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 1687[1] 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 thermodynamic, 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
$$i\hbar\frac{\partial\psi(\mathbf{x},t)}{\partial t} = \frac{\hbar^2}{2m}\left(\frac{\partial^2\psi(\mathbf{r},t)}{\partial x^2} + \frac{\partial^2\psi(\mathbf{r},t)}{\partial y^2} + \frac{\partial^2\psi(\mathbf{r},t)}{\partial z^2}\right) + V(\mathbf{x})\psi(\mathbf{x},t),$$
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 ρ

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

and

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

where ρ 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

$$m\frac{dy^{(2)}(t)}{dt} = -kx(t) = -ky^{(1)}(t), \tag{8.2}$$

and

$$\frac{dy^{(1)}(t)}{dt} = y^{(2)}(t). \tag{8.3}$$

8.3 Finite difference methods

These methods fall under the general class of one-step methods. The algoritm 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 Δ, 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

$$\Delta(t_i, y_i(t_i)) = (y'(t_i) + \cdots + y^{(p)}(t_i)\frac{h^{p-1}}{p!}). \tag{8.5}$$

We define

$$y'(t_i) = f(t_i, y_i)$$

and if we truncate Δ 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

$$f'_{2c}(x) = \frac{f(x+h) - f(x)}{h} + O(h),$$

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

$$y_1^{(2)} = v_0 - hy_0k/m + O(h^2).$$

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 $y_{n+1}^{(1)}$ using the velocity at $y_{n+1}^{(2)}$, that is (in a slightly more generalized form)

$$y_{n+1}^{(1)} = y_n^{(1)} + hy_{n+1}^{(2)} + O(h^2)$$

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

$$\Delta(t_i, y_i(t_i)) = f(t_i) + \frac{h}{2}\frac{df(t_i, y_i)}{dt} + O(h^3).$$

The second derivative can be rewritten as

$$y'' = f' = \frac{df}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} f$$

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

$$y_{i+1} = y(t = t_i + h) = y(t_i) + h(f(t_i, y_i) + \ldots f^{(p-1)}(t_i, y_i)\frac{h^{p-1}}{p!}) + O(h^{p+1}).$$

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

$$y^{(1)}_{n+1} = y^{(1)}_n + \frac{h}{2}\left(y^{(2)}_{n+1} + y^{(2)}_n\right) + O(h^2)$$

and

$$y^{(2)}_{n+1} = y^{(2)}_n + ha_n + O(h^2),$$

yielding

$$y^{(1)}_{n+1} = y^{(1)}_n + hy^{(2)}_n + \frac{h^2}{2}a_n + O(h^3)$$

implying 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

$$y^{(2)}_{n+1/2} = y^{(2)}_{n-1/2} + ha_n + O(h^2),$$

and

$$y^{(1)}_{n+1} = y^{(1)}_n + hy^{(2)}_{n+1/2} + O(h^2).$$

Note that this method needs the calculation of $y^{(2)}_{1/2}$. This is done using for example Euler's method

$$y^{(2)}_{1/2} = y^{(2)}_0 + \frac{h}{2}a_0 + O(h^2).$$

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

$$y^{(2)}_{n+1} = y^{(2)}_n + ha_{n+1/2} + O(h^2), \tag{8.7}$$

and

$$y^{(1)}_{n+1} = y^{(1)}_n + hy^{(2)}_{n+1/2} + O(h^2). \tag{8.8}$$

8.3.2 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 predicition for the solution by computing $y_{i+1} \approx y(t_i) + hk_1$ by Euler's method.
3. Use the predicition 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 $O(h^2)$, 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

$$y_{i+1} \approx y_i + k_2 + O(h^3).$$

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

$$\int_{t_i}^{t_{i+1}} f(t,y)dt \approx \frac{h}{6}\left[f(t_i,y_i) + 4f(t_{i+1/2},y_{i+1/2}) + f(t_{i+1},y_{i+1})\right] + O(h^5).$$

This means in turn that we have

$$y_{i+1} = y_i + \frac{h}{6}\left[f(t_i,y_i) + 4f(t_{i+1/2},y_{i+1/2}) + f(t_{i+1},y_{i+1})\right] + O(h^5).$$

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
$$k_1 = hf(t_i,y_i), \tag{8.9}$$
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
$$k_2 = hf(t_i + h/2, y_i + k_1/2). \tag{8.10}$$
3. The improved slope at the midpoint is used to further improve the slope of $y_{i+1/2}$ by computing
$$k_3 = hf(t_i + h/2, y_i + k_2/2). \tag{8.11}$$
4. With the latter slope we can in turn predict the value of y_{i+1} via the computation of
$$k_4 = hf(t_i + h, y_i + k_3). \tag{8.12}$$
5. The final algorithm becomes then
$$y_{i+1} = y_i + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right). \tag{8.13}$$

Thus, the algorithm consists in first calculating k_1 with t_i, y_1 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.

