PY 502, Computational Physics, Fall 2018

Numerical Solutions of Classical Equations of Motion

Anders W. Sandvik, Department of Physics, Boston University

1 Introduction

Classical equations of motion, i.e., Newton's laws, govern the dynamics of systems ranging from very large, such as solar systems and galaxies, to very small, such as molecules in liquids and gases (in cases where quantum mechanical fluctuations can be neglected, which is often the case). In between these extremes, Newton's equations of motion apply, litterally, to "everything that moves".

Exact analytical solutions of the equations of motion exist only for simple systems, of the types that are discussed in elementary classical mechanics courses, and therefore numerical integration methods are very important in practice. Here we will discuss some commonly used differential equation solvers and use them to study the dynamics of mechanical systems, including ones that exhibit chaotic dynamics. We will limit the discussion to a single moving body, although the methods can be easily generalized to many-body systems as well—dynamics of many-body systems will be discussed later in connection with molecular dynamics simulations.

While some of the numerical schemes that we will discuss here are particularly suitable for integrating classical equations of motions, we will also described methods, such as the classic Runge-Kutta algorithm, that are more generally applicable to a large class of ordinary differential equations. The discussion of systems with chaotic dynamics, although here introduced in the context of classical mechanics, is also of relevance more broadly in studies of nonlinear dynamics.

2 Basic algorithms for equations of motion

Consider a single object (here regarded as a point particle) with mass m moving in one dimension. With its time-dependent position denoted $x(t)$, the differential equation governing its dynamics is

$$\ddot{x}(t) = \frac{1}{m} F[x(t), \dot{x}(t), t], \tag{1}$$

where F is the total force acting on the particle, and $\dot{x}$ and $\ddot{x}$ are the first and second time derivatives of x. We have indicated that the force may depend on x, $\dot{x}$ and t. These dependencies typically come from from a position dependent static potential, a velocity dependent damping (friction), and a time-dependent driving force, but there are other natural posibilities as well, e.g., a position dependent friction.

To study the system numerically, it is convenient to rewrite the second-order differential equation (1) as two coupled first-order equations. Giving the velocity its standard symbol $v(t)$, we have

$$\begin{aligned} \dot{x}(t) &= v(t) \\ \dot{v}(t) &= \frac{1}{m} F[x(t), v(t), t]. \end{aligned} \tag{2}$$

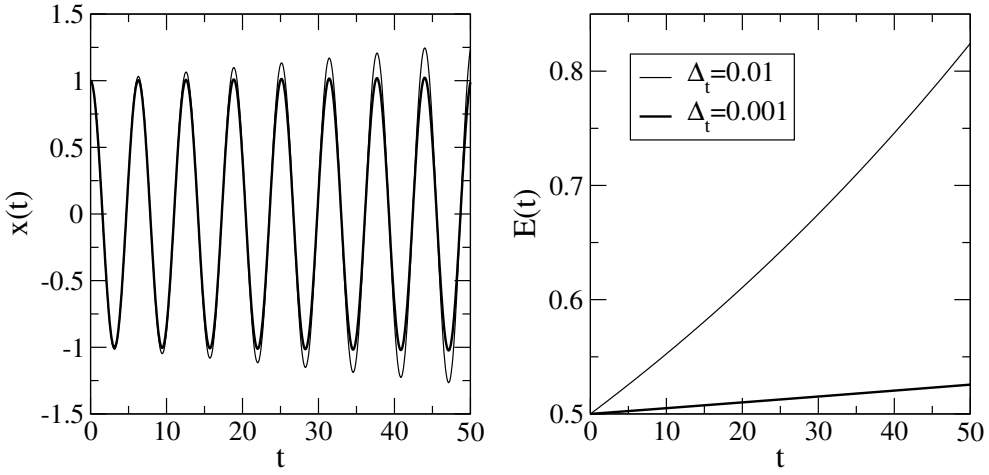

Figure 1: Time dependence of the position x and the energy $E = (1/2)kx^2 + (1/2)mv^2$ of a harmonic oscillator with $k = m = 1$ (which gives an oscillation period 2π) integrated using the Euler method with two different time steps; $\Delta_t = 10^{-2}$ and 10^{-3}.

To integrate this set of equations, we discretize the time-axis as $t = t_0, t_1, \ldots$, with a constant time step $t_{n+1} - t_n = \Delta_t$. The initial values $x_0 = x(t_0)$ and $v_0 = v(t_0)$ are used to start the integration process.

2.1 Euler algorithm

The simplest way to advance the time from t_n to t_{n+1} is to use the first-order approximation;

$$\begin{aligned} x_{n+1} &= x_n + \Delta_t v_n, \\ v_{n+1} &= v_n + \Delta_t a_n, \end{aligned} \tag{3}$$

where $a_n = F(x_n, v_n, t_n)/m$ is the acceleration. Clearly, the error made in each step of this algorithm is $O(\Delta_t^2)$. The method, which is called Euler's forward method, is in general not very useful in practice. For example, in systems with no damping or driving force, the energy should be conserved. However, with the Euler method the energy typically diverges with time, whereas in most higher-order methods the energy errors are bounded. Fig. 1 shows results obtained with the Euler method for the energy and the position of a harmonic oscillator with $k = m = 1$ ($F = -kx$). Even for a time step as as small as 10^{-3}, the energy error is as large as $\approx 5\%$ at $t = 50$ (corresponding to less than 9 oscillations); it grows faster than linearly with t.

There are several ways to proceed to derive more accurate, higher-order integration schemes; we will here discuss manipulations leading to a few practically useful formulas.

2.2 Leapfrog and Verlet algorithms

To simplify the discussion initially, we will here first assume that there is no damping, i.e., the force and the acceleration are velocity independent. We begin by writing the Taylor expansion of

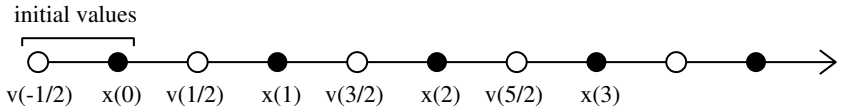

Figure 2: Position and velocity grids used in the leapfrog method. The position is calculated at integer multiples of the time step, $t_n = n\Delta_t$, $n = 0, 1, \ldots$, and the velocity is evaluated at the times $t_{n-1/2} = (n - 1/2)\Delta_t$ between these points. The integration starts with given $n = 0$ values.

$x_{n+1} = x(t_n + \Delta_t)$ to second order in Δ_t;

$$x(t_{n+1}) = x(t_n) + \Delta_t v(t_n) + \frac{1}{2}\Delta_t^2 a(x_n, t_n) + O(\Delta_t^3). \tag{4}$$

Noting that $v(t_n) + (\Delta_t/2)a(x_n, t_n) = v(t_{n+1/2})$ to order Δ_t^2, we can rewrite this as

$$x(t_{n+1}) = x(t_n) + \Delta_t v(t_n + \Delta_t/2) + O(\Delta_t^3). \tag{5}$$

We thus need a formula that propagates the velocity on a time grid with points $t_{n+1/2} = t_n + \Delta_t/2$, i.e., between the integer-labeled time points $t_n = t_0 + n\Delta_t$ used for the position. If we use the first-order expansion of the velocity, $v(t_{n+1/2}) = v(t_{n-1/2}) + \Delta_t a(t_{n-1/2}) + O(\Delta_t^2)$, the error remains $O(\Delta_t^3)$ in Eq. (5), but we have a problem since this requires the acceleration, and hence the position x (on which the force depends), on the grid points $t_{n+1/2}$ used only for the velocity. However, it appears intuitively clear, by symmetry, that we should actually use the acceleration at t_{n+1} to propagate the velocity from $t_{n+1/2}$ to $t_{n+3/2}$, i.e.,

$$v(t_{n+3/2}) = v(t_{n+1/2}) + \Delta_t a(x_{n+1}, t_{n+1}). \tag{6}$$

The scheme combining Eqs. (5) and (6) is often called the *leapfrog method*;

$$\begin{aligned} v_{n+1/2} &= v_{n-1/2} + \Delta_t a_n, \\ x_{n+1} &= x_n + \Delta_t v_{n+1/2}. \end{aligned} \tag{7}$$

The leapfrog grid is illustrated in Fig. 2.

We normally have initial conditions in the form of x_0 and v_0. In order to start the leapfrog method we need $v_{-1/2}$, which we can get, up to an error $\sim \Delta_t^2$, using

$$v_{-1/2} = v_0 - \frac{\Delta_t}{2} a_0 + O(\Delta_t^2). \tag{8}$$

The Δ_t^2 error may seem to spoil the $O(\Delta_t^3)$ error scaling of the leapfrog method. However, the $O(\Delta_t^3)$ error is a single-step error; we will see in Sec. 2.3 that the long-time error, i.e., after integrating a large number of steps, is $O(\Delta_t^2)$, and hence the error in (8) is of appropriate order.

It turns out that the position x in the leapfrog method in fact has a single-step error of order Δ_t^4, i.e., smaller than what might have been expected from the initial expansion (5). This can be seen most easily by deriving the method in another way, starting from two different Taylor expansions:

$$\begin{aligned} x_{n+1} &= x_n + \Delta_t v_n + \frac{1}{2}\Delta_t^2 a_n + \frac{1}{6}\Delta_t^3 \dot{a}_n + O(\Delta_t^4), \\ x_{n-1} &= x_n - \Delta_t v_n + \frac{1}{2}\Delta_t^2 a_n - \frac{1}{6}\Delta_t^3 \dot{a}_n + O(\Delta_t^4). \end{aligned} \tag{9}$$

Adding these two equations gives

$$x_{n+1} = 2x_n - x_{n-1} + \Delta_t^2 a_n + O(\Delta_t^4), \tag{10}$$

which is called the *Verlet algorithm*. It does not contain any velocities explicitly; the next x-value is obtained from two preceding x values. However, the Verlet algorithm is in fact completely equivalent to the leapfrog method (7). To see this, we use the fact that $(x_n - x_{n-1})/\Delta_t$ equals the velocity $v_{n-1/2}$, up to a correction $O(\Delta_t^2)$, and write

$$x_{n+1} = x_n + \Delta_t(v_{n-1/2} + \Delta_t a_n) + O(\Delta_t^4). \tag{11}$$

Noting that $v_{n-1/2} + \Delta_t a_n = v_{n+1/2}$, we get exactly the leapfrog method (7). It is important to note that the substitution $x_n - x_{n-1} \to \Delta_t v_{n-1/2}$ done in order to get Eq. (11) from Eq. (10) does not introduce any further errors in this equation, as it can be considered as a definition of the velocity in terms of a difference between the two x-values. Hence, the step error in the position is $O(\Delta_t^4)$ in the leapfrog method. For the velocity the error is $O(\Delta_t^2)$, due to the way it is defined as a discretized derivative. As we will see below in Sec. 2.3, the accumulated errors in x and v over a finite integration time both scale as (Δ_t^2).

An important feature of the leapfrog algorithm is that it is time-reversible, i.e., if we calculate x_{n+1} from x_n and $v_{n+1/2}$ according to Eqs. (7), and then reverse the time direction to go back to x_n, we use the same velocity $v_{n+1/2}$ as in the forward step, up to the change of its sign. Therefore, in the absence of numerical round-off errors, we arrive back exactly at the original x_n. This is not the case with the Euler metod, Eqs. (3), where we use v_n to go from x_n to x_{n+1}, but $-v_{n+1}$ to go backwards from x_{n+1} to x_n. If we carry out several leapfrog integration steps backwards in time, $t = t_N, \ldots, t_0$, after having completed a calculation from t_0 to t_N, we should arrive back at the original initial conditions at $t = t_0$. This way the time-reversibility can in fact be used as a check of the sensitivity of a calculation to round-off errors (which will slightly break the symmetry).

