2
Ordinary differential equations

2.1 Reduction to first order

An ordinary differential equation involves just one *independent* variable, x, and a *dependent* variable y. Obviously it involves *derivatives* of the dependent variable like $\frac{dy}{dx}$. The highest order differential, i.e. the term $\frac{d^N y}{dx^N}$ with the largest value of N appearing in the equation, defines the *order* N of the equation. So the most general ODE of order N can be written such that the Nth order derivative is equal to a function of all the lower order derivatives and the independent variable x:

$$\frac{d^N y}{dx^N} = f\left(\frac{d^{N-1}y}{dx^{N-1}}, \frac{d^{N-2}y}{dx^{N-2}}, \ldots, \frac{dy}{dx}, y, x\right). \tag{2.1}$$

Such an ordinary differential equation of order N in a single dependent variable y can always be reduced to a set of simultaneous coupled *first*-order equations involving N dependent variables. The simplest way to do this is to use a natural notation to denote by $y^{(i)}$ the ith derivative:

$$\frac{d^i y}{dx^i} = y^{(i)} \qquad i = 1, 2, \ldots, N-1. \tag{2.2}$$

When combined with the original equation, the total system can be written as a first-order vector differential equation whose components are

$$\frac{d}{dx} y^{(i)} = f_i(y^{(N-1)}, y^{(N-2)}, \ldots, y^{(1)}, y^{(0)}, x), \quad i = 0, 1, \ldots, N-1, \tag{2.3}$$

(where for notational consistency $y^{(0)} = y$). Explicitly in vector form:

$$\frac{d}{dx}\begin{pmatrix} y^{(0)} \\ y^{(1)} \\ \vdots \\ y^{(N-1)} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ f(y^{(N-1)}, y^{(N-2)}, \ldots, y^{(1)}, y^{(0)}, x) \end{pmatrix}. \tag{2.4}$$

https://doi.org/10.1017/CBO9781316155516.003 Published online by Cambridge University Press

Recognizing that the combined simultaneous vector system of dimension N with first-order derivatives is equivalent to a single scalar equation of order N, we often say that the order of the coupled vector system is still N. (Sorry if that seems confusing. In practice you get the hang of it.)

This is formal mathematics and applies to all equations, but precisely such a set of coupled first-order equations will often also arise directly in the formulation of the practical problem we are trying to solve. Suppose we are trying to track the position of a fluid element in a three-dimensional steady flow. If we know the fluid velocity $\boldsymbol{v}$ as a function of position $\boldsymbol{v}(\boldsymbol{x})$, then the equation of the track of a fluid element, i.e. the path followed by the element as it moves in time t, is

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{v}. \tag{2.5}$$

This is the equation we must solve to find a fluid streamline. It is of just the same form we derived by order reduction. Such a history of position as a function of time is called generically an orbit. Here the independent variable is t, and the dependent variable is $\boldsymbol{x}$. The vector $\boldsymbol{v}$ plays the role of the functions f_i.

Orbits may not be just in space, they may be in higher-dimensional phase-space. For example (see Fig. 2.1) to find the orbit of a charged particle (e.g. proton) of mass m_p and charge e moving at velocity $\boldsymbol{v}$ in a uniform magnetic field $\boldsymbol{B}$, we observe that it is subject to a force $e\boldsymbol{v} \times \boldsymbol{B}$. In the absence of any other forces, it has an equation of motion

$$\frac{d}{dt}\boldsymbol{v} = \frac{e}{m_p}\boldsymbol{v} \times \boldsymbol{B} = \boldsymbol{f}(\boldsymbol{v}), \tag{2.6}$$

in which the acceleration depends upon the velocity. This is a first-order vector differential equation, in three dimensions, where t plays the role of the

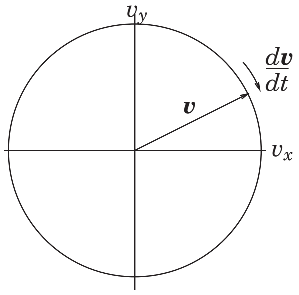

Figure 2.1 Orbit of the velocity of a particle moving in a uniform magnetic field is a circle in *velocity-space* perpendicular to the field ($B = B\hat{z}$ here).

independent variable, and the dependent variable is the vector velocity $\boldsymbol{v}$. The vector acceleration $\boldsymbol{f}$, which is the vector derivative function, depends upon all the components of $\boldsymbol{v}$.

