Chapter 3

Ordinary differential equations

In molecular dynamics the motion of particles in time is computed. Since the motion of particles in time is described by ordinary differential equations (ODE's), the content of this chapter is preliminary for the next chapter on molecular dynamics.

In general ODE's do not allow an exact solution. Simulating this motion in time therefore requires integrating ODE's numerically. This chapter discusses some ordinary differential equations and methods used to numerically integrate them.

In learning to solve ODE's with numerical integration it is helpful to start with an ODE that allows an analytic solution, so that results obtained with numerical integration can be compared to the analytic results. To this end the harmonic oscillator and its analytic solution are discussed in Section 3.1.

Secondly, Section 3.2 discusses nondimensionalization, as it is a process which can get more out of a computer simulation.

Finally, Section 3.3 discusses several methods of numerically integrating ODE's. It covers to some detail the Euler, Euler-Cromer, Verlet and Runge-Kutta integration schemes.

3.1 The harmonic oscillator

In introducing the numerical methods that solve odinary differential equations in this course, the methods are applied to solving the differential equation of the harmonic oscillator. The reason is that we can obtain an analytic solution to this equation, allowing us to compare and judge the quality of the numerical solutions.

In this section the analytic solution of the equation of motion of the harmonic oscillator is derived.

The differential equation of the harmonic oscillator represented in Fig. 3.1 is given by

$$m\ddot{x}(t) = -k(x_\mathrm{m} - x_\mathrm{e}),$$

where m is the mass the spring acts on, k is the stiffness of the spring, x_m is the position of the mass and x_e is the equilibrium position of the mass.

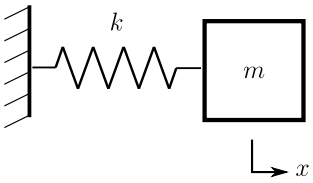

Figure 3.1: Schematic representation of the harmonic oscillator

Letting $x = x_\mathrm{m} - x_\mathrm{e}$ we can rewrite this to

$$\ddot{x}(t) = -\frac{k}{m}x, \tag{3.1}$$

which can be interpreted as a spring with an equilibrium length of zero.

3.1.1 Analytic solution

The solution to Eq. (3.1) is given by

$$x(t) = A\sin(\omega t + \delta), \tag{3.2}$$

where ω is the angular frequency of oscillation given by $\omega = 2\pi/T = \sqrt{k/m}$, where T is the period of oscillation.

In order to validate the numerical solutions the velocity is also required. The analytic solution for the velocity is given by the derivative of Eq. (3.2), equal to

$$\dot{x}(t) = \omega A\cos(\omega t + \delta). \tag{3.3}$$

The amplitude A and phase δ are determined by the intial conditions $x(0) = x_0$ and $v(0) = \dot{x}(0) = v_0$. Substituting these in Eqs. (3.2) and (3.3), respectively, and rewriting gives

$$\sin\delta = \frac{x_0}{A}, \tag{3.4}$$

$$\cos\delta = \frac{v_0}{A\omega}. \tag{3.5}$$

We can solve for the phase δ by dividing Eq. (3.4) by Eq. (3.5):

$$\tan\delta = \omega\frac{x_0}{v_0} \quad \Rightarrow \quad \delta = \arctan\left(\omega\frac{x_0}{v_0}\right). \tag{3.6}$$

We can solve for the amplitude A by squaring and adding Eqs. (3.4) and (3.5):

$$\sin^2\delta + \cos^2\delta = 1 = \frac{1}{A^2}(x_0^2 + (v_0/\omega)^2)$$
$$\Rightarrow \quad A = \sqrt{x_0^2 + (v_0/\omega)^2} \tag{3.7}$$

Ex. 2 — Write a small program that generates the analytical solution as time series and plot x vs. t.

3.2 Nondimensionalization

Using dimensions in our computations makes them easy to understand. Assigning dimensional units such as kilometers, kilograms and meters to the numbers we compute with provides an easy check for consistency. But it is not so that dimensions are necessary, and this is clear when one considers the computations performed by a computer. A computer just uses what we regard as pure numbers to perform its computations.

We can also perform our computations without the use of dimensions. The process of rewriting equations to a form that makes no use of dimensions is called 'nondimensionalization'. There are advantages to doing so.