An important consequence of the time-reversibility is that the errors are bounded for a system with periodic motion. This follows because integrating backward or forward in time for a full period T involves the acceleration at exactly the same x points (assuming now that the period is exactly a multiple of the time step). Hence, the error δ in some quantity, e.g., the energy, has to be the same at $t = \pm T$; $\delta(T) = \delta(-T)$. Integrating forward from $t = -T$ to 0, starting from the point reached in a previous integration from $t = 0$ to $-T$, we would expect an additional error similar to $\delta(T)$, because the initial conditions of these two integrations only differ by the very small amount $\delta(-T)$, so that the total error after a backward and a forward integration should be $\approx 2\delta(T)$. To be consistent with the time-reversibility, according to which this second integration in fact must bring us back to the original starting point and hence zero error, we must have $\delta(\pm T) = 0$. The error only vanishes completely at times $t - t_0 = T, 2T, \ldots$, but the important point is that there can be no steady error increase, which does affect methods with no time-reversal symmetry (as shown for the Euler method in Fig. 1). The general form of the long-time error scaling is discussed in detail in Sec. 2.3.

The bounded error of the leapfrog method when applied to systems with periodic motion is illustrated in Fig. 3, which shows results for the energy of the same harmonic oscillator as the one studied in Sec. 2.1 using the Euler method. The energy here oscillates with a period half of the periodicity 2π of the system. The vanishing of the error at every half period in this case is a consequence of the symmetric potential, which was not assumed in the discussion above. Note also

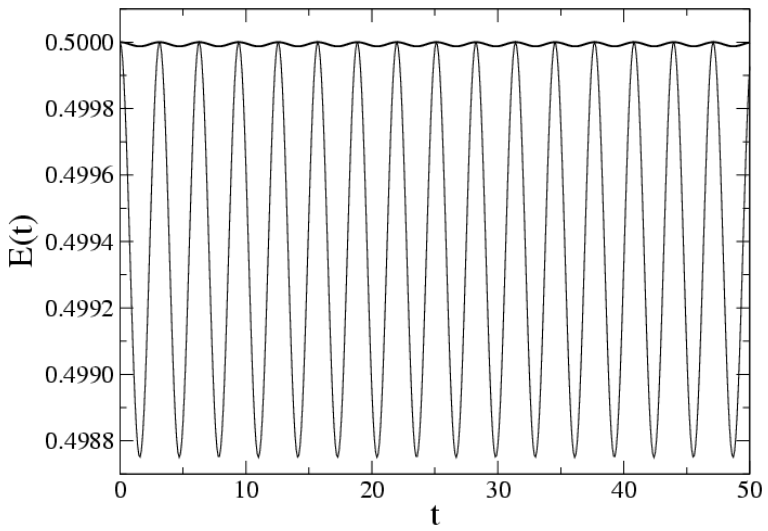

Figure 3: Time dependence of the energy of a harmonic oscillator with $k = m = 1$, integrated using the leapfrog method with $\Delta_t = 10^{-1}$ (thinner curve) and 10^{-2} (thicker curve).

how small the deviations are from the true energy $E = 1/2$ (compare with Fig. 1), even for Δ_t as large as 0.1.

There is yet a third equivalent formulation of the Verlet method, called the *velocity Verlet* algorithm. It is obtained from the original Verlet algorithm as follows: Adding x_{n+1} on both sides of Eq. (10),

$$2x_{n+1} = x_{n+1} + 2x_n - x_{n-1} + \Delta_t^2 a_n + O(\Delta_t^4). \tag{12}$$

we can define the velocity using a two-step difference;

$$v_n = \frac{1}{2\Delta_t}(x_{n+1} - x_{n-1}), \tag{13}$$

which when used in (12) gives the position in terms of x and v at the previous step only;

$$x_{n+1} = x_n + \Delta_t v_n + \frac{1}{2}\Delta_t^2 a_n. \tag{14}$$

In order to obtain an equation for the velocity, we first write the original Verlet equation (10) for x_n instead of x_{n+1};

$$x_n = 2x_{n-1} - x_{n-2} + \Delta_t^2 a_{n-1}, \tag{15}$$

which we add to (10). Rearranging the result in the following way;

$$x_{n+1} - x_{n-1} = x_n - x_{n-2} + \Delta_t^2(a_{n-1} + a_n), \tag{16}$$

and using the velocity definition (13), we obtain an equation for v_n;

$$v_n = v_{n-1} + \frac{1}{2}\Delta_t(a_{n-1} + a_n). \tag{17}$$

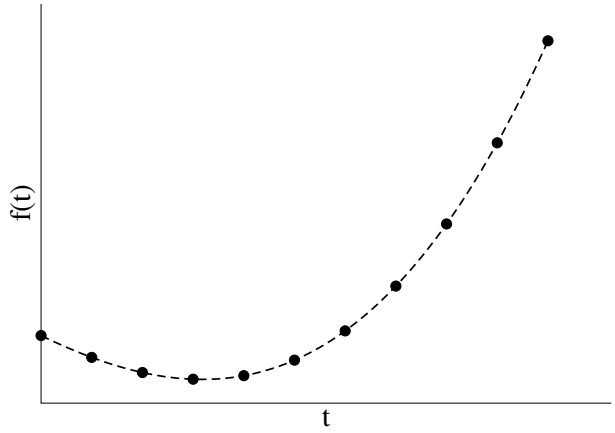

Figure 4: A function that is evaluated on a grid with spacing Δ_t (solid circles) can be approximated by an interpolating polynomial between those points (dashed curve), provided that the function is smooth on the scale Δ_t.

Shifting the time step by one and again writing down Eq. (14) for the position, we arrive at the velocity Verlet algorithm:

$$\begin{aligned} x_{n+1} &= x_n + \Delta_t v_n + \frac{1}{2}\Delta_t^2 a_n, \\ v_{n+1} &= v_n + \frac{1}{2}\Delta_t (a_{n+1} + a_n). \end{aligned} \tag{18}$$

This formulation of the Verlet algorithm is completely equivalent to (10) and (7) as far as the propagation of the position is concerned. It may preferrable to the leapfrog method in cases where there is some reason to use the same time grid for x and v, but note that more operations are required at each step of the velocity Verlet algorithm (however, the number of calls to the force/acceleration function is the same, which is typically the part dominating the processor time). It should also be noted that although the algorithms give identical results (up to round-off errors) for x_n, the accuracy in the velocity is four times higher in the leapfrog method becuase it is there defined using x points separated by a single time step, instead of two steps in the velocity Verlet method. However, nothing of course prohibits us from also calculating $v_{n+1/2}$ in terms of x_n and x_{n+1} in the velocity Verlet method or v_n in terms of x_{n-1} and x_{n+1} in the leapfrog method. In view of this, all results are completely equivalent in the two formulations.

2.3 Error build-up in the Verlet/leapfrog method

As we have seen above, the single-step error in the Verlet/leapfrog algorithm is $O(\Delta_t^4)$ for the position and $O(\Delta_t^2)$ for the velocity. We are clearly also very interested in the accumulated errors after a large number of steps have been carried out.

To study the accumulated error in the position x calculated using the Verlet/leapfrog method, we use the original Verlet form of the algorithm, Eq. (10). We introduce a symbol for the deviation δ_n of the calculated value x_n from the exact solution $x_n^{\rm ex}$,

$$x_n = x_n^{\rm ex} + \delta_n, \tag{19}$$

and insert this in Eq. (10). After rearranging, this gives

$$\delta_{n-1} - 2\delta_n + \delta_{n+1} = -(x^{\rm ex}_{n-1} - 2x^{\rm ex}_n + x^{\rm ex}_{n+1}) + \Delta_t^2 a_n + O(\Delta_t^4). \tag{20}$$

Under the assumption that the true solution, the acceleration (i.e., the force), and the deviation are all smooth functions on a time scale set by Δ_t (which is true for Δ_t smaller than some $\Delta_t^{\rm max}$ if the true solution and the force are well behaved functions), we can imagine constructing high-order polynomials that go through all the (t_n, δ_n) and $(t_n, x^{\rm ex}_n)$ points, as illustrated in Fig. 4. We can then use these interpolating polynomials to work in continuum time and use the second derivative of a function [$f(t) = \delta(t)$ or $x^{\rm ex}(t)$] in place of the discretized versions appearing in Eq. (20),

$$\frac{d^2 f(t_n)}{dt^2} = \frac{1}{\Delta_t}\frac{d}{dt}[f(t_{n+1/2}) - f(t_{n-1/2})] = \frac{1}{\Delta_t^2}(f_{n-1} - 2f_n + f_{n+1}), \quad \Delta_t \to 0, \tag{21}$$

to obtain from (20)

$$\ddot{\delta}(t_n) = -\ddot{x}^{\rm ex}(t_n) + a(t_n) + O(\Delta_t^2), \tag{22}$$

even though Δ_t is fixed and finite. The error introduced by this "quasi-continuum" approach is given by the order of the imagined interpolating polynomials and will clearly be much smaller than the $O(\Delta_t^4)$ error of the Verlet formula we are working with (under the assumption of smoothness on the scale set by Δ_t). By definition of the exact solution, $\ddot{x}^{\rm ex}(t_n) = a(t_n)$, and hence in Eq. (20) we are left with

$$\ddot{\delta}(t_n) \sim \Delta_t^2. \tag{23}$$

We can write an expression for the error $\delta(T)$ after a large number of integration steps in terms of the second derivative ($T = t_N - t_0$; we will set $t_0 = 0$):

$$\delta(T) = \int_0^T dt \dot{\delta}(t) + \delta(0) = \int_0^T dt \left(\int_0^t dt' \ddot{\delta}(t') - \dot{\delta}(0) \right) + \delta(0). \tag{24}$$

By definition of the initial conditions, the error $\delta(0) = 0$. Since the Verlet formula is completely symmetric with respect to reversal of the time direction ($n \to -n$), the Taylor expansion of the error around t_0 can have no contribution of first order (or any odd order), and hence also $\dot{\delta}(0) = 0$. With the second derivative bounded by the quadratic form (23), the integral (24) clearly can scale no worse than as $T^2\Delta_t^2$. Hence, the error in x after a finite number of steps scales in the worst case scenario as Δ_t^2 in the time discretization and as T^2 in the total integration time. Since there is an unknown prefactor in the Δ_t dependence of the error in Eq. (23)—the sign can even be mixed—the T-scaling can be much better in practice, as we have seen explicitly in Fig. 3 in the case of the harmonic oscillator. Note again that the above arguments hold only when Δ_t is sufficiently small for the solution to be smooth on this scale. The way the velocity is defined as a discrete derivative of the coordinate, its error will clearly also scale as Δ_t^2 for any t.

