3 Integration of ordinary differential equations

One of the most fundamental types of mathematical problems in applied physics is finding the solution to coupled set of *ordinary differential equations* (ODEs), such as the equations of motion of a set of objects under the influence of external and/or mutual forces. This chapter will discuss some elementary methods of integrating such equations.

3.1 Ordinary differential equations

When we talk about ODEs we are concerned with relations between an unknown scalar or vector-values function $\mathbf{y}(t)$ and its derivatives with respect to the dependent variable (t in this case – the following discussion associated this with 'time', but this could of course be also any other variable). Such equations hence formally take the form

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = \mathbf{f}(\mathbf{y}, t), \tag{3.1}$$

and we seek the solution $\mathbf{y}(t)$, subject to boundary conditions.

Many simple dynamical problems can be written in this form, including ones that involve second or higher derivatives. This is done through a procedure called **reduction to 1st order**. One does this by adding the higher derivatives, or combinations of them, as further rows to the vector $\mathbf{y}$.

For example, consider a simple pendulum with the equation of motion

$$\ddot{q} = -\frac{g}{l}\sin(q), \tag{3.2}$$

where q is the angle with respect to the vertical, g the gravitational acceleration constant and l the length of the pendulum. The function $q(t)$ is to be solved. Now define $p \equiv q$, yielding a state vector

$$\mathbf{y} \equiv \begin{pmatrix} q \\ p \end{pmatrix}, \tag{3.3}$$

and a first order ODE of the form:

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = \mathbf{f}(\mathbf{y}) = \begin{pmatrix} p \\ -\frac{g}{l}\sin(q) \end{pmatrix}. \tag{3.4}$$

A numerical approximation to the solution of an ODE is a set of values $\{y_0, y_1, y_2, \ldots\}$ at discrete times $\{t_0, t_1, t_2, \ldots\}$, obtained for certain boundary conditions. The most common boundary condition for ODEs is the **initial value problem** (IVP), where the state of $\mathbf{y}$ is known at the beginning of the integration interval. It is however also possible to have mixed boundary conditions where $\mathbf{y}$ is partially known at both ends of the integration interval.

There are many different methods for obtaining a discrete solution of an ODE system. We shall here discuss some of the most basic ones, restricting ourselves to the IVP problem, for simplicity.

3.2 Explicit Euler method

This solution method, sometimes also called "forward Euler", uses the discretization of the derivative in the following way:

$$\frac{dy}{dt} \rightarrow \frac{y_{n+1} - y_n}{\Delta t} \tag{3.5}$$

The discrete version of the ODE then becomes

$$\frac{y_{n+1} - y_n}{\Delta t} = f(y_n) \tag{3.6}$$

By rearranging things we obtain the following iteration scheme:

$$y_{n+1} = y_n + f(y_n)\Delta t, \tag{3.7}$$

where y can also be a vector. Δt is the integration step.

- This approach is the simplest of all.
- The method is called *explicit* because y_{n+1} is computed with a right-hand-side that only depends on things that are already known.
- The stability of the method can be a sensitive function of the stepsize, and will in general only be reached for a sufficiently small step size.
- It is recommended to refrain from using this scheme in practice, since there are other methods that offer higher accuracy at the same or lower computational cost. The reason is that the Euler method is only *first order accurate*. To see this, note that the truncation error in a single step is of order $\mathcal{O}_s(\Delta t^2)$, which follows simply from a Taylor expansion. To simulate over time T, we need however $N_s = T/\Delta t$ steps, producing a total error that scales as $N_s \mathcal{O}_s(\Delta t^2) = \mathcal{O}_T(\Delta t)$.

We remark in passing that for a method to reach a global error that scales as $\mathcal{O}_T(\Delta t^n)$ (which is then called an "n^{th}-order accurate scheme), a local truncation error one order higher is required, i.e. $\mathcal{O}_s(\Delta t^{n+1})$.