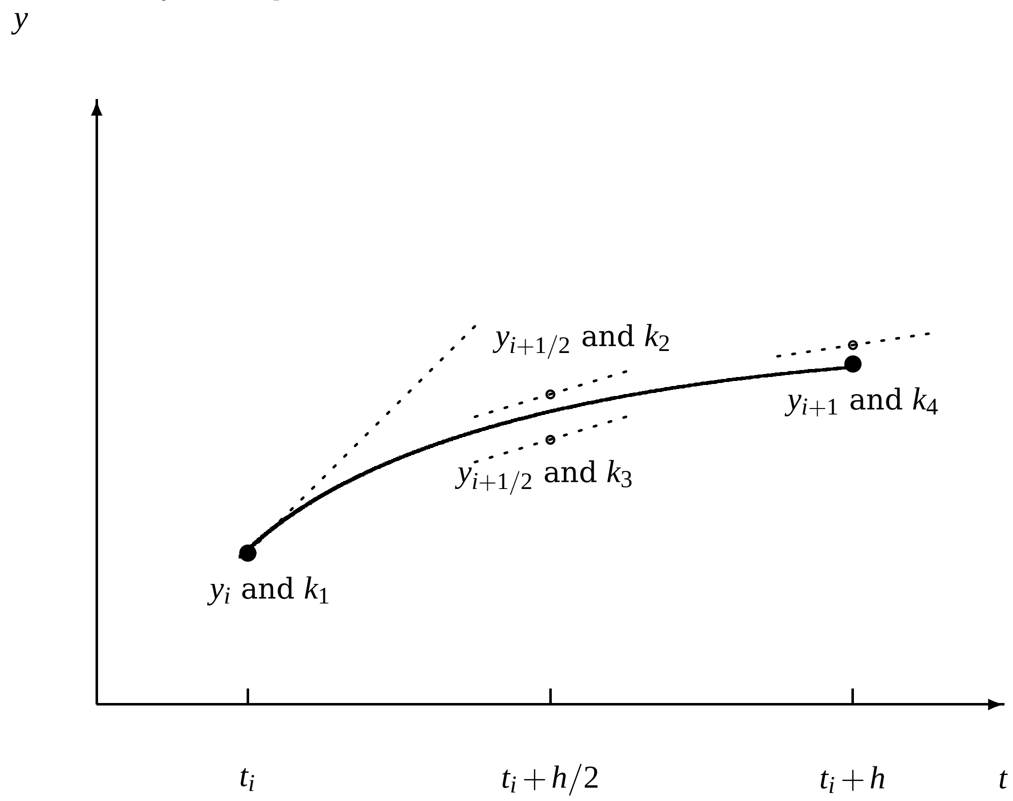


Fig. 8.1 Geometrical interpretation of the fourth-order Runge-Kutta method. The derivative is evaluated at four points, once at the intial point, twice at the trial midpoint and once at the trial endpoint. These four derivatives constitute one Runge-Kutta step resulting in the final value for $y_{i+1} = y_i + 1/6(k_1 + 2k_2 + 2k_3 + k_4)$.

8.5 Physics examples

8.5.1 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

$$F = -kx.$$

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

$$\frac{d^2x}{dt^2} = -\frac{k}{m}x = -\omega_0^2 x, \tag{8.14}$$

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

$$x(t) = Acos(\omega_0 t + \nu),$$

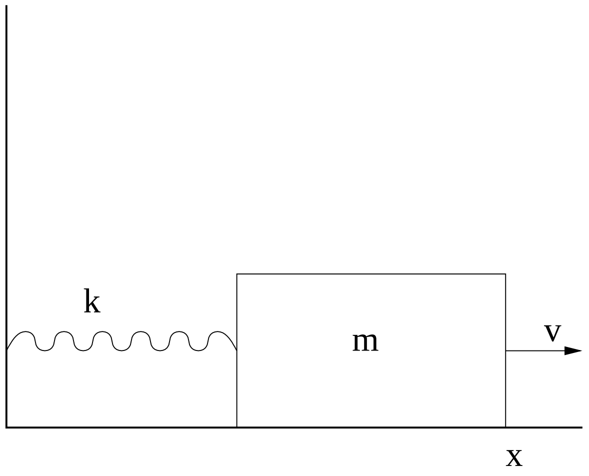


Fig. 8.2 Block tied to a wall with a spring tension acting on it.

where A is the amplitude and v 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 2π, 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 v.

In addition to the periodicity test, the total energy has also to be conserved.

Suppose we choose the initial conditions

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

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

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

   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.

