Finite difference methods, error analysis, stability analysis, stiff problems. (Initial Value Problems)

Finite Difference Methods

We don't plan to study highly complicated nonlinear differential equations. Our first goal is to see why a difference method is successful (or not). The crucial questions of **stability and accuracy** can be clearly understood for linear equations. Then we can construct difference approximations to a great variety of practical problems.

Another property we need is **computational speed**. That depends partly on the complexity of the equation $u' = f(u,t)$. Often we measure speed by the number of times that $f(u,t)$ has to be computed in each time step (that number could be one). When we turn to *implicit* methods and *predictor-corrector* methods, to improve stability, the cost per step goes up but we gain speed with a larger step Δt.

This begins with basic methods (forward Euler, backward Euler) and then improves. Each time, we test stability on $u' = a\,u$. When a is negative, Δt is often limited by $-a\,\Delta t \le C$. This has an immediate effect: the equation with $a = -99$ requires a much smaller Δt than $a = -1$. Let me organize the equations as scalar and vector, nonlinear in general or linear with constant coefficients a and A:

1 equation	$u' = f(u,t)$	$u' = au$	$\boldsymbol{a \approx \partial f/\partial u}$	$\boldsymbol{a \le 0}$
$\boldsymbol{N}$ **equations**	$u_i' = f_i(u,t)$	$u' = Au$	$\boldsymbol{A_{ij} \approx \partial f_i/\partial u_j}$	$\boldsymbol{\mathrm{Re}\,\lambda(A) \le 0}$

For an ordinary differential equation $u' = f(u,t)$, good codes will increase the accuracy (and keep stability) far beyond the $O(\Delta t)$ error in Euler's methods. You can rely on freely available software like `ode45` to make the two crucial decisions:

1. to choose an accurate difference method (and change the formula adaptively)

2. to choose a stable time step (and change Δt adaptively).

We will introduce accurate methods, and find the stability limits on Δt.

Stiff Differential Equations

First we call attention to one extra question: ***Is the equation stiff?*** Let me begin with a made-up example to introduce stiffness and its effects:

$$v(t) = \underset{\text{controls decay}}{e^{-t}} + \underset{\text{controls } \Delta t}{e^{-99t}}$$

The step Δt is 99 times smaller because of e^{-99t} that disappears anyway

Those decay rates -1 and -99 could be the eigenvalues of A, as in Example 1.

Example 1

$$\frac{d}{dt}\begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} -50 & 49 \\ 49 & -50 \end{bmatrix}\begin{bmatrix} v \\ w \end{bmatrix} \quad \text{with} \quad \begin{bmatrix} v(0) \\ w(0) \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}. \tag{1}$$

The solution has $v(t) = e^{-t} + e^{-99t}$ and $w(t) = e^{-t} - e^{-99t}$. The time scales are different by a factor of 99 (the condition number of A). **The solution will decay at the slow time scale of e^{-t}, but computing e^{-99t} may require a very small Δt for stability.** It is frustrating to have Δt controlled by the component that is decaying so fast.

Any explicit method will have a requirement $99\Delta t \le C$. We will see how this happens and how an implicit method (like `ode15s` and `od23t` in MATLAB) can avoid it.

Trefethen [ ] points to these applications where stiffness comes with the problem:

1. **Chemical kinetics** (reactions go at very different speeds)
2. **Control theory** (probably the largest application area for MATLAB)
3. **Circuit simulation** (components respond at widely different time scales)
4. **Method of Lines** (large systems come from partial differential equations).

Example 2 The N by N second difference matrix K produces a large stiff system:

$$\textbf{Method of Lines} \qquad \frac{du}{dt} = \frac{-Ku}{(\Delta x)^2} \quad \text{has} \quad \frac{du_i}{dt} = \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2}. \tag{2}$$

This comes from the heat equation $\partial u/\partial t = \partial^2 u/\partial x^2$, by discretizing only the space derivative. Example 1 had eigenvalues -1 and -99, while Example 2 has N eigenvalues—but the difficulty is essentially the same! The most negative eigenvalue here is about $a = -4/(\Delta x)^2$. So a small Δx (for accuracy) will require a *very small* Δt (for stability).