3.3 Implicit Euler method

In a so-called "backwards Euler" scheme, one uses

$$y_{n+1} = y_n + f(y_{n+1})\Delta t, \tag{3.8}$$

which seemingly represents only a tiny change compared to the explicit scheme. The problem is now that the right-hand-side contains references to the value y_{n+1}, which is not know yet, when we integrate from small to large n. If the ODE is *linear*, e.g.

$$f(y_{n+1}) = Ay_{n+1} + B \tag{3.9}$$

then a simple rearrangement of terms can solve this problem:

$$(1 - A\Delta t)y_{n+1} = y_n + B\Delta t, \tag{3.10}$$

which leads to the following explicit expression for y_{n+1}:

$$y_{n+1} = \frac{y_n + B\Delta t}{1 - A\Delta t} \tag{3.11}$$

For *non-linear* equations this is not that easy. It requires solving a *root finding problem*, which can be done numerically using for instance a *Newton iteration procedure*. Because of their complexity we will not treat implicit integration methods for non-linear equations here. We will discuss this in Section 3.6.

- This approach has excellent stability properties, and for some problems, is in fact essentially always stable even for extremely large timestep. Note however that the accuracy will usually become very bad when using such large steps.
- This stability property makes implicit Euler sometimes useful for *stiff equations* where the derivatives (suddenly) can become very large.

3.4 Stability of the Euler method

Let's look at a simple problem to investigate the stability of forward and backwards Euler. Suppose we have the ODE

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -A\,y, \tag{3.12}$$

with $A > 0$ and $y(0) = y_0$. Here we of course know the analytic solution, given by $y(t) = y_0 \exp(-A\,t)$. This shows that there is a characteristic time scale τ of the ODE:

$$\tau = \frac{1}{A} \tag{3.13}$$

which is the time scale by which $y(t)$ changes as a result of the ODE.

What does *explicit* Euler give for this problem? We can work this out as

$$y_{n+1} = y_n + \dot{y}_n \Delta t = y_n - \alpha\, y_n \Delta t = y_n(1 - A\,\Delta t). \tag{3.14}$$

Hence every step gives us a factor $(1 - A\,\Delta t)$, and after n steps we have

$$y_n = (1 - A\,\Delta t)^n\, y_0. \tag{3.15}$$

We see that for $0 < 1 - A\,\Delta t < 1$ (i.e. equivalently for $\Delta t < 1/A$) the y_n monotonically decline. This is acceptable. For $-1 < 1 - A\,\Delta t < 0$ (i.e. equivalently for $1/A < \Delta t < 2/A$), the series oscillates but still declines overall, which is already somewhat problematic. But for $1 - A\,\Delta t < -1$ (i.e. for $\Delta t > 2/A$), the solution blows up and oscillates, which is as bad as it gets. So here the integration clearly becomes unstable for a too large timestep.

Now let's look at the *implicit* Euler instead. Here we have

$$y_{n+1} = y_n + \dot{y}_{n+1} \Delta t = y_n - A\, y_{n+1} \Delta t, \tag{3.16}$$

yielding

$$y_{n+1} = \frac{y_n}{1 + A\,\Delta t}, \tag{3.17}$$

which declines and is unconditionally stable for arbitrarily large timestep. Whether it is accurate for such large time steps is another issue.

It is therefore clear that for explicit schemes the time step *must* obey:

$$\Delta t = \frac{\eta}{A} = \eta\,\tau \tag{3.18}$$

where $0 < \eta \leq 1$ is the *safety factor*. For $\eta = 1$ we get $y_{n+1} = 0$, which is not unstable, but extremely inaccurate. So we must put η to a rasonably small number, say $\eta = 0.25$ or so. The smaller the better, but also the more costly a numerical integration becomes because we then need more time steps to reach some desired end-time $t_{\rm end}$.