8.5.1.1 Program to solve the differential equations for a sliding block

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

http://folk.uio.no/mhjensen/compphys/programs/chapter08/cpp/program1.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 d^2x/dt^2 =-kx
with k the spring tension and m the mass of the block.
The angular frequency is omega^2 = 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
    dx/dt = v and
    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
y[1] == v
dy[0]/dt = v
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;
// output file as global variable
ofstream ofile;
// function declarations
void derivatives(double, double *, double *);
void initialise ( double&, double&, int&);
void output( double, double *, double);
void runge_kutta_4(double *, double *, int, double, double,
     double *, void (*)(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
  dydt = new 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*0.5;
    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,dym); // 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.

8.5.2 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. (8.14) to

$$\frac{d^2x}{dt^2} = -\omega_0^2 x - \nu\frac{dx}{dt}, \tag{8.15}$$

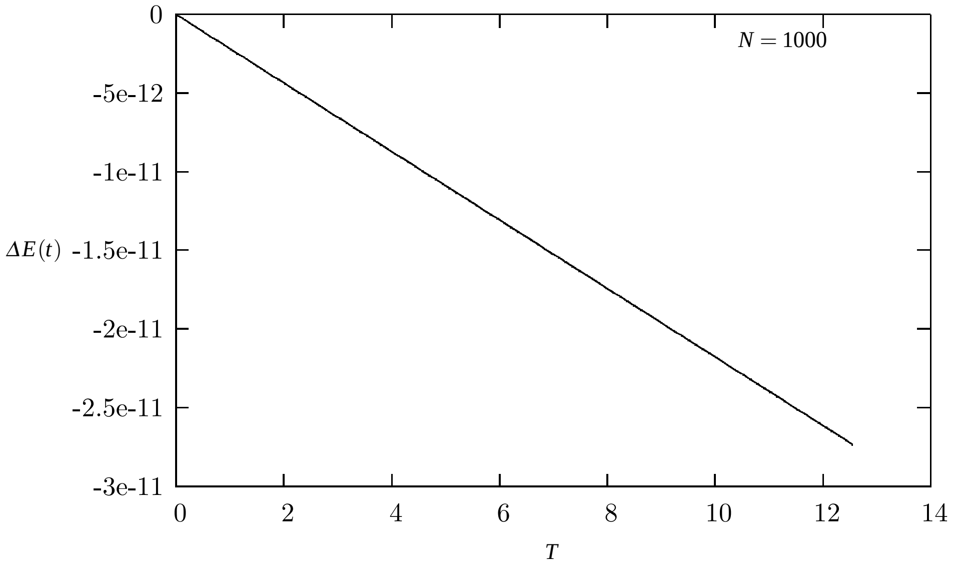
where ν is the damping coefficient, being a measure of the magnitude of the drag term.

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

$$F(t) = Bcos(\omega t),$$

and we rewrite Eq. (8.15) as

$$\frac{d^2x}{dt^2} = -\omega_0^2 x - \nu\frac{dx}{dt} + F(t). \tag{8.16}$$



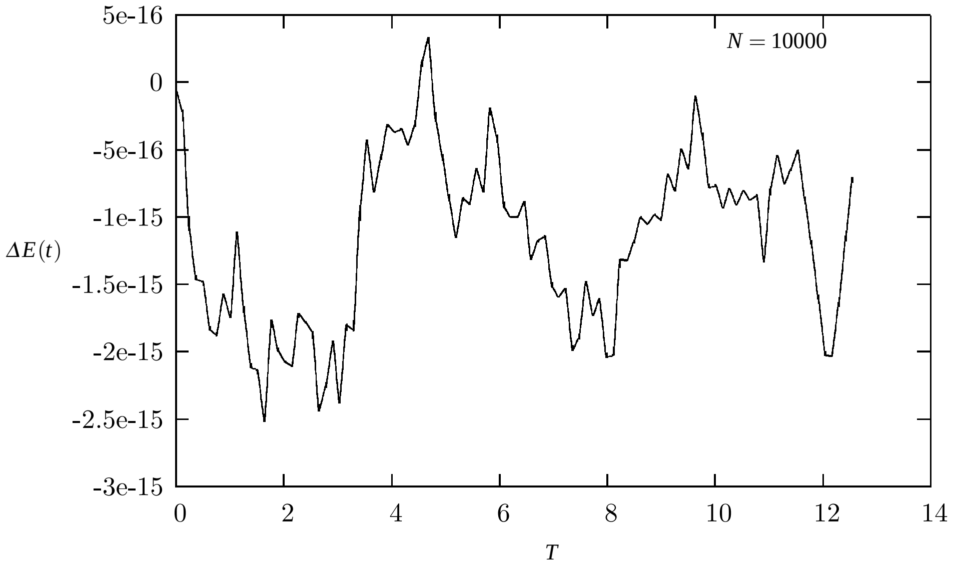


Fig. 8.3 Plot of $\Delta E(t) = E_0 - E_{\text{computed}}$ for $N = 1000$ and $N = 10000$ time steps up to two periods. The initial position $x_0 = 1$ m and initial velocity $v_0 = 0$ m/s. The mass and spring tension are set to $k = m = 1$.

