Intro to numerical methods for ordinary differential equations

Marc Kjerland

July 14, 2016

1 Introduction

This handout will be a reference for simple numerical methods for ordinary differential equations (ODEs).

We are interested solving an *initial value problem* for a differentiable function of one variable, $u(t)$. The values of u are unknown except at a single time, $t = t_0$, but we know the rate of change of u via the differential equation

$$u'(t) = f(u(t)), \tag{1}$$

where $u'(t) = \frac{\mathrm{d}u}{\mathrm{d}t}$ and where f is a well-behaved function which is given, possibly based on physical laws, and which depends on the current physical state u.

The initial value problem is given as

$$\begin{cases} u'(t) = f(u(t)), \\ u(t_0) = u_0. \end{cases} \tag{2}$$

In general f can also be a function of t, for example if there is some external time-dependent forcing like a solar cycle. For comparison:

$$u'(t) = f(t, u(t)) \tag{3}$$

Equation (3) is said to be *non-autonomous* and (1) is *autonomous*. For simplicity we'll only consider the autonomous case, where f is independent of t, but the following methods can all be readily extended to the non-autonomous case.

Note that there is a close connection between the methods in this paper and the quadrature methods for finding the area under a curve: the rectangle method, the trapezoid rule, Simpson's rule, etc.

1.1 Systems of equations

All of the methods listed here can be applied to systems of differential equations:

$$\begin{cases} u_1' = f_1(u_1, u_2, \ldots, u_n), \\ u_2' = f_2(u_1, u_2, \ldots, u_n), \\ \ldots \\ u_n' = f_n(u_1, u_2, \ldots, u_n). \end{cases} \tag{4}$$

This can be written concisely as

$$\vec{u}' = \vec{f}(\vec{u}). \tag{5}$$

For simplicity in notation, we can leave off the $\vec{}$ and just write u and f.

2 Euler's method

We can solve the initial value problem (2) on a discretized time domain by replacing the derivative $u'(t)$ with a finite difference:

$$\frac{u(t + \Delta t) - u(t)}{\Delta t} \approx f(u(t)). \tag{6}$$

Moving some terms around we have

$$u(t + \Delta t) \approx u(t) + \Delta t f(u(t)) \tag{7}$$

Using this, we can find an approximate solution for u recursively, starting with $u(t_0)$ to compute $u(t_0 + \Delta t)$, then using this to compute $u(t_0 + 2\Delta t)$, and so on.

We'll use the following simplifying notation: $t_n = t_0 + n\Delta t$ and $u_n = u(t_n)$. Frequently in this handout we'll assume that we know the value of u_n at the current time $t = t_n$, and we want to solve for u_{n+1} at the next time t_{n+1}.

Using this notation we can rewrite (7) as:

$$\boxed{u_{n+1} = u_n + \Delta t f(u_n)} \qquad \text{(Forward Euler)}$$

This is known as Euler's method, or more specifically the Forward Euler method, named after the 18th Century Swiss mathematician Leonhard Euler. This is the simplest *explicit* method, where the value at the next time step, $u(t + \Delta t)$, is given explicitly as a function of known values at previous times.

3 Accuracy

Surely replacing the derivative term with a finite difference will introduce some error into our solution, and intuitively we expect the error to depend on the size of Δt which is no longer infinitesimal. Study of the error term is known as the *accuracy* of the method.

How accurate is the Forward Euler method? Let's expand $u(t + \Delta t)$ in a Taylor series (See Appendix A and B):

$$\begin{aligned} u(t + \Delta t) &= u(t) + \Delta t\, u'(t) + \frac{\Delta t^2}{2} u''(t) + \dots \\ &= u(t) + \Delta t\, u'(t) + \mathcal{O}(\Delta t^2). \end{aligned}$$

Remember that $u'(t) = f(u(t))$, so

$$u(t + \Delta t) - u(t) = \Delta t f(u(t)) + \mathcal{O}(\Delta t^2). \tag{8}$$

The *local truncation error* at each integration step is proportional to Δt^2. Over multiple integration steps these errors will accumulate to a *global truncation error*; since the total number of integration steps is proportional to Δt^{-1}, the global truncation error is proportional to $\Delta t^2 \cdot \Delta t^{-1} = \Delta t$. So if you want to reduce the errors in your solution by a factor of 2, you must reduce the step size by a factor of 2, which means you will perform twice as many calculations.