2.4 Verlet/leapfrog methods for damped systems

In the derivation of the Verlet algorithm, and its leapfrog formulation, we assumed that the force-function contains no explicit velocity dependence. With a velocity dependent force, $F(x, v, t)$, the problem is that the acceleration $a_n = a(x_n, v_n, t_n)$ in (7) should be evaluated at the time-points

corresponding to the position x_n, on which we do not have the velocity. To circumvent this problem, we first separate the damping term from the rest of the force terms and write

$$a(x, v, t) = \frac{1}{m}[F(x, t) - G(v)]. \tag{25}$$

If we simply approximate $a(x_n, v_n, t_n)$ by $[F(x_n, t_n) - G(v_{n-1/2})]/m$, we make an error of order Δ_t. Since a is multiplied by Δ_t^2 to obtain x_{n+1}, the resulting algorithm acquires a contribution to the x error scaling as Δ_t^3; an order lower than the Verlet algorithm without damping term. To remedy this, we can use a scheme based on an intermediate estimation $\hat{x}_{n+1}$ of the position, obtained using the above approximation $G(v_{n-1/2})$ in place of $G(v_n)$. We can subsequently correct for the $O(\Delta_t^3)$ error in $\hat{x}_{n+1}$ by doing a second step, where we use a better approximation of the velocity; $v_n = (\hat{x}_{n+1} - x_{n-1})/(2\Delta_t)$. The error in this velocity, and hence in $G(v_n)$, is $O(\Delta_t^2)$, i.e., one order smaller than in the first approximation and sufficient to render x_{n+1} calculated with it accurate to the same order as in the original leapfrog algorithm. To summarice this modified algorithm, these are the steps to be performed in the leapfrog version of the Verlet algorithm with a damping term:

$$\begin{aligned}
\hat{v}_{n+1/2} &= v_{n-1/2} + \Delta_t[F(x_n, t_n) - G(v_{n-1/2})]/m, \\
\hat{x}_{n+1} &= x_n + \Delta_t \hat{v}_{n+1/2}, \\
v_n &= (\hat{x}_{n+1} - x_{n-1})/(2\Delta_t), \\
v_{n+1/2} &= v_{n-1/2} + \Delta_t[F(x_n, t_n) - G(v_n)]/m, \\
x_{n+1} &= x_n + \Delta_t v_{n+1/2}.
\end{aligned} \tag{26}$$

This algorithm requires more work than the simple leapfrog method without damping. However, in cases where the processor time is dominated by evaluating the function F, the differences are in practice only minor.

In the important special case of linear damping, i.e.,

$$G(v) = \gamma v = \frac{x_{n+1} - x_{n-1}}{2\Delta_t} + O(\Delta_t^2), \tag{27}$$

the algorithm can be simplified. Starting from the Verlet form (10) we can write

$$x_{n+1} = 2x_n - x_{n-1} + \frac{\Delta_t^2}{m}[F_n - \gamma(x_{n+1} - x_{n-1})/(2\Delta_t)] + O(\Delta_t^4), \tag{28}$$

which we can rearrange as

$$x_{n+1}(1 + \gamma\Delta_t/2m) = 2x_n - x_{n-1}(1 - \gamma\Delta_t/2m) + \frac{\Delta_t^2}{m}F_n + O(\Delta_t^4), \tag{29}$$

or

$$\begin{aligned}
x_{n+1} &= x_n + \frac{(1 - \gamma\Delta_t/2m)(x_n - x_{n-1}) + \Delta_t^2 F_n/m}{1 + \gamma\Delta_t/2m}, \\
&= x_n + \frac{\Delta_t(1 - \gamma\Delta_t/2)v_{n-1/2} + \Delta_t^2 F_n}{1 + \gamma\Delta_t/2}.
\end{aligned} \tag{30}$$

In this expression we can identify the velocity $v_{n+1/2}$ and obtain the leapfrog algorithm in the presence of linear damping;

$$\begin{aligned}
v_{n+1/2} &= \frac{(1 - \gamma\Delta_t/2m)v_{n-1/2} + \Delta_t F_n/m}{1 + \gamma\Delta_t/2m}, \\
x_{n+1} &= x_n + \Delta_t v_{n+1/2}.
\end{aligned} \tag{31}$$

Using the velocity Verlet algorithm (18) with damping, we have a problem analogous to that in the leapfrog method; to evaluate v_{n+1} we need a_{n+1}, which itself depends explicitly on v_{n+1}. We can proceed in a way similar to what we did above, first obtaining an estimate $\hat{v}_{n+1}$ based on approximating $a_{n+1} = a(x_{n+1}, v_{n+1}, t_{n+1})$ by $a(x_{n+1}, v_n + a_n\Delta_t, t_{n+1})$ and then refining;

$$\begin{aligned} x_{n+1} &= x_n + \Delta_t v_n + \frac{1}{2}\Delta_t^2 a_n, \\ \hat{v}_{n+1} &= v_n + \frac{1}{2}\Delta_t[a_n + a(x_{n+1}, v_n + \Delta_t a_n, t_{n+1})], \\ v_{n+1} &= v_n + \frac{1}{2}\Delta_t[a_n + a(x_{n+1}, \hat{v}_{n+1}, t_{n+1})]. \end{aligned} \tag{32}$$

Also in this case the algorithm can be further simplified in the case of linear damping. The expression corresponding to Eq. (16) in the presence of linear damping is

$$x_{n+1} - x_{n-1} = x_n - x_{n-2} + \frac{\Delta_t^2}{m}(F_{n-1} + F_n) - \frac{\gamma}{2}\Delta_t(x_{n+1} - x_{n-1} + x_n - x_{n-2}), \tag{33}$$

which leads to the following algorithm [the formula for the position in the original velocity Verlet algorithm with no damping, Eq. (18), remains unchanged]:

$$\begin{aligned} x_{n+1} &= x_n + \Delta_t v_n + \frac{1}{2}\Delta_t^2 a_n, \\ v_{n+1} &= \frac{1}{1 + \Delta_t\gamma/2m}[v_n(1 - \Delta_t\gamma/2m) + \frac{\Delta_t}{2m}(F_{n-1} + F_n)]. \end{aligned} \tag{34}$$

2.5 The Runge-Kutta method

The Runge-Kutta (RK) method can be considered the most classic of all high-order schemes. There is a whole range of methods of this type of different orders, but the RK name is most commonly associated with the fourth-order variant [discretization error $O(\Delta_t^5)$]. We will here outline one way of constructing this algorithm; however, without proving the error scaling. We will first give a complete proof of the much simpler second-order RK method.

For simplicity, before considering the equations of motion, we will consider the slightly easier case of a single first-order differential equation,

$$\dot{x}(t) = f[x(t), t]. \tag{35}$$

The second-order RK method for this equation corresponds to the mid-point rule of function integration: If we know the value of the function $f[x(t), t]$ at the mid-point of an interval $[t_n, t_{n+1}]$, its integral is known up to an error $O(\Delta_t^3)$:

$$\int_{t_n}^{t_{n+1}} f[x(t), t]dt = \Delta_t f[x(t_{n+1/2}), t_{n+1/2}] + O(\Delta_t^3). \tag{36}$$

In the case of a function $f(t)$, we could simply evaluate it at $t_{n+1/2}$, but in the present case we do not know the value of $x_{n+1/2} = x(t_{n+1/2})$ and hence we need to approximate it. We only need an

approximant linear in Δ_t to keep the error of the integral cubic, so we can simply use the Euler formula for (35):

$$\hat{x}_{n+1/2} = x_n + \frac{\Delta_t}{2} f(x_n, t_n) + O(\Delta_t^2), \tag{37}$$

where $\hat{}$ is again used to indicate an intermediate step of the calculation. We then arrive at the second-order RK formula,

$$x_{n+1} = x_n + \Delta_t f(\hat{x}_{n+1/2}, t_{n+1/2}) + O(\Delta_t^3), \tag{38}$$

which is typically written in the algorithm form

$$\begin{aligned} k_1 &= \Delta_t f(x_n, t_n), \\ k_2 &= \Delta_t f(x_n + k_1/2, t_{n+1/2}), \\ x_{n+1} &= x_n + k_2. \end{aligned} \tag{39}$$

In the case of the equations of motion for x and v, we use the Euler formula for both quantities in the first step to obtain approximants for $x_{n+1/2}$ and $v_{n+1/2}$ and then use these to obtain better estimates in the same way as we did above, in Sec. 2.4, giving

$$\begin{aligned} k_1 &= \Delta_t a(x_n, v_n, t_n), \\ l_1 &= \Delta_t v_n, \\ k_2 &= \Delta_t a(x_n + l_1/2, v_n + k_1/2, t_{n+1/2}), \\ l_2 &= \Delta_t (v_n + k_1/2), \\ v_{n+1} &= v_n + k_2, \\ x_{n+1} &= x_n + l_2. \end{aligned} \tag{40}$$

This algorithm is rarely used in practice, since the Verlet/leapfrog method is both simpler and more accurate. The algorith does, however, serve as a good warm-up for studying the significantly more complicated fourth-order RK algorithm.

We again first consider the single equation (35). Again using results from simple function integration, we know that if the values of the function $f[x(t), t]$ are evaluated at the time points $t_n, t_{n+1/2}$, and t_n, we can apply Simpson's formula to obtain the integral over the range $[t_n, t_{n+1}]$ up to an error $O(\Delta_t^5)$, i.e.,

$$x_{n+1} = x_n + \frac{\Delta_t}{6}(f_n + 4f_{n+1/2} + f_{n+1}) + O(\Delta_t^5). \tag{41}$$

We hence need to find approximations for $f_{n+1/2}$ and f_{n+1} up to errors $O(\Delta_t^4)$. The somewhat obscure scheme used for this purpose is illustrated in Fig. 5. As in the second-order derivation above, we begin by using an Euler formula to obtain an approximation for $x_{n+1/2}$. This time there will be two such estimates, and we use a superscript to distinguish between them;

$$\begin{aligned} \hat{x}^1_{n+1/2} &= x_n + \frac{\Delta_t}{2} f(x_n, t_n) \\ \hat{x}^2_{n+1/2} &= x_n + \frac{\Delta_t}{2} f(\hat{x}^1_{n+1/2}, t_{n+1/2}). \end{aligned} \tag{42}$$

Since f is the derivative of x, what we have done is to first extrapolate $x_{n+1/2}$ from x_n using derivative information at the initial point (x_n, t_n), denoted z_0 in Fig. 5. The second extrapolation

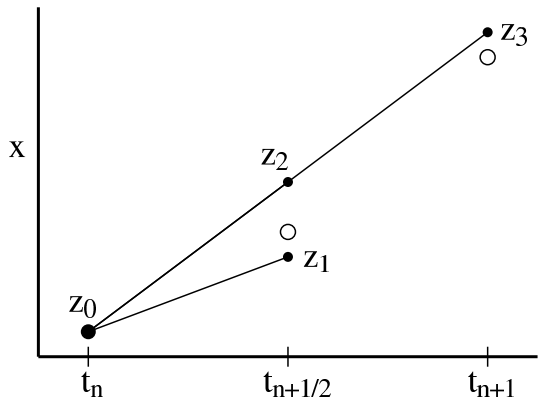

Figure 5: Illustration of the scheme used to estimate the values of the function f at the true (t, x) points $(t_{n+1/2}, x_{n+1/2})$ and (t_{n+1}, x_{n+1}) (here shown as open circles) based on the known point $z_0 = (t_n, x_n)$ (large solid circle). A point z_1 is first generated using $f(z_0)$ as the derivative. The points z_2 and z_3 are obtained using $f(z_1)$ as the derivative.