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, ν 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. (8.16) as

$$L\frac{d^2Q}{dt^2} + \frac{Q}{C} + R\frac{dQ}{dt} = V(t). \tag{8.17}$$

The circuit is shown in Fig. 8.4.

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,

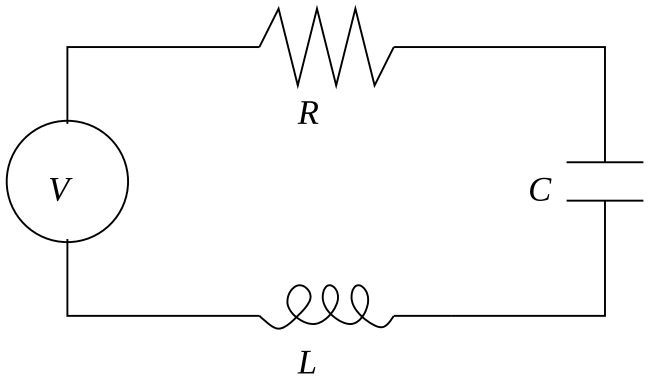


Fig. 8.4 Simple RLC circuit with a voltage source V.

which is a consequence of energy conservation, we have

$$V(t) = IR + LdI/dt + Q/C,$$

and using

$$I = \frac{dQ}{dt},$$

we arrive at Eq. (8.17).

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 dwelve into the problems of the pendulum, from almost harmonic oscillations to chaotic motion!

8.5.3 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.

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

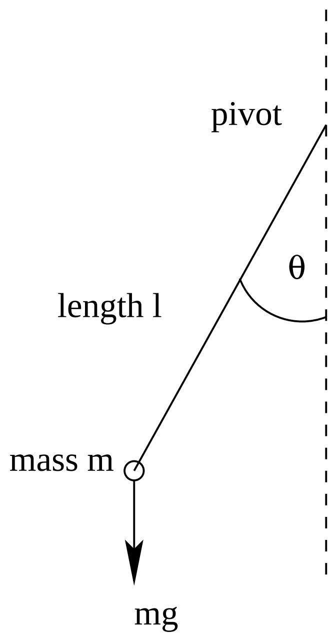


Fig. 8.5 A simple pendulum.

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\theta,$$

which is exactly of the same form as Eq. (8.14). We can thus check our solutions for small values of θ 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

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

where v 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

$$ml\frac{d^2\theta}{dt^2} + v\frac{d\theta}{dt} + mg\sin(\theta) = A\cos(\omega t), \tag{8.18}$$

with A and ω 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. (8.18) 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

$$\frac{d\theta}{d\hat{t}} = \hat{v},$$

and

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

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.

8.6 Physics Project: the pendulum

8.6.1 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 θ, we have $\sin(\theta) \approx \theta$ and our equations become

$$\frac{d\theta}{d\hat{t}} = \hat{v},$$

and

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

These equations are linear in the angle θ and are similar to those of the sliding block or the RLC circuit. With given initial conditions $\hat{v}_0$ and θ_0 they can be solved analytically to yield

$$\begin{aligned}\theta(t) \quad &= \left[\theta_0 - \frac{\hat{A}(1-\hat{\omega}^2)}{(1-\hat{\omega}^2)^2+\hat{\omega}^2/Q^2}\right] e^{-\tau/2Q} cos(\sqrt{1-\frac{1}{4Q^2}}\tau) \\ &+ \left[\hat{v}_0 + \frac{\theta_0}{2Q} - \frac{\hat{A}(1-3\hat{\omega}^2)/2Q}{(1-\hat{\omega}^2)^2+\hat{\omega}^2/Q^2}\right] e^{-\tau/2Q} sin(\sqrt{1-\frac{1}{4Q^2}}\tau) + \frac{\hat{A}(1-\hat{\omega}^2)cos(\hat{\omega}\tau) + \frac{\hat{\omega}}{Q} sin(\hat{\omega}\tau)}{(1-\hat{\omega}^2)^2+\hat{\omega}^2/Q^2},\end{aligned}$$

and

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

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

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

and

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

tracing the closed phase-space curve

$$\left(\frac{\theta}{\tilde{A}}\right)^2 + \left(\frac{\hat{v}}{\hat{\omega}\tilde{A}}\right)^2 = 1$$

with

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

This curve forms an ellipse whose principal axes are θ and $\hat{v}$. 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 $T = 2\pi/\hat{\omega}$. 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 θ as function of time τ for the case with $Q = 2$, $\hat{\omega} = 2/3$ and $\hat{A} = 0.5$. The length is set equal to 1 m and mass of the pendulum is set equal to 1 kg. The inital velocity is $\hat{v}_0 = 0$ 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, $\hat{A} = 0$ 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 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.