This "semidiscrete" method of lines is an important idea. Discretizing the space variables first produces a large system that can be given to an ODE solver. (We have ordinary differential equations in time.) If it wants to, that solver can vary the time step Δt and even the discretization formula as $u(t)$ speeds up or slows down.

This method splits the approximation of a PDE into two parts. Finite differences/finite elements in earlier chapters produce the first part (discrete in space). The upcoming stability-accuracy analysis applies to the second part (discrete in time). This idea is very simple and useful, even if it misses the good methods later in this chapter that take full advantage of space-time. For the heat equation $u_t = u_{xx}$, the useful fact $u_{tt} = u_{xxxx}$ allows us to cancel space errors with time errors—which we won't notice when they are separated in the semidiscrete method of lines.

Forward Euler and Backward Euler

The equation $u' = f(u,t)$ starts from an initial value $u(0)$. The key point is that the rate of change u' is determined by the current state u at any moment t. This model of reality, where all the history is contained in the current state $u(t)$, is a tremendous success throughout science and engineering. (It makes calculus almost as important as linear algebra.) But for a computer, continuous time has to change to discrete time. One differential equation allows many difference equations!

The simplest method (Euler is pronounced "Oiler") uses a forward difference:

$$\textbf{Forward Euler} \quad \frac{\boldsymbol{U_{n+1} - U_n}}{\boldsymbol{\Delta t}} = \boldsymbol{f(U_n, t_n)} \quad \text{is} \quad U_{n+1} = U_n + \Delta t\, f_n\,. \tag{3}$$

Over each Δt interval, the slope of U *doesn't change*. Figure 5.1 shows how the correct solution to $u' = au$ follows a smooth curve, while $U(t)$ is only piecewise linear. A better method (higher accuracy) will stay much closer to the curve by using more information than the one slope $f_n = f(U_n, t_n)$ at the start of the step.

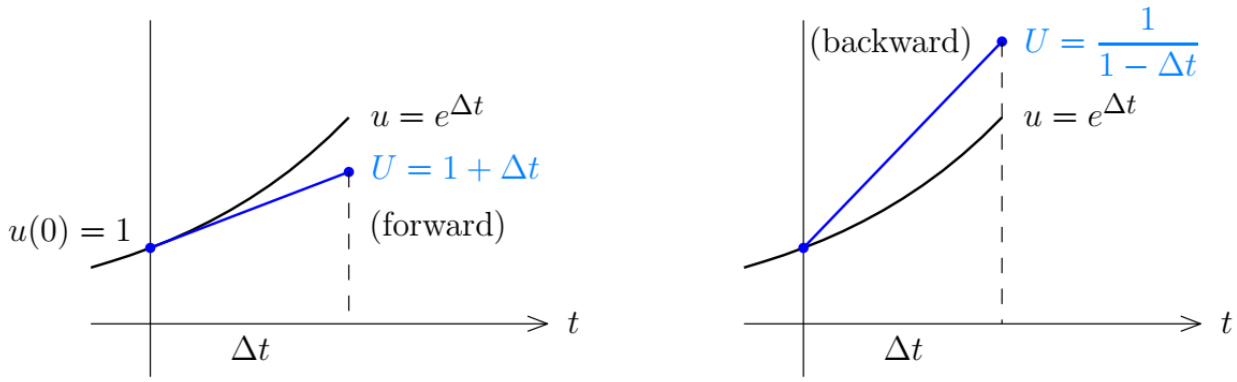


Figure 5.1: Forward Euler and Backward Euler for $u' = u$. One-step errors $\approx \frac{1}{2}(\Delta t)^2$.

Backward Euler comes from using f_{n+1} *at the end of the step*, when $t = t_{n+1}$:

$$\textbf{Backward Euler} \quad \frac{\boldsymbol{U_{n+1} - U_n}}{\boldsymbol{\Delta t}} = \boldsymbol{f(U_{n+1}, t_{n+1})} \quad \text{is} \quad U_{n+1} - \Delta t\, f_{n+1} = U_n\,. \tag{4}$$

This is an ***implicit method***. To find U_{n+1}, we have to solve equation (4). When f is linear in u, we are solving a linear system at each step (and the cost is low if the matrix is tridiagonal, like $I - \Delta t\, K$ in Example 2). We will comment later on iterations like Newton's method or predictor-corrector in the nonlinear case.