is based on derivative information at the estimated point $(\hat{x}^1_{n+1/2}, t_{n+2})$, denoted z_1 in the figure, but still using (x_n, t_n) as the point of departure. This second point at $t_{n+1/2}$ is denoted z_2 in the figure. The idea of this procedure is, roughly, that if the function f is smooth on the scale Δ_t, the actual point $x_{n+1/2}$ should be between z_1 and z_2. In the RK method the desired function value $f_{1+n/2}$ is approximated by the average of the function at these two estimated points;

$$f_{n+1/2} \approx \frac{1}{2}[f(x^1_{n+1/2}, t_{n+1/2}) + f(x^2_{n+1/2}, t_{n+1/2})]. \tag{43}$$

We will not prove here the remarkable fact that this approximation has the required accuracy, i.e., an $O(\Delta_t^4)$ error. In order to estimate x_{n+1}, the function value at $(\hat{x}^2_{n+1/2}, t_{n+1/2})$ (i.e., z_2) is used to obtain an approximation of the derivative,

$$\hat{x}_{n+1} = x_n + \Delta_t f(x^2_{n+1/2}, t_{n+1/2}). \tag{44}$$

This is in the spirit that the derivative should be evaluated at the mid-point between (x_n, t_n) and (x_{n+1}, t_{n+1}). Not knowing the actual point, we use the estimate z_2 for this. Again, it can be proven that this leads to an error $O(\Delta_t^4)$ in x_{n+1}. The fourt-order RK algorithm resulting from the above procedures is traditionally written as

$$\begin{aligned} k_1 &= \Delta_t f(x_n, t_n), \\ k_2 &= \Delta_t f(x_n + k_1/2, t_{n+1/2}), \\ k_3 &= \Delta_t f(x_n + k_2/2, t_{n+1/2}), \\ k_4 &= \Delta_t f(x_n + k_3, t_{n+1}), \\ x_{n+1} &= x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned} \tag{45}$$

The correctness of this scheme up to an error of order $O(\Delta_t^5)$ can be proven using a Taylor expansion.

Before adapting the RK algorithm to equations of motion, we write down a more general form of

the RK scheme for two coupled equations;

$$\begin{aligned} \dot{x}(t) &= f(x,y,t), \\ \dot{y}(t) &= g(x,y,t), \end{aligned} \tag{46}$$

for which the RK algorithm generalizes to

$$\begin{aligned} k_1 &= \Delta_t f(x_n, y_n, t_n), \\ l_1 &= \Delta_t g(x_n, y_n, t_n), \\ k_2 &= \Delta_t f(x_n + k_1/2, y_n + l_1/2, t_{n+1/2}), \\ j_2 &= \Delta_t g(x_n + k_1/2, y_n + l_1/2, t_{n+1/2}), \\ k_3 &= \Delta_t f(x_n + k_2/2, y_n + l_2/2, t_{n+1/2}), \\ l_3 &= \Delta_t g(x_n + k_2/2, y_n + l_2/2, t_{n+1/2}), \\ k_4 &= \Delta_t f(x_n + k_3, y_n + l_3, t_{n+1}), \\ l_4 &= \Delta_t g(x_n + k_3, y_n + l_3, t_{n+1}), \\ x_{n+1} &= x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\ y_{n+1} &= y_n + \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4), \end{aligned} \tag{47}$$

which in turn easily generalizes to any number of coupled equations.

In the case of the equations of motion, we make the identification $x \to v$, $y \to x$, $f \to a$, $g \to v$ (note that v corresponds to a function $g(x,y,t) = x$ in the notation used above). We then obtain

$$\begin{aligned} k_1 &= \Delta_t a(x_n, v_n, t_n), \\ l_1 &= \Delta_t v_n, \\ k_2 &= \Delta_t a(x_n + l_1/2, v_n + k_1/2, t_{n+1/2}), \\ l_2 &= \Delta_t (v_n + k_1/2), \\ k_3 &= \Delta_t a(x_n + l_2/2, v_n + k_2/2, t_{n+1/2}), \\ l_3 &= \Delta_t (v_n + k_2/2), \\ k_4 &= \Delta_t a(x_n + l_3, v_n + k_3, t_{n+1}), \\ l_4 &= \Delta_t (v_n + k_3), \\ v_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\ x_{n+1} &= x_n + \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4). \end{aligned} \tag{48}$$

The RK method is clearly not time-reversal symmetric, and hence the errors are unbounded even for periodic motion. Fig. 6 shows the time dependence of the energy obtained using the RK method for the same harmonic oscillator that was previously studied with the Euler (Fig. 1) and leapfrog (Fig. 3) methods. Comparing the RK and leapfrog methods at $\Delta = 0.1$, it can be seen that the errors are considerably smaller in the case of the RK method within the time range shown. However, the error grows linearly with time, and hence for long times the leapfrog method will in fact perform better in this case although its single-step error scaling is worse. However, for aperiodic motion, or motion with a very long period, there may be no practical advantage of the bounded error of the

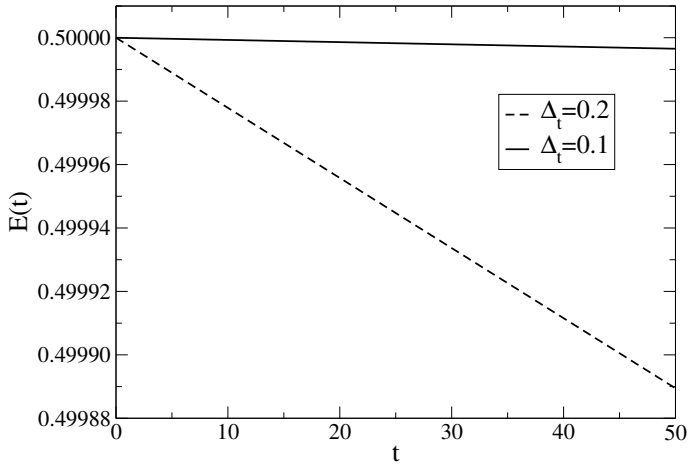

Figure 6: Time dependence of the energy of a harmonic oscillator with $k = m = 1$ integrated using the 4th-order Runge-Kutta method with time steps $\Delta_t = 0.2$ and 0.1.

leapfrog method—the error can still grow large within a period—and then the RK method is often preferrable.

An advantage of the RK method is that it can be used with a variable time step. This is not possible in the Verlet/leapfrog method, because data at t_n and $t_{n+1/2}$ are needed to advance the time to t_{n+1}. Using different time-steps $t_{n+1} - t_n$ and $t_n - t_{n-1}$ would clearly ruin the symmetry of the scheme and lead to a worse error scaling. In the RK method, only data at t_n are needed to advance to t_{n+1}, and hence Δ_t can be chosen arbitrarily in each step. There are adaptive methods that use this property of the Runge-Kutta scheme to automatically adjust the time step to keep the error within specified limits.

2.6 Motion in more than one dimension

The methids we have discussed above generalize very easily to motion in more than one dimension. The equations of motion become vector equations;

$$\begin{aligned} \dot{x}(t) &= \vec{v}(t) \\ \dot{\vec{v}}(t) &= \frac{1}{m}\vec{F}[\vec{x}(t), \vec{v}(t), t], \end{aligned} \tag{49}$$

which separate into equations for the components x_α, v_α ($\alpha = 1, 2, 3$ in three dimensions). These equations are coupled only through the force function, the components F_α of which typically depend on all components of $\vec{x}$ and, in the case of damped motion, $\vec{v}$.

As a simple example, we consider planetary motion, which takes place in a single plane (in the case of a single planet considered here), i.e., we study the equations of motion in two dimensions. For a planet of mass m moving in the gravitational field of a star with mass M, the force experienced by m is given by

$$\vec{F}(r) = -\frac{GMm}{r^3}\vec{r}, \tag{50}$$

where $\vec{r}$ is the position vector of the planet with respect to the star and $r = |\vec{r}|$. In principle this is a two-body problem, but a result of elementary mechanics is that it can be reduced to a one-body problem for a reduced mass $\mu = Mm/(m+M)$. We will here for simplicity assume that $M \gg m$ and treat the star as a stationary body, using the original m for the mass of the planet. We then have the equations of motion for the x and y coordinates of the planet:

$$\begin{aligned} \dot{x} &= v_x, \\ \dot{v}_x &= -GMx/r^3, \\ \dot{y} &= v_y, \\ \dot{v}_y &= -GMy/r^3, \end{aligned} \tag{51}$$

where $r = \sqrt{x^2+y^2}$. The leapfrog algorithm (7) for these equations is

$$\begin{aligned} v_x(n+1/2) &= v_x(n-1/2) - \Delta_t GMx(n)[x^2(n)+y^2(n)]^{-3/2}, \\ x(n+1) &= x(n) + \Delta_t v_x(n+1/2), \\ v_y(n+1/2) &= v_y(n-1/2) - \Delta_t GMy(n)[x^2(n)+y^2(n)]^{-3/2}, \\ y(n+1) &= y(n) + \Delta_t v_y(n+1/2). \end{aligned} \tag{52}$$

This scheme of course generalizes to other types of forces. The Runge-Kutta method can also easily be used in more than one dimension.

2.7 Program example; taking down a satellite

As an example of a numerical solution of equations of motion, we will here show how to calculate the orbit of a satellite that is subject to earth's gravitational force, atmospheric drag, as well as the thrust of its own internal rocket motor. Specifically, we will study the case of a satellite that is to be brought back to earth after having served its mission in a circular orbit, at a specified altitude. At the start of the calculation, the rocket motor, which produces a force directed opposite to the satellite's velocity vector, is turned on for a specified duration of time. As the satellite slows down, it exits the orbit and begins to accelerate towards earth. As it enters the atmosphere, air drag will slow down the fall, and eventually the satellite crashes onto the ground. The program, which is available on the course web-site in the file 'crash.f90', produces a file with the time dependence of the altitude, the angular position, and the velocity.

2.7.1 Modeling the forces acting on a satellite

The problem is illustrated schematcally in Fig. 7. The satellite has mass m and its position and velocity vectors are $\vec{r}$ and $\vec{v}$, respectively. Its dynamics is governed by a force

$$\vec{F} = F_{\text{gravity}}(r)\vec{e}_r + F_{\text{rocket}}(t)\vec{e}_v + F_{\text{drag}}(r,v)\vec{e}_v, \tag{53}$$

which is a sum of forces due to gravitation, the internal rocket motor, and air drag. All the forces in this problem keep the motion in the plane defined by the original circular orbit, and hence we consider a two dimensional space; $\vec{r} = x\vec{e}_x + y\vec{e}_y$. The explicit dependencies of the forces on $\vec{r}$,

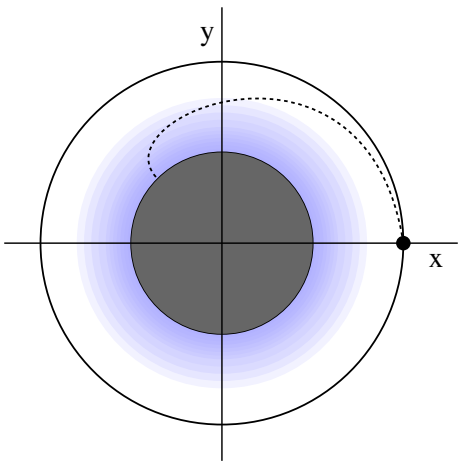

Figure 7: Schematic illustration of the satellite landing problem. Earth, seen from above the north pole, is shown in dark gray; the atmosphere is indicated by the lighter shade of gray. The satellite is initially in a circular orbit around the equator. At the start of the calculation it is at the position indicated, at angle $\alpha = 0$ with the x axis. The rocket motor is turned on and the satellite begins to descend, eventually crashing at a location to be determined by the calculation.

