Notes for CS542G

Robert Bridson

October 12, 2007

1 Stability of Forward Euler

The linear analysis of FE begins by plugging in the test equation

$$\frac{dy}{dt} = \lambda y, \qquad y(0) = 1$$

to get:

$$y_{n+1} = y_n + \Delta t \lambda y_n$$

We next **assume** that Δt is held constant through the iteration. This sometimes is the case, but certainly not always—we will look later at adaptive time step selection, where Δt may be modified to make a better accuracy/speed trade-off. With this assumption, the FE solution is:

$$y_n = (1 + \Delta t \lambda)^n$$

The factor $(1 + \Delta t \lambda)$ is called the **amplification factor**: it indicates how much y_n will grow or decay.

For the FE solution to be stable, it's clear we need

$$|1 + \Delta t \lambda| < 1$$

Remember that λ models an eigenvalue of the Jacobian matrix $\partial f / \partial y$, and thus can be complex in general. We can understand the **stability region** of FE (those values of $\lambda \Delta t$ in the complex plane that give rise to stable discrete solutions) then as just a unit circle centered at the point -1 (on the negative real axis).

Also remember that from the exact solution of the test equation, we're really only interested in λ with zero or negative real part (i.e. so the exact solution doesn't unstably blow up). Clearly FE's stability region is a pretty small subset of the left-half of the complex plane we might be interested in: this means that in general our choice of Δt can't be too large.

The first case we'll look at in detail is when λ is real and negative. In this case, the stability restriction is simple:

$$\Delta t < \frac{2}{|\lambda|}$$

For problems of this type, FE is **conditionally stable**.

The second case is when λ is pure imaginary. The stability region of FE does not include any of the imaginary axis—the circle is only tangent to it at the origin 0. Thus for problems of this type, FE is **unconditionally unstable**: no matter how small Δt is, the solution will always exponentially grow. This would suggest FE is a poor choice of a method for problems like this.

It should be underscored that regardless of stability, FE can be proved to converge to the correct solution in the limit $\Delta t \to 0$ (under mild smoothness conditions on f). Even for pure imaginary λ, as Δt gets smaller the rate of exponential growth becomes gentler. For a fixed end-time, FE can still be made to work—though its first order accuracy and stability properties suggest there may be much more efficient methods.

2 Monotonicity and Forward Euler

For λ real and negative, the exact solution monotonically decays towards zero. Note that the FE solution $y_n = (1 + \lambda\Delta t)^n y_0$ might not have this behaviour. In particular, if $\Delta t > 1/|\lambda|$, the amplification factor is negative, and so the discrete solution changes sign each time step, oscillating back and forth. This is quite unlike the true behaviour of the solution, and in some contexts may not make any physical sense and cause significant problems (e.g. if the solution is supposed to model the concentration of a particular chemical, which cannot be negative). Thus we somtimes are interested in a stronger condition than stability: **monotonicity**. For FE (and real, negative λ) the monotonicity time step restriction is

$$\Delta t < \frac{1}{|\lambda|}$$

This is half the stable time step in this case.

3 Leap-frog

The next method we will look at will attempt to improve on the first order accuracy of Forward Euler. If we are restricted to just using y_n and $f(y_n, t_n)$, it's hard to do much better: we need more information. One possibility is to use the previous value, y_{n-1}. Take a look at a Taylor series expansion of both y_{n+1} and y_{n-1}, assuming again that the time step Δt is held constant:

$$\begin{aligned} y(t + \Delta t) &= y(t) + \Delta t \frac{dy}{dt} + \frac{\Delta t^2}{2}\frac{d^2y}{dt^2} + \frac{\Delta t^3}{6}\frac{d^3y}{dt^3} + O(\Delta t^4) \\ y(t - \Delta t) &= y(t) - \Delta t \frac{dy}{dt} + \frac{\Delta t^2}{2}\frac{d^2y}{dt^2} - \frac{\Delta t^3}{6}\frac{d^3y}{dt^3} + O(\Delta t^4) \end{aligned}$$

Subtracting the two expansions gives

$$y(t + \Delta t) - y(t - \Delta t) = 2\Delta t \frac{dy}{dt} + O(\Delta t^3)$$

Note that the quadratic terms cancel: this will give us an extra order of accuracy. Before going on to the time integration algorithm, it should be pointed out that this also gives us a **central difference** approximation to the first derivative of y:

$$\frac{dy}{dt}(t) = \frac{y(t + \Delta t) - y(t - \Delta t)}{2\Delta t} + O(\Delta t^2)$$

It's called central because the terms to the left and right of the point in question are perfectly symmetric. Forward Euler, on the other hand, is based on a **one-sided difference**, $(y(t + \Delta t) - y(t))/\Delta t$. Generally speaking, for the same number of terms, central differences are more accurate—however, as we shall see, that doesn't necessarily mean they are better.