One advantage of nondimensionalization is that the solution to the resulting nondimensional equation can be converted to the solution to a range of real world problems. I.e. one simulation provides the results that can be converted to multiple real world problems. Another advantage has to do with the limitations on the precision of numbers the computer can represent. When the numbers that appear in a problem range between orders of magnitude too far apart for a computer to represent simultaneously with the required accuracy, nondimensionalization might allow one to represent the problem on a scale that a computer can represent properly. Another advantage is that in some cases nondimensionalization reduces the number of parameters in the equation.

Any dimensional quantity Q can be thought of as being a product of a dimensionless quantity (a numerical value) $\{Q\}$ and a dimensional unit $[Q]$. The nondimensional number $\{Q\}$ is said to be measured in units of $[Q]$. To nondimensionalize an equation, the dimensional quantities in the equation can therefore be multiplied with an inverse dimensional unit $1/U$, such that $[U] = [Q]$ and therefore $Q/U = \{Q\}/\{U\}$, a dimensionless number. An interesting choice of U would be a unit that only depends on characteristics of the system under consideration. For example, given a characteristic length σ, we define a nondimensional variable x' to nondimensionalize the variable length x according to $x' := x/\sigma$.

In the following, the differential equation of the harmonic oscillator, Eq. (3.1), is nondimensionalized.

First we have to find enough characteristic quantities. For each unit in the equation we define a unit that is just a multiple of the SI unit, but depends on the characteristics of a harmonic oscillator. So we could define a unit of length σ representing the equilibrium length, a unit of mass m_0 representing the oscillating mass, and a unit of time τ representing the period of oscillation.

In order to nondimensionalize Eq. (3.1) we replace the dimensional quantities x, t, k and m with nondimensional quantities x', t', k' and m' by multiplying the dimensional quantities with an appropriate combination of the inverse of units σ, m_0 and τ. This can be achieved by choosing $x' = x/\sigma$, $t' = t/\tau$, $k' = k\tau^2/m_0$ and $m' = m/m_0$. Note that for this equation we have now defined enough units to nondimensionalize any dimensional quantity. For example, we could define a unit of density as $\rho_0 = m_0/\sigma^3$.

Instead of solving Eq. (3.1) we can now solve the nondimensional differential equation

$$\frac{d^2x'}{dt'^2} = -\frac{k'}{m'}x'. \tag{3.8}$$

To obtain a solution we set k' and m' to whatever we find appropriate. We also provide dimensionless initial conditions $x'_0 = (1/\sigma)x_0$ and $v'_0 = (\tau/\sigma)v_0$, which are determined by the particular problem to be solved. This results in the solution x' as a function of t'.

We can now obtain the solution to a real world dimensional problem by computing $x = x'\sigma$ and $t = t'\tau$. Here σ and τ can be varied to describe different real world problems.

The choice of reference units is somewhat arbitrary. One can use any combination of units that together provide all the physical dimensions that occur in the equation. Another set of reference units could be the position σ_0, the mass density ρ_0 and the system energy ϵ_0.

The number of quantities to keep in mind can grow quite large. One may find it convenient to give an overview of all the quantities in a table. An example of such an overview is given in Tab. 3.1. It lists all dimensional quantities we have in mind and their relation to the nondimensional quantities in the simulation.

Ex. 3 — Nondimensionalize Eq. (3.1) using the reference units σ_0, ρ_0 and ϵ_0 and fill in the table using these units.

3.3 Numerical integration of ODE's

Many odinary differential equations do not allow for an analytic solution. To obtain a solution therefore requires an approximate approach, as provided by numerical integration.

The solution to Eq. (3.1) can be approached by integrating it numerically. To approach the exact solution, we estimate the value of x at discrete times

	Dimensional quantity	Quantity in simulation
reference units	position σ	$\sigma' = 1$
	time τ	$\tau' = 1$
	mass m_0	$m_0' = 1$
measurable quantities	position x	$x' = x/\sigma$
	time t	$t' = t/\tau$
	mass m	$m' = m/m_0$
	stiffness k	$k' = k\tau^2/m_0$
	angular frequency ω	$\omega' = \omega\tau$
	period T	$T' = T/\tau$
	density ρ	$\rho' = \rho\sigma^3/m_0$
	energy ϵ	$\epsilon' = \epsilon\tau^2/(m_0\sigma^2)$

Table 3.1: An overview of a nondimensionalization of the harmonic oscillator

$t_i = i\Delta t$, where Δt is the size of some arbitrary time interval. Five numerical integration schemes are introduced: the Euler, Euler-Cromer, Verlet, leapfrog and Runge-Kutta integration schemes.

