding thereby additional computations. The RKF45 method requires at each step the computations of the following six values

\[
\begin{align*}
    k_1 &= hf(t_k, y_k), \\
    k_2 &= hf(t_k + \frac{1}{4}h, y_k + \frac{1}{4}k_1), \\
    k_3 &= hf(t_k + \frac{3}{8}h, y_k + \frac{3}{32}k_1 + \frac{9}{32}k_2), \\
    k_4 &= hf(t_k + \frac{12}{13}h, y_k + \frac{1932}{2197}k_1 + \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3), \\
    k_5 &= hf(t_k + \frac{16}{27}h, y_k + \frac{6656}{12825}k_1 - \frac{28561}{56430}k_2 + \frac{9602}{105}k_3 + \frac{363}{50}k_4), \\
    k_6 &= hf(t_k + \frac{1}{2}h, y_k - \frac{8}{27}k_1 + 2k_2 - \frac{355}{112}k_3 + \frac{10}{21}k_4 - \frac{1}{28}k_5).
\end{align*}
\]

Then an approximation to the solution of the ordinary differential equation is made using a Runge-Kutta method of order four:

\[
y_{k+1} = y_k + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4101}k_4 - \frac{1}{5}k_5,
\]

where the four function values \( k_1, k_3, k_4, \) and \( k_5 \) are used. Notice that \( k_2 \) is not used here. A better value for the solution is determined using a Runge-Kutta method of order five as follows

\[
z_{k+1} = y_k + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6.
\]

The optimal time step \( \alpha h \) is then determined by

\[
\alpha = \left( \frac{\xi h}{2|z_{k+1} - y_{k+1}|} \right)^{1/4}.
\]
with \( \xi \) our defined tolerance. For more details behind the derivation of this method, see for example Ref. [42].

## 8.6 Physics examples

### 8.6.1 Ideal harmonic oscillations

Our first example is the classical case of simple harmonic oscillations, namely a block sliding on a horizontal frictionless surface. The block is tied to a wall with a spring, portrayed in e.g., Fig. 8.2. If the spring is not compressed or stretched too far, the force on the block at a given position \( x \) is

\[
F = -kx.
\]

![Diagram of block tied to a wall with a spring tension acting on it.](image)

**Fig. 8.2** Block tied to a wall with a spring tension acting on it.

The negative sign means that the force acts to restore the object to an equilibrium position. Newton’s equation of motion for this idealized system is then

\[
m \frac{d^2x}{dt^2} = -kx,
\]

or we could rephrase it as

\[
\frac{d^2x}{dt^2} = -\frac{k}{m}x = -\omega_0^2 x, \tag{8.15}
\]

with the angular frequency \( \omega_0^2 = k/m \).

The above differential equation has the advantage that it can be solved analytically with solutions on the form

\[
x(t) = Acos(\omega_0 t + \nu),
\]

where \( A \) is the amplitude and \( \nu \) the phase constant. This provides in turn an important test for the numerical solution and the development of a program for more complicated cases which cannot be solved analytically.
As mentioned earlier, in certain cases it is possible to rewrite a second-order differential equation as two coupled first-order differential equations. With the position $x(t)$ and the velocity $v(t) = dx/dt$ we can reformulate Newton’s equation in the following way

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

and

$$\frac{dv(t)}{dt} = -\omega_0^2 x(t).$$

We are now going to solve these equations using the Runge-Kutta method to fourth order discussed previously. Before proceeding however, it is important to note that in addition to the exact solution, we have at least two further tests which can be used to check our solution.

Since functions like $\cos$ are periodic with a period $2\pi$, then the solution $x(t)$ has also to be periodic. This means that

$$x(t + T) = x(t),$$

with $T$ the period defined as

$$T = \frac{2\pi}{\omega_0} = \frac{2\pi}{\sqrt{k/m}}.$$

Observe that $T$ depends only on $k/m$ and not on the amplitude of the solution or the constant $\nu$.

In addition to the periodicity test, the total energy has also to be conserved. Suppose we choose the initial conditions

$$x(t = 0) = 1 \text{ m} \quad v(t = 0) = 0 \text{ m/s},$$

meaning that block is at rest at $t = 0$ but with a potential energy

$$E_0 = \frac{1}{2}kx(t = 0)^2 = \frac{1}{2}k.$$

The total energy at any time $t$ has however to be conserved, meaning that our solution has to fulfill the condition

$$E_0 = \frac{1}{2}kx(t)^2 + \frac{1}{2}mv(t)^2.$$

An algorithm which implements these equations is included below.

1. Choose the initial position and speed, with the most common choice $v(t = 0) = 0$ and some fixed value for the position. Since we are going to test our results against the periodicity requirement, it is convenient to set the final time equal $t_f = 2\pi$, where we choose $k/m = 1$. The initial time is set equal to $t_i = 0$. You could alternatively read in the ratio $k/m$.

2. Choose the method you wish to employ in solving the problem. In the enclosed program we have chosen the fourth-order Runge-Kutta method. Subdivide the time interval $[t_i, t_f]$ into a grid with step size

$$h = \frac{t_f - t_i}{N},$$

where $N$ is the number of mesh points.

3. Calculate now the total energy given by
and use this when checking the numerically calculated energy from the Runge-Kutta iterations.
4. The Runge-Kutta method is used to obtain $x_{i+1}$ and $v_{i+1}$ starting from the previous values $x_i$ and $v_i$.
5. When we have computed $x(v)_{i+1}$ we upgrade $t_{i+1} = t_i + h$.
6. This iterative process continues till we reach the maximum time $t_f = 2\pi$.
7. The results are checked against the exact solution. Furthermore, one has to check the stability of the numerical solution against the chosen number of mesh points $N$.

### 8.6.1.1 Program to solve the differential equations for a sliding block

The program which implements the above algorithm is presented here, with a corresponding

```cpp
/* This program solves Newton’s equation for a block sliding on a horizontal frictionless surface. The block is tied to a wall with a spring, and Newton’s equation takes the form $m \ddot{x} = -kx$ with $k$ the spring tension and $m$ the mass of the block. The angular frequency is $\omega = \sqrt{k/m}$ and we set it equal 1 in this example program.

Newton’s equation is rewritten as two coupled differential equations, one for the position $x$ and one for the velocity $v$
$$\frac{dx}{dt} = v \quad \text{and} \quad \frac{dv}{dt} = -x$$
when we set $k/m=1$

We use therefore a two-dimensional array to represent $x$ and $v$ as functions of $t$
$$y[0] = x \quad y[1] = v \quad \frac{dy[0]}{dt} = v \quad \frac{dy[1]}{dt} = -x$$

The derivatives are calculated by the user defined function derivatives.

The user has to specify the initial velocity (usually $v_0=0$) the number of steps and the initial position. In the programme below we fix the time interval $[a,b]$ to $[0,2\pi]$.

*/
#include <cmath>
#include <iostream>
#include <fstream>
#include <iomanip>
#include "lib.h"

using namespace std;

ofstream ofile;

void derivatives(double, double*, double*); void initialise ( double&, double&, int&);
void output( double, double*, double);

int main(int argc, char* argv[]) {
    // declarations of variables
    double *y, *dydt, *yout, t, h, tmax, E0; double initial_x, initial_v; int i, number_of_steps, n; char *outfilename;
    // Read in output file, abort if there are too few command-line arguments
    if ( argc <= 1 ) { cout << "Bad Usage: " << argv[0] << " read also output file on same line" << endl; exit(1); }
    else { outfilename=argv[1]; ofile.open(outfilename); // this is the number of differential equations $n = 2$; // allocate space in
```
memory for the arrays containing the derivatives
(double[n]; y = new double[n]; yout = new double[n]; // read in
the initial position, velocity and number of steps initialise
(initial.x, initial.v, number_of_steps); // setting initial
values, step size and max time tmax h = 4.*acos(-1.)/( (double)
number_of_steps); // the step size tmax = h*number_of_steps; //
the final time y[0] = initial.x; // initial position y[1] =
initial.v; // initial velocity t=0.; // initial time E0 =
0.5*y[0]*y[0]+0.5*y[1]*y[1]; // the initial total energy // now
we start solving the differential equations using the RK4 method
while (t <= tmax){ derivatives(t, y, dydt); // initial
derivatives runge_kutta_4(y, dydt, n, t, h, yout, derivatives);
for (i = 0; i < n; i++) { y[i] = yout[i]; } t += h; output(t,
y, E0); // write to file } delete [] y; delete [] dydt; delete
[] yout; ofile.close(); // close output file return 0; } // End
of main function

// Read in from screen the number of steps, // initial position and
initial speed void initialise (double& initial_x, double&
initial_v, int& number_of_steps)
{ cout << "Initial position = "; cin >> initial_x; cout << "Initial speed = "; cin >> initial_v;
cout << "Number of steps = "; cin >> number_of_steps; }
// end of function initialise
// this function sets up the derivatives for this special case void
derivatives(double t, double *y, double *dydt)
{ dydt[0]=y[1]; // derivative of x
dydt[1]=-y[0]; // derivative of v
}
// end of function derivatives
// function to write out the final results void output(double t,
double *y, double E0) { ofile << setiosflags(ios::showpoint |
ios::uppercase); ofile << setw(15) << setprecision(8) << t;
ofile << setw(15) << setprecision(8) << y[0]; ofile << setw(15) <<
setprecision(8) << y[1]; ofile << setw(15) << setprecision(8) <<
cos(t); ofile << setw(15) << setprecision(8) << 0.5*y[0]*y[0]+0.5*y[1]*y[1]-E0 << endl; } // end of function
output