The first example to study is the linear scalar equation $u' = au$. Compare forward and backward Euler, for one step and for n steps:

$$\textbf{Forward} \qquad U_{n+1} = (1 + a\,\Delta t)U_n \quad \text{leads to} \quad \boldsymbol{U_n = (1 + a\,\Delta t)^n U_0}\,. \tag{5}$$

Backward $(1 - a\,\Delta t)U_{n+1} = U_n$ leads to $\boldsymbol{U_n = (1 - a\,\Delta t)^{-n}U_0}$. (6)

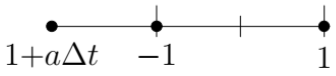
Forward Euler is like compound interest, at the rate a. Each step starts with U_n and adds the interest $a\,\Delta t\,U_n$. As the stepsize Δt goes to zero and we need $T/\Delta t$ steps to reach time T, this discrete compounding approaches continuous compounding. The discrete solution U_n approaches the continuous solution of $u' = au$:

$$(1 + a\,\Delta t)^{T/\Delta t} \quad \text{approaches} \quad e^{aT} \quad \text{as} \quad \Delta t \to 0\,.$$

This is the convergence of U to u that we will prove below, more generally. It holds for backward Euler too, because $(1 - a\,\Delta t)^{-1} = 1 + a\Delta t + \text{higher order terms}$.

The stability question arises for very negative a. The true solution $e^{-at}u(0)$ is extremely stable, approaching zero. (If this is your own money, you should change banks and get $a > 0$.) Backward Euler will be stable because it divides by $1 - a\,\Delta t$ (which is greater than 1 when a is negative). *Forward Euler will explode if* $1 + a\,\Delta t$ *is smaller than* -1*, because its powers grow exponentially*:

Instability $1 + a\,\Delta t < -1$ when $a\,\Delta t < -2$ (7)



That is a pretty sharp borderline between stability and instability, at $-a\,\Delta t = 2$. For $u' = -20u$ which has $a = -20$, the borderline is $\Delta t = \frac{2}{20} = \frac{1}{10}$. Compare the results at time $T = 2$ from $\Delta t = \frac{1}{11}$ (22 steps) and $\Delta t = \frac{1}{9}$ (18 steps):

$$\text{Stable } \Delta t = \frac{1}{11} \qquad (1 + a\,\Delta t)^{22} = \left(-\frac{9}{11}\right)^{22} \approx .012$$

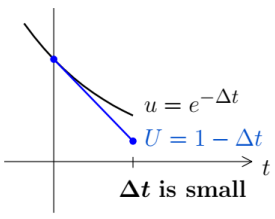
$$\text{Unstable } \Delta t = \frac{1}{9} \qquad (1 + a\,\Delta t)^{18} = \left(-\frac{11}{9}\right)^{18} \approx 37.043$$

I would describe backward Euler as absolutely stable (**A-stable**) because it is stable whenever Re $a < 0$. Only an implicit method can be A-stable. Forward Euler is a **stable method**(!) because it succeeds as $\Delta t \to 0$. For small enough Δt, it is on the stable side of the borderline.

In this example a *good quality approximation* requires more than stability (even $\Delta t = \frac{1}{11}$ is too big). Those powers of $-\frac{9}{11}$ alternate in sign, while e^{-20t} stays positive.

Accuracy and Convergence

Since forward and backward differences are *first order accurate*, it is no surprise that the errors from forward and backward Euler are $O(\Delta t)$. This error $e = u - U$ is



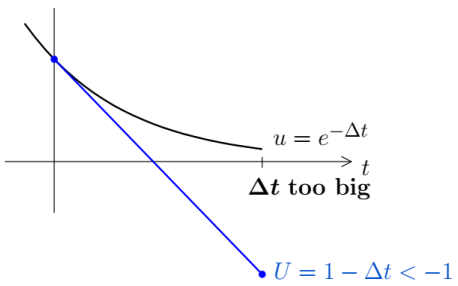


Figure 5.2: Forward Euler (stable and unstable, with $-a\Delta t$ going above 2).

measured at a fixed time T. As Δt decreases, so does the new error added at each step. But the number of steps to reach that time increases, keeping $n\,\Delta t = T$.