Are there more accurate methods? Perhaps with a global truncation error of $\mathcal{O}(\Delta t^2)$ or better? There are, as we'll see in the next section. A method with error $\mathcal{O}(\Delta t^n)$ is said to be *order-n* accurate or *nth-order* accurate. The Forward Euler method is *order-1* accurate.

4 Higher order methods

There is a family of numerical methods which use multiple values of u' between t and $t + \Delta t$ to get a more accurate estimate for $u(t + \Delta t)$. These methods (just like higher-order quadrature methods) require only that the higher derivates of f are bounded.

The first example is the *midpoint method*, a 2nd-order method.

4.1 Midpoint method

The basic idea of the midpoint method is that the finite difference should be more accurate if it is centered around the derivative:

$$\frac{u(t+\Delta t)-u(t)}{\Delta t} \approx u'(t+\frac{\Delta t}{2}). \tag{9}$$

This can be verified by replacing $u(t+\Delta t)$ and $u(t)$ with Taylor expansions of u around $t+\frac{\Delta t}{2}$:

$$\begin{aligned} u(t+\Delta t) &= u(t+\frac{\Delta t}{2}) + \frac{\Delta t}{2}u'(t+\frac{\Delta t}{2}) + \frac{\Delta t^2}{8}u''(t+\frac{\Delta t}{2}) + \mathcal{O}(\Delta t^3) \\ u(t) &= u(t+\frac{\Delta t}{2}) - \frac{\Delta t}{2}u'(t+\frac{\Delta t}{2}) + \frac{\Delta t^2}{8}u''(t+\frac{\Delta t}{2}) + \mathcal{O}(\Delta t^3) \\ \implies u(t+\Delta t) - u(t) &= \Delta t\, u'(t+\frac{\Delta t}{2}) + \mathcal{O}(\Delta t^3). \end{aligned}$$

So, if we apply this to our differential equation (1), we have

$$u_{n+1} = u_n + \Delta t\, f(u_{n+1/2}) + \mathcal{O}(\Delta t^3). \tag{10}$$

Unfortunately we don't know the value of $u_{n+1/2}$. However, we know how to approximate it with a Taylor expansion:

$$\begin{aligned} u(t+\frac{\Delta t}{2}) &= u(t) + \frac{\Delta t}{2}u'(t) + \mathcal{O}(\Delta t^2) \\ \implies u_{n+1/2} &= u_n + \frac{\Delta t}{2}f(u_n) + \mathcal{O}(\Delta t^2). \end{aligned} \tag{11}$$

Putting this all together we have:

$$\boxed{u_{n+1} = u_n + \Delta t\, f\left(u_n + \tfrac{\Delta t}{2} f(u_n)\right)} \qquad \text{(Midpoint method)}$$

From (10) we see that the midpoint method has local error of order Δt^3, and as long as f is differentiable the remainder term in (11) won't result in larger errors. Thus the midpoint method has a *global* error of Δt^2, so we say it is a 2nd-order method.

This accuracy comes at a cost: for each integration step that the Forward Euler method would take, the midpoint method requires two integration steps: first to find $u_{n+1/2}$ then to find u_{n+1}. However, this is a small price to pay for the increased order of accuracy.

4.2 Runge-Kutta methods

Following the example of the midpoint method, we can add more and more intermediate terms to our calculations to increase the order of accuracy. Two

German mathematicians, C. Runge and M.W. Kutta, explored these methods around the year 1900 and now they are known as the Runge-Kutta methods. The midpoint method is a simple case of a Runge-Kutta method and is sometimes referred to as RK2.

One of the most popular Runge-Kutta methods is 4th-order and requires four calculations at each integration step. It is often referred to as RK4:

$$\boxed{\begin{aligned} u_{n+1} &= u_n + \frac{\Delta t}{6}(k_1 + k_2 + k_3 + k_4) \\ k_1 &= f(u_n) \\ k_2 &= f(u_n + \frac{\Delta t}{2} k_1) \\ k_3 &= f(u_n + \frac{\Delta t}{2} k_2) \\ k_4 &= f(u_n + \Delta t k_3). \end{aligned}} \tag{RK4}$$

Matlab's standard ODE solver `ode45` is a 4th order Runge-Kutta solver (but it isn't RK4: it also uses an adaptive timestep feature).

5 Stability

Sometimes a differential equation is especially challenging to solve numerically, and if your timestep isn't small enough your solution will be very wrong. This problem is known as *stability*, and particularly troublesome equations are sometimes called "stiff."