If, for our proton orbit, $\boldsymbol{B}$ is not uniform, but varies with position, then we need to know both position $\boldsymbol{x}$ and velocity $\boldsymbol{v}$ at all times along the orbit to solve it. The system then involves six-dimensional vectors consisting of the components of $\boldsymbol{x}$ and $\boldsymbol{v}$:

$$\frac{d}{dt}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ (v_2B_3 - v_3B_2)e/m_p \\ (v_3B_1 - v_1B_3)e/m_p \\ (v_1B_2 - v_2B_1)e/m_p \end{pmatrix}. \tag{2.7}$$

Very often, to find *analytic* solutions it feels more natural to eliminate some of the dependent variables with the result that the order of the ODE is raised. So, for example for a uniform magnetic field in the 3-direction, the dynamics perpendicular to it separate out into

$$\frac{d}{dt}v_1 = \Omega v_2,\ \frac{d}{dt}v_2 = -\Omega v_1 \quad \Rightarrow \quad \frac{d^2v_1}{dt^2} = -\Omega^2 v_1,\ \frac{d^2v_2}{dt^2} = -\Omega^2 v_2 \tag{2.8}$$

(writing $\Omega = eB/m_p$). The second-order *uncoupled* equations are familiar to us as simple harmonic oscillator equations, having solutions like $\cos\Omega t$ and $\sin\Omega t$. So they are easier to solve *analytically*. But the original first-order equations, even though they are *coupled*, are far easier to solve *numerically*. So we don't generally do the elimination in computational solutions.

2.2 Numerical integration of initial-value problems

2.2.1 Explicit integration

Now we consider how in practice to solve a first-order ordinary differential equation in which all the boundary conditions are imposed at the same position in the independent variable. Such boundary conditions constitute what is called an "initial-value problem." We start integrating forward in the independent variable (e.g. time or space) from a place where the initial values are specified. To simplify the discussion we will consider a single (scalar) dependent variable y, but note that the generalization to a vector of dependent variables is usually immediate, so the treatment is fine for higher-order equations that have been reduced to vectorial first-order form.

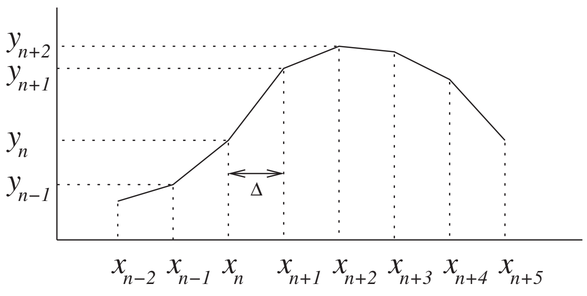
y_{n+2}
y_{n+1}
y_n
y_{n-1}
Δ
x_{n-2} x_{n-1} x_n x_{n+1} x_{n+2} x_{n+3} x_{n+4} x_{n+5}

Figure 2.2 Illustrating finite difference representation of a continuous function.

In general, numerical solution of differential equations requires us to represent the solution, which is usually continuous, in a discrete manner where the values are given at a series of points rather than continuously. See Fig. 2.2. The natural way to discretize the derivative is to write

$$\frac{dy}{dx} \approx \frac{y_{n+1} - y_n}{x_{n+1} - x_n} = f(y, x), \tag{2.9}$$

where the index n denotes the value at the nth discrete step, and therefore

$$y_{n+1} = y_n + f(y, x)(x_{n+1} - x_n). \tag{2.10}$$

This equation tells us how y changes from one step to the next. Starting from an initial position we can step discretely as far as we like, choosing the size of the independent variable step $(x_{n+1} - x_n) = \Delta$ appropriately.

A question that arises, though, is what to use for x and y inside the derivative function $f(y, x)$. The x value can be chosen more or less at will[1] but before we've actually made the step, we don't know where we are going to end up in y, so we can't easily decide where in y to evaluate f. The easiest answer, but not generally the best, is to recognize that at any point in stepping from n to $n + 1$ along the orbit, we already have the value y_n. So we could just use $f(y_n, x_n)$. This choice is said to be "explicit," and is sometimes called the Euler method. The reason why this method is not the best is because it tends to have poor *accuracy* and poor *stability*.