/* This function upgrades a function y (input as a pointer) and
returns the result yout, also as a pointer. Note that these
variables are declared as arrays. It also receives as input the
starting value for the derivatives in the pointer dydx. It receives
also the variable n which represents the number of differential
equations, the step size h and the initial value of x. It receives
also the name of the function *derivs where the given derivative is
computed */ void runge_kutta_4(double *y, double *dydx, int n,
double x, double h, double *yout, void (*derivs)(double, double *,
double *)) { int i; double xh, hh, h6; double *dym, *dyt, *yt; //allocate space for local vectors dym = new double [n]; dyt = new
double [n]; yt = new double [n]; hh = h/2.0; h6 = h/6.; xh =
x+hh; for (i = 0; i < n; i++) { yt[i] = y[i]+hh*dydx[i]; } (*derivs)(xh,yt,dyt); // computation of k2, eq. 3.60 for (i=0;
i < n; i++) { yt[i] = y[i]+hh*dyt[i]; } (*derivs)(xh,yt,dyt); //
computation of k3, eq. 3.61 for (i=0; i < n; i++) { yt[i] =
y[i]+h*dym[i]; dym[i] += dyt[i]; } (*derivs)(x+h,yt,dyt); //
computation of k4, eq. 3.62 // now we upgrade y in the array yout
for (i = 0; i < n; i++) { yout[i] =
y[i]+h6*(dydx[i]+dyt[i]+2.0*dym[i]); } delete []dym; delete []
dyt; delete [] yt; } // end of function Runge-kutta 4
In Fig. 8.3 we exhibit the development of the difference between the calculated energy and the exact energy at $t = 0$ after two periods and with $N = 1000$ and $N = 10000$ mesh points. This figure demonstrates clearly the need of developing tests for checking the algorithm used. We see that even for $N = 1000$ there is an increasing difference between the computed energy and the exact energy after only two periods.

*Fig. 8.3* Plot of $\Delta E(t) = E_0 - E_{\text{computed}}$ for $N = 1000$ and $N = 10000$ time steps up to two periods. The initial position $x_0 = 1$ m and initial velocity $v_0 = 0$ m/s. The mass and spring tension are set to $k = m = 1$. 
8.6.2 Damping of harmonic oscillations and external forces

Most oscillatory motion in nature does decrease until the displacement becomes zero. We call such a motion for damped and the system is said to be dissipative rather than conservative. Considering again the simple block sliding on a plane, we could try to implement such a dissipative behavior through a drag force which is proportional to the first derivative of \( x \), i.e., the velocity. We can then expand Eq. (8.15) to

\[
\frac{d^2x}{dt^2} = -\omega_0^2x - \nu \frac{dx}{dt},
\]

where \( \nu \) is the damping coefficient, being a measure of the magnitude of the drag term.

We could however counteract the dissipative mechanism by applying e.g., a periodic external force

\[ F(t) = B\cos(\omega t), \]

and we rewrite Eq. (8.16) as

\[
\frac{d^2x}{dt^2} = -\omega_0^2x - \nu \frac{dx}{dt} + F(t). \tag{8.17}
\]

Although we have specialized to a block sliding on a surface, the above equations are rather general for quite many physical systems.

If we replace \( x \) by the charge \( Q \), \( \nu \) with the resistance \( R \), the velocity with the current \( I \), the inductance \( L \) with the mass \( m \), the spring constant with the inverse capacitance \( C \) and the force \( F \) with the voltage drop \( V \), we rewrite Eq. (8.17) as

\[
L\frac{d^2Q}{dt^2} + \frac{Q}{C} + R\frac{dQ}{dt} = V(t). \tag{8.18}
\]

The circuit is shown in Fig. 8.4.

![Fig. 8.4 Simple RLC circuit with a voltage source \( V \).](image)

How did we get there? We have defined an electric circuit which consists of a resistance \( R \) with voltage drop \( IR \), a capacitor with voltage drop \( Q/C \) and an inductor \( L \) with voltage drop \( LdI/dt \). The circuit is powered by an alternating voltage source and using Kirchhoff’s law, which is a consequence of energy conservation, we have

\[ V(t) = IR + LdI/dt + Q/C, \]

and using
we arrive at Eq. (8.18).

This section was meant to give you a feeling of the wide range of applicability of the methods we have discussed. However, before leaving this topic entirely, we’ll delve into the problems of the pendulum, from almost harmonic oscillations to chaotic motion!

### 8.6.3 The pendulum, a nonlinear differential equation

Consider a pendulum with mass $m$ at the end of a rigid rod of length $l$ attached to say a fixed frictionless pivot which allows the pendulum to move freely under gravity in the vertical plane as illustrated in Fig. 8.5.

![Fig. 8.5 A simple pendulum.](image)

The angular equation of motion of the pendulum is again given by Newton’s equation, but now as a nonlinear differential equation

$$ml\frac{d^2\theta}{dt^2} + mg\sin(\theta) = 0,$$

with an angular velocity and acceleration given by

$$v = l\frac{d\theta}{dt},$$

and

$$a = l\frac{d^2\theta}{dt^2}.$$

For small angles, we can use the approximation
\[ \sin(\theta) \approx \theta. \]

and rewrite the above differential equation as
\[ \frac{d^2 \theta}{dt^2} = -\frac{g}{l} \theta, \]

which is exactly of the same form as Eq. (8.15). We can thus check our solutions for small values of \( \theta \) against an analytical solution. The period is now
\[ T = \frac{2\pi}{\sqrt{l/g}}. \]

We do however expect that the motion will gradually come to an end due a viscous drag torque acting on the pendulum. In the presence of the drag, the above equation becomes
\[ ml \frac{d^2 \theta}{dt^2} + \nu \frac{d \theta}{dt} + mg \sin(\theta) = 0, \]

where \( \nu \) is now a positive constant parameterizing the viscosity of the medium in question. In order to maintain the motion against viscosity, it is necessary to add some external driving force. We choose here, in analogy with the discussion about the electric circuit, a periodic driving force. The last equation becomes then
\[ ml \frac{d^2 \theta}{dt^2} + \nu \frac{d \theta}{dt} + mg \sin(\theta) = A \cos(\omega t), \tag{8.19} \]

with \( A \) and \( \omega \) two constants representing the amplitude and the angular frequency respectively. The latter is called the driving frequency.

If we now define the natural frequency
\[ \omega_0 = \sqrt{g/l}, \]

the so-called natural frequency and the new dimensionless quantities
\[ \hat{t} = \omega_0 t, \]

with the dimensionless driving frequency
\[ \hat{\omega} = \frac{\omega}{\omega_0}, \]

and introducing the quantity \( Q \), called the quality factor,
\[ Q = \frac{mg}{\omega_0 \nu}, \]

and the dimensionless amplitude
\[ \hat{A} = \frac{A}{mg} \]

we can rewrite Eq. (8.19) as
\[ \frac{d^2 \theta}{d\hat{t}^2} + \frac{1}{Q} \frac{d \theta}{d\hat{t}} + \sin(\theta) = \hat{A} \cos(\hat{\omega} \hat{t}). \]

This equation can in turn be recast in terms of two coupled first-order differential equations as follows
8.7 Physics Project: the pendulum

8.7.1 Analytic results for the pendulum

Although the solution to the equations for the pendulum can only be obtained through numerical efforts, it is always useful to check our numerical code against analytic solutions. For small angles $\theta$, we have $\sin(\theta) \approx \theta$ and our equations become

$$\frac{d\theta}{dt} = \dot{\theta},$$

and

$$\frac{d\dot{\theta}}{dt} = -\frac{\dot{\theta}}{Q} - \sin(\theta) + \hat{A}\cos(\hat{\omega}t).$$

These are the equations to be solved. The factor $Q$ represents the number of oscillations of the undriven system that must occur before its energy is significantly reduced due to the viscous drag. The amplitude $\hat{A}$ is measured in units of the maximum possible gravitational torque while $\hat{\omega}$ is the angular frequency of the external torque measured in units of the pendulum’s natural frequency.

$$\frac{d\theta}{dt} = \dot{\theta},$$

and

$$\frac{d\dot{\theta}}{dt} = -\frac{\dot{\theta}}{Q} - \theta + \hat{A}\cos(\hat{\omega}t).$$