To see why the error $u(T) - U(T)$ is $O(\Delta t)$, the key is stability. We need to know that *earlier errors don't increase as they are carried forward to time* T. Forward Euler is the simplest difference equation, so it is the perfect example to follow through first. (The next sections will apply the same idea to partial differential equations.) Euler's difference equation for $u' = f(u,t) = au$ is

$$U_{n+1} = U_n + \Delta t\, f(U_n, t_n) = U_n + a\,\Delta t\, U_n\,. \tag{8}$$

The true solution at time $n\,\Delta t$ satisfies (8) except for a **discretization error DE**:

$$u_{n+1} = u_n + \Delta t\, u_n' + \mathrm{DE} = u_n + a\,\Delta t\, u_n + \mathrm{DE}_{n+1}\,. \tag{9}$$

That error DE is of order $(\Delta t)^2$ because the second-order term is missing (it should be $\frac{1}{2}(\Delta t)^2 u_n''$, but Euler didn't keep it). Subtracting (8) from (9) gives a difference equation for the error $e = u - U$, propagating forward in time:

Error equation
$$e_{n+1} = e_n + a\,\Delta t\, e_n + \mathrm{DE}_{n+1}\,. \tag{10}$$

You could think of this one-step error DE_{n+1} as a deposit like $(\Delta t)^2$ into your account. Once the deposit is made, it will grow or decay according to the error equation. To reach time $T = N\,\Delta t$, each error DE_k at step k has $N - k$ more steps to go:

$$e_N = (1 + a\,\Delta t)^{N-1}\,\mathrm{DE}_1 + \cdots + (1 + a\,\Delta t)^{N-k}\,\mathrm{DE}_k + \cdots + \mathrm{DE}_N\,. \tag{11}$$

Now stability plays its part. If a is negative, those powers are all less than 1 (in absolute value)—*provided* $1 + a\,\Delta t$ *does not go below* -1. If a is positive, those powers of $1 + a\,\Delta t$ are all less than $(e^{a\,\Delta t})^N = e^{aT}$. Then the error e_N has N terms in (11), and every term is less than $c(\Delta t)^2$ for a fixed constant c:

Error bound
$$|\boldsymbol{e_N}| = |\boldsymbol{u_N - U_N}| \le \boldsymbol{N\,c\,(\Delta t)^2 = c\,T\,\Delta t}\,. \tag{12}$$

The errors along the way, of size $(\Delta t)^2$, combined after $N = T/\Delta t$ steps into an overall Δt error. This depended on stability—the local errors didn't explode.

Notice that the error growth follows the difference equation in (10), not the differential equation. The stiff example with $a\,\Delta t = (-20)(\frac{1}{9})$ gave a large $1 + a\,\Delta t$, even when $e^{a\,\Delta t}$ was small. We still call forward Euler a **stable method**, because as soon as Δt is small enough the danger has passed. Backward Euler also gives $|e_N| = O(\Delta t)$. The problem with these first-order methods is their low accuracy.

The local discretization error DE tells us the accuracy. For Euler that error $\mathrm{DE} \approx \frac{1}{2}(\Delta t)^2 u''$ is the first term that Euler misses in the Taylor series for $u(t + \Delta t)$. Better methods will capture that term exactly, and miss on a higher-order term. The discretization error (we find it by comparing $u(t+\Delta t)$ with U_{n+1}, assuming $u(t)$ agrees with U_n) begins with a power $(\Delta t)^{p+1}$:

Local discretization error $\quad \mathrm{DE} \approx c(\Delta t)^{p+1}\dfrac{d^{p+1}u}{dt^{p+1}}\,.$ (13)

The method decides c and $p+1$. With stability, $T/\Delta t$ steps give a global error of order $(\Delta t)^p$. The derivative of u shows whether the problem is hard or easy.