How does this work for systems of coupled ODEs? Consider the equation

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = -\mathbf{A}\,\mathbf{y}, \tag{3.19}$$

where $\mathbf{y}$ is a vector of N elements and $\mathbf{A}$ is an $N \times N$ matrix. What time step should we take? The answer lies in the *eigenvalues* of the matrix A. There are typically N such eigenvalues $\alpha_1 \cdots \alpha_N$, each corresponding to a characteristic time scale $\tau_1 \cdots \tau_N$. The condition for the time step is then

$$\Delta t = \eta \min(\tau_1, \cdots, \tau_N) = \frac{\eta}{\max(\alpha_1, \cdots, \alpha_N)} \tag{3.20}$$

In other words: forward integration schemes require time steps smaller than the *smallest* characteristic time scale of the system.

3.5 Implicit midpoint rule

If we use

$$y_{n+1} = y_n + f\left(\frac{y_n + y_{n+1}}{2}\right)\Delta t \tag{3.21}$$

we obtain the implicit midpoint rule. This is *second order accurate*, but still implicit, so difficult to use in practice. Interestingly, it is also time-symmetric, i.e. one can formally integrate backwards and recover exactly the same steps (modulo floating point round-off errors) as in a forward integration.

3.6 Implicit integration of non-linear equations

Now consider a *non-linear* ordinary differential equation:

$$\frac{dy(t)}{dt} = f(y) \tag{3.22}$$

where $f(y)$ is a non-linear function of y. If we discretize this in implicit form we get

$$y_{n+1} = y_n + f(y_{n+1})\Delta t, \tag{3.23}$$

At each time step we know the value of y_n but we must find a solution for y_{n+1} to the above equation. We are now confronted with the classical problem of *root finding*: We must find a solution (root) of a simple but non-linear algebraic equation. In our case:

$$y_{n+1} - y_n - f(y_{n+1})\Delta t = 0 \tag{3.24}$$

where we must find a solution to y_{n+1}. We can write this in more abstract form:

$$F(y_{n+1}) \equiv y_{n+1} - y_n - f(y_{n+1})\Delta t = 0 \tag{3.25}$$

We now apply a Newton-Raphon iteration with initial guess $y_{n+1}^{(0)} = y_n$:

$$J^{(i)} \cdot (y_{n+1}^{(i+1)} - y_{n+1}^{(i)}) = -F(y_{n+1}^{(i)}) \tag{3.26}$$

where $J^{(i)}$ is the Jacobian:

$$J^{(i)} = \frac{\partial F(y_{n+1}^{(i)})}{\partial y_{n+1}^{(i)}} \tag{3.27}$$

These equations hold for ODEs of single and multiple variables. In the latter case $y_{n+1}^{(i)}$ and $F(y_{n+1}^{(i)})$ are vectors $J^{(i)}$ is a matrix.

3.7 Stiffness of coupled sets of ODEs

One may ask the following question about implicit methods. The implicit Eurer method (backward Euler) is not more accurate than forward Euler. The only advantage of the implicit method over the forward integration method is that it remains stable even for integration steps Δt that are much too large, much larger than the characteristic time scale of the system. But what is the advantage of a numerical method that is stable for large Δt but inaccurate? The answer lies in the comparison of the characteristic time scales of the system of equation.

Consider, for example, the following ODE:

$$\frac{dy(t)}{dt} = A\left(e^{i\omega t} - y(t)\right) \tag{3.28}$$

This equation has two characteristic time scales:

$$\tau_{\text{osc}} = \frac{1}{\omega}, \qquad \tau_{\text{coupling}} = \frac{1}{A}, \tag{3.29}$$

The first one, τ_{osc}, is the oscillation time scale of the $e^{i\omega t}$ function on the right-hand-side of the ODE. The second one, τ_{coupling}, is the time scale by which the function $y(t)$ is forced toward the $e^{i\omega t}$ function. For this example we happen to be able to find an analytic solution:

$$y(t) = \frac{A}{i\omega + A} e^{i\omega t} + be^{-At} \tag{3.30}$$

The first part is the particular solution, the second part the homogeneous solution with an integration constant b. The integration constant b is set by the initial condition $y(0)$:

$$b = y(0) - \frac{A}{i\omega + A} \tag{3.31}$$

The homogenous part of the solution thus takes care of the initial condition. But since e^{-At} decays on a time scale $\tau_{\text{coupling}} = 1/A$ this initial condition is soon "forgotten" and the solution approaches the particular solution. If $\tau_{\text{coupling}} \ll \tau_{\text{osc}}$ this period is much shorter than the typical time scale of the oscillation. The ODE is then said to be *stiff*. Most of the time the solution is nearly perfectly equal to the particular solution. If we are interested in this "most of the time" period, and we do not care too much if we accurately model the decaying homogeneous part (which anyway only makes a difference very shortly after $t = 0$ and soon decays), then we may *wish* to model on time scales of τ_{osc}. Indeed, with the exception of the first few decay times, the solution changes on time scales of τ_{osc}. However, if we choose $\Delta t \simeq eta\tau_{\text{osc}}$ a forward integration scheme would blow up, because forward integration schemes require time steps smaller than the *smallest* characteristic time scale. It is here that an implicit scheme helps: such a scheme allows us to take time steps Δt appropriate for the typical time scales of change occurring during the simulation:

$$\Delta t = \eta\, \tau_{\text{osc}} \tag{3.32}$$

instead of $\Delta t = \eta \min(\tau_{\text{coupling}}, \tau_{\text{osc}})$. This means that if $\tau_{\text{coupling}} \gtrsim \tau_{\text{osc}}$ then we model on the smallest time scale, but if $\tau_{\text{coupling}} < \tau_{\text{osc}}$ we model on the typical scale by which the system chances after the initial condition (the homogeneous part) has decayed. The best of both worlds. Implicit methods can thus help to solve *stiff ODEs*.

Another example of stiffness is from coupled sets of ODEs. Consider for instance the following coupled set of N ODEs:

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = -\mathbf{A}\,\mathbf{y} + \mathbf{g}(t) \tag{3.33}$$

The matrix has eigenvalues $\alpha_1 \cdots \alpha_N$, and thus characteristic time scales $\tau_1 \cdots \tau_N$ with $\tau_i = 1/\alpha_i$. In addition we have the characteristic time scale τ_g from the function $\mathbf{g}(t)$. Again, the time step for explicit integrators is set by the smallest time scale (cf. Eq. 3.20):

$$\Delta t = \eta \min(\tau_1, \cdots, \tau_N, \tau_g) \tag{3.34}$$

If

$$\frac{\max(\tau_1, \cdots, \tau_N, \tau_g)}{\min(\tau_1, \cdots, \tau_N, \tau_g)} \tag{3.35}$$

is large, then the system is said to be *stiff*. Implicit methods can help overcome the time constraint of the smallest τ if we are interested in the behavior at larger time scales.

3.8 Runge-Kutta methods

The Runge-Kutta schemes form a whole class of versatile integration methods. Let's derive one of the simplest Runge-Kutta schemes.

1. We start from the exact solution,

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f\left(y(t)\right) \mathrm{d}t. \tag{3.36}$$

2. Next, we approximate the integral with the (implicit) trapezoidal rule:

$$y_{n+1} = y_n + \frac{f(y_n) + f(y_{n+1})}{2} \Delta t. \tag{3.37}$$

3. Runge proposed in 1895 to predict the unknown y_{n+1} on the right hand side by an Euler step, yielding a *2nd order accurate Runge-Kutta scheme*, sometimes also called predictor-corrector scheme:

$$\begin{aligned} k_1 &= f(y_n, t_n), && (3.38)\\ k_2 &= f(y_n + k_1 \Delta t, t_{n+1}), && (3.39)\\ y_{n+1} &= \frac{k_1 + k_2}{2} \Delta t. && (3.40) \end{aligned}$$

   Here the step done with the derivate of equation (3.37) is called the 'predictor' and the one done with equation (3.38) is the corrector step.

