Simple time-stepping

Our previous examples were all stationary problems. However, many practical simulations describe processes that change over time. In this part we want to start looking a bit closer onto time-domain partial differential equations and their efficient implementation.

Finite Difference Approximation for the time-derivative

We want to approximate the time derivative $\frac{du}{dt}$. Remember that

$$\frac{du}{dt} \approx \frac{u(x+h) - u(x)}{h}$$

for sufficiently small h.

There are three standard approximations for the time-derivative:

- The forward difference:

$$\frac{du}{dt} \approx \frac{u(x+h) - u(x)}{h}.$$

- The backward difference:

$$\frac{du}{dt} \approx \frac{u(x) - u(x-h)}{h}.$$

- The centered difference:

$$\frac{du}{dt} \approx \frac{u(x+h) - u(x-h)}{2h}.$$

To understand the error of these schemes we can use Tayler expansions to obtain

$$\frac{u(x+h) - u(x)}{h} = u'(x) + \frac{1}{2}hu''(x) + \dots$$

$$\frac{u(x) - u(x-h)}{h} = u'(x) - \frac{1}{2}hu''(x) + \dots$$

$$\frac{u(x+h) - u(x-h)}{2h} = u'(x) + \frac{1}{6}h^2u'''(x) + \dots$$

Hence, the error of the first two schemes decreases linearly with h and the error in the centred scheme decreases quadratically with h.

The 3-point stencil for the second derivative

For simplicity we denote $u_i := u(x), u_{i+1} := u(x+h), u_{i-1} := u(x-h)$. We want to approximate

$$\frac{d}{dx}\left[\frac{du}{dx}\right].$$

The trick is to use an approximation around half-steps for the outer derivative, resulting in

$$\frac{d}{dx}\left[\frac{du}{dx}\right] \approx \frac{1}{h}\left[u'_{i+\frac{1}{2}} - u'_{i-\frac{1}{2}}\right].$$

The derivatives at the half-steps are now again approximated by centered differences, resulting in

$$\begin{aligned}\frac{d}{dx}\left[\frac{du}{dx}\right] &\approx \frac{1}{h}\left[\frac{u_{i+1}-u_i}{h} - \frac{u_i - u_{i-1}}{h}\right] \\ &= \frac{u_{i+1} - 2u_i - u_{i-1}}{h^2} \\ &= u''(x) + \mathcal{O}(h^2)\end{aligned}$$

This is the famous second order finite difference operator that we have already used before. Its error is quadratically small in h.

Application to time-dependent problems

We now want to solve

$$\begin{aligned}\frac{dU}{dt} &= f(U,t) \\ U(0) &= U_0,\end{aligned}$$

where $U(t) : \mathbb{R} \to \mathbb{R}^n$ is some vector valued function.

The idea is replace $\frac{dU}{dt}$ by a finite difference approximation.

- Forward Euler Method

$$\frac{U_{n+1} - U_n}{\Delta t} = f(U_n, t_n)$$

- Backward Euler Method

$$\frac{U_{n+1} - U_n}{\Delta t} = f(U_{n+1}, t_{n+1})$$

The forward Euler method is an explicit method. We have that

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

and the right-hand side only has known values.

In contrast to this is the backward Euler Method, which is an implicit method since

$$U_{n+1} = U_n + \Delta t f(U_{n+1}, t_{n+1}).$$

We hence need to solve a linear or nonlinear system of equations to compute U_{n+1}.

Stability of forward Euler

We consider the model problem

$$u' = \alpha u$$

for $\alpha < 0$. Note that the explicit solution of this problem is $u(t) = u_0 e^{\alpha t}$. For $t \to \infty$ we have $u(t) \to 0$ if $\alpha < 0$.

The forward Euler method can now be written as

$$\begin{aligned}U_{n+1} &= (1 + \alpha\Delta t)U_n \\ &= (1 + \alpha\Delta t)^n U_0.\end{aligned}$$

Hence, in order for the solution to decay we need that $|1 + \alpha\Delta t| < 1$ or equivalently

$$-1 < 1 + \alpha\Delta t < 1,$$

from which we obtain $|\alpha\Delta t| < 2$ (if α negative). Now consider the problem $\$\frac{dU}{dt} = AU\$$

with $A \in \mathbb{R}^{n\times n}$ diagonalizable. For any eigenpair $(\lambda, \hat{U})$ of A satisfying $A\hat{U} = \lambda\hat{U}$ the function $U(t) = e^{\lambda t}\hat{U}$ is a solution for this problem. Therefore, for forward Euler to converge we require that

$$\Delta t < \frac{2}{|\lambda_{max}(A)|},$$

where λ_{max} is the largest eigenvalue by magnitude.

As example let us take a look at the problem $\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2} with$u(x, 0) = u_0(x),u(0, t) = u(1, t) = 0$. We can discretise the right-hand side using our usual second order finite dfference scheme. For the left-hand side, we use the forward Euler method. This gives us the recurrence equation

$$U_{n+1} = U_n + \Delta t A U_n,$$

with $A = \frac{1}{h^2}\text{tridiag}(1, -2, 1)$.

The eigenvalues of A are given explicitly as

$$\lambda_k = -\frac{1}{h^2} 4 \sin^2 \frac{k\pi}{2(n+1)}$$

We therefore have that $|\lambda_{max}| \sim \frac{4}{h^2}$. Hence, for forward Euler to be stable we require that

$$\frac{\Delta t}{h^2} \lesssim \frac{1}{2}.$$

Hence, we need that $\Delta t \sim h^2$, meaning that the number of required time steps grows qudratically with the discretisation accuracy.

Stability of backward Euler

For backward Euler we obtain

$$\begin{aligned} U_{n+1} &= (1 - \alpha\Delta t)^{-1} U_n \\ &= (1 - \alpha\Delta t)^{-n} U_0. \end{aligned}$$

We now require that $|(1 - \alpha\Delta t)^{-1}| < 1$. But for $\alpha > 0$ this is always true. Hence, the backward Euler method is unconditionally stable.

Implicit vs explicit methods

This analysis is very typical. In computational sciences we always have to make a choice between implicit and explicit methods. The advantage of implicit methods are the very good stability properties, allowing us for the backward Euler method to choose the time-discretisation independent of the spatial discretisation. For explicit methods we have to be much more careful and in the case of Euler we have the quadratic dependency between time-steps and spatial discretisation. However, a single time-step is much cheaper for explicit Euler as we do not need to solve a linear or nonlinear system of equations in each step. The right choice of solver depends on a huge number of factors and is very application dependent.

Time-Stepping Methods in Software

In practice we do not necesarily use explicit or implicit Euler. There are many better methods out there. The Scipy library provides a number of time-stepping algorithms. For PDE problems PETSc has an excellent infrastructure of time-stepping methods built in to support the solution of time-dependent PDEs.

By Timo Betcke