Here is a very simple example which turns out to be a stiff problem. Consider a linear drag law, or Stokes' law, where the frictional force F_d on an object or a fluid parcel is proportional to the velocity u:

$$F_d = -\alpha u, \tag{12}$$

for some $\alpha > 0$. For an object with unit mass, by Newton's second law the acceleration $\frac{\mathrm{d}u}{\mathrm{d}t}$ is given by:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -\alpha u. \tag{13}$$

This is very simple to solve analytically [what is the solution?] but let's apply the Forward Euler method:

$$\begin{aligned} \frac{u_{n+1} - u_n}{\Delta t} &= -\alpha u_n \\ \implies u_{n+1} &= u_n - \Delta t\, \alpha u_n \\ &= (1 - \Delta t\, \alpha) u_n. \end{aligned} \tag{14}$$

If the timestep Δt is small enough, we'll get the result we expect. But what happens if the drag coefficient α is large, or Δt is large, so that $\Delta t \geq \alpha^{-1}$?

[Try an example! $u_n = 5, \alpha = 10, \Delta t = 0.5$]

When $\Delta t > \alpha^{-1}$ the method becomes *unstable* and the solutions will be unphysical.

The stability of a numerical method depends strongly on the differential equation itself, and this issue is especially important for numerical solution of partial differential equations (PDEs).

5.1 CFL condition

In fluid dynamics there is an important stability criteria known as the *CFL condition* (Courant, Friedrichs, Lewy), sometimes called the Courant condition. It specifies an upper limit for the timestep size Δt based on the spatial grid size Δx and the speed C of the fastest characteristic wave:

$$\Delta t \leq \frac{\Delta x}{C} \tag{CFL}$$

In an explicit numerical scheme, this condition must be satisfied or the solution will be unphysical.

For example: long water waves propagate with speed $u \pm \sqrt{gh}$, where u is the water current speed and h is the height of the water column. So we would take $C = \max_x(|u| + \sqrt{gh})$ to find a constraint for the timestep. This means we may have to take smaller time steps if there are areas of fast flow, high waves, or deeper bathymetry.

Another example is the speed of sound in a compressible gas, which is much higher than the speed of gravity waves; this is one reason why the compressible Euler equations are not used in meteorology.

The CFL condition creates a costly problem - if you want fine spatial resolution you will reduce Δx, but by the CFL conditon you must also reduce Δt. [See: Curse of Dimensionality]

6 Implicit methods

Stability constraints can sometimes be severe, requiring Δt to be so small that your numerical solution is very costly to compute. One way to avoid this

is to use an *implicit* method.

Going back to our original ODE (1), recall we replaced the derivative on the left hand side with a difference equation from u_n to u_{n+1} and we used u_n on the right hand side, like so:

$$\frac{u_{n+1} - u_n}{\Delta t} = f(u_n). \tag{15}$$

What if we instead used u_{n+1} in the right hand side?

$$\frac{u_{n+1} - u_n}{\Delta t} = f(u_{n+1}). \tag{16}$$

Rearranging the terms we have

$$\boxed{u_{n+1} + \Delta t\, f(u_{n+1}) = u_n} \qquad \text{(Backward Euler)}$$

Now we would have to solve an algebraic equation for u_{n+1}, and often this is done using an iterative solution method such as Newton's method.

For a linear system the Backward Euler method is much simpler:

$$u' = Au \tag{17}$$

where A is a scalar or matrix. Then we have

$$\begin{aligned} &\frac{u_{n+1} - u_n}{\Delta t} = Au_{n+1} \\ \implies &u_{n+1} - \Delta t A u_{n+1} = u_n \\ \implies &(\mathbb{1} - \Delta t A)\, u_{n+1} = u_n \\ \implies &u_{n+1} = (\mathbb{1} - \Delta t A)^{-1}\, u_n, \end{aligned} \tag{18}$$

where $\mathbb{1}$ is an identity matrix (or 1 in the scalar case).

Thus to solve the linear equation (17) the Backwards Euler method is given by

$$\boxed{u_{n+1} = (\mathbb{1} - \Delta t A)^{-1} u_n} \qquad \text{(Backward Euler, linear)}$$

The Backward Euler method is the simplest implicit method, and it has the same accuracy as the Forward Euler method (order 1). Higher order implicit methods exist, as well as hybrid implicit-explicit methods.

Implicit methods always require some extra calculation steps such as iteration or matrix inversion* which in practice can be costly. However, they are invaluable for certain problems (as we'll see in the next example).