[1] If f is independent of y, then we should use $f(x_{n+1/2})$ where $x_{n+1/2} = (x_{n+1} + x_n)/2$, and we then have a formula accurate to second order for performing simple integration $\int_0^{x_n} f(x)dx = y_n - y_0 = \sum_{k=0}^{n-1} f(x_{k+1/2})(x_{k+1} - x_k)$. There are fancier, higher-order, ways to do integration, but this is by far the most straightforward.

2.2.2 Accuracy and Runge–Kutta schemes

To illustrate the problem of accuracy, consider the derivative function f expanded as a Taylor series about the x_n, y_n position, writing $x - x_n = \delta x$, $y - y_n = \delta y$. The derivative function f is a function of both x and y. However, the solution for the orbit can be written $y = y(x)$. Therefore, the function evaluated on the orbit, $f(y(x), x)$, is a function only of x, and we can write its (total) derivative as $\frac{df}{dx}$. The Taylor expansion of this function is simply[2]

$$f(y(x), x) = f(y_n, x_n) + \frac{df_n}{dx}\delta x + \frac{d^2 f_n}{dx^2}\frac{\delta x^2}{2!} + O(\delta x^3). \tag{2.11}$$

We use the notation df_n/dx (etc.) to indicate values evaluated at position n. If we substitute this Taylor expansion for f into the differential equation we are trying to solve, $dy/dx = d\delta y/d\delta x = f$, and integrate term by term, we get the *exact* solution of the differential equation:

$$\delta y = f_n \delta x + \frac{df_n}{dx}\frac{\delta x^2}{2!} + \frac{d^2 f_n}{dx^2}\frac{\delta x^3}{3!} + O(\delta x^4). \tag{2.12}$$

We subtract from it whatever the finite difference *approximate* equation is. In the case of eq. (2.10) it is $\delta y^{(1)} = f_n \delta x$, and we find that the error in y_{n+1} is

$$\delta y - \delta y^{(1)} = \epsilon = \frac{df_n}{dx}\frac{\delta x^2}{2!} + \frac{d^2 f_n}{dx^2}\frac{\delta x^3}{3!} + O(\delta x^4). \tag{2.13}$$

This tells us that the explicit Euler difference scheme is accurate only to *first order* in the size of the step δx (when the first derivative of f is non-zero) because an error of order δx^2 is present. That means if we make the step a factor of two smaller, the cumulative error, when integrating over a set total distance, gets smaller by (approximately) a factor of two. (Because each step's error is four times smaller, but there are twice as many steps.) That's not very good. We would have to take very small steps, δx, to get good accuracy.

We can do better. Our error arose because we approximated the derivative function by using its value only at x_n. But once we've moved to the next position, and know (with some inaccuracy) the value y_{n+1} and hence f_{n+1} there, we can evaluate *better* the f we should have used. This process is illustrated in Fig. 2.3. In fact, by substitution from eq. (2.11) it is easy to see that if we use, instead, the advancing equation

$$\delta y = \frac{1}{2}(f_n + f_{n+1}^{(1)})\delta x, \tag{2.14}$$

[2] The notation $O(\epsilon^n)$ denotes additional terms whose magnitude is proportional to a (usually small) parameter ϵ raised to the power n or higher.

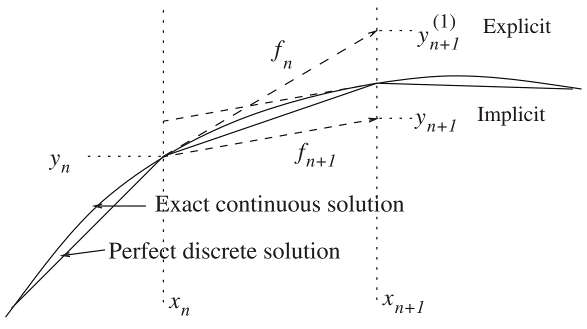

Figure 2.3 Optional steps using derivative function evaluated at n or $n+1$.

where $f_{n+1}^{(1)} = f(y_{n+1}^{(1)}, x_{n+1})$ is the value of f obtained at the end of our first (explicit Euler) advance $y_{n+1}^{(1)} = y_n + f_n\delta x$, then we would obtain for our *approximate* advancing scheme

$$\delta y = f_n\delta x + \frac{df_n}{dx}\frac{\delta x^2}{2!} + O(\delta x^3), \tag{2.15}$$

which now agrees with the *first two* terms of the full exact expansion (2.12), and whose error, in comparison with that expression, is now of *third* order, rather than second. This second-order accurate scheme gives cumulative errors proportional to δx^2 and so converges much more quickly as we shorten the step length.

