

Mat1062: Introductory Numerical Methods for PDE

Mary Pugh

January 15, 2009

1 Ownership

These notes are the joint property of Rob Almgren and Mary Pugh.

2 Time discretization

In the following, we consider the initial value boundary value problem:

$$\begin{cases} u_t = Du_{xx} & \text{for } x \in [X_L, X_R], t > 0 \\ u(X_L, t) = u_L(t) \\ u(X_R, t) = u_R(t) \\ u(x, 0) = u_0(x) & \text{for } x \in [X_L, X_R] \end{cases}$$

Having discretized in space, we now have an evolution problem for a vector $U(t)$ of length $N - 1$:

$$\frac{dU}{dt} = LU = MU + R(t), \quad U(0) \text{ given.} \quad (1)$$

This notation is slightly confusing, since L is *not a matrix*: we use it to denote this “affine linear” operation.

To solve this system of ODEs, we choose a time step k , define time levels $t_n = nk$ for $n = 0, 1, \dots$, and introduce $u^n \in \mathbb{R}^{N-1}$. The vector u^n will be determined by a discrete dynamical system and it is our hope that this can be done in such a way that $u_j^n \approx U_j(t_n) \approx u(x_j, t^n)$. Recall that the *superscript* n denotes the n th time level rather than the n th power.

If we want the solution value at a specific time $t = T$, it will be convenient if $T = t_M = Mk$ for some integer M ; if not, we will have to do an interpolation in time to get the final answer.

There is an immense variety of schemes for numerical “integration” (solution) of ordinary differential equations, often with very high order accuracy and various other desirable properties (Runge-Kutta methods, for example). When the ODE system arises from discretization of a PDE, the major errors usually do not come from the time integration. It is most important to have a scheme that is simple, efficient, and especially *stable*, as we discuss below.

Our three basic methods all use the *same* approximation for the time derivative in $u_t = Du_{xx}$:

$$u_t(\tilde{t}) \approx \frac{1}{k} (u^{n+1} - u^n) \quad \text{for } \tilde{t} \in [t_n, t_n + k],$$

as $k \rightarrow 0$. Our methods differ only in the precise time value \tilde{t} at which the operator LU is evaluated.

Instead of matching u_t to LU at level n ($\tilde{t} = t_n$), the beginning of the interval, it would be more accurate to match it to a value in the middle, somewhere between levels n and $n + 1$ ($\tilde{t} = t_{\text{mid}} = (t_n + t_{n+1})/2$, say). Unfortunately, there are no intermediate grid values — all we have to work with is u^n (approximating $U(t_n)$) and u^{n+1} (approximating $U(t_{n+1})$). And so one would have to approximate $LU(t_{\text{mid}})$ using these, by linear interpolation, for example.

We consider three time-stepping methods, summarized by

$$\frac{1}{k} (u^{n+1} - u^n) = (1 - \theta) Lu^n + \theta Lu^{n+1} \approx LU(\tilde{t})$$

where $\theta \in [0, 1]$ and $\tilde{t} = (1 - \theta)t_n + \theta t_{n+1}$. We call θ an *implicitness parameter*. In the three cases of most interest, we have special names for the methods:

$$\frac{1}{k} (u^{n+1} - u^n) = \begin{cases} Lu^n, & \text{Forward Euler } (\theta = 0) \\ \frac{1}{2} (Lu^{n+1} + Lu^n), & \text{Crank-Nicolson } (\theta = \frac{1}{2}) \\ Lu^{n+1}, & \text{Backward Euler } (\theta = 1). \end{cases}$$

As you may guess from the terminology, the forward Euler method is a *fully explicit* method, and the backward Euler method is sometimes called the

fully implicit method. (There is a large variety of fully explicit methods, most notably “linear multistep” methods of which second-order Adams-Bashforth is occasionally useful for PDEs.)

When we substitute the expression for L , we obtain the formula for u^{n+1} :

$$(I - \theta kM)u^{n+1} = (I + (1 - \theta)kM)u^n + k(\theta R^{n+1} + (1 - \theta)R^n). \quad (2)$$

Note that this formula tells us exactly at what times we should evaluate the time-dependent boundary conditions. The actual computations we do to find u^{n+1} are very different, depending on whether θ is zero or not.