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 θ is shown in Fig. 8.8. The corresponding phase-space curve is shown in Fig. 8.9. This curve

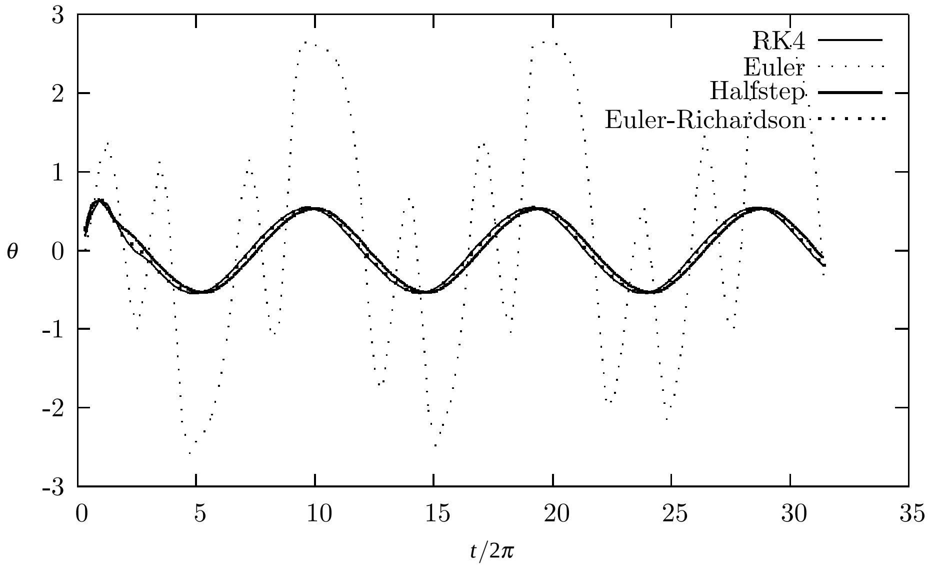


Fig. 8.6 Plot of θ as function of time τ with $Q = 2$, $\hat{\omega} = 2/3$ and $\hat{A} = 0.5$. The mass and length of the pendulum are set equal to 1. The initial velocity is $\hat{v}_0 = 0$ and $\theta_0 = 0.01$. Four different methods have been used to solve the equations, Euler's method from Eq. (8.6), the half-step method, Euler-Richardson's method in Eqs. (8.7)-(8.8) and finally the fourth-order Runge-Kutta scheme RK4. Only $N = 100$ integration points have been used for a time interval $t \in [0, 10\pi]$.

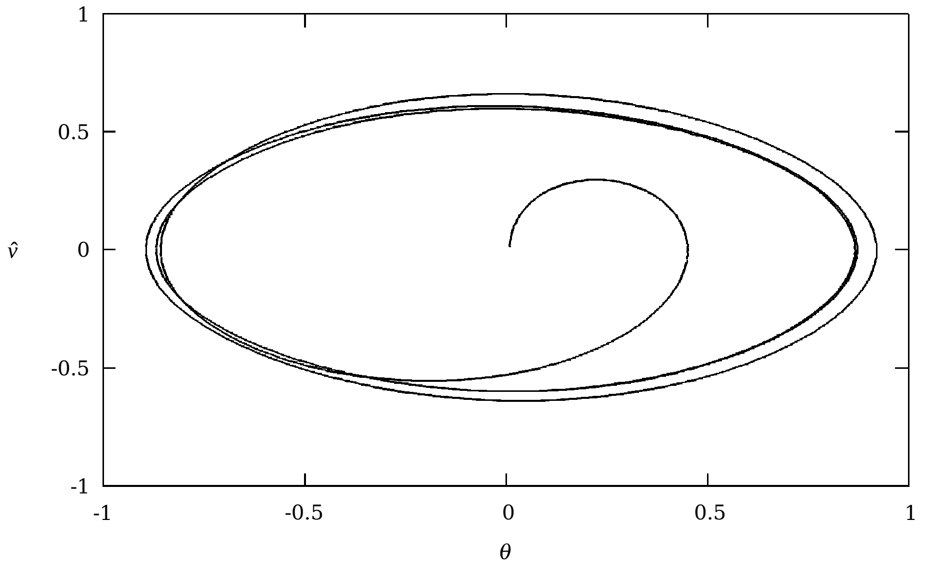


Fig. 8.7 Phase-space curve of a linear damped pendulum with $Q = 2$, $\hat{\omega} = 2/3$ and $\hat{A} = 0.5$. The inital velocity is $\hat{v}_0 = 0$ and $\theta_0 = 0.01$.