Error estimates like (13) appear everywhere in numerical analysis. The $1,-2,1$ second difference has error $\frac{1}{12}(\Delta x)^4 u''''$. Partial differential equations (Laplace, wave, heat) produce similar terms. In one line we find $c=-\frac{1}{2}$ for *backward Euler*:

$$(u_{n+1}-u_n)-\Delta t\,u'_{n+1}\approx\left(\Delta t\,u'_n+\frac{(\Delta t)^2}{2}\,u''_n\right)-\Delta t(u'_n+\Delta t\,u''_n)\approx-\frac{(\Delta t)^2}{2}\,u''_n. \quad (14)$$

The global estimate $|u-U|\le C\,\Delta t\,\max|u''|$ shows no error when u is linear and u'' is zero (Euler's approximation of constant slope becomes true). For nonlinear equations, the key is in the subtraction of (8) from (9). A "Lipschitz" bound L on $\partial f/\partial u$ replaces the number a in the error equation:

$$|f(u,t)-f(U,t)|\le L|u-U| \qquad \text{gives} \qquad e_{n+1}\le(1+L\Delta t)\,e_n+\mathrm{DE}_{n+1}\,. \quad (15)$$

Second-Order Methods

Here is a first way to increase the accuracy. We could center the equation at the midpoint $(n+\frac{1}{2})\Delta t$ of the step, by averaging $f(U_n,t_n)$ and $f(U_{n+1},t_{n+1})$. The result is an *implicit second-order method* use in MATLAB's `ode23t`:

Trapezoidal rule/Crank-Nicolson $\quad \dfrac{\boldsymbol{U_{n+1}-U_n}}{\boldsymbol{\Delta t}}=\dfrac{\boldsymbol{1}}{\boldsymbol{2}}(\boldsymbol{f_{n+1}+f_n})\,.$ (16)

So $U_{n+1}-\frac{1}{2}\Delta t\,f_{n+1}=U_n+\frac{1}{2}\Delta t\,f_n$. For our model $u'=f(u)=au$, this is

$$(1-\frac{1}{2}a\,\Delta t)\,U_{n+1}=(1+\frac{1}{2}a\,\Delta t)\,U_n \quad \text{which gives} \quad U_{n+1}=\left(\frac{1+\frac{1}{2}a\,\Delta t}{1-\frac{1}{2}a\,\Delta t}\right)U_n\,. \quad (17)$$

The true solution has $u_{n+1}=e^{a\Delta t}u_n$. Problem 1 will find $\mathrm{DE}\approx(\Delta t)^3$. The equation is stable. So $N=T/\Delta t$ steps produce $|e_N|=|u_N-U_N|\le cT(\Delta t)^2$.

How to improve forward Euler and still keep it explicit? We don't want U_{n+1} on the right side of the equation, but we are happy with U_{n-1} (from the previous step!). Here is a combination that gives second-order accuracy in a **two-step method**:

"Adams-Bashforth" $\quad \dfrac{\boldsymbol{U_{n+1}-U_n}}{\boldsymbol{\Delta t}}=\dfrac{\boldsymbol{3}}{\boldsymbol{2}}\boldsymbol{f(U_n,t_n)}-\dfrac{\boldsymbol{1}}{\boldsymbol{2}}\boldsymbol{f(U_{n-1},t_{n-1})}\,.$ (18)

Remember that $f(U_{n-1}, t_{n-1})$ is already computed in the previous step, going from $n-1$ to n. So this **explicit multistep method** requires no extra work, and improves the accuracy. To see how $(\Delta t)^2$ comes out correctly, write u' for f:

$$\frac{3}{2}u'_n - \frac{1}{2}u'_{n-1} \approx \frac{3}{2}u'_n - \frac{1}{2}(u'_n - \Delta t\, u''_n) = u'_n + \frac{1}{2}\Delta t\, u''_n\,.$$

Multiplied by Δt in (18), that new term $\frac{1}{2}(\Delta t)^2 u''_n$ is exactly what Euler missed. *Each extra term in the difference equation can increase the accuracy by* 1.

A third possibility uses the already computed value U_{n-1} (instead of the slope f_{n-1}). With $\frac{3}{2}, -\frac{4}{2}, \frac{1}{2}$ chosen for second-order accuracy, we get an implicit **backward difference method**:

$$\textbf{Backward differences} \qquad \frac{\boldsymbol{3U_{n+1} - 4U_n + U_{n-1}}}{\boldsymbol{2\Delta t}} = \boldsymbol{f(U_{n+1}, t_{n+1})}\,. \tag{19}$$

What about stability? The implicit method (17) is stable even in the stiff case, when a is very negative. $1 - \frac{1}{2}a\,\Delta t$ (left side) will be larger than $1 + \frac{1}{2}a\,\Delta t$ (right side). (19) is also stable. The explicit method (18) will be stable if Δt is small enough, but there is always a limit on Δt for stiff systems.