The reason we obtained a more accurate step was that we used a more accurate value for the average (over the step) of the derivative function. It is straightforward to improve the average even more, so as to obtain even higher-order accuracy. But to do that requires us to obtain estimates of the derivative function part way through the step as well as at the ends. That's because we need to estimate the first and second derivatives of f.

A Runge–Kutta method consists of taking a series of steps, each one of which uses the estimate of the derivative function obtained from the previous one, and then taking some weighted average of their derivatives. Specifically, the *fourth-order (accurate) Runge–Kutta* scheme, which is by far the most popular and is illustrated in Fig. 2.4, uses:

$$\begin{aligned}
f^{(0)} &= f(y_n, x_n)\\
f^{(1)} &= f\left(y_n + f^{(0)}\frac{\Delta}{2}, x_n + \frac{\Delta}{2}\right)\\
f^{(2)} &= f\left(y_n + f^{(1)}\frac{\Delta}{2}, x_n + \frac{\Delta}{2}\right)\\
f^{(3)} &= f(y_n + f^{(2)}\Delta, x_n + \Delta),
\end{aligned} \tag{2.16}$$

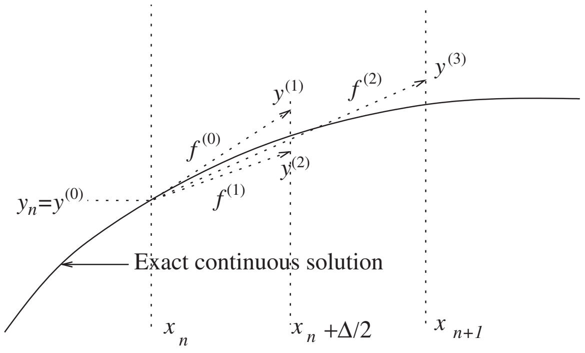

Figure 2.4 Runge–Kutta fourth-order scheme with four partial steps, evaluates the derivative function $f^{(k)}$ with $k = 0, 1, 2, 3$, at four places (x_k, y_k), each determined by extrapolation along the previous derivative.

in which two steps are to the half-way point $\frac{\Delta}{2}$. Then the following combination,

$$y_{n+1} = y_n + \left(\frac{f^{(0)}}{6} + \frac{f^{(1)}}{3} + \frac{f^{(2)}}{3} + \frac{f^{(3)}}{6} \right) \Delta + O(\Delta^5), \tag{2.17}$$

gives an approximation accurate to fourth order.[3]

The Runge–Kutta method costs more computation per step, because it requires four evaluations of the function $f(y, x)$, rather than just one. But that

[3] **Enrichment:** The proof is rather hard work. Here's an outline. Use notation $F(\delta x) = f(y(x_n + \delta x), \delta x)$ to refer to values of the derivative function along the exact orbit. Suppose we compose $\Delta y = (\frac{1}{6}F(0) + \frac{1}{3}F(\frac{\Delta}{2}) + \frac{1}{3}F(\frac{\Delta}{2}) + \frac{1}{6}F(\Delta))\Delta$. This is the form corresponding to eqs. (2.16) and (2.17), but using the exact f on the orbit, rather than the intermediate $f^{(n)}$ estimates. By substituting from eq. (2.11) it is easy but tedious to show that this Δy is equal to $y(x_n + \Delta) - y_n + O(\Delta^5)$; that is, it is an accurate representation of the y-step to fourth order. Actually it is instructive to realize that the symmetry of the δx positions, 0, $\frac{\Delta}{2}$, and Δ ensures that all even-order errors in this Δy are zero while the first-order error is zero because the coefficients sum to unity. The factor of two difference between the center and the end coefficients is the choice that makes the third-order error zero. In itself this has not proved the scheme to be fourth-order accurate, because there are non-zero differences approximately proportional to $\frac{\partial f}{\partial y}$ between the f-values and the F-values: $f^{(1)} - F(\frac{\Delta}{2}) = \frac{\partial f}{\partial y}(y^{(1)} - y(\frac{\Delta}{2}))$, $f^{(2)} - F(\frac{\Delta}{2}) = \frac{\partial f}{\partial y}(y^{(2)} - y(\frac{\Delta}{2}))$, and $f^{(3)} - F(\Delta) = \frac{\partial f}{\partial y}(y^{(3)} - y(\Delta))$. (Because of the order of the y-differences one can quickly see that $\frac{\partial^2}{\partial y^2}$ terms don't need to be kept.) The successive differences such as $(y^{(1)} - y(\frac{\Delta}{2}))$ can be expressed in terms of the Taylor expansion eq. (2.12), and then the f-differences are gathered in the combination of eq. (2.17): $\frac{1}{6}, \frac{1}{3}, \frac{1}{3}, \frac{1}{6}$. One can then evaluate the $\frac{\partial f}{\partial y}$ terms and demonstrate that they cancel to fourth order.