- If $\theta = 0$ (forward Euler method), then this is the explicit formula

$$u^{n+1} = u^n + k(Mu^n + R^n).$$

This says, just compute the time derivative in the current configuration, and advance u as though you moved at that speed for the entire time k . Only a matrix multiplication is required; but since the matrix is tridiagonal it is very fast to evaluate: u_j^{n+1} is given by a difference formula in terms of u_{j-1}^n , u_j^n , and u_{j+1}^n .

- If $\theta > 0$, then we must solve a linear system to get u^{n+1} . In this problem, with only one space dimension, this system is tridiagonal. We can thus obtain the solution in time $\mathcal{O}(N)$ by a simplified version of Gaussian elimination; pivoting is not usually necessary for problems arising from diffusive PDEs. For problems in more space dimensions, this linear solve can become the most time-consuming part of the whole computation. Each component u_j^{n+1} depends on *all* components u_j^n at the previous level. (This statement requires some thought — it’s claiming that the inverse of the matrix $I - \theta kM$ is a full matrix.)

3 Ordinary Differential Equations and Stability

Ordinary differential equations (ODEs) are fundamental models for time evolution problems. An initial value problem has the form

$$\frac{dU}{dt} = f(U(t), t) \quad \text{for } t \in [0, T], \quad \text{with } U(0) = U_0.$$

Here $U(t)$ is an element of \mathbb{C}^N or \mathbb{R}^N , and f is a function from $\mathbb{C}^N \times [0, T]$ into \mathbb{C}^N if $U(t) \in \mathbb{C}^N$ (else it’s from $\mathbb{R}^N \times [0, T]$ into \mathbb{R}^N .)

We name the type of the ODE depending on properties of the function f . The equation is *linear* if $f(\mathbf{U}, t) = A(t)\mathbf{U} + B(t)$ for some matrix $A(t)$ and some vector $B(t)$. That is, if f is “affine linear” in \mathbf{u} . It is *linear and homogeneous* if $f(\mathbf{U}, t) = A(t)\mathbf{U}$, that is, $B = 0$ so f is linear in \mathbf{u} . If $f(\mathbf{U}, t) = f(\mathbf{U})$, independent of t , then the equation is *autonomous*. If $f(\mathbf{U}, t) = f(t)$, independent of \mathbf{U} , then the initial value problem can be solved by the Fundamental Theorem of Calculus: $\mathbf{U}(t) = \mathbf{U}_0 + \int_0^t f(s) ds$.

3.1 Discretization of ODE

Choose a time step $k > 0$. Define time “levels” $t_n = nk$, for $n = 0, 1, \dots, M$ where $Mk = T$. We want to construct a sequence of points u^0, u^1, \dots, u^M , so that $u^n \approx U(nk)$ when k is small. Here u^n is the solution of the fully discrete model, and $U(nk)$ is the exact solution of the ODE, evaluated at the discrete time points. We use a superscript n to label time levels, so that we may later use a subscript to label spatial grid points.

We need a scheme for constructing the discrete approximation; such a scheme must consist of two things. First, we need a rule for picking the initial value v_0 . Since $U(t)$ and u^n are members of the same space, we can usually just pick $u^0 = u_0$. Second, we need a rule of the form $u^{n+1} = F(u^n, \dots, u^0; k)$ for computing each step of the approximation in terms of earlier steps. In fact, F could well be different for each step n . But we shall be most interested in one-level methods, for which $u^{n+1} = F(u^n; k)$, and F is the same for each $n > 0$.

A very rich class of schemes are the “linear multistep formulas,” for which u^{n+1} is a linear function of $u^n, \dots, u^{n-n_0}, f^n, \dots, f^{n-n_0}$, and possibly f^{n+1} , where $f^n = f(u^n, t_n)$. Here, n_0 is a fixed integer and the scheme is called an n_0 -level scheme. A multistep scheme is *explicit* if the formula for u^{n+1} does not include f^{n+1} ; it is *implicit* if it does include f^{n+1} .