Here is a quick way to find the stability limit in (18) when a is real. The limit occurs when the growth factor is exactly $G = -1$. ***Set $U_{n+1} = -1$ and $U_n = 1$ and $U_{n-1} = -1$ in (18). Solve for a when $f(u,t) = au$:***

$$\textbf{Stability limit in (18)} \quad \frac{-2}{\Delta t} = \frac{3}{2}a + \frac{1}{2}a \quad \text{gives} \quad a\,\Delta t = -1\,. \quad \text{So} \quad C = 1\,. \tag{20}$$

Those second-order methods (17)–(18)–(19) are definitely useful! The reader might suggest including both U_{n-1} and f_{n-1} to increase the accuracy to $p = 3$. Sadly, this method is violently unstable (Problem 4). We may use older values of U in backward differences or $f(U)$ in Adams formulas, but including both U and $f(U)$ for even higher accuracy produces instability for all Δt.

Multistep Methods: Explicit and Implicit

By using p older values of either U or $f(U)$ (already computed at previous steps!), the accuracy can be increased to order p. Each ∇U is $U(t) - U(t - \Delta t)$ and $p = 2$ is (19):

$$\textbf{Backward differences} \quad \left(\nabla + \frac{1}{2}\nabla^2 + \cdots + \frac{1}{p}\nabla^p\right) U_{n+1} = \Delta t\, f(U_{n+1}, t_{n+1})\,. \tag{21}$$

Using older values of the right side $f(U)$ gives an **Adams-Bashforth method**:

$$U_{n+1} - U_n = \Delta t(b_1\, f_n + \cdots + b_p\, f_{n-p+1}) \quad \text{with} \quad f_n = f(U_n, t_n)\,. \tag{22}$$

The table shows the numbers b up to $p = 4$, starting with Euler for $p = 1$.

order of accuracy	b_1	b_2	b_3	b_4	limit on $a\Delta t$ for stability	constant c in error DE
$p = 1$	1				-2	$1/2$
$p = 2$	$3/2$	$-1/2$			-1	$5/12$
$p = 3$	$23/12$	$-16/12$	$5/12$		$-6/11$	$3/8$
$p = 4$	$55/24$	$-59/24$	$37/24$	$-9/24$	$-3/10$	$251/720$

The fourth-order method is often a good choice, although astronomers go above $p = 8$. At the stability limit $G = -1$ as in (20). The local error DE $\approx c(\Delta t)^{p+1}u^{(p+1)}$ is a problem of step control. Whether it is amplified by later steps is a problem of stability control.

Implicit methods have an extra term $c_0 f_{n+1}$ at the new time level. Properly chosen, that adds one extra order of accuracy—as it did for the trapezoidal rule, which is the second method in the new table. Backward Euler is the first:

order of accuracy	c_0	c_1	c_2	c_3	limit on $a\Delta t$ for stability	constant c in error DE
$p = 1$	1				$-\infty$	$-1/2$
$p = 2$	$1/2$	$1/2$			$-\infty$	$-1/12$
$p = 3$	$5/12$	$8/12$	$-1/12$		-6	$-1/24$
$p = 4$	$9/24$	$19/24$	$-5/24$	$1/24$	-3	$-19/720$

Every row of both tables adds to 1, so $u' = 1$ is solved exactly.

You see that the error constants and stability are all in favor of implicit methods. So is the user, except when solving for U_{n+1} becomes expensive. Then there is a simple and successful ***predictor-corrector*** method, or else ***Newton's method***.

Predictor-Corrector

P: Use the explicit formula to *predict* a new U^*_{n+1}
E: Use u^*_{n+1} to *evaluate* the right side f^*_{n+1}
C: Use f^*_{n+1} in the implicit formula to *correct* to a new U_{n+1}.