$\vec{v}$, and t have been indicated above; the unit vectors are in the direction of $\vec{r}$ and $\vec{v}$, respectively; $\vec{e}_r = \vec{r}/r$ and $\vec{e}_v = \vec{v}/v$. We will now specify their exact forms of the force terms.

We already discussed motion in earth's gravitational field above in Sec. 2.6; the force F_{gravity} is given by Eq. (50), where M is earth's mass, $M_e = 5.976 \cdot 10^{24}$ kg, and the constant of gravitation $G = 6.672 \cdot 10^{-11}$ m^3/kgs^2. To relate the initial altitude h_0 to r, and to determine when the satellite crashes, we also need earth's radius; $r_e = 6378$ km.

For the braking process used to bring the satellite out of its orbit, we assume that the rocket engine produces a thrust corresponding to a constant decelerating force opposite to the direction of motion. We neglect the reduction of the satellite's mass as fuel is consumed, i.e., we assume a contribution to the total acceleration that is constant in magnitude;

$$\frac{\vec{F}_{\text{rocket}}}{m} = -\Theta(T_{\text{break}} - t) B \vec{e}_v, \tag{54}$$

where T_{brake} is the time the motor stays on, and $\Theta(t)$ is the theta function; $\Theta(t) = 1$ for $t \geq 0$, $\Theta(t) = 0$ for $t < 0$. In the program, we will take $B = 5\text{m/s}^2$ for the deceleration.

We also need a model of atmospheric drag. We will assume that the damping force due to the air depends quadratically on the velocity;

$$\vec{F}_{\text{drag}}(h) = C_d \rho(h) v^2 \vec{e}_v \tag{55}$$

where C_d is the drag coefficient and $\rho(h)$ is the density of the atmosphere at altitude h and $h = r - r_e$. The quadratic form is appropriate at high velocity, whereas a linear velocity dependence is closer to reality at low velocities. Here we will not incorporate a cross-over to such a low-velocity linear form, because the velocity will be high most of the time and the details would anyway be highly dependent on the shape of the satellite. We will adjust the drag coefficient C_d, which has the unit m^2, so that a reasonable terminal velocity for a free fall is achieved.

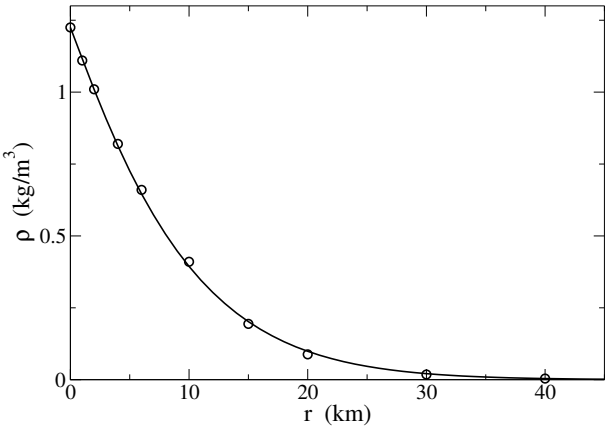

Figure 8: Density of the atmosphere as a function of altitude above sea level. The circles are values for the so-called "1976 standard atmosphere", and the curve was calculated using the model (57) with $k_1 = 1.2 \cdot 10^4$ m and $k_2 = 2.2 \cdot 10^4$ m.

The terminal velocity is determined by $F_{\rm drag}(h = 0, v)/m = g$, or

$$\frac{C_d}{m} = \frac{g}{\rho(0)v^2}. \tag{56}$$

Our values for GM and earth's radius r_e correspond to a gravitational acceleration $g = 9.77355$ m/s^2. Since we are just interested in a rough estimate, we will neglect the fact that the true terminal velocity will be affected by the non-constant air denity, and, to a lesser extent, the altitude dependence of g. We can later use our program to check the true terminal velocity of a body of given C_d/m that drops from a high altitude. Let us here assume that the shape and mass of the satellite are such that the terminal velocity (56) is 100m/s if ρ and g are taken as constants equal to their sea-level values. With the density of air at sea level taken as 1.225 kg/m^3, we then get $C_d/m \approx 8 \cdot 10^{-4}$ m^2/kg, which we will use in the program.

We will use the following simple model for the atmospheric density

$$\rho(h) = \cdot \exp\left[-\left(\frac{h}{k_1} + \left(\frac{h}{k_{3/2}}\right)^{3/2}\right)\right] \cdot 1.225 \text{ kg/m}^3, \tag{57}$$

with k_1 and k_2 adjusted to give an approximate agreement with tabulated values for the density. This form was arrived at simply by trial and error, using tabulated data up to $r = 40$km. With $k_1 = 1.2 \cdot 10^4$ m and $k_2 = 2.2 \cdot 10^4$ m the agreement with the data is good, as shown in Fig. 8. For altitudes > 40 km it is not clear how well this model reproduces actual conditions. In fact, the atmospheric density at very high altitudes varies considerably with time and is influenced by, e.g., the solar wind (and, of course, even at low altitudes the actual air density depends on the weather!). The model density (57) suffices for semi-realistic illustration purposes.

2.7.2 Computer program

The computer program carrying out the calculations is available on the course web site, in the file 'crash.f90'. We here discuss its main subroutines and functions.

The program uses a module `systemparam`, which appears at the beginning of the program file and contains some of the parameters and variables used in several of the program units:

```
module systemparam

real(8), parameter :: pi=3.141592653589793d0
real(8), parameter :: gm=3.987d14  ! Earth's mass times constant of gravity
real(8), parameter :: arocket=5.d0 ! Retardation due to rocket motor
real(8), parameter :: dragc=8.d-4  ! Air drag coefficient
real(8), parameter :: re=6.378d6   ! Earth's radius

real(8) :: dt,dt2  ! time step, half of the time step
real(8) :: tbrake  ! run-time of rocket motor

end module systemparam
```

The parameters, all of which were discussed above, are explained in the module by comments; SI units are used throughout the program. The time step `dt` and the run-time `tbrake` of the rocket motor are read from the keyboard by the main program. A variable `dt2` with half the time step is also defined. The statement `use systemparam` appears in the beginning of most of the program units, which thus have access to all the declarations of the modeule.

The main program `satellitecrash` reads the following input data from the user; the initial altitude, the run-time of the rocket motor, the time step to be used in the integration of the equations of motion, the fraction of the time-points to be recorded, and the maximim time to integrate (used in case inappropriate input data are used, for which the satellite may never crash, or may do so only after an exceedingly long time).

Initial conditions for the position and the velocity are set according to the altitude entered (to which the earth's radius is added to give the distance r_0 from earth's center). The initial position is $x = r_0, y = 0$, in accordance with Fig. 7. The initial velocity is that corresponding to a circular orbit of radius r_0; $v_x = 0$, $v_y = \sqrt{GM/r_0}$.

The main program repeatedly calls a subroutine `rkstep`, which uses the Runge-Kutta algorithm to advance the time by a step Δ_t=`dt`. Before each call to this routine, the following information is written to a file 'sat.dat': the time, the normalized angular position (from 0 to 1, with 0 corresponding to the initial angle 0, as calculated using the x and y coordinates in the subroutine `polarposition`), the altitude, and the velocity. The program stops when a negative altitude $h = r - r_e$ has been reached (the satellite has crashed), or when the maximum integration time is exceeded.

In the integration subroutine `rkstep(t0,x0,y0,vx0,vy0)`, the argument list contains the present time, coordinates, and velocities. On exit, the coordinates and velocities have been replaced by their values at the next time step. When the subroutine is executed, the time value at the next step `t1` as well as the half-step `th` between `t0` and `t1` are first set; `t1=t0+dt; th=t0+dt2`. The implementation of the Runge-Kutta algorithm (48) for the x and y components follows:

```
call accel(x0,y0,vx0,vy0,t0,ax,ay)
kx1=dt2*ax
ky1=dt2*ay
```

```
 lx1=dt2*vx0
 ly1=dt2*vy0
 call accel(x0+lx1,y0+ly1,vx0+kx1,vy0+ky1,th,ax,ay)
 kx2=dt2*ax; ky2=dt2*ay
 lx2=dt2*(vx0+kx1)
 ly2=dt2*(vy0+ky1)
 call accel(x0+lx2,y0+ly2,vx0+kx2,vy0+ky2,th,ax,ay)
 kx3=dt*ax
 ky3=dt*ay
 lx3=dt*(vx0+kx2)
 ly3=dt*(vy0+ky2)
 call accel(x0+lx3,y0+ly3,vx0+kx3,vy0+ky3,t1,ax,ay)
 kx4=dt2*ax
 ky4=dt2*ay
 lx4=dt2*(vx0+kx3)
 ly4=dt2*(vy0+ky3)
 x1=x0+(lx1+2.d0*lx2+lx3+lx4)/3.d0
 y1=y0+(ly1+2.d0*ly2+ly3+ly4)/3.d0
 vx1=vx0+(kx1+2.d0*kx2+kx3+kx4)/3.d0
 vy1=vy0+(ky1+2.d0*ky2+ky3+ky4)/3.d0
```

In these lines of code, note that the variables corresponding to $k_1, k_2, k_4, l_1, l_2, l_4$ (but not k_3, l_3) in Eq. (48) have been defined with a factor $1/2$ (using the half time step `dt2`) in order to avoid unnecessary divisions by 2. At the end of the subroutine the new coordinates and velocities are copied to the variables of the argument list; `x0=x1; y0=y1; vx0=vx1; vy0=vy1`.

The accelerations needed in the Runge-Kutta algorithm are obtained by calls to the subroutine `accel(x,y,vx,vy,t,ax,ay)`, where `ax` and `ay` are the accelerations in the x and y directions. This subroutine implements the various F/m terms discussed above, in the following way:

```
r=sqrt(x**2+y**2)
v2=vx**2+vy**2
v1=sqrt(v2)

  !*** evaluates the acceleration due to gravitation
r3=1.d0/r**3
ax=-gm*x*r3
ay=-gm*y*r3


  !*** evaluates the acceleration due to air drag
if (v1 > 1.d-12) then
  ad=dragc*airdens(r)*v2
  ax=ax-ad*vx/v1
  ay=ay-ad*vy/v1
end if

  !*** evaluates the acceleration due to rocket motor thrust
if (t < tbrake .and. v1 > 1.d-12) then
  ax=ax-arocket*vx/v1
  ay=ay-arocket*vy/v1
end if
```

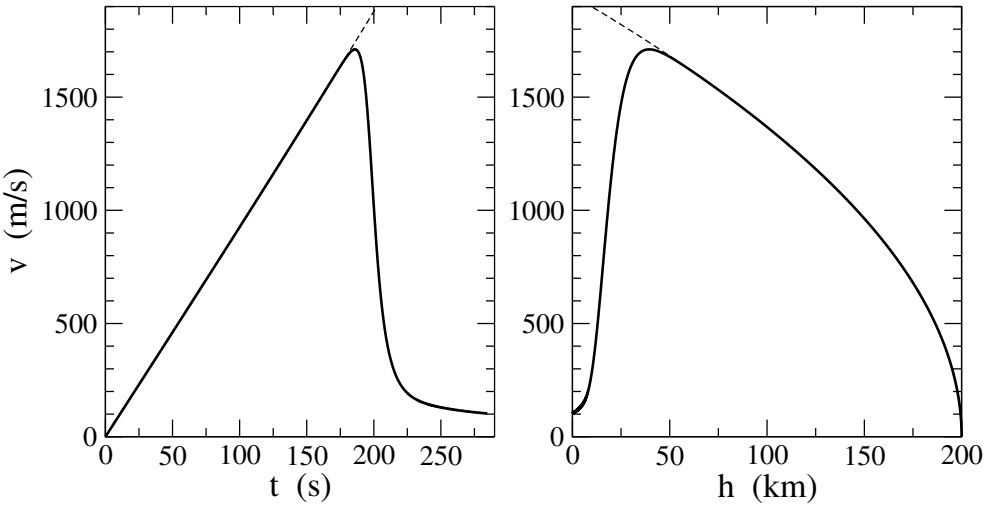

Figure 9: Time and altitude dependence of the velocity a body dropped from an altitude of 200 km in the presence of the atmosphere (solid curves) and without atmosphere (dashed curves).