The *order of accuracy* of a discrete scheme is the largest p so that

$$\|U(t+k) - F(U(t), U(t-k), \dots; k)\| \text{ is } \mathcal{O}(k^{p+1}), \text{ as } k \rightarrow 0 \quad (3)$$

where $U(t)$ is a true solution of the continuous problem. This quantity is called the *local truncation error*. The scheme is *consistent* if $p > 0$. It does not matter what norm we use, since all norms on a finite-dimensional space are equivalent. As we will see, to evaluate the local error and check consistency, we need only apply the discrete formula once and then do Taylor expansions.

A scheme is *convergent* if

$$\|u^n - U(t)\| \rightarrow 0 \quad \text{as } k \rightarrow 0 \text{ with } nk = t,$$

where again $U(t)$ is a continuous solution. Of course, this is the important property a numerical scheme must have if it is to be of any use. But it is difficult to check directly, since it involves both the complete solution of the difference approximation, and the exact solution of the differential equation.

Fact: Fix $t > 0$ and use it to determine the time step k via $k = t/n$. If a scheme converges, then it must be consistent, and

$$\|u^n - U(t)\| \sim \mathcal{O}(k^p), \quad \text{as } n \rightarrow \infty.$$

The intuition is that we make an error of size $\mathcal{O}(k^{p+1})$ on each step, and we make $\mathcal{O}(k^{-1})$ steps to go a fixed time t , so the total error should be $\mathcal{O}(k^p)$. And so, if the scheme works at all then the local analysis tells us how accurate the overall result is. Another way of saying this is: if the local truncation error goes to zero with rate $p + 1$ then this information of the local truncation error is enough to give us information about how good the solution computed at time t is.

The missing link between consistency and convergence is “stability.” If a scheme is “unstable,” then although it may be consistent, its discrete solutions look nothing like the solutions of the continuous problem. For multistep schemes for nonlinear ODEs, stability is a rather tricky topic. It is rather simpler for linear PDEs, as we discuss below.

3.2 Consistent schemes

It is easy to construct consistent schemes by using the approximation

$$u_t \approx \frac{1}{k}(u^{n+1} - u^n),$$

which we may match to f at t_n , at t_{n+1} , or at some interpolated intermediate point.

Forward Euler We set

$$\frac{1}{k}(u^{n+1} - u^n) = f^n, \quad \text{so} \quad u^{n+1} = u^n + kf^n$$

This scheme is explicit: it gives us a easily-evaluated formula for u^{n+1} in terms of quantities which are known at time level n .

Backward Euler We take

$$\frac{1}{k}(u^{n+1} - u^n) = f^{n+1}, \quad \text{so} \quad u^{n+1} = u^n + kf^{n+1}$$

which means that at each step we have to solve the nonlinear equation

$$u^{n+1} - kf(u^{n+1}) = u^n$$

for the new value u^{n+1} . This scheme is implicit. For a linear problem, the equation to be solved is linear. How easy it is to solve depends on the problem; for evolution problems associated with a PDE it is easy to solve in one space dimension, but can be very hard to solve in multiple dimensions.

Trapezoid We want to match the discrete difference of v to the value of f in between levels n and $n + 1$. But since there is no $u^{n+\frac{1}{2}}$ and hence no $f^{n+\frac{1}{2}}$, we must approximate this intermediate value by an average. We take

$$\frac{1}{k}(u^{n+1} - u^n) = \frac{1}{2}(f^n + f^{n+1}), \quad \text{so} \quad u^{n+1} = u^n + \frac{1}{2}k(f^n + f^{n+1})$$

and we must solve the nonlinear equation

$$u^{n+1} - \frac{1}{2}kf(u^{n+1}) = u^n + \frac{1}{2}kf^n$$

for u^{n+1} . This is also an implicit method. When it is used for partial differential equations, it is called the *Crank-Nicolson method*.

Leapfrog Here we attempt to achieve the intermediate value described above by instead widening the difference in v , setting

$$\frac{1}{2k}(u^{n+1} - u^{n-1}) = f^n, \quad \text{so} \quad u^{n+1} = u^{n-1} + 2kf^n.$$

This is a “two-level” method, since u^{n+1} depends on both u^n (via f^n) and u^{n-1} . We need to do something special at the first step to get u^1 when we have only u^0 , such as one step of forward Euler. Unfortunately, as we will see, though consistent, this method is completely *unstable*.