is often more than compensated by the ability to take larger steps than with the Euler method for the same accuracy.

2.2.3 Stability

The second, and possibly more important, weakness of explicit integration is in respect of stability. Consider a linear differential equation

$$\frac{dy}{dx} = -ky, \tag{2.18}$$

where k is a positive constant. This of course has the solution $y = y_0 \exp(-kx)$. But suppose we integrate it numerically using the explicit scheme

$$y_{n+1} = y_n + f(y_n, x_n)(x_{n+1} - x_n) = y_n(1 - k\Delta). \tag{2.19}$$

This finite difference equation has the solution

$$y_n = y_0(1 - k\Delta)^n, \tag{2.20}$$

as may be verified by the simple observation that $y_{n+1}/y_n = (1 - k\Delta)$. (This ratio is called the amplification factor.) If $k\Delta$ is a small number, then no difficulties will arise, and our scheme will produce approximately correct results. However, a choice $k\Delta > 2$ compromises not only accuracy but also *stability*. The resulting solution has alternating sign; it oscillates; but also its magnitude *increases* with n and will tend to infinity at large x. It has become unstable, as illustrated in Fig. 2.5.

In general, an explicit discrete advancing scheme requires the step in the independent variable to be less than some value (in this case $2/k$) in order to achieve stability.

An *implicit* advancing scheme, by contrast, is one in which the value of the derivative used to advance the variable is taken at the *end* of the step rather than at the *beginning*. For our example equation, this would be a numerical scheme of the form

$$y_{n+1} = y_n + f(y_{n+1}, x_{n+1})(x_{n+1} - x_n) = y_n - ky_{n+1}\Delta. \tag{2.21}$$

It is easy to rearrange this equation into

$$y_{n+1}(1 + k\Delta) = y_n. \tag{2.22}$$

This has solution

$$y_n = y_0(1 + k\Delta)^{-n}. \tag{2.23}$$

For positive $k\Delta$ (the case of interest) this finite difference equation *never* becomes unstable, no matter how big $k\Delta$ is, because the solution consists of

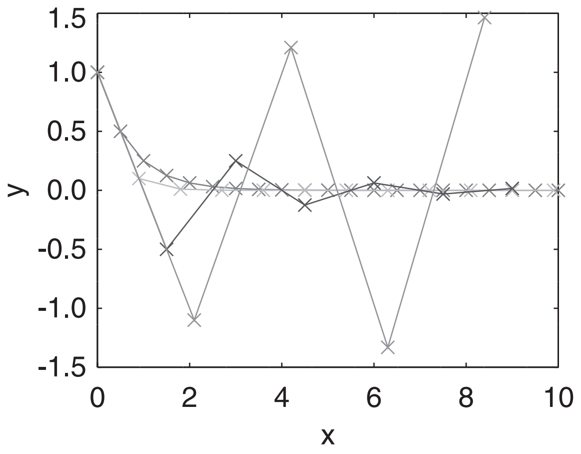

Figure 2.5 Explicit numerical integration of eq. (2.18), using eq. (2.19) leads to an oscillatory instability if the step length is too long. Four step lengths are shown $\Delta/k = 0.5, 0.9, 1.5, 2.1$.

successive powers of an amplification factor $1/(1 + k\Delta)$ whose magnitude is always less than 1. This is a characteristic of implicit schemes. They are generally stable even for large steps.