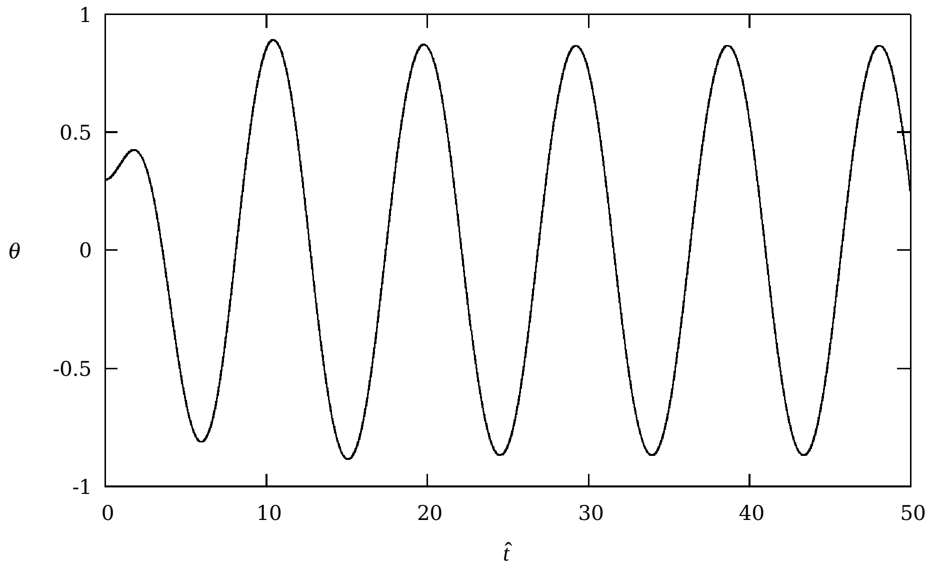


Fig. 8.8 Plot of θ as function of time τ with $Q = 2$, $\hat{\omega} = 2/3$ and $\hat{A} = 0.5$. The mass of the pendulum is set equal to 1 kg and its length to 1 m. The inital velocity is $\hat{v}_0 = 0$ and $\theta_0 = 0.3$.

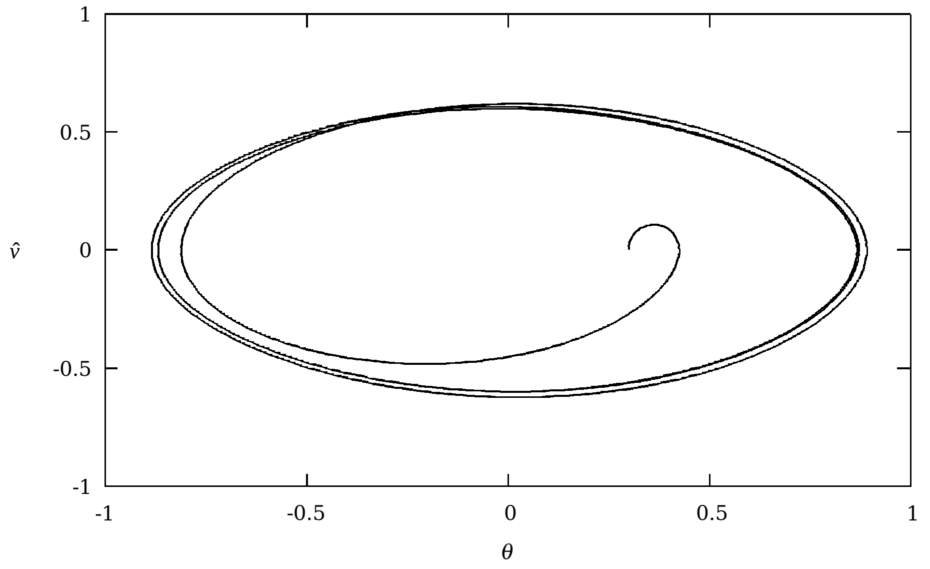


Fig. 8.9 Phase-space curve with $Q = 2$, $\hat{\omega} = 2/3$ and $\hat{A} = 0.5$. The mass of the pendulum is set equal to 1 kg and its length $l = 1$ m.. The inital velocity is $\hat{v}_0 = 0$ and $\theta_0 = 0.3$.