Higher order Runge-Kutta schemes

A variety of further Runge-Kutta schemes of different order can be defined. Perhaps the most commonly used is the classical 4^{th}-order Runge-Kutta scheme:

$$\begin{aligned} k_1 &= f(y_n, t_n) && (3.41)\\ k_2 &= f\left(y_n + k_1\frac{\Delta t}{2}, t_n + \frac{\Delta t}{2}\right) && (3.42)\\ k_3 &= f\left(y_n + k_2\frac{\Delta t}{2}, t_n + \frac{\Delta t}{2}\right) && (3.43)\\ k_4 &= f\left(y_n + k_3\Delta t, t_n + \Delta t\right). && (3.44) \end{aligned}$$

These four function evaluations per step are then combined in a weighted fashion to carry out the actual update step:

$$y_{n+1} = y_n + \left(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}\right)\Delta t + \mathcal{O}(\Delta t^5). \tag{3.45}$$

We note that the use of higher order schemes also entails more function evaluations per step, i.e. the individual steps become more complicated and expensive. Because of this, higher order schemes are not always better; they usually are up to some point, but sometimes even a simple second-order accurate scheme can be the best choice for certain problems.

3.9 Adaptive step sizes

One issue left open in our discussion thus far is the choice of the optimum integration step. How can we get an optimum compromise between accuracy, efficiency and stability?

To control the accuracy, we need a way to estimate the local integration error, and a scheme for adjusting the step size such that a prescribed desired maximum error is guaranteed. One useful idea for controlling this lies in solving a step of size Δt twice, once with Δt (case a), and once by doing two steps of size $\frac{\Delta t}{2}$ (case b). The difference between the results then yields an estimate of the truncation error.

For example, suppose we use a p-th order scheme (e.g. Runge-Kutta forth order), then we have for the two end states:

$$\begin{aligned} y_a - y(t_0 + \Delta t) &= A \cdot (\Delta t)^{p+1} + \mathcal{O}(\Delta t^{p+2}), && (3.46)\\ y_b - y(t_0 + \Delta t) &= 2 \times \left[A \cdot (\Delta t/2)^{p+1}\right] + \mathcal{O}(\Delta t^{p+2}). && (3.47) \end{aligned}$$

This yields an error estimate of

$$\epsilon = |y_a - y_b| = A \cdot (\Delta t)^{p+1} \cdot (1 - 2^{-p}). \tag{3.48}$$

Assuming we are given a local error bound ϵ_0 for each step, we can now construct an algorithm where this is respected in every integration step:

1. Take a step of size Δt (yielding y_a), and two of stepsize $\frac{\Delta t}{2}$ (yielding y_b). We can then get an error estimate as $y = |y_a - y_b|$.
2. If $\epsilon > \epsilon_0$, discard the step, halve the timestep, $\Delta t' = \frac{\Delta t}{2}$, and try again.
3. If $\epsilon \ll \epsilon_0$, keep y_b and double the step for the next step, $\Delta t' = 2\Delta t$.
4. Else if $\epsilon < \epsilon_0$, keep y_b and retain Δt for the next step.

When does it make sense to double the step in point 3? This should only be done if the estimated new error is still smaller than the error bound, i.e. for

$$\epsilon \cdot 2^{p+1} < \epsilon_0. \tag{3.49}$$

But there is another complication. So far, our error estimate has been entirely local, disregarding the fact that we might need to do many integration steps. If we want to guarantee a certain error globally, even when all the errors add up with the same sign, we have to lower ϵ_0 with smaller step size Δt, because we then need to do more steps!

We can for example accounting for this by setting

$$\epsilon_0 = \frac{\Delta t}{T}\epsilon_0^{\text{global}}, \tag{3.50}$$

where T is the total integrated time, and $\epsilon_0^{\text{global}}$ is our prescribed error bound for the whole integration. This means we loose effectively one power of Δt again, because of

$$\epsilon \simeq \alpha \cdot (1 - 2^{-p}) \cdot \Delta t^{p+1} < \frac{\Delta t}{T}\epsilon_0^{\text{global}}. \tag{3.51}$$

Instead of step doubling one can also use a continuous step adjustment. The idea is to scale the timestep such that the estimated error matches the desired error ϵ_0:

$$(\Delta t)^{\text{desired}} = (\Delta t) \cdot \left(\frac{\epsilon_0}{\epsilon}\right)^{1/(p+1)}, \tag{3.52}$$

where ϵ is the error if a step of size Δt is taken, and ϵ_0 is the error level that one wants. $(\Delta t)^{\text{desired}}$ is then an estimate of the timestep that would deliver this error level.