For ordinary differential equations, it is very worthwhile to construct schemes which are more complicated and more highly accurate than these. For partial differential equations, the errors due to space discretization are

often more important, and so there may be no profit in using an exceptionally accurate time-stepper if your spatial discretization is limited in its accuracy.

Okay, let's do a local truncation error analysis. To do this, we take a solution of the ODE and plug it into the discrete scheme. This means substituting $U(t)$ and $U(t+k)$ into the difference formula. The difference formula holds exactly for the discrete solution u^n . It will not hold exactly for time-samples of the ODE's solution $U(t)$. How much it fails to hold is the local truncation error.

Here $U(t)$ is a solution of the differential equation, so

$$U_t = f(U(t), t).$$

Let's consider only autonomous problems for simplicity. Differentiating the ODE, we obtain

$$\begin{aligned} U_{tt} &= \frac{d}{dt}f(U(t)) = f_U U_t = f f_U \\ U_{ttt} &= f^2 f_{UU} + f f_U^2 \\ &\vdots \end{aligned}$$

We are writing these formulas as though U and f are scalars; if they are vectors, then we need to be a little more careful about the matrix notation, but the conclusions are the same.

For a smooth solution $U(t)$, we have

$$U(t_n + k) = U(t_n) + kU_t(t_n) + \frac{1}{2}k^2U_{tt}(\xi)$$

for some $\xi \in (t_n, t_n + k)$.

Forward Euler The scheme is

$$u^{n+1} = u^n + kf^n = u^n + kf(u^n)$$

and so the local truncation error is

$$\begin{aligned} &\|U(t_n + k) - (U(t_n) + kf(U(t_n)))\| \\ &= \|U(t_n) + kU_t(t_n) + \frac{1}{2}k^2U_{tt}(\xi) - (U(t_n) + kf(U(t_n)))\| \\ &= \|k(U_t(t_n) - f(U(t_n))) + \frac{1}{2}k^2U_{tt}(\xi)\| \\ &= \frac{1}{2}k^2\|U_{tt}(\xi)\| = \frac{1}{2}k^2\|f(U(\xi)) f_U(U(\xi))\| \end{aligned}$$

That's as far as we can go without knowing some more information about the ODE we're studying. We would like to know that the right-hand side can be bounded by Ck^2 . When might this happen?

In a practical situation, there's some time T that you want to compute your solution up to some time T and would like to know about the local truncation error on that time interval. Because $U(t)$ is a continuous function of time, there will be an upper and lower bound for U on this interval:

$$m \leq U(t) \leq M \quad \text{for all } t \in [0, T].$$

Because $f(U)$ is a continuous function of U (required in order to guarantee existence of a solution) you know that $|f(U)|$ is also continuous and so there is some upper bound M_1 so that

$$|f(U)| \leq M_1 \quad \text{for all } U \in [m, M]$$

implying $|f(U(t))| \leq M_1$ for all $t \in [0, T]$. Similarly, if $f_U(U)$ is a continuous function of U then there is an upperbound M_2 so that $|f_U(U(t))| \leq M_2$ for all $t \in [0, T]$. Combining these, one obtains the desired

$$\|U(t_n + k) - (U(t_n) + kf(U(t_n)))\| \leq Ck^2,$$

showing that $p = 1$ in (3) and Forward Euler is a first-order accurate scheme.

Note: You don't need f_U to be continuous in order to be guaranteed a solution. For this reason, for some ODEs, Forward Euler will fail to be first-order accurate. Also, if you use Forward Euler on two different ODEs then the ODE with smaller bounds on f and f_U will be the one that results in smaller local truncation errors.

Backward Euler Here we have to control

$$\|U(t_n + k) - (U(t_n) + kf(U(t_n + k)))\|. \quad (4)$$

To do this, we will need to use Taylor's theorem three times. First of all,

$$U(t_n + k) = U(t_n) + kU_t(t_n) + \frac{1}{2}k^2U_{tt}(\xi) \quad \text{for some } \xi \in (t_n, t_n + k). \quad (5)$$

Secondly,

$$f(U(t_n + k)) = f(U(t_n)) + f_U(\eta) (U(t_n + k) - U(t_n)) \quad (6)$$

for some η between $U(t_n)$ and $U(t_n + k)$. Finally,

$$U(t_n + k) = U(t_n) + kU_t(\rho) \quad \text{for some } \rho \in (t_n, t_n + k). \quad (7)$$