3.3.1 Euler integration

For the Euler method the estimation goes as follows. Suppose we know the position and velocity at some time t_i, which we denote by x_i and v_i, i.e. we know $x_i = x(t_i)$ and $v_i = v(t_i)$. We can then estimate the position at the next time step $t_{i+1} = t_i + \Delta t$ by assuming the velocity stays constant during the time interval Δt. We thus estimate $x_{i+1} \approx x_i + v_i\Delta t$. Now, using this to compute the position at the next time step x_{i+2} would require the velocity v_{i+1}. Analogous to computing x_{i+1}, this can be estimated by $v_{i+1} = v_i + a_i\Delta t$. Here a_i is the acceleration at time step t_i, which for a harmonic oscillator we can compute from Eq. (3.1): $a_i = \ddot{x}_i = -\omega^2 x_i$, where $\omega = \sqrt{k/m}$.

Thus, given an intial position x_0 and intial velocity v_0, we can now estimate the solution $x(t)$ at times $t_i = t_0 + i\Delta t$, with $i = 1, 2, ..., M$, and $M = t_{\max}/\Delta t$ by computing

$$x_{i+1} = x_i + v_i\Delta t, \tag{3.9}$$

$$v_{i+1} = v_i + a_i\Delta t, \tag{3.10}$$

where $a_i = -\omega^2 x_i$.

A single integration step using the Euler method is visualized in Fig. 3.2. Given the exact solution $x_i = x(t_i)$ and $v_i = \dot{x}_i(t_i)$, the approximation of the solution at the next time step $x_{i+1} = x(t_{i+1})$ is visualized.

There are two major disadvantages to the Euler integration scheme. One is that the integration scheme recovers the Taylor expansion of $x(t_i + \Delta t) =$

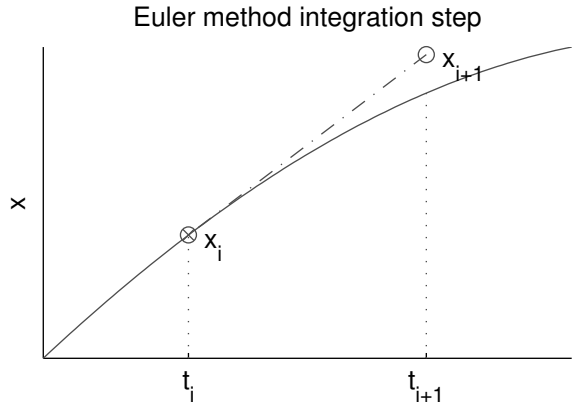

Figure 3.2: Visualization of an Euler method integration step

$x(t_i) + \dot{x}(t_i)\Delta t + \ldots$ up to first order in Δt only, making the error per step (commonly referred to as the local error) proportional to Δt^2. The other is that the scheme tends to 'overshoot' the analytic solution at every step, thereby accumulating local errors into a global error (the total error over a trajectory of length $t_{\max}$) that in simulations is found to be proportional to the employed time step.[1] An algorithm with this property is first order accurate. Consequently, decreasing the time step Δt by a factor h (and simultaneously increasing the number of steps by a factor h to conserve $t_{\max}$) makes the error only a factor h smaller, hence making your time step smaller is therefore not that effective in improving the accuracy of the calculated trajectory. Higher order schemes exist for which the error decreases by a factor h^n with $n > 1$.

3.3.2 Euler-Cromer integration

The Euler scheme has the disadvantage of constantly overshooting the exact solution. The Euler-Cromer scheme avoids this by estimating the position x_{i+1} using not the velocity at the current time, v_i, but the estimated velocity at the next time, v_{i+1}. It is as if it anticipates where to go next, avoiding the constant overshooting. Of course this requires that we estimate v_{i+1} first.