These equations are linear in the angle $\theta$ and are similar to those of the sliding block or the RLC circuit. With given initial conditions $\theta_0$ and $\dot{\theta}_0$ they can be solved analytically to yield

$$\theta(t) = \left[ \theta_0 - \frac{\hat{A}(1-\hat{\omega}^2)}{(1-\hat{\omega}^2)^2 + \hat{\omega}^2/Q^2} \right] e^{-\tau/2Q} \cos(\sqrt{1-\frac{1}{4Q^2}} \tau)$$

$$+ \left[ \dot{\theta}_0 + \frac{\hat{A}(1-\hat{\omega}^2)/2Q}{(1-\hat{\omega}^2)^2 + \hat{\omega}^2/Q^2} \right] e^{-\tau/2Q} \sin(\sqrt{1-\frac{1}{4Q^2}} \tau) + \frac{\hat{A}(1-\hat{\omega}^2)\cos(\hat{\omega}t) + \hat{\omega}^2\sin(\hat{\omega}t)}{(1-\hat{\omega}^2)^2 + \hat{\omega}^2/Q^2},$$

and

$$\dot{\theta}(t) = \left[ \dot{\theta}_0 - \frac{\hat{A}\hat{\omega}^2/Q}{(1-\hat{\omega}^2)^2 + \hat{\omega}^2/Q^2} \right] e^{-\tau/2Q} \cos(\sqrt{1-\frac{1}{4Q^2}} \tau)$$

$$- \left[ \theta_0 + \frac{\dot{\theta}_0 + \hat{A}(1-\hat{\omega}^2)/2Q}{(1-\hat{\omega}^2)^2 + \hat{\omega}^2/Q^2} \right] e^{-\tau/2Q} \sin(\sqrt{1-\frac{1}{4Q^2}} \tau) + \frac{\hat{\omega}^2(1-\hat{\omega}^2)\sin(\hat{\omega}t) + \hat{\omega}\cos(\hat{\omega}t)}{(1-\hat{\omega}^2)^2 + \hat{\omega}^2/Q^2},$$

with $Q > 1/2$. The first two terms depend on the initial conditions and decay exponentially in time. If we wait long enough for these terms to vanish, the solutions become independent of the initial conditions and the motion of the pendulum settles down to the following simple orbit in phase space

$$\theta(t) = \frac{\hat{A}(1-\hat{\omega}^2)\cos(\hat{\omega}t) + \hat{\omega}^2\sin(\hat{\omega}t)}{(1-\hat{\omega}^2)^2 + \hat{\omega}^2/Q^2},$$

and

$$\dot{\theta}(t) = \frac{\hat{\omega}\hat{A}(1-\hat{\omega}^2)\sin(\hat{\omega}t) + \hat{\omega}\cos(\hat{\omega}t)}{(1-\hat{\omega}^2)^2 + \hat{\omega}^2/Q^2},$$

 tracing the closed phase-space curve.
for a time interval $(8.8)$ and finally the fourth-order Runge-Kutta scheme RK4. 

Irrespective of the initial conditions, the trajectory in $\theta$ Fig. 8.6 forms an ellipse whose principal axes are $\theta$ and $\dot{\theta}$. This curve is closed, as we will see from the examples below, implying that the motion is periodic in time, the solution repeats itself exactly after each period $T = \frac{2\pi}{\dot{\omega}}$. Before we discuss results for various frequencies, quality factors and amplitudes, it is instructive to compare different numerical methods. In Fig. 8.6 we show the angle $\theta$ as function of time $\tau$ for the case with $Q = 2$, $\dot{\omega} = 2/3$ and $\dot{A} = 0.5$. The length is set equal to 1 m and mass of the pendulum is set equal to 1 kg. The initial velocity is $\dot{\theta}_0 = 0$ and $\theta_0 = 0.01$. Four different methods have been used to solve the equations, Euler’s method from Eq. (8.6), Euler-Richardson’s method in Eqs. (8.7)-(8.8) and finally the fourth-order Runge-Kutta scheme RK4. We note that after few time steps, we obtain the classical harmonic motion. We would have obtained a similar picture if we were to switch off the external force, $\dot{A} = 0$ and set the frictional damping to zero, i.e., $Q = 0$. Then, the qualitative picture is that of an idealized harmonic oscillation without damping. However, we see that Euler’s method performs poorly and after a few steps its algorithmic simplicity leads to results which deviate considerably from the other methods. In the discussion hereafter we will thus limit ourselves to present results obtained with the fourth-order Runge-Kutta method.

The corresponding phase space plot is shown in Fig. 8.7, for the same parameters as in Fig. 8.6. We observe here that the plot moves towards an ellipse with periodic motion. This stable phase-space curve is called a periodic attractor. It is called attractor because, irrespective of the initial conditions, the trajectory in phase-space tends asymptotically to

$$
\left(\frac{\theta}{\bar{A}}\right)^2 + \left(\frac{\dot{\theta}}{\dot{\bar{A}}}\right)^2 = 1
$$

with

$$
\bar{A} = \frac{\dot{\bar{A}}}{\sqrt{(1 - \dot{\omega})^2 + \dot{\omega}^2/Q^2}}.
$$

This curve forms an ellipse whose principal axes are $\theta$ and $\dot{\theta}$. This curve is closed, as we will see from the examples below, implying that the motion is periodic in time, the solution repeats itself exactly after each period $T = \frac{2\pi}{\dot{\omega}}$. Before we discuss results for various frequencies, quality factors and amplitudes, it is instructive to compare different numerical methods. In Fig. 8.6 we show the angle $\theta$ as function of time $\tau$ for the case with $Q = 2$, $\dot{\omega} = 2/3$ and $\dot{A} = 0.5$. The length is set equal to 1 m and mass of the pendulum is set equal to 1 kg. The initial velocity is $\dot{\theta}_0 = 0$ and $\theta_0 = 0.01$. Four different methods have been used to solve the equations, Euler’s method from Eq. (8.6), Euler-Richardson’s method in Eqs. (8.7)-(8.8) and finally the fourth-order Runge-Kutta scheme RK4. We note that after few time steps, we obtain the classical harmonic motion. We would have obtained a similar picture if we were to switch off the external force, $\dot{A} = 0$ and set the frictional damping to zero, i.e., $Q = 0$. Then, the qualitative picture is that of an idealized harmonic oscillation without damping. However, we see that Euler’s method performs poorly and after a few steps its algorithmic simplicity leads to results which deviate considerably from the other methods. In the discussion hereafter we will thus limit ourselves to present results obtained with the fourth-order Runge-Kutta method.

The corresponding phase space plot is shown in Fig. 8.7, for the same parameters as in Fig. 8.6. We observe here that the plot moves towards an ellipse with periodic motion. This stable phase-space curve is called a periodic attractor. It is called attractor because, irrespective of the initial conditions, the trajectory in phase-space tends asymptotically to
such a curve in the limit $\tau \to \infty$. It is called periodic, since it exhibits periodic motion in time, as seen from Fig. 8.6. In addition, we should note that this periodic motion shows what we call resonant behavior since the driving frequency of the force approaches the natural frequency of oscillation of the pendulum. This is essentially due to the fact that we are studying a linear system, yielding the well-known periodic motion. The non-linear system exhibits a much richer set of solutions and these can only be studied numerically.

![Phase-space curve of a linear damped pendulum with $Q = 2$, $\dot{\omega} = 2/3$ and $\hat{A} = 0.5$. The initial velocity is $\dot{v}_0 = 0$ and $\theta_0 = 0.01$.](image)

In order to go beyond the well-known linear approximation we change the initial conditions to say $\theta_0 = 0.3$ but keep the other parameters equal to the previous case. The curve for $\theta$ is shown in Fig. 8.8. The corresponding phase-space curve is shown in Fig. 8.9. This curve demonstrates that with the above given sets of parameters, after a certain number of periods, the phase-space curve stabilizes to the same curve as in the previous case, irrespective of initial conditions. However, it takes more time for the pendulum to establish a periodic motion and when a stable orbit in phase-space is reached the pendulum moves in accordance with the driving frequency of the force. The qualitative picture is much the same as previously. The phase-space curve displays again a final periodic attractor.

If we now change the strength of the amplitude to $\hat{A} = 1.35$ we see in Fig. 8.10 that $\theta$ as function of time exhibits a rather different behavior from Fig. 8.8, even though the initial conditions and all other parameters except $\hat{A}$ are the same. The phase-space curve is shown in Fig. 8.11.

We will explore these topics in more detail in Exercise 8.2 below, where we extend our discussion to the phenomena of period doubling and its link to chaotic motion.

### 8.7.2 The pendulum code

The program used to obtain the results discussed above is presented here. The enclosed code solves the pendulum equations for any angle $\theta$ with an external force $A \cos(\omega t)$. It employs
Fig. 8.8 Plot of $\theta$ as function of time $\tau$ with $Q = 2$, $\hat{\omega} = 2/3$ and $\hat{A} = 0.5$. The mass of the pendulum is set equal to 1 kg and its length to 1 m. The initial velocity is $\hat{v}_0 = 0$ and $\theta_0 = 0.3$.

Fig. 8.9 Phase-space curve with $Q = 2$, $\hat{\omega} = 2/3$ and $\hat{A} = 0.5$. The mass of the pendulum is set equal to 1 kg and its length $l = 1$ m. The initial velocity is $\hat{v}_0 = 0$ and $\theta_0 = 0.3$. 
Fig. 8.10 Plot of $\theta$ as function of time $\tau$ with $Q = 2$, $\dot{\omega} = 2/3$ and $\dot{A} = 1.35$. The mass of the pendulum is set equal to 1 kg and its length to 1 m. The initial velocity is $\dot{\theta}_0 = 0$ and $\theta_0 = 0$.3. Every time $\theta$ passes the value $\pm \pi$ we reset its value to swing between $\theta \in [-\pi, \pi]$. This gives the vertical jumps in amplitude.