The stability is much improved if another E step evaluates f_{n+1} with the corrected u_{n+1}. In principle we should continue the corrector to convergence. Frequently 1–2 corrections are enough, and the extra calculation can be put to use. By comparing the predicted U^*_{n+1} and corrected U_{n+1} the code can estimate the local error and change Δt:

$$\text{local error DE} \approx \frac{c}{c^* - c}(U_{n+1} - U^*_{n+1}), \tag{23}$$

where c^* and c are the error constants in the tables for the predictor and corrector.

Implicit methods often have a **Newton loop** inside each time step, to compute U_{n+1}. The kth iteration in the Newton loop is a linear system, with the Jacobian matrix $A_{ij}^{(k)} = \partial f_i/\partial u_j$ evaluated at the latest approximation $U_{n+1}^{(k)}$ with $t = t_{n+1}$.

Here is the kth i to solve $U_{n+1} - c_0 f(U_{n+1}, t_{n+1}) =$ old values:

$$\textbf{Newton iteration} \quad (I - c_0 A^{(k)})(U_{n+1}^{(k+1)} - U_{n+1}^{(k)}) = c_0 f(U_{n+1}^{(k)}, t_{n+1}) + \text{old values} . \tag{24}$$

When $f(u) = Au$ is linear, you only need one iteration ($k = 0$ starts with $U_{n+1}^{(0)} = U_n$). For nonlinear $f(u)$, Newton's rapid convergence squares the error at each step (when it gets close). The price is a new evaluation of $f(u)$ and its matrix $A^{(k)}$ of derivatives.

Runge-Kutta Methods

A-stable methods have no stability limit. Multistep methods achieve high accuracy with one or two evaluations of f. If more evaluations are not too expensive, ***Runge-Kutta methods*** are definitely competitive (and these are self-starting). They are compound one-step methods, using Euler's $U_n + \Delta t\, f_n$ ***inside f:***

$$\textbf{Simplified Runge-Kutta} \qquad \frac{U_{n+1} - U_n}{\Delta t} = \frac{1}{2}\big[f_n + f(U_n + \Delta t\, f_n, t_{n+1})\big]. \tag{25}$$

You see the compounding of f. For the model equation $u' = au$ the result is

$$u_{n+1} = u_n + \frac{1}{2}\Delta t\big[au_n + a(u_n + \Delta t\, au_n)\big] = \big(1 + a\Delta t + \frac{1}{2}a^2\Delta t^2\big)u_n = G\, u_n\,. \tag{26}$$

This confirms the second-order accuracy; the growth factor G agrees with $e^{a\Delta t}$ through $(\Delta t)^2$. There will be a stability threshold $a\Delta t = C$, where $|G| = 1$:

$$\textbf{Stability limit} \qquad |G| = \Big|1 + C + \frac{1}{2}C^2\Big| = 1 \quad (\text{for complex } C = a + ib)$$

The complete stability limit is not a point but a closed curve in the complex plane. Figure 5.____ shows all the complex numbers C at which $|G| = 1$.

The famous version of Runge-Kutta is compounded *four times* and achieves $p = 4$:

$$\textbf{Runge-Kutta} \qquad \frac{U_{n+1} - U_n}{\Delta t} = \frac{1}{3}(k_1 + 2k_2 + 2k_3 + k_4) \tag{27}$$

$$k_1 = \frac{1}{2} f(U_n, t_n) \qquad k_3 = \frac{1}{2} f(U_n + \Delta t\, \boldsymbol{k_2}, t_{n+1/2})$$
$$k_2 = \frac{1}{2} f(U_n + \Delta t\, \boldsymbol{k_1}, t_{n+1/2}) \qquad k_4 = \frac{1}{2} f(U_n + 2\Delta t\, \boldsymbol{k_3}, t_{n+1})$$

For this one-step method, no special starting instructions are necessary. It is simple to change Δt as you go. The growth factor reproduces $e^{a\Delta t}$ through $\frac{1}{24} a^4 \Delta t^4$. The error constant is the next coefficient $\frac{1}{120}$. Among highly accurate methods, Runge-Kutta is especially easy to code and run—probably the easiest there is. MATLAB's workhorse is `ode45`.