Plugging in $dy/dt = f(y, t)$, we finally get the so-called **leap-frog** scheme:

$$y_{n+1} = y_{n-1} + 2\Delta t f(y_n, t_n)$$

From the Taylor series above, we find the local truncation error is $O(\Delta t^3)$, thus the method is (globally) second order accurate. This is one better than Forward Euler. For small enough time steps, the final error of FE should be halved when the time step is halved; for leap-frog the error is quartered (one half squared) when the time step is halved.

Before getting too enthusiastic, let's do the linear stability analysis for leap-frog. Plugging in the test equation, we get

$$y_{n+1} = y_{n-1} + 2\Delta t \lambda y_n$$

This is now a linear difference equation, with constant coefficients (since we're assuming Δt is held constant). In general the solution will be a linear combination of exponentials. We can find the base of these exponential by substituting in a solution of the form r^n:

$$r^{n+1} = r^{n-1} + 2\Delta t \lambda r^n$$

Dividing through by r^{n-1} and rearranging gives a quadratic for r:

$$r^2 - 2\Delta t\lambda r - 1 = 0$$

This has two solutions in general:

$$r = \Delta t\lambda \pm \sqrt{(\Delta t\lambda)^2 + 1}$$

The stability of the method will be determined by the root with largest magnitude. (Even though it might be possible to start an integration with only the smaller root taking part, floating-point rounding will almost certainly guarantee both roots will end up present in computed solutions.) In particular, leap-frog is only stable when both roots satisfy $|r| < 1$. Unfortunately, working out the accompanying stability region is fairly hard-going algebra. We will content ourselves with an exact analysis of the two cases λ purely real or purely imaginary, and rely on plotting the magnitude of $|r|_{max}$ for general complex λ to understand the method.

First let's look at Forward Euler's problem case: when λ is pure imaginary. First consider $\Delta t < 1/|\lambda|$. In this case, the discriminant of the quadratic is positive, and thus the roots are complex with real part $\pm\sqrt{(\Delta t\lambda)^2 + 1)}$ and imaginary part $\Delta t\lambda$. Thus the magnitude of both roots is:

$$\begin{aligned} |r| &= \sqrt{(\Delta t|\lambda|)^2 - ((\Delta t|\lambda|)^2 + 1)} \\ &= 1 \end{aligned}$$

Marvelously this matches the behaviour of the magnitude of the exact solution *perfectly*! Of course this doesn't imply the discrete solution is equal to the exact solution: in fact, while both can easily be seen to be composed of sine and cosine oscillations, the frequency of the discrete solution won't quite match the frequency $|\lambda|$ of the exact solution. The difference between the frequencies is sometimes referred to as a **phase error**.

Still in the pure imaginary λ case, if $\Delta t > 1/|\lambda|$, then the roots r are pure imaginary. It's not difficult to see that the largest magnitude root has

$$\begin{aligned} |r|_{max} &= \Delta t|\lambda| + \sqrt{(\Delta t|\lambda|)^2 - 1} \\ &> \Delta t|\lambda| \\ &> 1 \end{aligned}$$

Thus leap-frog is **conditionally stable** for pure imaginary λ, with time step restriction

$$\Delta t < \frac{1}{|\lambda|}$$

This isn't bad at all: since the exact solution is oscillating at frequency λ, it's hard to see how a general-purpose numerical method could hope to get good results with time steps much larger than this. (We will return to the question of how small a time step is reasonable for a given problem later on...)

Now let's take a look at the case where λ is real and negative. Then the discriminant is always positive, giving two real roots. The smaller one is guaranteed to be positive:

$$r_+ = \sqrt{1 + (\Delta t|\lambda|)^2} - \Delta t|\lambda|$$

It's not hard to see that r_+ is also always bounded by 1: it behaves very well (giving a nice monotone solution that decays to zero with second order accuracy). It's the other root,

$$r_- = -\sqrt{1 + (\Delta t|\lambda|)^2} - \Delta t|\lambda|,$$

that's a problem: it's negative, and always below -1 no matter how small Δt is taken, giving an oscillatory exponential blow-up. Thus leap-frog is **unconditionally unstable** for the case when λ is real. It should be noted again that though in exact arithmetic it might be possible to avoid this problematic root from appearing in the discrete solution, floating point errors will generally always force it to appear, and thence to grow quickly to dominate the error.