2.3 Multidimensional stiff equations: implicit schemes

The question of stability for an order-one system (the scalar problem) is generally not very interesting; because instability of the explicit scheme occurs only when the step size is longer than the characteristic spatial scale of the problem $1/k$. If you've chosen your step size so large, you are already failing to get an accurate solution of the equation. However, multi-dimensional (i.e. higher order) sets of (vector) equations may have multiple solutions that have very different scale-lengths in the independent variable. A classic example is an order-two homogeneous linear system with constant coefficients

$$\frac{d}{dx}\mathbf{y} = \mathbf{A}\mathbf{y}, \quad \text{where for example} \quad \mathbf{A} = \begin{pmatrix} 0 & -1 \\ 100 & -101 \end{pmatrix}. \tag{2.24}$$

For any such linear system the solution can be constructed by consideration of the *eigenvalues* of the matrix $\mathbf{A}$: those numbers for which there exists a solution to $\mathbf{Ay} = \lambda\mathbf{y}$. If these are λ_j and the corresponding eigenvectors are $\mathbf{y}_j$, then $\mathbf{y} = \mathbf{y}_j \exp(\lambda_j x)$ are solutions to the equation. The complete solution can be constructed as a sum of these different characteristic solutions, weighted by coefficients to satisfy the initial conditions. The point of our particular example matrix is that its eignvalues are -100 and -1. Consequently, in order to integrate numerically a solution that has a significant quantity of the second, slowly changing, solution $\lambda_2 = -1$, it is necessary nevertheless to ensure the stability of the first, rapidly changing, solution, $\lambda_1 = -100$. Otherwise, if the first solution is unstable, no matter how little of that solution we start with, it will eventually grow exponentially large and erroneously dominate our result. If an explicit advancing scheme is used, then stability requires $|\lambda_1|\Delta < 2$ as well as $|\lambda_2|\Delta < 2$, and the λ_1 condition is by far the most restrictive. There are then at least $\sim |\lambda_1/\lambda_2|$ steps during the decay of the (λ_2) solution of interest. Because this ratio is large, an explicit scheme is computationally expensive, requiring many steps. In general, the *stiffness* of a differential equation system can be measured by the ratio of the largest to the smallest eigenvalue magnitude. If this ratio is large, the system is stiff, and that means it is hard to integrate explicitly.

Using an implicit scheme avoids the necessity for taking very small steps. It does so at the cost of solving the matrix problem $(\mathbf{I} - \Delta.\mathbf{A})\mathbf{y}_{n+1} = \mathbf{y}_n$. This requires the inversion of a matrix in order to evaluate

$$\mathbf{y}_{n+1} = (\mathbf{I} - \Delta.\mathbf{A})^{-1}\mathbf{y}_n. \tag{2.25}$$

For a linear problem like the one we are considering, the single inversion, done once and for all, is a relatively small cost compared with the gain obtained by being able to take decent length steps.

All of this probably seems rather elaborate for a linear, constant coefficient, system, since we are actually able to solve it analytically when we know the eigenvalues and eigenvectors. However, it becomes much more significant when we realize that the stability of a *non-linear* system, or one in which the coefficients vary with x or y, for which numerical integration may be essential, is generally very well described by expressing it approximately as a linear system in the neighborhood of the region under consideration, and then evaluating the stability of the linearized system. The matrix that arises from linearization when the (vectorial) derivative function is $\mathbf{f}(\mathbf{y}, x)$ has the elements $A_{ij} = \partial f_i/\partial y_j$. An implicit solution then requires $(\mathbf{I} - \Delta.\mathbf{A})$ to be inverted for every step, because it is changing with position (if the derivative function is non-linear).

In short, implicit schemes lead to greater stability, which is very important with stiff systems, but they require matrix inversion.

2.4 Leap-Frog schemes

Codes such as particle-in-cell simulations of plasmas, atomistic simulation, or any codes that do a large amount of orbit integration, generally want to use as cheap a scheme as possible to maintain their speed. The step size is often dictated by questions other than the accuracy of the orbit integration. In such situations, Runge–Kutta or implicit schemes are rarely used. Instead, an accurate orbit integration can be obtained by recognizing that Newton's second law of motion (acceleration proportional to force) is a second-order vector equation that can most conveniently be split into two first-order vector equations involving position, velocity, and acceleration. The velocity we want for the equation governing the evolution of position, $d\boldsymbol{x}/dt = \boldsymbol{v}$, is the average velocity during the motion between two positions. An unbiassed estimate of that velocity is to take it to be the velocity at the *center* of the position step. So if the times at which we evaluate the position $\boldsymbol{x}_n$ of the particle are denoted t_n, then we want the velocity at time $(t_{n+1} + t_n)/2$. We might call this time $t_{n+1/2}$ and the velocity $\boldsymbol{v}_{n+1/2}$. Also, the acceleration we want for the equation for the evolution of velocity, $d\boldsymbol{v}/dt = \boldsymbol{a}$, is the average acceleration between two velocities. If the velocities are represented at $t_{n-1/2}$ and $t_{n+1/2}$, then the time at which we want the acceleration is t_n, and we might call that acceleration $\boldsymbol{a}_n$.