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 θ 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

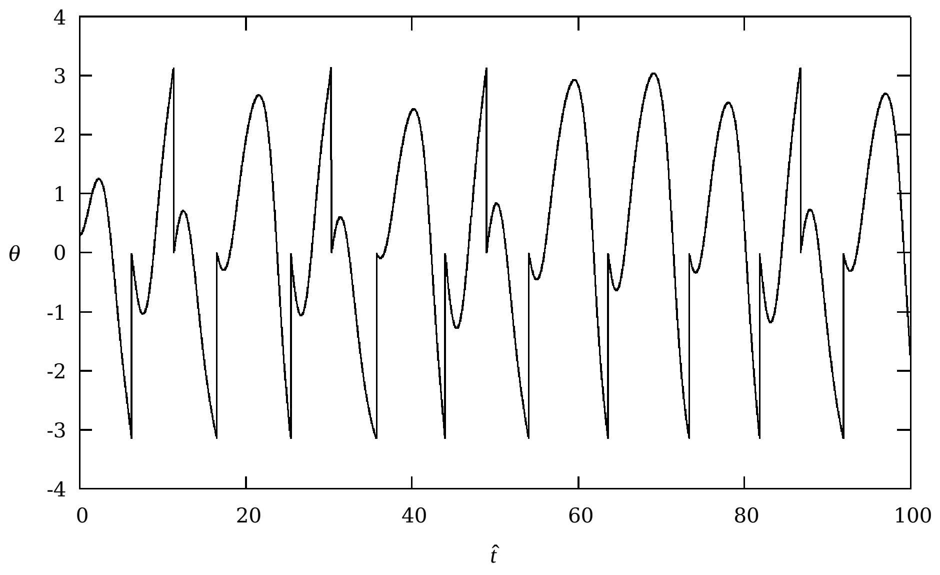


Fig. 8.10 Plot of θ as function of time τ with $Q = 2$, $\hat{\omega} = 2/3$ and $\hat{A} = 1.35$. The mass of the pendulum is set equal to 1 kg and its length to 1 m. The inital velocity is $\hat{v}_0 = 0$ and $\theta_0 = 0.3$. Every time θ passes the value $\pm\pi$ we reset its value to swing between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

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.

8.6.2 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 θ with an external force $Acos(\omega t)$. It employes 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 `. classpendulum`. 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

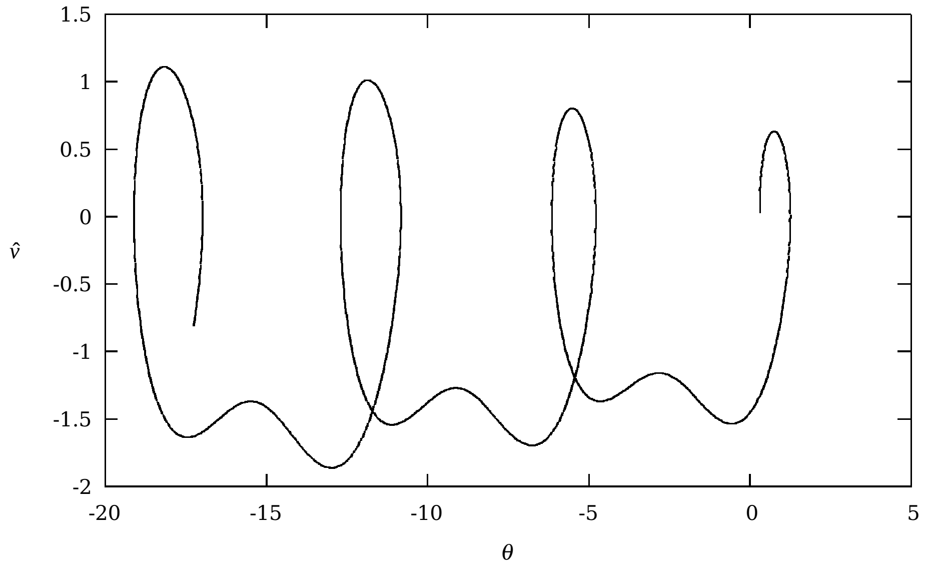


Fig. 8.11 Phase-space curve after 10 periods with $Q = 2$, $\hat{\omega} = 2/3$ and $\hat{A} = 1.35$. The mass of the pendulum is set equal to 1 kg and its length $l = 1$ m. The inital velocity is $\hat{v}_0 = 0$ and $\theta_0 = 0.3$.

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

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

m*l*(phi)'' + viscosity*(phi)' + m*g*sin(phi) = A*cos(omega*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*l)
t_roof = omega_0*t
omega_roof = omega/omega_0
Q = (m*g)/(omega_0*reib)
A_roof = A/(m*g)

and we get a dimensionless equation

(phi)'' + 1/Q*(phi)' + sin(phi) = A_roof*cos(omega_roof*t_roof)