Using the Euler-Cromer scheme, given the intial position and velocity x_0 and v_0, we can estimate the solution $x(t)$ at time t_i, $i = 1, 2, ..., M$, by computing

$$v_{i+1} = v_i + a_i \Delta t, \tag{3.11}$$

$$x_{i+1} = x_i + v_{i+1} \Delta t, \tag{3.12}$$

where for the harmonic oscillator $a_i = -\omega^2 x_i$ with $\omega = \sqrt{k/m}$.

[1] A run of length $t_{\max}$ requires $t_{\max}/\Delta t$ steps, each contributing a local error $\mathcal{O}(\Delta t^2)$, thus accumulating a global error proportional to $t_{\max}\Delta t$.

A single integration step using the Euler-Cromer method is visualized in Fig. 3.3. Given the exact solution $x_i = x(t_i)$, it shows the approximation of the solution at the next time step $x_{i+1} = x(t_{i+1})$.

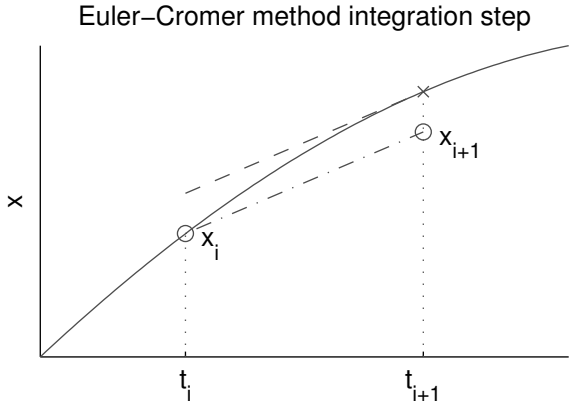

Figure 3.3: Visualization of an Euler-Cromer method integration step

Simulations of a harmonic oscillator using the Euler-Cromer scheme show that it performs much better than the original Euler scheme. Nevertheless, the simulations also reveal that the Euler-Cromer scheme is still only first order accurate, just like the Euler scheme. An explanation for this apparent contradiction is provided below.

3.3.3 Verlet integration

A higher order scheme is the Verlet integration scheme. It is derived by considering the Taylor series expansion of the position $x(t)$. The Taylor series expansion of x_{i+1} truncated to second order is

$$x_{i+1} \approx x_i + \dot{x}_i \Delta t + \frac{1}{2}\ddot{x}_i \Delta t^2. \tag{3.13}$$

Doing the same for x_{i-1} gives

$$x_{i-1} \approx x_i - \dot{x}_i \Delta t + \frac{1}{2}\ddot{x}_i \Delta t^2. \tag{3.14}$$

Adding Eqs. (3.13) and (3.14) we get an approximation for x_{i+1} in terms of $\ddot{x}_i$, x_i and x_{i-1}:

$$x_{i+1} \approx 2x_i - x_{i-1} + \ddot{x}_i \Delta t^2. \tag{3.15}$$

We are thus let to the Verlet algorithm, which calculates the next position from

$$x_{i+1} = 2x_i - x_{i-1} + a_i \Delta t^2. \tag{3.16}$$

Given initial conditions $x(0) = x_0$ and $\dot{x}(0) = v_0$, we still need $x_{-1} = x(-\Delta t)$ to obtain x_1. When this information is not available, it can be generated by an Euler approximation: $x_{-1} = x_0 - v_0 \Delta t$.

The advantages of the Verlet integration scheme are that it is rather simple and compact, yet it is second order accurate. This means that reducing the step size by a factor h (while simultaneously increasing the number of steps by a factor h to conserve $t_{\max}$) makes the global error in the numerical solution h^2 times as small.[2] This in contrast to the Euler and Euler-Cromer schemes, which are only first order accurate. Also, unlike the Euler and Euler-Cromer algorithms but in agreement with Newton's equations of motion, the trajectory generated by the Verlet scheme is time reversible. Furthermore, the Verlet scheme is a symplectic integrator, meaning that the trajectories generated by the algorithm conserve volume in phase space (as do Newton's equations of motion). The latter two properties contribute to the long term stability of trajectores calculated by the Verlet scheme, as reflected by a total energy that fluctuates around an average value rather than decreasing to zero or growing to infinity. This stability is in marked contrast to the Euler and Runge-Kutta (see below) methods, which see the total energy grow to infinity or decrease to zero, respectively. Note that long term stability is not to be confused with long term accuracy.