Use of this in practice requires that we obtain for each step also an estimate of the error ϵ that is made. Then we can use this expression to determine the timestep for the next step. Such a 'built-in' error estimate can be delivered by so-called embedded Runge-Kutta schemes, which compute it more cheaply than done with the step-doubling technique.

A typical algorithm for continuous adaptive step size control would then for example work as follows:

1. Advance the system for a step Δt and estimate the error ϵ of the step at the same time (we here assume a p-th order scheme with $\mathcal{O}(\epsilon) = \Delta t^{p+1}$).

2. Calculate the new step size as

$$(\Delta t)^{\text{new}} = \beta \cdot (\Delta t) \cdot \left(\frac{\epsilon_0}{\epsilon}\right)^{1/(p+1)} \tag{3.53}$$

   where $\beta \sim 0.9$ is an empirical "safety factor".

3. If $\epsilon < \epsilon_0$ accept the step taken in (1), otherwise discard it and try again with new step size.

The well-known Runge-Kutta-Fehlberg integration method is of this type.

3.10 The leapfrog

Suppose we have a second order differential equation of the type

$$\ddot{x} = f(x). \tag{3.54}$$

This could of course be brought into standard form, $\dot{\mathbf{y}} = \tilde{\mathbf{f}}(\mathbf{y})$, by defining something like $\mathbf{y} = (x, \dot{x})$ and $\tilde{\mathbf{f}} = (\dot{x}, f(x))$, followed by applying a Runge-Kutta scheme as introduced above.

However, there is also another approach in this case, which turns out to be particularly simple and interesting. Let's define $v \equiv \dot{x}$. Then the so-called Leapfrog integration scheme is the mapping $(x_n, v_n) \to (x_{n+1}, v_{n+1})$ defined as:

$$v_{n+\frac{1}{2}} = v_n + f(x_n)\frac{\Delta t}{2}, \tag{3.55}$$

$$x_{n+1} = x_n + v_{n+\frac{1}{2}}\,\Delta t, \tag{3.56}$$

$$v_{n+1} = v_{n+\frac{1}{2}} + f(x_{n+1})\frac{\Delta t}{2}. \tag{3.57}$$

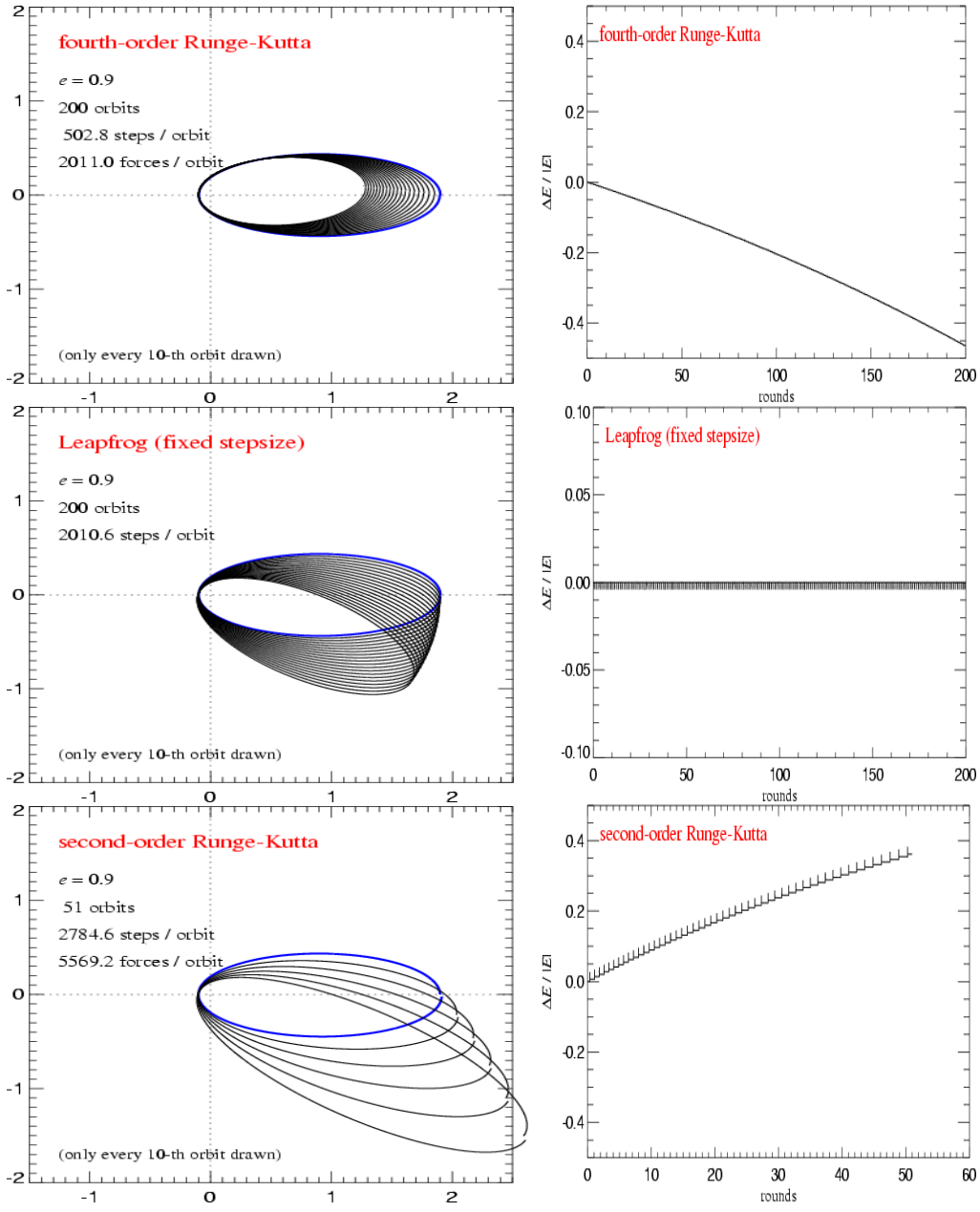
- This scheme is 2nd-order accurate (proof through Taylor expansion).
- It requires only 1 evaluation of the right hand side per step (note that $f(x_{n+1})$ can be reused in the next step.
- The scheme can be written in a number of alternative ways, for example by combining the two half-steps of two subsequent steps. One then gets

$$x_{n+1} = x_n + v_{n+\frac{1}{2}}\,\Delta t, \tag{3.58}$$

$$v_{n+\frac{3}{2}} = v_{n+\frac{1}{2}} + f(x_{n+1})\,\Delta t. \tag{3.59}$$

  One here sees the time-centered nature of the formulation very clearly, and the interleaved advances of position and velocity give it the name leapfrog.

The performance of the leapfrog on certain problems is found to be surprisingly good, better than that of other schemes such as Runge-Kutta which have formally the same or even a better error order. This is illustrated below for the Kepler problem, i.e. the integration of the motion of a small point mass in the gravitational field of a large mass.



We see that the long-term evolution is entirely different. Unlike the RK schemes, the leapfrog does not build up a large energy error. So why is the leapfrog behaving here so much better than other 2nd order or even 4th order schemes?

3.11 Symplectic integrators

The reason for these beneficial properties lies in fact that the leapfrog is a so-called symplectic method. These are structure-preserving integration methods that observe important special properties of Hamiltonian systems: Such systems have first conserved integrals (such as the energy), they also exhibit phase-space conservation as described by the Liouville theorem, and more generally, they preserve Poincare's integral invariants.

Symplectic transformations

- A linear map $F : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ is called symplectic if $\omega(F\xi, F\eta) = \omega(\xi, \eta)$ for all vectors $\xi, \eta \in \mathbb{R}^{2d}$, where w gives the area of the parallelgram spanned by the the two vectors.
- A differentiable map $g : U \to \mathbb{R}^{2d}$ is called symplectic if its Jacobian matrix is everywhere symplectic, i.e. $\omega(g'\xi, g'\eta) = \omega(\xi, \eta)$.
- **Poincare's theorem** states that the time evolution generated by a Hamiltonian in phase-space is a symplectic transformation.