In fact, leap-frog can be shown to be **unconditionally unstable** for any complex λ with a nonzero real part. Its stability region is just an interval of the imaginary axis.

Also, as a point of future reference, this type of instability (where part of the discrete solution is nice and monotone, the other part is unstable and oscillatory) is a common problem plaguing central difference schemes. Even if stable, care must usually be taken to make sure spurious oscillations do not occur. This comes out of the fact that the central difference approximation to the first derivative involves $y(t + \Delta t)$ and $y(t - \Delta t)$ but not the middle value $y(t)$: this often means the odd-indexed values are only loosely coupled to the even-indexed values, allowing for zigzag oscillations between the two.

4 Adams-Bashforth

Our experience with leap-frog suggests using y_{n-1} was the wrong piece of information to use, in general. More success can be had by using $f(y_{n-1}, t_{n-1})$ instead. That is, we're interested in a method of the following form:

$$y_{n+1} = y_n + \Delta t \left(b_0 f(y_n, t_n) + b_1 f(y_{n-1}, t_{n-1})\right)$$

Here b_0 and b_1 are some constant coefficients we'll need to figure out to get higher accuracy (or possibly better stability). Note that Forward Euler is the case where $b_0 = 1$ and $b_1 = 0$—we're obviously interested in $b_1 \neq 0$.

While the coefficients may be derived from a Taylor series analysis, it gets somewhat complicated due to f being evaluated at different points. We'll instead take a short-cut approach to deriving the coefficients that give a certain accuracy. Taylor's theorem essentially tells us that locally (in an $O(\Delta t)$ neighbourhood), a smooth function is just a polynomial of degree p up to an error of $O(\Delta t^{p+1})$. Therefore, if a numerical scheme is *exact* for polynomials of degree p, it should have global accuracy of order p.

In particular, if we force our method to be exact for functions $y(t) = 1$, $y(t) = t$, …, $y(t) = t^p$, which correspond to $f(y, t) = 0$, $f(y, t) = 1$, …, $f(y, t) = pt^{p-1}$, then as long the method is linear (as the one we're looking at now is) it will be exact for all the polynomials. In fact, any basis for the polynomials will do the trick.

Note this accuracy short-cut doesn't necessarily go the other way: a $p'th$ order numerical scheme might not, in general, be exact for any polynomial. It sometimes might even make sense to avoid being exact—for example, take a look at the following modification to Forward Euler:

$$y_{n+1} = y_n + \Delta t f(y_n, t_n) - \epsilon \Delta t^2 y_n$$

Here ϵ is a positive real constant that the user might tune. Note that while Forward Euler does solve the constant and linear polynomial cases exactly, this scheme doesn't—though as is clear, it's still first order since the local truncation error is still $O(\Delta t^2)$. At first glance you might say this is stupid: it looks like we're arbitrarily adding to the error of the scheme, but if you think hard about where Forward Euler fails, you might see the advantage of this extra error...

Returning to the problem of determining b_0 and b_1: we would like the new method to be second order accurate. Thus we will force it to solve for quadratic polynomials exactly. To make life particularly simple, we'll just check the case where $n = 0$. Plugging in the constant $y(t) = 1$ with $f(y, t) = 0$ gives:

$$1 = 1$$

which obviously is always true. Plugging in the linear $y(t) = t$ with $f(y, t) = 1$ gives:

$$\Delta t = 0 + \Delta t(b_0 + b_1)$$

or, dividing through by Δt:

$$1 = b_0 + b_1$$

Finally plugging in the quadratic $y(t) = t^2$ with $f(y, t) = 2t$ gives:

$$\Delta t^2 = 0 + \Delta t(0 - 2\Delta t b_1)$$

which gives $b_1 = -1/2$. The linear equation then implies $b_0 = 3/2$, giving us the following method:

$$y_{n+1} = y_n + \Delta t \left(\frac{3}{2} f(y_n, t_n) - \frac{1}{2} f(y_{n-1}, t_{n-1}) \right)$$

This algorithm is the second order **Adams-Bashforth** method, sometimes abbreviated as AB2. In general, Adams-Bashforth is a family of methods, where the p'th order method is of the form:

$$y_{n+1} = y_n + \Delta t \left(b_0 f(y_n, t_n) + b_1 f(y_{n-1}, t_{n-1}) + \cdots b_{p-1} f(y_{n-p+1}, t_{n-p+1}) \right)$$

with coefficients $b_0, \ldots, b_{p-1}$ selected to make the error p'th order accurate. Forward Euler is in fact the same thing as AB1, the first-order Adams-Bashforth scheme.