The disadvantages of Verlet integration are that it only works efficiently for differential equations of the form $\ddot{x} = f(x)$, and that the velocities are not readily available.

3.3.4 Leapfrog integration

Closely related to the Verlet scheme is the leapfrog scheme. An auxiliary variable w_i is introduced, defined as

$$w_i = (x_i - x_{i-1})/\Delta t. \tag{3.17}$$

The above Verlet scheme is then readily rewritten as

$$w_{i+1} = w_i + a_i \Delta t \tag{3.18}$$

$$x_{i+1} = x_i + w_{i+i} \Delta t. \tag{3.19}$$

The difference between this second order scheme and the superficially identical first order Euler-Cromer scheme is subtle. In the Euler-Cromer scheme, the velocity v_i is the numerical approximation of the velocity at the integer timestep: $v_i \approx v(t_i)$. In the leapfrog scheme, the velocity w_i is by definition equal to the average velocity between t_{i-1} and t_i, which makes w_i a numerical approximation for the velocity at the half-integer time step,

[2] The local error in the Verlet scheme is $\mathcal{O}(\Delta t^4)$, a significant improvement over the $\mathcal{O}(\Delta t^2)$ of the Euler scheme. Unfortunately, the factor 2 in Eq. (3.16) makes the global error grow quicker than linear with the number of steps, resulting in a global error over a run of length $t_{\max}$ proportional to Δt^2. Note that to achieve this performance, x_0 and x_{-1} must be exactly identical to $x(t_0)$ and $x(t_{-1})$, respectively. Bootstrapping the Verlet scheme with the Euler step $x_{-1} = x_0 - v_0 \Delta t$ reduces the global accuracy to first order.

$w_i \approx v((i-\frac{1}{2})\Delta t)$.[3] Continuing this line of thought, one may approximate the velocity at the integer timestep by the average $v(t_i) \approx \frac{1}{2}(w_i + w_{i+1})$. The alternation of full step positions x_i and midstep velocities w_i is reflected in the name of the scheme, leapfrog integration, after the child's game. Just like the Verlet scheme, it only works efficiently for differential equations of the form $\ddot{x} = f(x)$. Note the time symmetry underlying the Verlet and leapfrog schemes; symmetry is often a good property to have in a numerical scheme.

3.3.5 Runge-Kutta integration

The last integration scheme we discuss is the second order accurate Runge-Kutta scheme, also called the midpoint method. It is thus, as Verlet integration, second order accurate, but in contrast to the Verlet scheme it can be applied to differential equations of the form $\ddot{x} = f(x, \dot{x})$.

The Runge-Kutta scheme works by using the velocity and accelerations at intermediate times $t_{i+1/2}$ in estimating t_{i+1}, so

$$x_{i+1} = x_i + \Delta t \bar{v}_i \tag{3.20}$$

$$v_{i+1} = v_i + \Delta t \bar{a}_i, \tag{3.21}$$

where $\bar{v}_i = v_{i+1/2}$ and $\bar{a}_i = a_{i+1/2}$. To compute $\bar{v}_i$ and $\bar{a}_i$ we use the Euler scheme. For the harmonic oscillator the velocity at the intermediate timestep can be computed by

$$\bar{v}_i = v_i + \frac{\Delta t}{2} a_i,$$

where $a_i = -\omega^2 x_i, \omega = \sqrt{k/m}$ by Eq. (3.1). The intermediate acceleration can be computed by substituting the intermediate position in Eq. (3.1);

$$\bar{a}_i = -\frac{k}{m}\bar{x}_i,$$

where $\bar{x}_i = x_{i+1/2}$, approximated with the Euler method by

$$\bar{x}_i = x_i + \frac{\Delta t}{2} v_i.$$

A single integration step using the midpoint method is visualized in Fig. 3.4. Given the exact solution $x_i = x(t_i)$ the approximation of the solution at the next time step $x_{i+1} = x(t_{i+1})$ is visualized.