The operations here should be self-explanatory; the air drag calculation uses a function `airdens(r)` which implements the athmospheric density model (57). The condition `v1 > 1.d-12` used in the air drag and rocket motor contributions is included for safety, in order to avoid an unlikely division by zero. In the case of the air drag, the force at zero velocity is zero so there should be no contribution anyway. In the case of the rocket motor contribution, a zero velocity would happen only if the motor runs long enough for the satellite to come to a halt, which should not happen in situations of interest (the way the brake force is defined, it always acts in the direction opposite to the velocity, and hence one cannot make the satellite fly off into space with this program). If the velocity happens to be exactly zero at some time step, the motor will be neglected for that step.

2.7.3 Some results

In order to test the program, we can give a high initial altitude, > 200 km, for which there is no significant air drag, and run the motor for 0 seconds. The satellite should then continue forever in its circular orbit. We can also set the atmospheric drag coefficient `dragc=0.d0` and the initial velocity `vy=0` (instead of the value of the circular orbit used). The satellite should then experience free fall. If we use a very low altutude, e.g., 1 km, the gravitational force is constant to a good approximation, and the time to reach the ground should be $t = \sqrt{2r_0/g}$ (with $g \approx 9.774$ m/s^2). The program passed these tests.

Nest, we will calculate the actual terminal velocity of a body falling fram a high altitude in the presence of the atmosphere. We have chosen the drag coefficient $C_d/m = 8{\cdot}10^{-4}$ m^2/kg, so that the terminal velocity according to the relation (56) is ≈ 100 m/s (our values correspond more precisely to 99.865 m/s). The actual terminal velocity will be different, however, because of the altitude dependence of g and ρ. Since the atmospheric effect is larger, we would expect a slightly higher terminal velocity. Running the program with an initial velocity 0 and an initial altitude of 200 km produced the time and altitude dependencies of the velocity shown in Fig. 9, where results are also

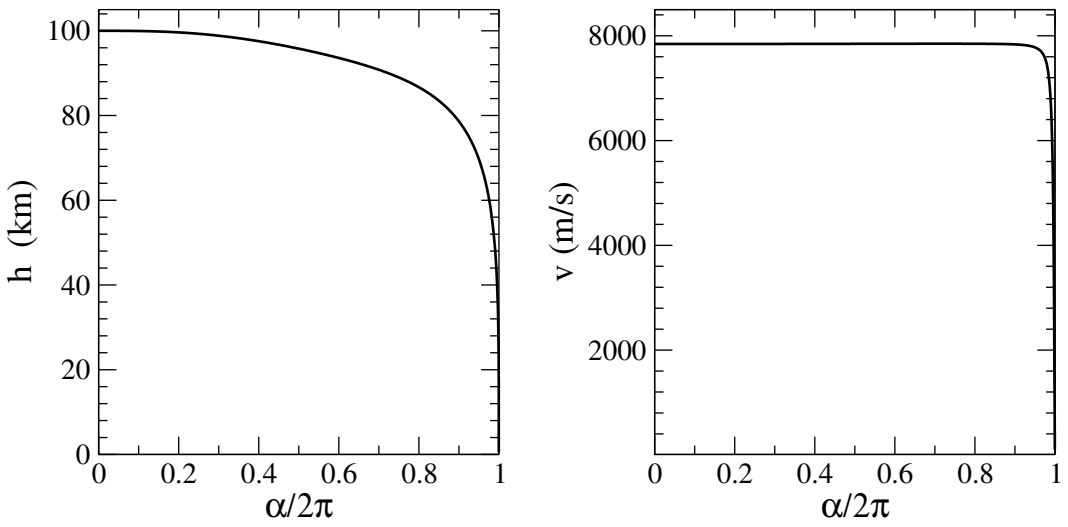

Figure 10: Altitude and velocity versus angular distance traveled for a satellite initially in a circular orbit at 100 km and then subjected to atmospheric drag.

shown for the case of no atmosphere (obtained by setting `dragc=0.d0`). In these graphs the effects of the atmosphere are seen to become strong quite sharply at an altitude of approximately 40 km. The terminal velocity is 102.6 m/s. These calculations were run with a time step of 0.1 s, however, 1 s produces almost indistinguishable results (but gives a less precise crash time).

It is also interesting to investigate the slow-down solely due to atmospheric drag of a satellite in a low orbit, i.e., without igniting its rocket motor. With the atmospheric density model used, the density at 200 km altitude is only $\approx 10^{-19}$ of that at sea-level, and the satellite stays in orbit almost indefinitely (in reality a satellite at that altitude must be regularly propelled to stay in orbit). At 100 km, the relative density in our model is $\approx 1.5 \cdot 10^{-8}$, which actually has a significant effect on the orbit; by coincidence, the satellite in this case crashes almost exactly after one revolution around earth, as shown in Fig. 10. Notice that the magnitude of the velocity is affected very little until the satellite has dropped to ≈ 50 km. It is hence primarily the direction of the velocity that is affected at the initial stages of the decent. The calculation was run with $\Delta_t = 1$ s, but even 10 s gives an acceptable convergence in this case.

Fig. 11 show results for an initial altitude of 120 km. In this case the satellite stays up for tens of orbits, eventually crashing after almost 88 hours. The points represent the altitude versus the angular position sampled every 30 s (the integration was done using $\Delta_t = 1$ s).

Finally, we turn on the rocket engine. We investigate a satellite in a 200 km orbit, keeping the rocket engine running for various lengths of time. Results for the altitude versus the angular position are shown in Fig. 12. Running the motor for 5 s does not sufficiently slow down the satellite and it instead assumes an almost elliptical orbit; it will eventually crash, but only after a very large number of orbits. After 10 s and 20 s motor runs the satellite crashes without completing another full orbit.

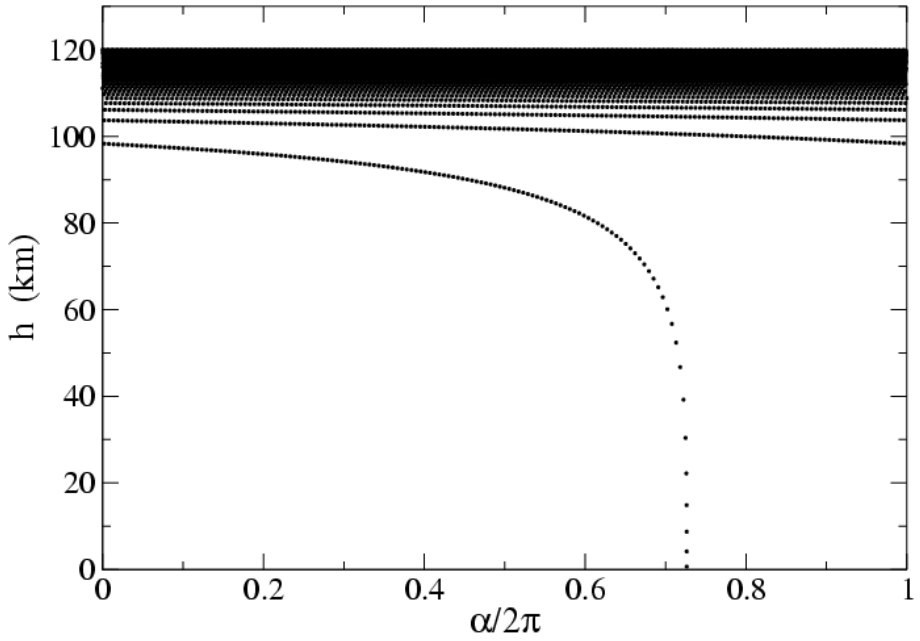

Figure 11: Altitude versus angular position of a satellite initially in a circular orbit at 120 km and then, at the beginning of the simulation, subjected to atmospheric drag. The points represent sampling of the path at intervals of 30 s.

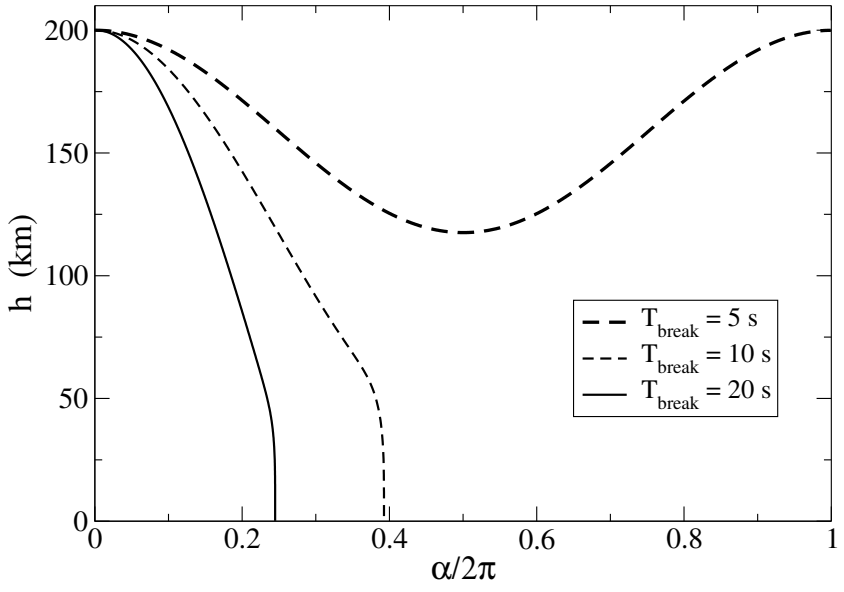

Figure 12: Altitude versus the angular position of a satellite initially in a circular orbit at 200 km altitude. The rocket engine, which causes the satellite to decelerate at 5 m/s^2, is turned on at angular position $\alpha = 0$ for different durations of time. In the case of the 5 s burn, the satellite stays in orbite for a long time in an almost elliptical orbit; only the first orbit is shown here.

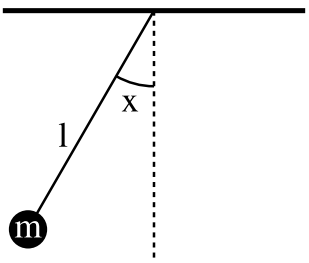

Figure 13: A pendulum of mass m, suspended by a weightless rod of length l. The definition of the angle x is indicated.