Plugging (5)–(7) into the truncation error (4) yields

$$\begin{aligned} \|U(t_n + k) - (U(t_n) + kf(U(t_n + k)))\| &= \|\frac{1}{2}k^2U_{tt}(\xi) - k^2f_U(\eta)u_t(\rho)\| \\ &= \|\frac{1}{2}k^2f(U(\xi)) f_{UU}(U(\xi)) - k^2f(U(\rho)) f_U(\eta)\| \\ &\leq \frac{1}{2}k^2\|f(U(\xi)) f_{UU}(U(\xi))\| + k^2\|f(U(\rho))\| \|f_U(\eta)\| \\ &\leq Ck^2. \end{aligned}$$

To get the constant C we have to use the same logic as we did at the end of the local error analysis of the forward Euler method. This allows us to conclude that the method is first-order accurate.

Trapezoid Proceeding as above, one needs to control

$$\|U(t_n + k) - (U(t_n) + \frac{1}{2}k(f(U(t_n + k)) + f(U(t_n))))\|$$

Applying Taylor's theorem four times, one can find that the local truncation error equals the nasty expression

$$\begin{aligned} &\|k^3/6 [f(U(\xi))^2 f_{UU}(U(\xi)) + f(U(\xi))f_{UU}(U(\xi))^2] \\ &\quad - k^3/4 f(U(\rho)) f_U(U(\rho)) f_U(U(t_n)) - k^3/2 f_{UU}(\eta) f(U(\tilde{\rho}))^2\| \end{aligned}$$

where $\xi, \rho, \tilde{\rho} \in (t_n, t_n + k)$ and η is between $U(t_n)$ and $U(t_n + k)$. Using the triangle inequality and the prior logic, one can bound the above by Ck^3 , concluding that the method is second-order accurate.

Note that in order for this method to be second order accurate we're demanding even more of $f(U)$ — its second derivative has to be a continuous function of U . If one is computing an ODE for which this isn't true then this method will not be as accurate.

The above is all quite rigorous and carefully written out and makes it clear how much (or how little) smoothness one needs in a solution for a time-stepper to achieve its maximal accuracy. Another way of doing these calculations is with asymptotic notation. Let's revisit the Trapezoidal Method calculation using this approach.

First,

$$U(t_n + k) = U(t_n) + kU_t(t_n) + k^2/2 U_{tt}(t_n) + k^3/6 U_{ttt}(t_n) + \dots \quad (8)$$

Similarly,

$$f(V) = f(U) + f_U(U) (V - U) + 1/2 f_{UU}(U) (V - U)^2 + \dots \quad (9)$$

Plugging (8) into (9) yields

$$\begin{aligned} f(U(t_n + k)) &= f(U(t_n)) + k f_U(U(t_n)) u_t(t_n) \\ &+ k^2 \left[\frac{1}{2} f_U(U(t_n)) u_{tt}(t_n) + \frac{1}{2} f_{UU}(U(t_n)) u_t(t_n)^2 \right] + \dots \end{aligned} \quad (10)$$

Plugging (8) and (10) into the local truncation error

$$U(t_n + k) - (U(t_n) + k/2 f(U(t_n + k)) + k/2 f(U(t_n)))$$

yields

$$\begin{aligned} &U(t_n + k) - (U(t_n) + k/2 f(U(t_n + k)) + k/2 f(U(t_n))) \\ &= -\frac{1}{12} k^3 \left[f(U(t_n))^2 f_{UU}(U(t_n)) + f(U(t_n)) f_U(U(t_n))^2 \right] + \dots \end{aligned}$$

This shows that the lowest-order term in a k -expansion of the local truncation error is on the order of k^3 , and so the trapezoid rule is second-order accurate.

I encourage you to do the k -expansions for the truncation errors for forward Euler and backward Euler. As you might expect, since the trapezoid rule is “halfway in between” forward and backward Euler, the k^2 terms of their truncation error cancel each other, resulting in the trapezoid rule being one order more accurate.

In general, you get a gain in accuracy when the difference formulas are *symmetric*.

Leapfrog This method is also second-order accurate, as you can easily guess by symmetry. The computation is much the same as for trapezoid.