Fig. 8.11 Phase-space curve after 10 periods with $Q = 2$, $\dot{\omega} = 2/3$ and $\dot{A} = 1.35$. The mass of the pendulum is set equal to 1 kg and its length $l = 1$ m. The initial velocity is $\dot{\theta}_0 = 0$ and $\theta_0 = 0.3$. 
several methods for solving the two coupled differential equations, from Euler’s method to adaptive size methods coupled with fourth-order Runge-Kutta. It is straightforward to apply this program to other systems which exhibit harmonic oscillations or change the functional form of the external force.

We have also introduced a class where we define various methods for solving ordinary and coupled first order differential equations. This is done via the class pendulum. This methods access variables which belong only to this particular class via the private declaration. As such, the methods we list here can easily be reused by other types of ordinary differential equations. In the code below, we list only the fourth order Runge Kutta method, which was used to generate the above figures. For the full code see programs/chapter08/program2.cpp.

http://folk.uio.no/mhjensen/compphys/programs/chapter08/cpp/program2.cpp

```cpp
#include <stdio.h> include <iostream.h> include <math.h> include
#include <fstream.h> /*
Different methods for solving ODEs are presented We are solving the following equation:

\[ m \cdot l \cdot (\phi)'' + \text{viscosity} \cdot (\phi)' + m \cdot g \cdot \sin(\phi) = A \cdot \cos(\omega_0 t) \]

If you want to solve similar equations with other values you have to rewrite the methods 'derivatives' and 'initialise' and change the variables in the private part of the class Pendulum

At first we rewrite the equation using the following definitions:

\[ \omega_0 = \sqrt{g \cdot l} \quad t_{\text{roof}} = \omega_0 \cdot t \quad \omega_{\text{roof}} = \omega_0 / (m \cdot g) \quad Q = (m \cdot g) / (\omega_0 \cdot \omega_{\text{roof}}) \quad A_{\text{roof}} = A / (m \cdot g) \]

and we get a dimensionless equation

\[ (\phi)'' + 1/Q \cdot (\phi)' + \sin(\phi) = A_{\text{roof}} \cdot \cos(\omega_{\text{roof}} \cdot t_{\text{roof}}) \]

This equation can be written as two equations of first order:

\[ (\phi)' = v (v)' = -v/Q - \sin(\phi) + A_{\text{roof}} \cdot \cos(\omega_{\text{roof}} \cdot t_{\text{roof}}) \]

All numerical methods are applied to the last two equations. The algorithms are taken from the book "An introduction to computer simulation methods" */

class pendulum { private: double Q, A_roof, omega_0, omega_roof, g;
    // double y[2]; //for the initial-values of phi and v int n; // how many steps double delta_t, delta_t_roof; // Definition of methods to solve ODEs public: void derivatives(double, double*, double*); void initialise(); void euler(); void euler_cromer(); void midpoint(); void euler_richardson(); void half_step(); void rk2();
    //runge-kutta-second-order void rk4(double, double*, double*, double); // we need it in function rk4() and asc() void rk4(); //runge-kutta-fourth-order void asc(); //runge-kutta-fourth-order with adaptive stepsize control ;

    // This function defines the particular coupled first order ODEs void pendulum::derivatives(double t, double* in, double* out) { /* Here we are calculating the derivatives at (dimensionless) time t 'in' are the values of phi and v, which are used for the calculation The results are given to 'out' */
        out[0]=in[1]; //out[0] = (phi)' = v if(Q)
Here we define all input parameters.