3 Chaotic motion

Many mechanical systems exhibit chaotic motion in some regions of their parameter spaces. Essentially, the term chaotic motion, or chaos, refers to aperiodic motion and sensitivity of the time evolution to the initial conditions. A chaotic system is in practice unpredictable on long time scales, althoght the motion is in principle deterministic, because minute changes in the initial conditions can lead to large changes in the behavior after some time (the time-scale on which this unpredictability sets in is governed by the so-called Lyapunov exponent). Although a chaotic system is unpredictable, its motion is not completely random. In particular, the way the system approaches chaos often exhibits universality, i.e., seemingly different systems make the transition from regular, periodic motion to chaotic motion in very similar ways, often through a series of quantitatively universal period dublings (bifurcations).

We will here only touch briefly on this subjecy, by studying one of the simplest model systems exhibiting chaotic motion; a periodically driven damped pendulum. We will limit the discussion to how chaotic motion can be characterized and quantified graphically based on numerical simulation data; methods that can be directly applied to other systems as well.

3.1 The driven damped pendulum

For a pendulum of mass m suspended by a rigid rod of length l (assumed to be massless), the potential energy is

$$V(x) = mgl[1 - \cos(x)], \tag{58}$$

where x is the displacement angle, as shown in Fig. 13. For small angles, this system can be approximated by a harmonic oscillator with spring constant $k = mgl$, but here we will keep the full form of the potential. In addition, we include a damping $\gamma\dot{x}$, which could arise from the suspension mechanism at the upper end of the rod. We will also consider a periodic driving force $F_{\rm dr} = Q\sin(t/\Omega)$, which could be due to, e.g., a time-dependent electrical field if the pendulum is charged. The equation of motion is thus

$$\ddot{x} = -k\sin(x) - \gamma v + Q\sin(t/\Omega), \tag{59}$$

where $v = \dot{x}$ is the angular velocity. This equation cannot be solved analytically when $\gamma \neq 0$ and $Q \neq 0$. We will discuss the trajectory this system follows in its phase space $x(t), v(t)$, using numerical results obtained with the Runge-Kutta method.

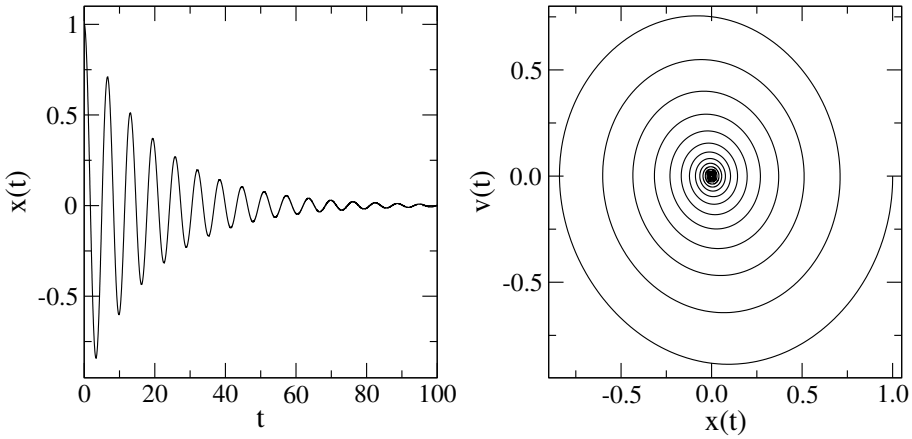

Figure 14: Damped motion of a pendulum with $k = 1, \gamma = 0.1$, $Q = 0$, and initial condition $x = 1, v = 0$. Left: displacement angle versus time; Right: phase space trajectory.

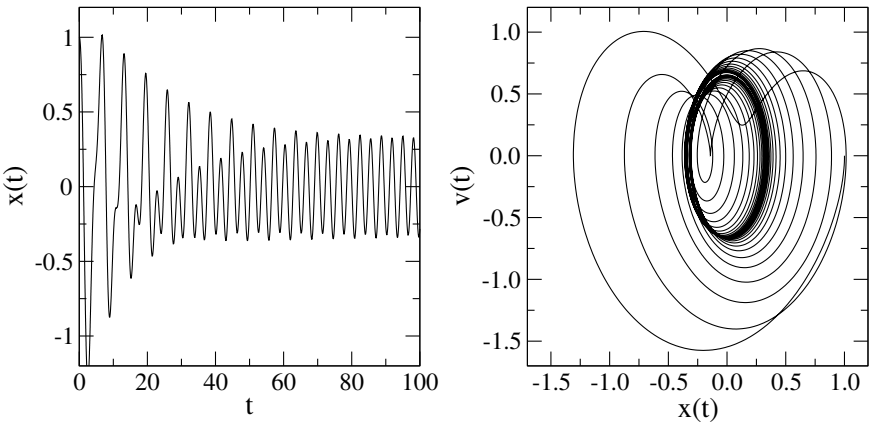

Figure 15: Motion of a damped driven pendulum with $k = 1$, $\gamma = 0.1$, $Q = 1$, $\Omega = 2$, and initial condition $x = 1, v = 0$. Left: displacement angle versus time; Right: phase space trajectory.

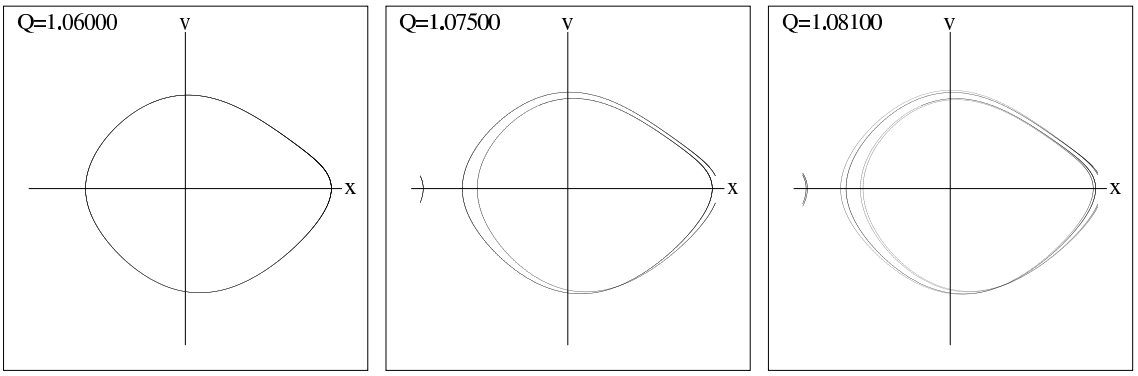

Figure 16: Limit cycles of period 1, 2, and 4 times the driving period $2\pi/\Omega$ for a pendulum with $k = 1$, $\gamma = 0.1$, $\Omega = 2/3$, and three different values of the driving amplitude Q. Note that the angle x is periodic.

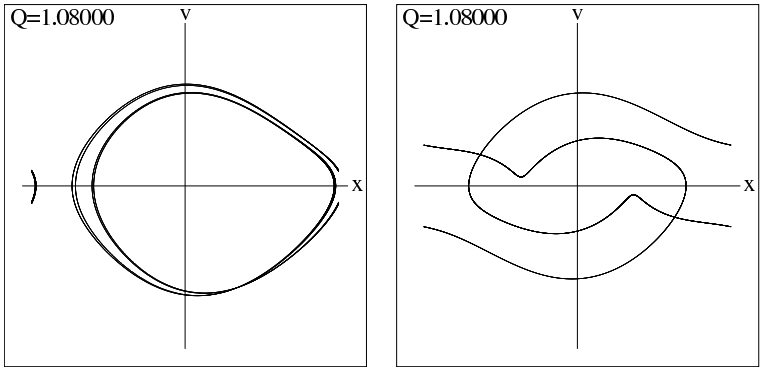

Figure 17: Limit cycles for a pendulum with $k = 1$, $\gamma = 0.1$, $\Omega = 2/3$, $Q = 1.08$, for two different initial conditions; $x_0 = 0.7$ (left) and $x_0 = 0.8$ (right); $v_0 = 0$ in both cases.

For $\gamma = 0$ and $Q = 0$, the pendulum exhibits simple periodic motion. In the presence of damping, $\gamma > 0$, it eventually comes to rest; this is illustrated by a plot of $x(t)$ in the left panel of Fig. 14. Looking at the phase space, shown in the right panel of the same figure, the trajectory is seen to spiral toward the point $(x = 0, v = 0)$. This point is said to be an attractor of this dynamical system. In Fig. 15 a driving force of amplitude $Q = 1$ and $\Omega = 2$ (period $2\pi/\Omega = \pi$) has been added. In this case, the system is seen to initially exhibit damped, irregular motion (transient behavior), but eventually it settles into a periodic motion with the same frequency as the driving force. There is a phase-space attractor in this case as well; a closed loop. In general, an attractor is a region of phase space that a system approaches at long times for a set of boundary conditions. There can be more than one attractor, in which case the space of initial conditions subdivides into *basins of attraction*. The basins of attraction can be very complex (fractal), reflecting a sensitive dependence on the initial conditions. An attractor is also often referred to as a 'limit set'. A periodic attractor is a limit-cycle in this terminology.

Periodic attractors more complicated than the single closed loop are also possible. Turning up the driving force Q, a damped pendulum ca undergo a series of period doublings, i.e., the period of the motion changes from $2\pi/\Omega$ to $\pi/\Omega, \pi/2\Omega$, etc., at certain values of Q. The corresponding limit-cycle

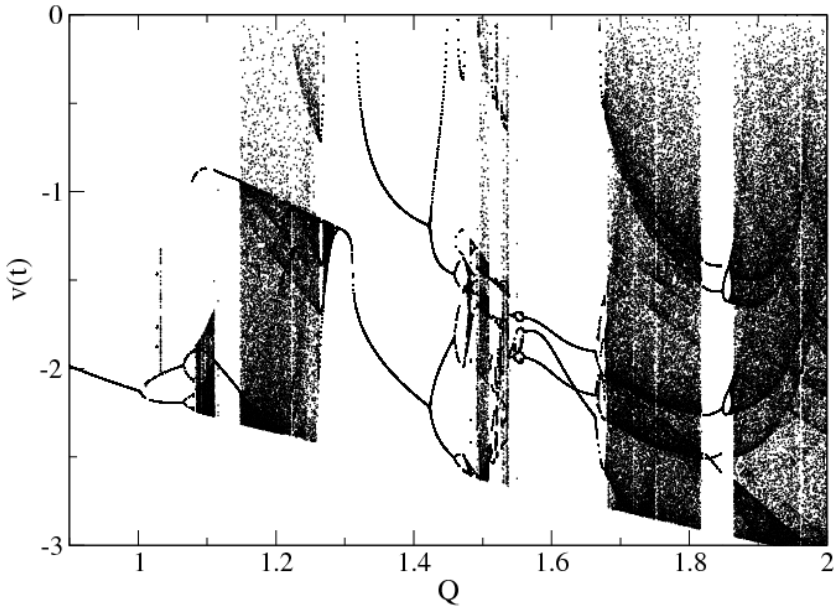

Figure 18: Bifurcation diagram for the $k = 1, \gamma = 0.1, \Omega = 2/3, Q = 2/3$ pendulum.

in phase space then develops a more complex loop structure. Examples of such doublings are shown in Fig. 16. Here the graphing of the trajectory was started only after a time long enough for the transient behavior to have died away, so that only the attractor itself is seen. It can also be noted in these graphs that the attractor is not symmetric with respect to changing the signs of x and v. This is an example of *symmetry breaking*. There are also corresponding attractors which are obtained by $(x, v) \to (-x, -v)$ in the figure (under which the equations of motion are symmetric). There are in fact yet other attractors in addition to those shown in Fig. 16 in this region of the parameter space. The attractors shown in Fig. 16 were obtained with the initial condition $x_0 = 1, v_0 = 0$. Fig. 17 shows two attractors for a pendulum with $Q = 1.08$ and the other paremeters equal to those in Fig. 16. In both ceses $v_0 = 0$, but the initial angle x_0 was different. For $x_0 = 0.7$, shown in the left panel, the limit cycle is a symmetry-broken one with period four, as the one for $Q = 1.081$ in Fig. 16, but for $x_0.8$, shown in the right panel, a symmetric attractor is obtained. There is also of course an $x \to -x$, $v \to v$ counterpart to the symmetry-broken attractor. Which one of the three attractors that is realized (i.e., the basin of attraction to which the initial conditions belong) depends in a complex way on (x_0, v_0).