The above suggests that there is a close connection between exact solutions of Hamiltonians and symplectic transformations. Also, two consecutive symplectic transformations are again symplectic.

Separable Hamiltonians

Dynamical problems that are described by Hamiltonians of the form

$$H(p, q) = \frac{p^2}{2m} + U(q) \tag{3.60}$$

are quite common. These systems have separable Hamiltonians that can be written as

$$H(p, q) = H_{\text{kin}}(p) + H_{\text{pot}}(q). \tag{3.61}$$

Now we will allude to the general idea of *operator splitting*. Let's try to solve the two parts of the Hamiltonian individually:

1. For the part $H = H_{\text{kin}} = \frac{p^2}{2m}$, the equations of motion are

$$\dot{q} = \frac{\partial H}{\partial p} = \frac{p}{m}, \tag{3.62}$$

$$\dot{p} = -\frac{\partial H}{\partial q} = 0. \tag{3.63}$$

These equations are straightforwardly solved and give

$$q_{n+1} = q_n + p_n \Delta t, \tag{3.64}$$

$$p_{n+1} = p_n. \tag{3.65}$$

Note that this solution is exact for the given Hamiltonian, for arbitrarily long time intervals Δt. Given that it is a solution of a Hamiltonian, the solution constitutes a symplectic mapping.

2. The potential part, $H = H_{\text{pot}} = U(q)$, leads to the equations

$$\dot{q} = \frac{\partial H}{\partial p} = 0 \tag{3.66}$$

$$\dot{p} = -\frac{\partial H}{\partial q} = -\frac{\partial U}{\partial q}. \tag{3.67}$$

This is solved by

$$q_{n+1} = q_n, \tag{3.68}$$

$$p_{n+1} = p_n - \frac{\partial U}{\partial q}\Delta t. \tag{3.69}$$

Again, this is an exact solution independent of the size of Δt, and therefore a symplectic transformation.

Let's now introduce an operator $\varphi_{\Delta t}(H)$ that describes the mapping of phase-space under a Hamiltonian H that is evolved over a time interval Δt, then it is easy to see that the leapfrog is given by

$$\varphi_{\Delta t}(H) = \varphi_{\frac{\Delta t}{2}}(H_{\text{pot}}) \circ \varphi_{\Delta t}(H_{\text{kin}}) \circ \varphi_{\frac{\Delta t}{2}}(H_{\text{pot}}) \tag{3.70}$$

for a separable Hamiltonian $H = H_{\text{kin}} + H_{\text{pot}}$.

- Since each individual step of the leapfrog is symplectic, the concatenation is also symplectic.

- In fact, the leapfrog generates the exact solution to a modified Hamiltonian H_{leap}, where $H_{\text{leap}} = H + H_{\text{err}}$. The difference lies in the 'error Hamiltonian' H_{err}, which is given by

$$H_{\text{err}} \propto \frac{\Delta t^2}{12}\left\{\{H_{\text{kin}}, H_{\text{pot}}\}, H_{\text{kin}} + \frac{1}{2}H_{\text{pot}}\right\} + \mathcal{O}(\Delta t^3), \tag{3.71}$$

where the curly brackets are Poisson brackets. This can be demonstrated by expanding

$$\mathrm{e}^{(H+H_{\text{err}})\Delta t} = \mathrm{e}^{H_{\text{pot}}\frac{\Delta t}{2}}\ \mathrm{e}^{H_{\text{kin}}\Delta t}\ \mathrm{e}^{H_{\text{pot}}\frac{\Delta t}{2}} \tag{3.72}$$

with the help of the Baker-Campbell-Hausdorff formula.

- This property explains the superior long-term stability of the integration of conservative systems with the leapfrog. Because it respects phase-space conservation, secular trends are largely absent, and the long-term energy error stays bounded and reasonably small.