```c++
void pendelum::initialise()
// double m,l,omega,A,omega_0,v_0,t_end; cout<"Solving the differential eqation of the pendulum!\n"; cout<"The initial conditions at t=0 are Aa n do m e g a n " ;c o u t < < " F u r t h e r m o r e t h e r ei sav i s c o u sd r a g 

```


8.8 Exercises

8.1. In the pendulum example we rewrote the equations as two differential equations in terms of so-called dimensionless variables. One should always do that. There are at least two good reasons for doing this.

- By rewriting the equations as dimensionless ones, the program will most likely be easier to read, with hopefully a better possibility of spotting eventual errors. In addition, the various constants which are pulled out of the equations in the process of rendering the equations dimensionless, are reintroduced at the end of the calculation. If one of these constants is not correctly defined, it is easier to spot an eventual error.
- In many physics applications, variables which enter a differential equation, may differ by orders of magnitude. If we were to insist on not using dimensionless quantities, such differences can cause serious problems with respect to loss of numerical precision.

An example which demonstrates these features is the set of equations for gravitational equilibrium of a neutron star. We will not solve these equations numerically here, rather, we will limit ourselves to merely rewriting these equations in a dimensionless form.

The equations for a neutron star

The discovery of the neutron by Chadwick in 1932 prompted Landau to predict the existence of neutron stars. The birth of such stars in supernovae explosions was suggested by Baade and Zwicky 1934. First theoretical neutron star calculations were performed by Tolman, Oppenheimer and Volkoff in 1939 and Wheeler around 1960. Bell and Hewish were the first to discover a neutron star in 1967 as a radio pulsar. The discovery of the rapidly rotating Crab pulsar (rapidly rotating neutron star) in the remnant of the Crab supernova observed by the Chinese in 1054 A.D. confirmed the link to supernovae. Radio pulsars are rapidly rotating with periods in the range $0.033 \, \mathrm{s} \leq P \leq 4.0 \, \mathrm{s}$. They are believed to be powered by rotational energy loss and are rapidly spinning down with period derivatives of order $\dot{P} \sim 10^{-12} - 10^{-16}$. Their high magnetic field $B$ leads to dipole magnetic braking radiation proportional to the magnetic field squared. One estimates magnetic fields of the order of $B \sim 10^{11} - 10^{13} \, \mathrm{G}$. The total number of pulsars discovered so far has just exceeded 1000 before the turn of the millenium and the number is increasing rapidly.

The physics of compact objects like neutron stars offers an intriguing interplay between nuclear processes and astrophysical observables, see Refs. [46–48] for further information and references on the physics of neutron stars. Neutron stars exhibit conditions far from those encountered on earth; typically, expected densities $\rho$ of a neutron star interior are of the order of $10^3$ or more times the density $\rho_d \approx 4 \cdot 10^{11} \, \mathrm{g/cm}^3$ at ‘neutron drip’, the density at which nuclei begin to dissolve and merge together. Thus, the determination of an equation of state (EoS) for dense matter is essential to calculations of neutron star properties. The EoS determines properties such as the mass range, the mass-radius relationship, the crust thickness and the cooling rate. The same EoS is also crucial in calculating the energy released in a supernova explosion.
Clearly, the relevant degrees of freedom will not be the same in the crust region of a neutron star, where the density is much smaller than the saturation density of nuclear matter, and in the center of the star, where density is so high that models based solely on interacting nucleons are questionable. Neutron star models including various so-called realistic equations of state result in the following general picture of the interior of a neutron star. The surface region, with typical densities $\rho < 10^6 \text{g/cm}^3$, is a region in which temperatures and magnetic fields may affect the equation of state. The outer crust for $10^6 \text{g/cm}^3 < \rho < 2 \cdot 10^{11} \text{g/cm}^3$ is a solid region where a Coulomb lattice of heavy nuclei coexist in $\beta$-equilibrium with a relativistic degenerate electron gas. The inner crust for $2 \cdot 10^{11} \text{g/cm}^3 < \rho < 4 \cdot 10^{11} \text{g/cm}^3$ consists of a lattice of neutron-rich nuclei together with a superfluid neutron gas and an electron gas. The neutron liquid for $4 \cdot 10^{11} \text{g/cm}^3 < \rho < 10^{15} \text{g/cm}^3$ contains mainly superfluid neutrons with a smaller concentration of superconducting protons and normal electrons. At higher densities, typically $2 - 3$ times nuclear matter saturation density, interesting phase transitions from a phase with just nucleonic degrees of freedom to quark matter may take place. Furthermore, one may have a mixed phase of quark and nuclear matter, kaon or pion condensates, hyperonic matter, strong magnetic fields in young stars etc.

**Equilibrium equations**

If the star is in thermal equilibrium, the gravitational force on every element of volume will be balanced by a force due to the spacial variation of the pressure $P$. The pressure is defined by the equation of state (EoS), recall e.g., the ideal gas $P = Nk_B T$. The gravitational force which acts on an element of volume at a distance $r$ is given by

$$F_{\text{Grav}} = -\frac{Gm}{r^2} \rho / c^2,$$

where $G$ is the gravitational constant, $\rho(r)$ is the mass density and $m(r)$ is the total mass inside a radius $r$. The latter is given by

$$m(r) = \frac{4\pi r^2}{c^2} \int_0^r \rho(r') r'^2 dr'$$

which gives rise to a differential equation for mass and density

$$\frac{dm}{dr} = 4\pi r^2 \rho(r) / c^2.$$

When the star is in equilibrium we have

$$\frac{dP}{dr} = -\frac{Gm(r)}{r^2} \rho(r) / c^2.$$

The last equations give us two coupled first-order differential equations which determine the structure of a neutron star when the EoS is known.

The initial conditions are dictated by the mass being zero at the center of the star, i.e., when $r = 0$, we have $m(r = 0) = 0$. The other condition is that the pressure vanishes at the surface of the star. This means that at the point where we have $P = 0$ in the solution of the differential equations, we get the total radius $R$ of the star and the total mass $m(r = R)$. The mass-energy density when $r = 0$ is called the central density $\rho_c$. Since both the final mass $M$ and total radius $R$ will depend on $\rho_c$, a variation of this quantity will allow us to study stars with different masses and radii.
**Dimensionless equations**

When we now attempt the numerical solution, we need however to rescale the equations so that we deal with dimensionless quantities only. To understand why, consider the value of the gravitational constant \( G \) and the possible final mass \( m = M_0 \). The latter is normally of the order of some solar masses \( M_0 \), with \( M_0 = 1.989 \times 10^{30} \text{ Kg} \). If we wish to translate the latter into units of MeV/c\(^2\), we will have that \( M_0 \sim 10^{60} \text{ MeV/c}^2 \). The gravitational constant is in units of \( G = 6.67 \times 10^{-45} \times h c \text{ (MeV/c}^2\text{)}^{-2} \). It is then easy to see that including the relevant values for these quantities in our equations will most likely yield large numerical roundoff errors when we add a huge number \( dP/dr \) to a smaller number \( P \) in order to obtain the new pressure. We list here the units of the various quantities and in case of physical constants, also their values. A bracketed symbol like \([\hat{P}]\) stands for the unit of the quantity inside the brackets.

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>([\hat{P}])</td>
<td>MeVfm(^{-3})</td>
</tr>
<tr>
<td>([\hat{\rho}])</td>
<td>MeVfm(^{-3})</td>
</tr>
<tr>
<td>([\hat{\rho}])</td>
<td>fm(^{-3})</td>
</tr>
<tr>
<td>([\hat{m}])</td>
<td>MeVc(^{-2})</td>
</tr>
<tr>
<td>(M_0)</td>
<td>(1.989 \times 10^{30} \text{ Kg} = 1.1157467 \times 10^{60} \text{ MeVc}^{-2})</td>
</tr>
<tr>
<td>1 Kg</td>
<td>(= 10^{30}/1.78266270D0 \text{ MeVc}^{-2})</td>
</tr>
<tr>
<td>([\hat{r}])</td>
<td>m</td>
</tr>
<tr>
<td>(G)</td>
<td>(hc6.67259 \times 10^{-45} \text{ MeV}^{-2}c^{-4})</td>
</tr>
<tr>
<td>(hc)</td>
<td>197.327 MeVfm</td>
</tr>
</tbody>
</table>

\(\hat{r} = r/R_0\), mass-energy density \(\hat{\rho} = \rho/\rho_s\), pressure \(\hat{P} = P/\rho_s\) and mass \(\hat{m} = m/M_0\).

The constants \(M_0\) and \(R_0\) can be determined from the requirements that the equations for \(d\hat{m}/d\hat{r}\) and \(d\hat{\rho}/d\hat{r}\) should be dimensionless. This gives

\[
\frac{dM_0\hat{m}}{dR_0\hat{r}} = 4\pi R_0^2\hat{r}^2 \hat{\rho}_s \hat{\rho},
\]

yielding

\[
\frac{d\hat{m}}{d\hat{r}} = 4\pi R_0^3\hat{r}^2 \hat{\rho}_s \hat{\rho} / M_0.
\]

If these equations should be dimensionless we must demand that

\[
4\pi R_0^3\rho_s / M_0 = 1.
\]

Correspondingly, we have for the pressure equation

\[
\frac{d\rho_s\hat{P}}{dR_0\hat{r}} = -GM_0 \frac{\hat{m}\rho_s \hat{\rho}}{R_0^2\hat{r}^2}
\]

and since this equation should also be dimensionless, we will have

\[
GM_0/R_0 = 1.
\]

This means that the constants \(R_0\) and \(M_0\) which will render the equations dimensionless are given by

\[
R_0 = \frac{1}{\sqrt{\hat{\rho}_s G4\pi}},
\]
and

\[ M_0 = \frac{4\pi \rho_s}{(\sqrt{\rho_s G^4 \pi})^3}. \]

However, since we would like to have the radius expressed in units of 10 km, we should multiply \( R_0 \) by \( 10^{-19} \), since 1 fm = \( 10^{-15} \) m. Similarly, \( M_0 \) will come in units of MeV/c\(^2\), and it is convenient therefore to divide it by the mass of the sun and express the total mass in terms of solar masses \( M_\odot \).

The differential equations read then

\[ \frac{d\hat{P}}{dr} = -\frac{\hat{n}\hat{\rho}}{r^2}, \quad \frac{d\hat{n}}{dr} = \hat{r}^2 \hat{\rho}. \]

In the solution of our problem, we will assume that the mass-energy density is given by a simple parametrization from Bethe and Johnson [49]. This parametrization gives \( \rho \) as a function of the number density \( n = N/V \), with \( N \) the total number of baryons in a volume \( V \). It reads

\[ \rho(n) = 236 \times n^{2.54} + nm_n, \quad (8.20) \]

where \( m_n = 938.926 \text{MeV/c}^2 \), the mass of the neutron (averaged). This means that since \( [n] = \text{fm}^{-3} \), we have that the dimension of \( \rho \) is \([\rho] = \text{MeV/c}^2\text{fm}^{-3} \). Through the thermodynamic relation

\[ P = -\frac{\partial E}{\partial V}, \quad (8.21) \]

where \( E \) is the energy in units of MeV/c\(^2\) we have

\[ P(n) = n \frac{\partial \rho(n)}{\partial n} - \rho(n) = 363.44 \times n^{2.54}. \]

We see that the dimension of pressure is the same as that of the mass-energy density, i.e., \([P] = \text{MeV/c}^2\text{fm}^{-3} \).

Here comes an important point you should observe when solving the two coupled first-order differential equations. When you obtain the new pressure given by

\[ P_{\text{new}} = \frac{dP}{dr} + P_{\text{old}}, \]