In order to systematically investigate the behavior versus a parameter of the system, e.g., the driving force Q, it is useful to not consider the full phase space, but only a cut through it (or, more generally, for a system with a higher-dimensional phase space, a surface in that space). For example, we can graph the velocity at the point where the angle x passes the value 0 from above. Such a construct is called a *Poincaré section*. For periodic motion, the Poincaré section is a discrete set of points; for the limit cycles shown in Fig. 16 there are 1, 2, and 4 points in the Poincaré section through $x = 0$. By graphing the set of points versus Q, we can investigate how the periodicity of the motion evolves (although in general it may be difficult, or even be impossible, to know exactly what Poincaré section to use in order to capture the actual periodicity this way, since a limit cycle may have a looping portion that does not cross the chosen cut). Such a plot is shown

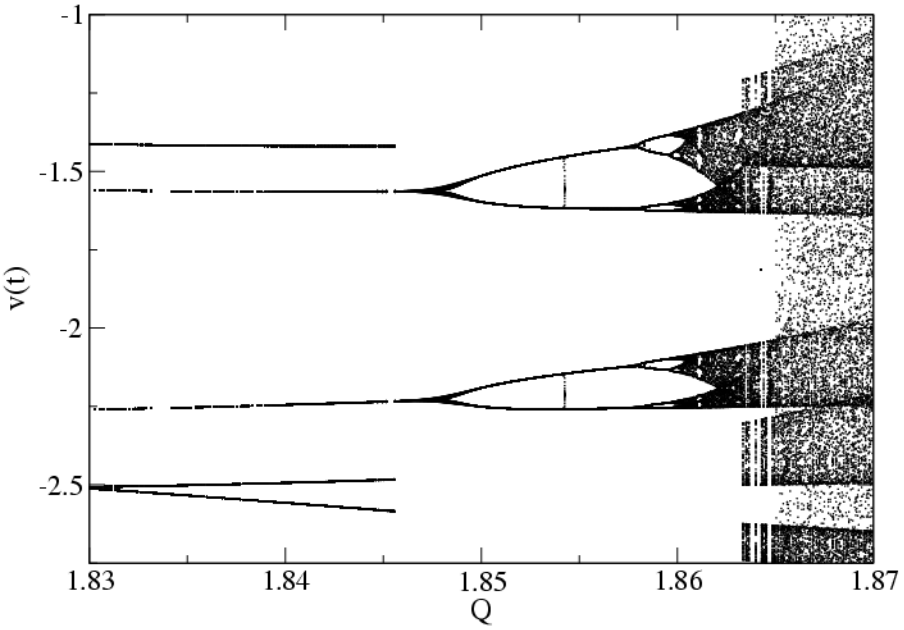

Figure 19: The parameter region $Q = 1.83 - 1.87$ of the bifurcation diagram in Fig. 18 shown in greater detail.

in Fig. 18, for the velocity as x crosses zero from above (hence all the velocities are negative). A very complex behavior can be observed, with regions of period doublings and regions where there does not appear to be a discrete set of points at all—such regions correspond to chaotic behavior.[1] Plots like these, often referred to as bifurcation diagrams, have been investigated in great detail for many systems. It turns out that the bifurcations—the points where the periodicity doubles—have a universal structure, i.e., many different systems show a quantitatively similar behavior. Chaotic behavior occurs after an infinite number of increasingly closely spaced period-doubling bifurcations, and this series of bifurcations exhibits universality. in Fig 19, the region $Q = 1.83 - 1.87$ of Fig. 18 is shown in greater detail (in order to get a sufficient density of points, a new run with a finer Q-spacing had to be carried out). Here several period-doubling bifurcations can be seen preceding the chaotic regime for $Q > 1.863$.

The universality of period-doubling bifurcations is contained in the so-called logistic map. This is a discrete recursion relation, where the next value in a sequence of numbers is given by

$$x_{n+1} = \mu x_n (1 - x_n). \tag{60}$$

One can liken the discrete index n to the successive points on a Poincaré section. A graph similar to Fig. 18 is obtained by iterating the logistic map numerous times for each of a sequence of values of the control parameter μ, plotting the points (μ, x_n), $n = 0, 1, 2, \ldots$ (after some initial number of steps, to avoid transients). Such a graph is shown in Fig. 20. Clearly, there are great similarities with the bifurcations seen in Fig. 19, and in fact the two plots could be scaled in such a way that they coincide perfectly. This is true in general for systems that approach chaos through a

[1]The same initial conditions, $x_0 = 1, v_0 = 0$ were used for all Q values in this plot. In regions where there are several basins of attraction, the plot could hence look different for other initial conditions.

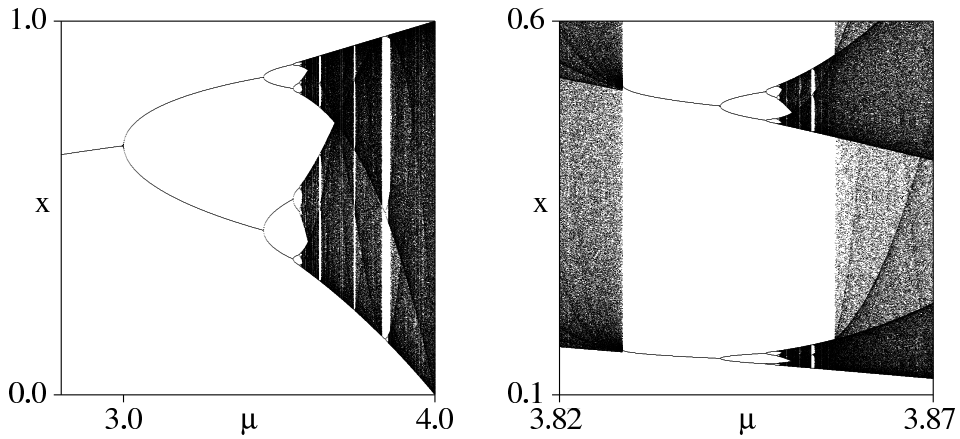

Figure 20: Bifurcation diagram for the logistic map. In the right panel, a region of the left panel is shown on a more detailed scale, to reveal other bifurcation sequences.

sequence of period-doubling bifurcations, as was first noted by Mitchell Feigenbaum in 1974. The period-doublings are self-similar, i.e., they look exactly the same on different scales; the smaller "bubbles": seen in Fig. 20 can be magnified and then look exactly as the bigger ones, and they themselves bifurcate into smaller bubbles, etc., until chaos sets in in the limit of an infinite number of bifurcations. There is not just one period-doubling sequence in the logistic map; they appear in an infinite number of places in a fractal fashion. An example is shown in the right panel of Fig. 20, which is a magnification of a region of the left panel where only some dim structures can be seen. Upon magnification, these features can be seen to constitute exactly the same type of bifurcation as in the left graph (but "upside down"). In addition to its famous period doubling bifurcations, the logistic map also exhibits other universal features, e.g., associated with the "window" structure (the almost empty spaces between the chaotic regimes, which correspond to periodic motion).

As we have seen, period-doubling bifurcations do occur in the damped pendulum, and the similarity with the logistic map can be seen clearly, e.g., in Fig. 19. However, not all transitions to chaos in the pendulum are of this form; several "routes tho chaos" are known, but the period-doubling is the most well studied and understood.

The phase space attractor for a pendulum in a chaotic state is shown in Fig. 21. In this case there is no limit-cycle, and instead the attractor fills a finite region of the two-dimensional phase space. There is clearly some structure in the attractor, with some regions more likely to be occupied than others.

One can also represent the phase space behavior of a periodically driven systems by *stroboscopic sampling*. This amounts to plotting x and v at time intervals equal to the period $2\pi/\Omega$ of the driving force. For a system with periodicity equal to that of the driving force, this plot will clearly have a single point, and if the plot has n points the periodicity is $n2\pi/\Omega$. For a chaotic systems the points fall on an extended region of the phase space, and this region has a fractal dimensionality for chaotic motion. Examples of stroboscopic sampling are shown in Fig. 22. Plots like these are also oftent referred to as Poincaré sections, and the sets of occupied points are also attractors. The right panel in Fig. 22 is an example of a *strange attractor*; one with a fractal dimensionality (i.e., a Hausdorff dimensionality between 1 and 2 in this case). Stroboscopic sampling is often the easiest

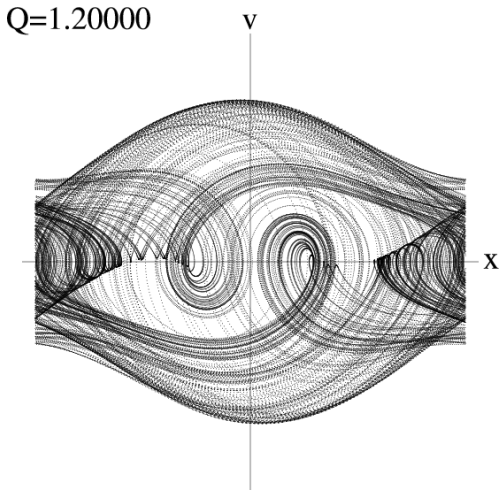

Figure 21: Phase space trajectory (attractor) for the $k = 1, \gamma = 0.1, \Omega = 2/3, Q = 1.2$ pendulum; this system is chaotic.

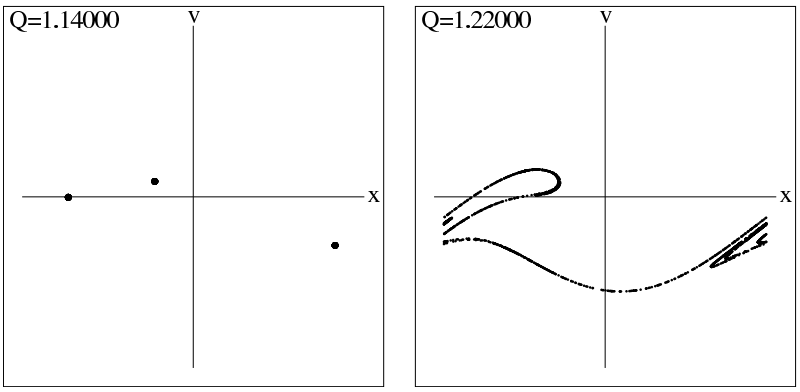

Figure 22: Stroboscopic sampling plots for the $k = 1, \gamma = 0.1, \Omega = 2/3$ pendulum at two different driving apliitudes. The left graph shows a case where the period is 3Ω, and the right graph shows a chaotic case with a strange attractor.

way to examine whether a periodically driven system is chaotic.