[*Here is an excellent essay why you should never actually invert a matrix: http://www.johndcook.com/blog/2010/01/19/dont-invert-that-matrix/]

Note 1: The Backward Euler method will fail if $(\mathbb{1} - \Delta t A)$ is not invertible, but in practice this is usually not a problem (see what happens in the next example).

Note 2: In the nonlinear case, an approximation can be made to f at each time step such as replacing it locally with with a tangent line. Then linear algebra can be used to get a solution.

6.1 Application to stiff problem

Implicit methods are best suited for stiff problems. Let's return to the scalar ODE for linear drag law (Stokes law):

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -\alpha u \tag{19}$$

and now let's apply the Backward Euler method:

$$\begin{aligned} & \frac{u_{n+1} - u_n}{\Delta t} = -\alpha u_{n+1} \\ \implies & u_{n+1} = u_n - \Delta t\, \alpha u_{n+1} \\ \implies & u_{n+1} + \Delta t\, \alpha u_{n+1} = u_n \\ \implies & (1 + \Delta t\, \alpha) u_{n+1} = u_n \\ \implies & u_{n+1} = \frac{u_n}{1 + \Delta t\, \alpha}. \end{aligned} \tag{20}$$

Compare this with the explicit case (14). In this implicit case, the term $(1 + \Delta t\, \alpha)$ is always positive so u_{n+1} will always be the same sign as u_n, and this will drive the solution towards zero as expected. In fact, Δt could be quite large and it will still give a qualitatively correct result (although the accuracy will suffer).

So, implicit methods help ignore stability constraints and thus allow for larger Δt, resulting in fewer timesteps in total. The trade-off is that each timestep will be more costly, especially with high-dimensional systems, and the cost also usually increases with the size of Δt.

A "Big Oh" notation

The "Big Oh" notation is often used in applied math and computer science when discussing roughly how large (or small) a function is depending on some variable. You can think of it like an Order of Magnitude. In this handout we

are specifically interested how a small parameter like Δt affects the error or remainder term in our approximations.

We say $g(h) = \mathcal{O}(h)$ if there is a finite constant C such that $\frac{g(h)}{h} \to C \neq 0$ as $h \to 0$. So roughly $g(h) \approx Ch$ for small values of h.

Example. Let $g(h) = -6h^2 + 25h^3 + 100h^4$. We are interested in values of h which are small (close to 0). In that case, $|h^4| << |h^3| << |h^2|$. So for small h, $g(h) \approx -6h^2$. Then we say $g(h) = \mathcal{O}(h^2)$.

In this handout if you see an expression like $\mathcal{O}(\Delta t^3)$, that means there is some error/remainder term which might not be known exactly but which is roughly the size of Δt^3.

In computer science, big-oh notation is usually used to describe how long an algorithm takes based on the problem size, represented by a large integer n, such as matrix multiplication or the famous *P vs NP* question.

Trivia (1): Multiplying two $n \times n$ matrices takes $\mathcal{O}(n^3)$ steps using the basic method, but the best algorithm takes $\mathcal{O}(n^{2.8})$ steps. For 1000×1000 matrices, in theory that's nearly four times faster!

Trivia (2): Computing the discrete Fourier transform of n data points based on the definition takes $\mathcal{O}(n^2)$ steps, but the Fast Fourier Transform does it in $\mathcal{O}(n \log n)$ steps. For 1000 data points that's a speedup of 100x !!

B Taylor series expansion

Taylor's Theorem tells us that we can represent a function $g(x)$ by its Taylor series expansion

$$g(x+h) = g(x) + hg'(x) + \frac{h^2}{2}g''(x) + \frac{h^3}{6}g'''(x) + \dots \tag{21}$$

within a small distance h from x. Furthermore, if g is nice (its higher derivatives aren't too big) then we can truncate the Taylor series and get a good approximation:

$$\begin{aligned}
g(x+h) &\approx g(x) + hg'(x) && +\mathcal{O}(h^2) \\
g(x+h) &\approx g(x) + hg'(x) + \frac{h^2}{2}g''(x) && +\mathcal{O}(h^3) \\
g(x+h) &\approx g(x) + hg'(x) + \frac{h^2}{2}g''(x) + \frac{h^3}{6}g'''(x) && +\mathcal{O}(h^4)
\end{aligned}$$

and so on, where the size of the error is roughly a power of h as $h \to 0$.

C Pronounciation guide

Euler: オイラー

Runge-Kutta: ルンゲ　クッタ