this comes as a function of \( r \). However, having obtained the new pressure, you will need to use Eq. (8.1) in order to find the number density \( n \). This will in turn allow you to find the new value of the mass-energy density \( \rho(n) \) at the relevant value of \( r \).

In solving the differential equations for neutron star equilibrium, you should proceed as follows

1. Make first a dimensional analysis in order to be sure that all equations are really dimensionless.
2. Define the constants \( R_0 \) and \( M_0 \) in units of 10 km and solar mass \( M_\odot \). Find their values. Explain why it is convenient to insert these constants in the final results and not at each intermediate step.
3. Set up the algorithm for solving these equations and write a main program where the various variables are defined.
4. Write thereafter a small function which uses the expressions for pressure and mass-energy density from Eqs. (8.1) and (8.20).
5. Write then a function which sets up the derivatives

\[ -\frac{\hat{m}\hat{\rho}}{r^2}, \quad \hat{r}^2 \hat{\rho}. \]
6. Employ now the fourth order Runge-Kutta algorithm to obtain new values for the pressure and the mass. Play around with different values for the step size and compare the results for mass and radius.

7. Replace the fourth order Runge-Kutta method with the simple Euler method and compare the results.

8. Replace the non-relativistic expression for the derivative of the pressure with that from General Relativity (GR), the so-called Tolman-Oppenheimer-Volkov equation

\[
\frac{d\dot{P}}{d\tau} = -\frac{(\dot{P} + \ddot{P})(\dot{P}^2 + \dot{\Pi})}{\dot{P}^2 - 2\dot{\Pi}},
\]

and solve again the two differential equations.

9. Compare the non-relativistic and the GR results by plotting mass and radius as functions of the central density.

8.2.

\[
ml\frac{d^2\theta}{dt^2} + mgsin(\theta) = 0,
\]

with an angular velocity and acceleration given by

\[
v = \frac{d\theta}{dt},
\]

and

\[
a = \frac{d^2\theta}{dt^2}.
\]

We do however expect that the motion will gradually come to an end due a viscous drag torque acting on the pendulum. In the presence of the drag, the above equation becomes

\[
ml\frac{d^2\theta}{dt^2} + \nu\frac{d\theta}{dt} + mgsin(\theta) = 0, \tag{8.22}
\]

where \(\nu\) is now a positive constant parameterizing the viscosity of the medium in question. In order to maintain the motion against viscosity, it is necessary to add some external driving force. We choose here a periodic driving force. The last equation becomes then

\[
ml\frac{d^2\theta}{dt^2} + \nu\frac{d\theta}{dt} + mgsin(\theta) = Asin(\omega t), \tag{8.23}
\]

with \(A\) and \(\omega\) two constants representing the amplitude and the angular frequency respectively. The latter is called the driving frequency.

1. Rewrite Eqs. (8.22) and (8.23) as dimensionless equations.

2. Write then a code which solves Eq. (8.22) using the fourth-order Runge Kutta method. Perform calculations for at least ten periods with \(N = 100\), \(N = 1000\) and \(N = 10000\) mesh points and values of \(\nu = 1\), \(\nu = 5\) and \(\nu = 10\). Set \(l = 1.0\) m, \(g = 1\) m/s\(^2\) and \(m = 1\) kg. Choose as initial conditions \(\theta(0) = 0.2\) (radians) and \(v(0) = 0\) (radians/s). Make plots of \(\theta\) (in radians) as function of time and phase space plots of \(\theta\) versus the velocity \(v\). Check the stability of your results as functions of time and number of mesh points. Which case corresponds to damped, underdamped and overdamped oscillatory motion? Comment your results.

3. Now we switch to Eq. (8.23) for the rest of the project. Add an external driving force and set \(l = g = 1\), \(m = 1\), \(v = 1/2\) and \(\omega = 2/3\). Choose as initial conditions \(\theta(0) = 0.2\) and \(v(0) = 0\) and \(A = 0.5\) and \(A = 1.2\). Make plots of \(\theta\) (in radians) as function of time for at least 300 periods and phase space plots of \(\theta\) versus the velocity \(v\). Choose an appropriate time step. Comment and explain the results for the different values of \(A\).
4. Keep now the constants from the previous exercise fixed but set now \( A = 1.35, A = 1.44 \) and \( A = 1.465 \). Plot \( \theta \) (in radians) as function of time for at least 300 periods for these values of \( A \) and comment your results.

5. We want to analyse further these results by making phase space plots of \( \theta \) versus the velocity \( v \) using only the points where we have \( \omega t = 2n\pi \) where \( n \) is an integer. These are normally called the drive periods. This is an example of what is called a Poincare section and is a very useful way to plot and analyze the behavior of a dynamical system. Comment your results.

8.3. We assume that the orbit of Earth around the Sun is co-planar, and we take this to be the \( xy \)-plane. Using Newton’s second law of motion we get the following equations

\[
\frac{d^2x}{dt^2} = \frac{F_{Gx}}{M_{\text{Earth}}},
\]

and

\[
\frac{d^2y}{dt^2} = \frac{F_{Gy}}{M_{\text{Earth}}},
\]

where \( F_{Gx} \) and \( F_{Gy} \) are the \( x \) and \( y \) components of the gravitational force.

a) Rewrite the above second-order ordinary differential equations as a set of coupled first order differential equations. Write also these equations in terms of dimensionless variables. As an alternative to the usage of dimensionless variables, you could also use so-called astronomical units (AU as abbreviation). If you choose the latter set of units, one astronomical unit of length, known as 1 AU, is the average distance between the Sun and Earth, that is 1 AU = \( 1.5 \times 10^{11} \) m. It can also be convenient to use years instead of seconds since years match better the solar system. The mass of the Sun is \( M_{\text{Sun}} = M_\odot = 2 \times 10^{30} \) kg. The mass of Earth is \( M_{\text{Earth}} = 6 \times 10^{24} \) kg. The mass of other planets like Jupiter is \( M_{\text{Jupiter}} = 1.9 \times 10^{27} \) kg and its distance to the Sun is 5.20 AU. Similar numbers for Mars are \( M_{\text{Mars}} = 6.6 \times 10^{23} \) kg and 1.52 AU, for Venus \( M_{\text{Venus}} = 4.9 \times 10^{24} \) kg and 0.72 AU, for Saturn are \( M_{\text{Saturn}} = 5.5 \times 10^{26} \) kg and 9.54 AU, for Mercury are \( M_{\text{Mercury}} = 2.4 \times 10^{23} \) kg and 0.39 AU, for Uranus are \( M_{\text{Uranus}} = 8.8 \times 10^{25} \) kg and 19.19 AU, for Neptun are \( M_{\text{Neptun}} = 1.03 \times 10^{26} \) kg and 30.06 AU and for Pluto are \( M_{\text{Pluto}} = 1.31 \times 10^{22} \) kg and 39.53 AU. Pluto is no longer considered a planet, but we add it here for historical reasons.

Finally, mass units can be obtained by using the fact that Earth’s orbit is almost circular around the Sun. For circular motion we know that the force must obey the following relation

\[
F_G = \frac{M_{\text{Earth}} v^2}{r} = \frac{G M_\odot M_{\text{Earth}}}{r^2},
\]

where \( v \) is the velocity of Earth. The latter equation can be used to show that

\[
v^2r = GM_\odot = 4\pi^2 \text{AU}^3/\text{yr}^2.
\]

Discretize the above differential equations and set up an algorithm for solving these equations using the so-called Euler-Cromer.

b) Write then a program which solves the above differential equations for the Earth-Sun system using the Euler-Cromer method. Find out which initial value for the velocity that gives a circular orbit and test the stability of your algorithm as function of different time steps \( \Delta t \). Find a possible maximum value \( \Delta t \) for which the Euler-Cromer method does not yield stable results. Make a plot of the results you obtain for the position of Earth (plot the \( x \) and \( y \) values) orbiting the Sun.
Check also for the case of a circular orbit that both the kinetic and the potential energies are constants. Check also that the angular momentum is a constant. Explain why these quantities are conserved.

c) Modify your code by implementing the fourth-order Runge-Kutta method and compare the stability of your results by repeating the steps in b). Compare the stability of the two methods, in particular as functions of the needed step length $\Delta t$. Comment your results.

d) Kepler’s second law states that the line joining a planet to the Sun sweeps out equal areas in equal times. Modify your code so that you can verify Kepler’s second law for the case of an elliptical orbit. Compare both the Runge-Kutta method and the Euler-Cromer method and check that the total energy and angular momentum are conserved. Why are these quantities conserved? A convenient choice of starting values are an initial position of 1 AU and an initial velocity of 5 AU/yr.

e) Consider then a planet which begins at a distance of 1 AU from the sun. Find out by trial and error what the initial velocity must be in order for the planet to escape from the sun. Can you find an exact answer?

f) We will now study the three-body problem, still with the Sun kept fixed at the center but including Jupiter (the most massive planet in the solar system, having a mass that is approximately 1000 times smaller than that of the Sun) together with Earth. This leads us to a three-body problem. Without Jupiter, Earth’s motion is stable and unchanging with time. The aim here is to find out how much Jupiter alters Earth’s motion. The program you have developed can easily be modified by simply adding the magnitude of the force between Earth and Jupiter. This force is given again by

$$F_{\text{Earth-Jupiter}} = \frac{G M_{\text{Jupiter}} M_{\text{Earth}}}{r_{\text{Earth-Jupiter}}^2},$$

where $M_{\text{Jupiter}}$ is the mass of the Sun and $M_{\text{Earth}}$ is the mass of Earth. The gravitational constant is $G$ and $r_{\text{Earth-Jupiter}}$ is the distance between Earth and Jupiter.

We assume again that the orbits of the two planets are co-planar, and we take this to be the $xy$-plane. Modify your first-order differential equations in order to accommodate both the motion of Earth and Jupiter by taking into account the distance in $x$ and $y$ between Earth and Jupiter. Set up the algorithm and plot the positions of Earth and Jupiter using the fourth-order Runge-Kutta method. Include an adaptive solver to your Runge-Kutta method, using for example the adaptive scheme proposed by Fehlberg. Discuss the stability of the solutions using the standard Runge-Kutta4 solver and the adaptive scheme.

Repeat the calculations by increasing the mass of Jupiter by a factor of 10 and 1000 and plot the position of Earth. Study again the stability of the standard and the adaptive Runge-Kutta solvers.

g) Finally, using your optimal Runge-Kutta solver, we carry out a real three-body calculation where all three systems, Earth, Jupiter and the Sun are in motion. To do this, choose the center-of-mass position of the three-body system as the origin rather than the position of the sun. Give the sun an initial velocity which makes the total momentum of the system exactly zero (the center-of-mass will remain fixed). Compare these results with those from the previous exercise and comment your results. Extend your program to include all planets in the solar system (if you have time, you can also include the various moons, but it is not required) and discuss your results. Try to find data for the initial positions and velocities for all planets.