We can therefore construct an appropriate centered integration scheme as follows starting from position n.

(1) Move the particles using their velocities $\boldsymbol{v}_{n+1/2}$ to find the new positions $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{v}_{n+1/2}(t_{n+1} - t_n)$; this is called the drift step.
(2) Accelerate the velocities using the accelerations $\boldsymbol{a}_{n+1}$ evaluated at the new positions $\boldsymbol{x}_{n+1}$ to obtain the new velocities $\boldsymbol{v}_{n+3/2} = \boldsymbol{v}_{n+1/2} + \boldsymbol{a}_{n+1}(t_{n+3/2} - t_{n+1/2})$; this is called the kick step.
(3) Repeat (1) and (2) for the next time step $n + 1$, and so on.

Each of the drift and kick steps can be simple explicit advances. But because the velocities are suitably time-shifted relative to the positions and accelerations, the result is *second-order accurate*. Such a scheme is called a Leap-Frog scheme, in reference to the children's game where each of two players alternately jumps over the back of the other in moving to the new position. Velocity and position are jumping over one another in time; they never land at the same time (or place). See Fig. 2.6.

One trap for the unwary in a Leap-Frog scheme is the specification of initial values. If we want to calculate an orbit which has specified initial position $\boldsymbol{x}_0$ and velocity $\boldsymbol{v}_0$ at time $t = t_0$, then it is not sufficient simply to put the velocity initially equal to the specified $\boldsymbol{v}_0$. That is because the first velocity,

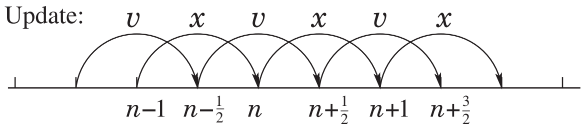

Figure 2.6 Staggered values for x and v are updated leap-frog style.

which governs the position step from t_0 to t_1 is *not* $\boldsymbol{v}_0$, but $\boldsymbol{v}_{1/2}$. To start the integration off correctly, therefore, we must take a *half-step* in velocity by putting $\boldsymbol{v}_{1/2} = \boldsymbol{v}_0 + \boldsymbol{a}_0(t_1 - t_0)/2$, before beginning the standard integration.

Leap-Frog schemes generally possess important conservation properties such as conservation of momentum, that can be essential for realistic simulation with a large number of particles.

Worked example. Stability of finite differences

Find the stability limits when solving by discrete finite differences of length Δ, using explicit or implicit schemes, the initial-value equation

$$\frac{d^2y}{dx^2} + 2\alpha\frac{dy}{dx} + k^2y = g(x), \tag{2.26}$$

where α is a positive constant $< |k|$.

First, reduce the equation to standard first-order form by writing

$$\begin{aligned}\frac{d}{dx}\begin{pmatrix} y^{(0)} \\ y^{(1)} \end{pmatrix} &= \begin{pmatrix} y^{(1)} \\ g(x) - 2\alpha y^{(1)} - k^2 y^{(0)} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 \\ -k^2 & -2\alpha \end{pmatrix}\begin{pmatrix} y^{(0)} \\ y^{(1)} \end{pmatrix} + \begin{pmatrix} 0 \\ g(x) \end{pmatrix}.\end{aligned} \tag{2.27}$$

Now, notice that, in either form, this equation has a sort of driving term $g(x)$ independent of y. It can be removed from the stability analysis by considering a new variable z, which is the difference between the finite difference numerical solution that we are trying to analyse, $y_n(x)$, and the exact solution of the differential equation, $y(x)$. Thus $z_n = y_n - y(x_n)$. The quantity z satisfies a discretized form of the *homogeneous* version of eq. (2.26), with the term $g(x)$ removed. Consequently analysing that homogeneous equation tells us whether the difference z between the numerical and the exact solutions is stable or not. That is what we really want to know. For stability, we pay no attention to $g(x)$. The resulting homogeneous equation is of exactly the same form as eq. (2.24): $\frac{d}{dx}\mathbf{z} = \mathbf{A}\mathbf{z}$. Each independent solution for $\mathbf{z}$ is thus an eigenvector of the matrix

$\mathbf{A}$ such that $\frac{d}{dx}\mathbf{z} = \lambda\mathbf{z}$. The eigenvalue λ decides its stability. An explicit (Euler) scheme of step length Δ will result in a step amplification factor, such that $\mathbf{z}_{n+1} = F\mathbf{z}_n$ with $F = 1 + \lambda\Delta$, and will be unstable if $|F| > 1$.