For a differential equation of the general form $\dot{\mathbf{x}} = f(t, \mathbf{x})$ where $\mathbf{x}$ denotes a vector, the Runge-Kutta scheme results in

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta t f\left(t_{i+1/2}, \mathbf{x}_i + \frac{\Delta t}{2} f(t_i, \mathbf{x}_i)\right). \tag{3.22}$$

[3] This approximation is sufficiently accurate for a run started with $x_0 = x(t_0)$ and $w_0 = \dot{x}(-\frac{1}{2}\Delta t)$ to generate a second order accurate trajectory.

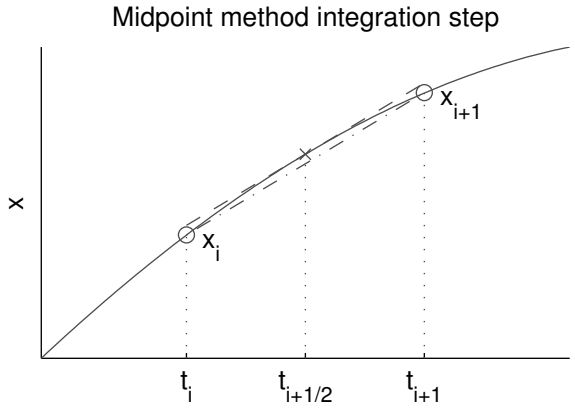

Figure 3.4: Visualization of a Midpoint method integration step

It is now shown how Eq. (3.1) can be solved using the general form of the midpoint method, Eq. (3.22).

In order to get Eq. (3.1) into the form $\dot{\mathbf{x}} = f(t, \mathbf{x})$ we must introduce the extra equation

$$\dot{x} = v.$$

From Eq. (3.1) we can now write

$$\dot{v} = -\frac{k}{m}x.$$

To rewrite these coupled differential equations in a suitable form, we use the vector $\mathbf{x} = [x, v]^{\mathrm{T}}$. Now, the equation

$$\dot{\mathbf{x}} = f(t, \mathbf{x}),$$

with

$$f(t, \mathbf{x}) = \begin{bmatrix} v \\ -\frac{k}{m}x \end{bmatrix}, \tag{3.23}$$

is equivalent to Eq. (3.1).

Now suppose we have the initial conditions $x(0) = x_0$ and $v(0) = v_0$, or

$$\mathbf{x}_0 = [x_0, v_0]^{\mathrm{T}},$$

stiffness k and mass m, then we can compute $\mathbf{x}_1$ by substituting this into Eq. (3.22) with $i = 0$ and $f(t, \mathbf{x})$ given by Eq. (3.23), and so on to compute $\mathbf{x}_2, \mathbf{x}_3, ...$

3.3.6 MATLAB ode45

In order to numerically solve a differential equation the above schemes can be implemented by hand. MATLAB comes with a family of implemented

numerical solvers, however. The MATLAB function ode45 implements a different member of the Runge-Kutta family of ODE solvers, namely the Dormand-Prince method.

In the simplest case the function call is

```
[t, y] = ode45(odefun, tspan, y0);
```

We will now first discuss the input arguments and output arguments.

The argument `odefun` is a function handle to a function m-file implementing the function $f(t, \mathbf{x})$, which is Eq. (3.23) for the harmonic oscillator. Suppose this function is implemented in MATLAB as in Listing 3.1, then `odefun` should be assigned with

```
odefun = @harmonic_oscillator_rhs;
```

```
function dxdt = harmonic_oscillator_rhs(t, x)

k = 1;
m = 1;

dxdt = [x(2);
        -k / m * x(1)];
```

Listing 3.1: content of harmonic_oscillator_rhs.m

The second argument `tspan` is a vector specifying the time interval the simulation should cover. It could be assigned with

```
tspan = [t0, tf];
```

where `t0` and `tf` are the start and end times, respectively.

The third argument `y0` is a vector specifying the initial conditions, which in this case could be defined with

```
y0 = [x0; v0];
```

Here `x0` and `v0` are the initial position and velocity, respectively.