h) The perihelion precession of Mercury. This part is optional but gives you an additional 30% on the final score!

An important test of the general theory of relativity was comparing its prediction for the perihelion precession of Mercury to the observed value. The observed value of the perihelion precession is $\frac{\Delta \theta}{\text{yr}}$. Now, we wish to calculate this value using your program. This is done by integrating the equations of motion for Mercury around its orbit. The equations of motion are:

$$\frac{d^2 \theta}{dt^2} = -\frac{GM}{r^3} \sin \theta,$$

where $\theta$ is the angle in radians between the direction of the sun and the velocity vector of Mercury. $G$ is the gravitational constant, and $M$ is the mass of the Sun. $r$ is the distance between Mercury and the Sun.

Using your Runge-Kutta solver, you can integrate these equations and calculate the perihelion precession $\Delta \theta$. Compare your result with the observed value and discuss any discrepancies.

Finally, you can extend your program to include all planets in the solar system (if you have time, you can also include the various moons, but it is not required) and discuss your results. Try to find data for the initial positions and velocities for all planets.
helion precession, when all classical effects (such as the perturbation of the orbit due to gravitational attraction from the other planets) are subtracted, is $43''$ (43 arc seconds) per century.

Closed elliptical orbits are a special feature of the Newtonian $1/r^2$ force. In general, any correction to the pure $1/r^2$ behaviour will lead to an orbit which is not closed, i.e. after one complete orbit around the Sun, the planet will not be at exactly the same position as it started. If the correction is small, then each orbit around the Sun will be almost the same as the classical ellipse, and the orbit can be thought of as an ellipse whose orientation in space slowly rotates. In other words, the perihelion of the ellipse slowly precesses around the Sun.

You will now study the orbit of Mercury around the Sun, adding a general relativistic correction to the Newtonian gravitational force, so that the force becomes

$$F_G = \frac{GM_{\text{Sun}} M_{\text{Mercury}}}{r^2} \left[ 1 + \frac{3l^2}{r^4 c^2} \right]$$

where $M_{\text{Mercury}}$ is the mass of Mercury, $r$ is the distance between Mercury and the Sun, $l = |r \times v|$ is the magnitude of Mercury's orbital angular momentum per unit mass, and $c$ is the speed of light in vacuum. Run a simulation over one century of Mercury's orbit around the Sun with no other planets present, starting with Mercury at perihelion on the $x$ axis. Check then the value of the perihelion angle $\theta_p$, using

$$\tan \theta_p = \frac{y_p}{x_p}$$

where $x_p$ $(y_p)$ is the $x$ $(y)$ position of Mercury at perihelion, i.e. at the point where Mercury is at its closest to the Sun. You may use that the speed of Mercury at perihelion is 12.44 AU/yr, and that the distance to the Sun at perihelion is 0.3075 AU. You need to make sure that the time resolution used in your simulation is sufficient, for example by checking that the perihelion precession you get with a pure Newtonian force is at least a few orders of magnitude smaller than the observed perihelion precession of Mercury. Can the observed perihelion precession of Mercury be explained by the general theory of relativity?

**8.4.** In this exercise we will implement a molecular dynamics (MD) code to model the behavior of a system of Argon atoms, and use this model to study statistical properties of the system. In all calculations, we will use so-called MD units. These assume that all the particles in a simulation are identical, so the masses and LJ parameters can be factored out of the equations. You will need to insert $A = A_0$ for every variable quantity $A$ in equations 8.25-8.30 above. For example, for velocity, $v = v L_0$. The time step $\Delta t$ must also be treated this way.

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Conversion factor</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>$L_0 = \sigma$</td>
<td>3.405 Å</td>
</tr>
<tr>
<td>Time</td>
<td>$t_0 = \sigma \sqrt{m/\epsilon}$</td>
<td>$2.1569 \cdot 10^4$ fs</td>
</tr>
<tr>
<td>Force</td>
<td>$F_0 = m \sigma / L_0^2 = \epsilon / \sigma$</td>
<td>$3.0303 \cdot 10^{-1}$ eV/Å</td>
</tr>
<tr>
<td>Energy</td>
<td>$E_0 = \epsilon$</td>
<td>$1.0318 \cdot 10^{-2}$ eV</td>
</tr>
<tr>
<td>Temperature</td>
<td>$t_0 = \epsilon / k_B$</td>
<td>119.74 K</td>
</tr>
</tbody>
</table>

**Table 8.1** Conversion factors $A_0$ from MD units for variable quantities.

In case you want to convert between your internal MD units and other units during input and output, the actual values of the conversion factors are listed in table 8.1. These are calculated using the argon mass, lattice constant and LJ parameters: $m = 39.948$ amu, $a = 5.260$
Å (solid argon), \( \sigma = 3.405 \, \text{Å}, \; \varepsilon = 1.0318 \cdot 10^{-2} \text{eV} \). Another common practice is putting \( E_0 = 4\varepsilon \), affecting the conversion factors \( F_0, T_0 \) and \( f_0 \).

Normally distributed random numbers are obtained by performing a Box-Muller transform on uniformly distributed numbers. Let \( u \) and \( v \) be uniform numbers in the interval \((-1,1)\). These numbers will only be accepted for the transformation if \( s = u^2 + v^2 \) is in the interval \((0,1)\). In that case, we obtain two normally distributed numbers \( n_1 \) and \( n_2 \) by multiplying \( u \) and \( v \) with a constant,

\[
n_1 = Su, \quad n_2 = Sv, \quad S = \sqrt{\frac{-\ln s}{s}}. \quad (8.24)
\]

\( n_1 \) and \( n_2 \) will have standard deviations of 1, but multiplying all generated numbers with a constant will give a distribution with that constant as the standard deviation.

a) Write a program that generates an \( N_c \times N_c \times N_c \) unit cell face centered cubic lattice of argon atoms. If you use an object oriented programming language, each atom and/or the entire lattice should be objects of a class.

For easy testing of the lattice arranger and the later MD implementation, you should already visualize your atoms. VMD is a visualization program with a simple output format and pretty graphics. It can be downloaded and run from your home area, and a description of its output format can be found in the appendix.

b) Consider first free particles with initial independent Maxwell-Boltzmann distributed velocities. These correspond to normally distributed values with standard deviation \( \sqrt{k_B T/m} \) for the desired temperature \( T \). In a system of \( N \) atoms, all \( 3N \) velocity components \( v \) are set using

\[
v = \sqrt{k_B T/m\xi} \quad (8.25)
\]

where \( \xi \) is a normally distributed number with mean 0 and standard deviation 1. Remove any initial total linear momentum from the system.

Integrate the dynamical equation (N2L) using the symplectic and numerically stable velocity Verlet algorithm. For each particle \( i \), the steps are as follows (currently setting \( U_i = 0 \)):

\[
v_i(t+\Delta t/2) = v_i(t) + \frac{F_i(t)}{2m} \Delta t \quad (8.26)
\]

\[
r_i(t+\Delta t) = r_i(t) + v_i(t+\Delta t/2)\Delta t \quad (8.27)
\]

\[
F_i(t+\Delta t) = -\nabla U_i(\{r\}) \quad (8.28)
\]

\[
v_i(t+\Delta t) = v_i(t+\Delta t/2) + \frac{F_i(t+\Delta t)}{2m} \Delta t \quad (8.29)
\]

The particles will now spread out into space. We are only interested in bulk atoms in a material, so the next step is implementing periodic boundary conditions. Every time the position of a particle is updated, the program must check if it has gone through one of the sides.

c) Create a function for calculating the force between all particles. Use the Lennard-Jones potential, which has the following form:

\[
U_{ij}(r_{ij}) = 4\varepsilon \left[ \left( \frac{\sigma}{r_{ij}} \right)^{12} - \left( \frac{\sigma}{r_{ij}} \right)^6 \right] \quad (8.30)
\]

where \( r_{ij} = r_i - r_j \). Differentiate the expression analytically for finding the force. Summing up the potential energy between all particles can also be useful.

Use the minimum image convention when calculating the distance between particles. E.g. the distance between atoms/atom replicate \( i \) and \( j \) in the \( x \) direction becomes \( \min_{\delta} (x_i - x_j + \delta L) \)
where $\delta \in \{-1, 0, 1\}$ and $L$ is the length of the simulation box in the $x$ direction. This limits the interaction range to half the system size, which is more than enough for our potential. You should now have a working MD program for simulating bulk argon in its solid, liquid and gas phases.

d) You probably notice that the force calculations are the most time consuming part of your program. The number of force terms is $\frac{1}{2}N(N-1)$ for each time step, which gives a workload scaling $\propto N^2$. We want to improve this by neglecting force terms for particles far apart. The LJ interaction is short ranged and can be neglected for distances over $r_{\text{cut}} \approx 3\epsilon$. A simple and efficient way of achieving this is by implementing Verlet lists.

Create arrays specifying the neighbours of all particles and a function to update this list e.g. every 10th timestep. An atom $i$ needs only to keep track of neighbour atoms with a lower index $j$. The force loops can now iterate over all atoms $i$ and atoms $j$ in the neighbour list of $i$. Compare the time usage of the program with and without Verlet lists for different system sizes.

e) An MD simulation of bulk material enables the measurement of macroscopic quantities. The ergodic hypothesis states that the time a system has one particular value of an observable $A$ is proportional to the phase space volume where $A$ has this value. This applies to systems in equilibrium studied for a long period of time. As a result, the time average and ensemble average of a variable are equal. If we average over long enough periods of time, we can predict equilibrium properties of real materials.

According to the central limit theorem, the velocity distribution of the particles will eventually evolve into a Maxwell-Boltzmann-distribution whatever the initial condition. Switch to initializing the velocities with uniformly distributed random numbers in the interval $[-v,v]$, for a reasonable $v$. Investigate the velocity distribution after equilibration, e.g. by dumping the velocities to a file and using the Matlab hist() function. Roughly how much time does it take for the velocities to reach a MB-distribution?

The easiest quantity to calculate is the total energy of the system. Sum up the kinetic and potential energies of all your argon atoms. Output the total energy for each time step of the simulation. The energy should be conserved, but some fluctuations are inevitable as the dynamics are discretized. How does the size of the fluctuations depend on the time step $\Delta t$?

The temperature of a MD system is non-trivial to calculate for general potential forms. The simplest estimate assumes equilibrium between the translational and potential degrees of freedom. According to the equipartition principle, the total kinetic energy is