To prove that the stability threshold $a\Delta t = -2.78$ is genuine, we reproduce the solution of $u' = -100u + 100 \sin t$. With $u(0) = 0$, Runge-Kutta gives

$$\begin{aligned} U_{120} &= 0.151 = u(3) \quad \text{with } \Delta t = \frac{3}{120} \quad \text{and } a\Delta t = -2.5 \\ U_{100} &= 670{,}000{,}000{,}000 \quad \text{with } \Delta t = \frac{3}{100} \quad \text{and } a\Delta t = -3 \end{aligned} \tag{28}$$

Differential-Algebraic Equations

Our system of equations might not come in the form $u' = f(u, t)$. The more general form $F(u', u, t) = 0$ appears in many applications. It may not be easy (or possible) to solve this system explicitly for u'. For large electrical networks (VLSI on chips), transistors produce a highly nonlinear dependence on u' and u.

Here is a simple model in which solving for the vector u' is impossible:

$$Mu' = f(u, t) \quad \text{with a singular matrix } M \text{ (rank } r < N). \tag{29}$$

Suppose we change variables from u to v, so as to diagonalize M. In the new variables we have r true differential equations and $N - r$ *algebraic equations* (*no derivatives*). The system becomes a ***differential-algebraic equation*** (**DAE**):

$$\textbf{DAE} \qquad \begin{aligned} \frac{dv_i}{dt} &= f_i(v_1, \ldots, v_N, t) \quad i = 1, \ldots, r \\ 0 &= f_i(v_1, \ldots, v_N, t) \quad i = r+1, \ldots, N. \end{aligned} \tag{30}$$

The $N - r$ algebraic equations restrict v to a surface in N-dimensional space. The r differential equations move the vector $v(t)$ along that surface.

Problem Set

1 The error in the trapezoidal (Crank-Nicolson) method ( ) comes from the difference

$$e^{a\Delta t} - \left[\frac{1+(a\Delta t/2)}{1-(a\Delta t/2)}\right] = e^{a\Delta t} - \left[\left(1+\frac{a\Delta t}{2}\right)\left(1+\frac{a\Delta t}{2}+\left(\frac{a\Delta t}{2}\right)^2+\cdots\right)\right]$$

This involves the two series that everyone should learn: the exponential series for $e^{a\Delta t}$ and the geometric series $1+x+x^2+\cdots$ for $\frac{1}{1-x}$.

Multiply inside the brackets to produce the correct $\frac{1}{2}(a\Delta t)^2$. Show that the $(a\Delta t)^3$ term is wrong by $c=\frac{1}{12}$. Then the error is DE $\approx \frac{1}{12}(\Delta t)^3\, u'''$.

2 Try Runge-Kutta on $u' = -100u + 100\sin t$ with $\Delta t = -.0275$ and $-.028$. Those are close to the stability limit $-.0278$.

3 For the backward difference error in (19), expand $\frac{1}{2}(3e^{a\Delta t}-4+e^{-a\Delta t})-a\Delta t\, e^{a\Delta t}$ into a series in $a\Delta t$. Show that the leading error is $-\frac{1}{3}(a\Delta t)^3$ so that $c=-\frac{1}{3}$.

References and further readings

1 A. R. Conn, N. I. M. Gould and Ph. L. Toint, Trust-Region Methods, SIAM 2000.

2. J. Dennis and R. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear equations, (republished by) SIAM, 1996.

3. R. Fletcher, Practical Methods of Optimization, 2nd edition Wiley, 1987 (republished in 2000).

4. P. Gill, W. Murray and M. H. Wright, Practical Optimization, Academic Press, 1981.

5. N. I. M. Gould, An Introduction to Algorithms for Continuous Optimization, 2006. Available for download at http://www.numerical.rl.ac.uk/nimg/course/lectures/paper/paper.pdf.

6. J. Nocedal and S. J. Wright, Numerical Optimization, Springer Verlag, 1999 (1st edition) or 2006 (2nd edition).

7. Mathematical Modeling and Computation of Real-Time Problems: An Interdisciplinary Approach (Mathematical Engineering, Manufacturing, and Management Sciences) 1st Edition by Rakhee Kulshrestha, Chandra Shekhar, Madhu Jain, Srinivas R. Chakravarthy