The call to `ode45` returns a column vector `t` with all the time points the solution was computed at and a matrix `y` whose rows correspond to the solution at a time point. The `y` is thus what $\mathbf{x}$ is in Eq. (3.23). We can now plot the position and velocity vs. time with the commands `plot(t, x(:, 1))` and `plot(t, x(:, 2))`, respectively. Figure 3.5 shows the plot resulting from the commands in Listing 3.2.

3.3.7 Other integration schemes

A lot more methods have been developed to numerically integrate ordinary differential equations. Important characteristics are of what order such a

```
% Simulation parameters
t0 = 0;                                 % start time
tf = 4 * pi;                            % end time
x0 = 0;                                 % initial position
v0 = 1;                                 % initial velocity

% Use ode45 to simulate
odefun = @harmonic_oscillator_rhs;      % function handle to f(t, x)
tspan = [t0, tf];                       % simulation time span
y0 = [x0; v0];                          % initial conditions
[t, x] = ode45(odefun, tspan, y0);      % call to ode45

% Plot result
plot(t, x(:, 1));
xlabel('time t'); ylabel('position x'); axis tight;
```

Listing 3.2: solving harmonic oscillator using `ode45` and plotting the result

numerical integration scheme is, how stable it is and which type of equation it can solve.

Higher order schemes converge faster, and thus are cheaper to use when the problem is sufficiently complicated. Using a lower order scheme might take longer because although the computation per timestep is cheaper, the number of timesteps required to obtain the solution can be so much greater than it is for a method of higher order that the total amount of computation required is greater.

Higher order schemes are given by e.g. Runge-Kutta schemes, the most famous being the RK4 method, multistep methods and predictor-corrector methods. Both Runge-Kutta and multi-step methods use more and more information to determine the next time step, leading to a higher order of accuracy. The difference is that Runge-Kutta schemes discard the information used to compute the next time step for each time step, whereas multi-step methods reuse the same information for the computation of several time steps. Predictor-corrector methods are a class of methods in which first a prediction of the solution at the next time step is made, and next this prediction is used to compute a better approximation to the same solution. Many more methods exist and an overview can be found on Wikipedia by searching for 'template:numerical integrators' [4].

Stability is an issue when the size of the time step required by a method is determined by stability reasons instead of accuracy reasons. For many algorithms a timestep that should lead to an accurate enough solution is not sufficient because it causes the solution to 'explode'. Equations whose

[4] the 'template:' part is necessary because the page is in the template namespace on Wikipedia, not the default main namespace

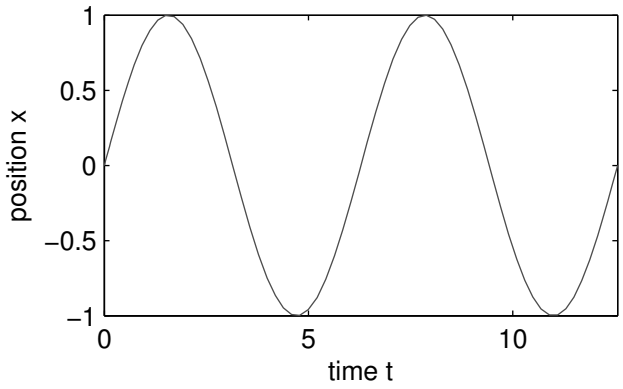

Figure 3.5: Position as a function of time of the harmonic oscillator as simulated with `ode45`

numerical integration by certain methods causes this behaviour are called stiff. A stable algorithm allows the size of the timestep to be determined by accuracy reasons only and therefore leads to a smaller number of timesteps required for obtaining a solution. In general, implicit methods are more stable and computationally less expensive in solving stiff problems. For implicit methods the solution for the next time step can not be expressed explicitly in solutions at previous time steps, requiring extra computation to solve for the next time step.

In general, integration schemes are restricted to a specific class of equations. For example, the midpoint method discussed before can only integrate equations of the form $y'(t) = f(t, y(t)),\ y(t_0) = y_0$. Also, numerically integrating partial differential equations generally requires very different methods from the ones just discussed. Examples of methods to numerically integrate partial differential equations are finite difference methods, the finite element method and the finite volume method. The last two are discussed later in this course.