$$E_k = \frac{3}{2} N k_B T$$

(8.31)

where $N$ is the number of atoms and $T$ is our estimate for the system temperature.

Invert the equation and measure the temperature for each time step. Don’t forget to equilibrate the system first. What mean temperature does the system settle on, and how does this compare to the initial temperature? How does the temperature fluctuations vary with the system size?

There are several ways of measuring the pressure $P$ of a many-atom system. The method we will use is derived from the virial equation for the pressure. In a volume $V$ with particle density $\rho = N/V$, the average pressure is

$$P = \rho k_B T + \frac{1}{3V} \langle \sum_{i<j} F_{ij} \cdot r_{ij} \rangle$$

(8.32)

where the sum runs over all interacting particle pairs. The vector products should be computed and summed up inside the force loops for efficiency.
f) In order to simulate the canonical ensemble, interactions with an external heat bath must be taken into account. Many methods have been suggested in order to achieve this, all with their pros and cons. Requirements for a good thermostat are:

- Keeping the system temperature around the heat bath temperature
- Sampling the phase space corresponding to the canonical ensemble
- Tunability
- Preservation of dynamics

The method closest to fulfilling these requirements which is in widespread use is the Nosé-Hoover thermostat, which is somewhat complicated to implement. We will focus on simpler methods. They will require negligible CPU time and should be applied for each time step.

Many thermostats work by rescaling the velocities of all atoms by multiplying them with a factor $\gamma$. The Berendsen thermostat uses

$$
\gamma = \sqrt{1 + \frac{\Delta t}{\tau} \left( \frac{T_{\text{bath}}}{T} - 1 \right)}
$$

(8.33)

with $\tau$ as the relaxation time, tuning the coupling to the heat bath. Though it satisfies Fourier's law of heat transfer (the transferred heat between two bodies is proportional to their temperature difference) it does a poor job at sampling the canonical ensemble.

Implement the Berendsen thermostat as a function in your code. $\tau = \Delta t$ will keep the (estimated) temperature exactly constant. It should be put to 10-20 times this value.

The Andersen thermostat simulates (hard) collisions between atoms inside the system and in the heat bath. Atoms which collide will gain a new normally distributed velocity with standard deviation $\sqrt{k_B T_{\text{bath}}/m}$. For all atoms, a random uniformly distributed number in the interval $[0, 1]$ is generated. If this number is less than $\Delta t/\tau$, the atom is assigned a new velocity. In this case, $\tau$ is treated as a collision time, and should have about the same value as the $\tau$ in the Berendsen thermostat. The Andersen thermostat is very useful when equilibrating systems, but disturbs the dynamics of e.g. lattice vibrations.

Implement the Andersen thermostat, and compare $T(t)$ graphs for simulations using the different methods. Again, be aware that our $T$ is just an approximation to the real temperature. Differences can also be seen in the dynamics.

The Andersen thermostat simulates (hard) collisions between atoms inside the system and in the heat bath. Atoms which collide will gain a new normally distributed velocity with standard deviation $\sqrt{k_B T_{\text{bath}}/m}$. For all atoms, a random uniformly distributed number in the interval $[0, 1]$ is generated. If this number is less than $\Delta t/\tau$, the atom is assigned a new velocity. In this case, $\tau$ is treated as a collision time, and should have about the same value as the $\tau$ in the Berendsen thermostat. The Andersen thermostat is very useful when equilibrating systems, but disturbs the dynamics of e.g. lattice vibrations.

Implement the Andersen thermostat, and compare $T(t)$ graphs for simulations using the different methods. Again, be aware that our $T$ is just an approximation to the real temperature. Differences can also be seen in the dynamics.

The Andersen thermostat simulates (hard) collisions between atoms inside the system and in the heat bath. Atoms which collide will gain a new normally distributed velocity with standard deviation $\sqrt{k_B T_{\text{bath}}/m}$. For all atoms, a random uniformly distributed number in the interval $[0, 1]$ is generated. If this number is less than $\Delta t/\tau$, the atom is assigned a new velocity. In this case, $\tau$ is treated as a collision time, and should have about the same value as the $\tau$ in the Berendsen thermostat. The Andersen thermostat is very useful when equilibrating systems, but disturbs the dynamics of e.g. lattice vibrations.

Implement the Andersen thermostat, and compare $T(t)$ graphs for simulations using the different methods. Again, be aware that our $T$ is just an approximation to the real temperature. Differences can also be seen in the dynamics.

The Andersen thermostat simulates (hard) collisions between atoms inside the system and in the heat bath. Atoms which collide will gain a new normally distributed velocity with standard deviation $\sqrt{k_B T_{\text{bath}}/m}$. For all atoms, a random uniformly distributed number in the interval $[0, 1]$ is generated. If this number is less than $\Delta t/\tau$, the atom is assigned a new velocity. In this case, $\tau$ is treated as a collision time, and should have about the same value as the $\tau$ in the Berendsen thermostat. The Andersen thermostat is very useful when equilibrating systems, but disturbs the dynamics of e.g. lattice vibrations.

Implement the Andersen thermostat, and compare $T(t)$ graphs for simulations using the different methods. Again, be aware that our $T$ is just an approximation to the real temperature. Differences can also be seen in the dynamics.

The Andersen thermostat simulates (hard) collisions between atoms inside the system and in the heat bath. Atoms which collide will gain a new normally distributed velocity with standard deviation $\sqrt{k_B T_{\text{bath}}/m}$. For all atoms, a random uniformly distributed number in the interval $[0, 1]$ is generated. If this number is less than $\Delta t/\tau$, the atom is assigned a new velocity. In this case, $\tau$ is treated as a collision time, and should have about the same value as the $\tau$ in the Berendsen thermostat. The Andersen thermostat is very useful when equilibrating systems, but disturbs the dynamics of e.g. lattice vibrations.

Implement the Andersen thermostat, and compare $T(t)$ graphs for simulations using the different methods. Again, be aware that our $T$ is just an approximation to the real temperature. Differences can also be seen in the dynamics.

The Andersen thermostat simulates (hard) collisions between atoms inside the system and in the heat bath. Atoms which collide will gain a new normally distributed velocity with standard deviation $\sqrt{k_B T_{\text{bath}}/m}$. For all atoms, a random uniformly distributed number in the interval $[0, 1]$ is generated. If this number is less than $\Delta t/\tau$, the atom is assigned a new velocity. In this case, $\tau$ is treated as a collision time, and should have about the same value as the $\tau$ in the Berendsen thermostat. The Andersen thermostat is very useful when equilibrating systems, but disturbs the dynamics of e.g. lattice vibrations.

Implement the Andersen thermostat, and compare $T(t)$ graphs for simulations using the different methods. Again, be aware that our $T$ is just an approximation to the real temperature. Differences can also be seen in the dynamics.

The Andersen thermostat simulates (hard) collisions between atoms inside the system and in the heat bath. Atoms which collide will gain a new normally distributed velocity with standard deviation $\sqrt{k_B T_{\text{bath}}/m}$. For all atoms, a random uniformly distributed number in the interval $[0, 1]$ is generated. If this number is less than $\Delta t/\tau$, the atom is assigned a new velocity. In this case, $\tau$ is treated as a collision time, and should have about the same value as the $\tau$ in the Berendsen thermostat. The Andersen thermostat is very useful when equilibrating systems, but disturbs the dynamics of e.g. lattice vibrations.

Implement the Andersen thermostat, and compare $T(t)$ graphs for simulations using the different methods. Again, be aware that our $T$ is just an approximation to the real temperature. Differences can also be seen in the dynamics.

The Andersen thermostat simulates (hard) collisions between atoms inside the system and in the heat bath. Atoms which collide will gain a new normally distributed velocity with standard deviation $\sqrt{k_B T_{\text{bath}}/m}$. For all atoms, a random uniformly distributed number in the interval $[0, 1]$ is generated. If this number is less than $\Delta t/\tau$, the atom is assigned a new velocity. In this case, $\tau$ is treated as a collision time, and should have about the same value as the $\tau$ in the Berendsen thermostat. The Andersen thermostat is very useful when equilibrating systems, but disturbs the dynamics of e.g. lattice vibrations.

Implement the Andersen thermostat, and compare $T(t)$ graphs for simulations using the different methods. Again, be aware that our $T$ is just an approximation to the real temperature. Differences can also be seen in the dynamics.

g) In a volume with PBC and atoms constituting a fluid, self-diffusion can be simulated. We are to measure the self-diffusion constant $D$ for liquid argon. This is achieved by finding the mean square displacement of all atoms after a given time,

$$
\langle r^2(t) \rangle = \frac{1}{N} \sum_{i=1}^{N} (r_i(t) - r_{\text{initial}}).
$$

(8.34)

From diffusion theory, we know that $\langle r^2(t) \rangle = 6Dt$ for a random walk in three dimensions, which is a good approximation to the motion of an atom in a fluid. Plot the mean square displacement as a function of time and extract the diffusion constant. Investigate the effect of temperature by finding $D$ for some temperatures in the liquid phase of argon. Remember that you are measuring the total distance travelled by the atoms, which must be continuous when an atom is displaced through a PBC boundary.

h) A radial distribution function $g(r)$, also called a pair correlation function, is a tool for characterizing the microscopic structure of a fluid. It is interpreted as the radial probability for finding another atom a distance $r$ from an arbitrary atom, or equivalently, the atomic density in a spherical shell of radius $r$ around an atom. It is commonly normalized by dividing it with the average particle density so that $\lim_{r \to \infty} g(r) = 1$. 


Estimate $g(r)$ for $r \in (0, \frac{L}{2}]$ in your argon system. The easiest way is to divide the distance interval into bins, loop over all pairs of particles and count how many distances belong in each bin. Time-averaging the function gives a better description of the system's general behaviour. Plot $g(r)$ for temperatures where the system is in solid and liquid phases. Does it appear as expected? How would the exact $g(r)$ look for a perfect crystal?