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

(phi)' = v
(v)' = -v/Q - sin(phi) +A_roof*cos(omega_roof*t_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 pendelum
   {
   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_step(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 pendelum::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)
      out[1]=-in[1]/((double)Q)-sin(in[0])+A_roof*cos(omega_roof*t); //out[1] = (phi)''
    else
      out[1]=-sin(in[0])+A_roof*cos(omega_roof*t); //out[1] = (phi)''
  }
// Here we define all input parameters.
  void pendelum::initialise()
  {
    double m,l,omega,A,viscosity,phi_0,v_0,t_end;
    cout<<"Solving the differential eqation of the pendulum!\n";
    cout<<"We have a pendulum with mass m, length l. Then we have a periodic force with
        amplitude A and omega\n";
    cout<<"Furthermore there is a viscous drag coefficient.\n";
    cout<<"The initial conditions at t=0 are phi_0 and v_0\n";
    cout<<"Mass m: ";
    cin>>m;
    cout<<"length l: ";
    cin>>l;
    cout<<"omega of the force: ";
    cin>>omega;
    cout<<"amplitude of the force: ";
    cin>>A;
    cout<<"The value of the viscous drag constant (viscosity): ";
    cin>>viscosity;
    cout<<"phi_0: ";
    cin>>y[0];
    cout<<"v_0: ";
    cin>>y[1];
    cout<<"Number of time steps or integration steps:";
    cin>>n;
    cout<<"Final time steps as multiplum of pi:";
    cin>>t_end;
    t_end *= acos(-1.);
    g=9.81;
    // We need the following values:
    omega_0=sqrt(g/((double)l)); // omega of the pendulum
    if (viscosity) Q= m*g/((double)omega_0*viscosity);
    else Q=0; //calculating Q
    A_roof=A/((double)m*g);
    omega_roof=omega/((double)omega_0);
    delta_t_roof=omega_0*t_end/((double)n); //delta_t without dimension
    delta_t=t_end/((double)n);
  }
// fourth order Run
  void pendelum::rk4_step(double t,double *yin,double *yout,double delta_t)
  {
    /*
      The function calculates one step of fourth-order-runge-kutta-method
      We will need it for the normal fourth-order-Runge-Kutta-method and
      for RK-method with adaptive stepsize control

      The function calculates the value of y(t + delta_t) using fourth-order-RK-method
      Input: time t and the stepsize delta_t, yin (values of phi and v at time t)
      Output: yout (values of phi and v at time t+delta_t)

    */
    double k1[2],k2[2],k3[2],k4[2],y_k[2];
    // Calculation of k1
    derivatives(t,yin,yout);
    k1[1]=yout[1]*delta_t;
    k1[0]=yout[0]*delta_t;
    y_k[0]=yin[0]+k1[0]*0.5;
    y_k[1]=yin[1]+k1[1]*0.5;
    /*Calculation of k2 */
    derivatives(t+delta_t*0.5,y_k,yout);
    k2[1]=yout[1]*delta_t;
    k2[0]=yout[0]*delta_t;
    y_k[0]=yin[0]+k2[0]*0.5;
    y_k[1]=yin[1]+k2[1]*0.5;
    /* Calculation of k3 */
    derivatives(t+delta_t*0.5,y_k,yout);
    k3[1]=yout[1]*delta_t;
    k3[0]=yout[0]*delta_t;
    y_k[0]=yin[0]+k3[0];
    y_k[1]=yin[1]+k3[1];
    /*Calculation of k4 */
    derivatives(t+delta_t,y_k,yout);
    k4[1]=yout[1]*delta_t;
    k4[0]=yout[0]*delta_t;
    /*Calculation of new values of phi and v */
    yout[0]=yin[0]+1.0/6.0*(k1[0]+2*k2[0]+2*k3[0]+k4[0]);
    yout[1]=yin[1]+1.0/6.0*(k1[1]+2*k2[1]+2*k3[1]+k4[1]);
  }

  void pendelum::rk4()
  {
    /*We are using the fourth-order-Runge-Kutta-algorithm
      We have to calculate the parameters k1, k2, k3, k4 for v and phi,
      so we use to arrays k1[2] and k2[2] for this
      k1[0], k2[0] are the parameters for phi,
      k1[1], k2[1] are the parameters for v
    */

    int i;
    double t_h;
    double yout[2],y_h[2]; //k1[2],k2[2],k3[2],k4[2],y_k[2];

    t_h=0;
    y_h[0]=y[0]; //phi
    y_h[1]=y[1]; //v
    ofstream fout("rk4.out");
    fout.setf(ios::scientific);
    fout.precision(20);
    for(i=1; i<=n; i++){
      rk4_step(t_h,y_h,yout,delta_t_roof);
      fout<<i*delta_t<<"\t\t"<<yout[0]<<"\t\t"<<yout[1]<<"\n";
      t_h+=delta_t_roof;
      y_h[0]=yout[0];
      y_h[1]=yout[1];
    }
    fout.close;
  }

  int main()
  {
    pendelum testcase;
    testcase.initialise();
    testcase.rk4();
    return 0;
  } // end of main function
```

8.7 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 addtion, 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.