The eigenvalues of $\mathbf{A}$ satisfy

$$0 = \begin{vmatrix} -\lambda & 1 \\ -k^2 & -\lambda - 2\alpha \end{vmatrix} = \lambda^2 + 2\alpha\lambda + k^2, \tag{2.28}$$

with solutions

$$\lambda = -\alpha \pm \sqrt{\alpha^2 - k^2}. \tag{2.29}$$

These solutions are complex if $k^2 > \alpha^2$, in which case,

$$|F|^2 = (1 - \alpha\Delta)^2 + (k^2 - \alpha^2)\Delta^2 = 1 - 2\alpha\Delta + k^2\Delta^2. \tag{2.30}$$

$|F|$ is greater than unity unless $\Delta < 2\alpha/k^2$, which is the stability criterion of the Euler scheme. If $\alpha = 0$, so that the homogeneous equation is an undamped harmonic oscillator, the explicit scheme is unstable for any Δ. A fully implicit scheme, by contrast, has $F = \mathbf{z}_{n+1}/\mathbf{z}_n = 1/(1 - \lambda\Delta)$, for which $|F|^2 = 1/(1 + 2\alpha\Delta + k^2)$, always less than one. The implicit scheme is always stable.

Exercise 2. Integrating ordinary differential equations

1. Reduce the following higher-order ordinary differential equations to first-order vector differential equations, which you should write out in vector format:

(a) $\frac{d^2y}{dt^2} = -1$,

(b) $Ay + B\frac{dy}{dx} + C\frac{d^2y}{dx^2} + D\frac{d^3y}{dx^2} = E$, and

(c) $\frac{d^2y}{dx^2} = 2\left(\frac{dy}{dx}\right)^2 - y^3$.

2. Accuracy order of ODE schemes. For notational convenience, we start at $x = y = 0$ and consider a small step in x and y of the ODE $dy/dx = f(y, x)$. The Taylor expansion of the derivative function along the orbit is

$$f(y(x), x) = f_0 + \frac{df_0}{dx}x + \frac{d^2f_0}{dx^2}\frac{x^2}{2!} + \dots. \tag{2.31}$$

(a) Integrate $\frac{dy}{dx} = f(y(x), x)$ term by term to find the solution for y to third order in x.

(b) Suppose $y_1 = f_0 x$. Find $y_1 - y(x)$ to second order in x.

(c) Now consider $y_2 = f(y_1, x)x$, show that it is equal to $f(y, x)x$ plus a term that is third order in x.

(d) Hence find $y_2 - y$ to second order in x.
(e) Finally, show that $y_3 = \frac{1}{2}(y_1 + y_2)$ is equal to y accurate to *second order* in x.

[Comment. The third-order term in part (c) involves the partial derivative $\partial f/\partial y$ rather than the derivative along the orbit. Proving rigorously that the fourth-order Runge–Kutta scheme really is fourth order is rather difficult because it requires keeping track of such partial derivatives.]

3. **Programming exercise.** Write a program to integrate numerically from $t = 0$ to $t = 4/\omega$ the ODE

$$\frac{dy}{dt} = -\omega y,$$

with ω a positive constant, starting from $y(0) = 1$, proceeding as follows.

(a) Use the explicit Euler scheme

$$y_{n+1} = y_n - \Delta t \omega y_n.$$

(b) Use the implicit scheme

$$y_{n+1} = y_n - \Delta t \omega y_{n+1}.$$

In each case, find numerically the *fractional* error at $t = 4/\omega$ for the following choices of timestep:

(i) $\omega \Delta t = 0.1$, (ii) $\omega \Delta t = 0.01$, and (iii) $\omega \Delta t = 1$.

(c) Find experimentally the timestep value at which the explicit scheme becomes unstable. Verify that the implicit scheme never becomes unstable.