Classic finite differences

YUJI NAKATSUKASA

Many slides by Ricardo Ruiz Baier
Mathematical Institute, Oxford

Computational Techniques InFoMM Centre for Doctoral Training
Michaelmas Term 2019, Week 6

UNIVERSITY OF OXFORD
Mathematical Institute

Oxford Mathematics

Plan

- Monday: Finite differences
- Tuesday: Finite differences + intro to Finite elements
- Wednesday: Finite elements
- Thurs: Finite elements, Demo on FEniCS by Federico Danieli

Plan for FD

1. Approximating derivatives from samples
2. Global error: u from u_{tt}
3. ODEs: Euler, Runge-Kutta, time-step stability
4. PDEs: time-space discretisation stability

- Often a closed-form solution for an ODE or a PDE cannot be derived explicitly and one has to resort to other techniques
- Numerically computed solutions could be obtained
- We proceed to discretize the problem, e.g. via finite differences
  1. represent the continuous space-time domain with a finite set of points
  2. replace differential operators with difference quotients
  3. determine the value of functions and coefficients on the points
  4. provide an approximation of the solution to the original problem
  5. want more accuracy? no (easy) way around: use more points/computing

$$\text{original PDE for } u(\boldsymbol{x},t) \xrightarrow{\text{Finite differences}} \text{Discrete difference equations} \xrightarrow{\text{Solution method}} \{u^n_{i,j,k}\} \approx u(\boldsymbol{x},t)$$

The finite difference (FD) method

Philosophy

- Rather simple to understand
- Easy to implement for regular domains
- Discretization is point-wise – fundamental tool: Taylor expansion
- Many fast solvers and packages (Fishpack, ClawPack, etc)
- Strong regularity requirements

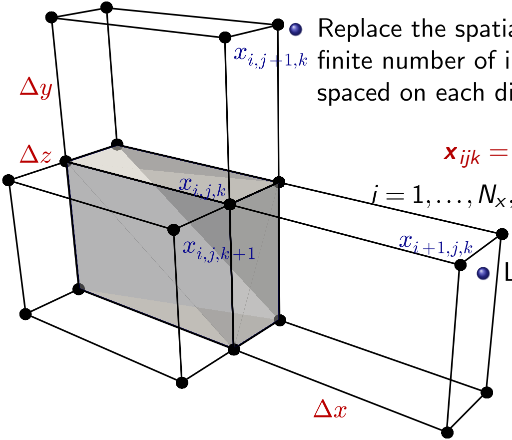
Further reading (highly recommended)

LeVeque, Finite Difference Methods for ODEs and PDEs. SIAM 2007.
Particularly Chapters 1, 2, and 9.

Others: Hairer(-Norsett)-Wanner (Classic), Iserles, Suli and Mayers

The finite difference (FD) method

Philosophy



- Replace the spatial domain with a mesh of a finite number of interior nodes (assumed unif. spaced on each direction)

$$\boldsymbol{x}_{ijk} = (i\Delta x, j\Delta y, k\Delta z),$$

$$i = 1, \ldots, N_x, j = 1, \ldots, N_y, k = 1, \ldots, N_z$$

- Likewise for the time domain:

$$t^n = n\Delta t, \quad n = 0, \ldots, N_t$$

- Properly define continuous functions and variables on the grid, e.g.

$$u(\boldsymbol{x}_{ijk}, t^n), \text{ to be approximated by } u^n_{ijk}$$

Let's slow down: 1D first...

The finite difference (FD) method
Philosophy

University of Oxford Mathematical Institute

- For v regular enough, Taylor's Theorem gives

$$v(x+\Delta x) = v(x) + \Delta x v'(x) + \frac{\Delta x^2}{2!}v''(x) + \frac{\Delta x^3}{3!}v^{(3)}(x) + \frac{\Delta x^4}{4!}v^{(4)}(x) + \cdots$$
$$v(x-\Delta x) = v(x) - \Delta x v'(x) + \frac{\Delta x^2}{2!}v''(x) - \frac{\Delta x^3}{3!}v^{(3)}(x) + \frac{\Delta x^4}{4!}v^{(4)}(x) + \cdots$$

- Then, for sufficiently small Δx, one has

$$\frac{v(x+\Delta x)-v(x)}{\Delta x} = v'(x) + \mathcal{O}(\Delta x), \quad \frac{v(x+\Delta x)-v(x-\Delta x)}{2\Delta x} = v'(x) + \mathcal{O}(\Delta x^2),$$
$$\frac{v(x+\Delta x)-2v(x)+v(x-\Delta x)}{\Delta x^2} = v''(x) + \mathcal{O}(\Delta x^2)$$

Definition

A term $E(\Delta x)$ is $\mathcal{O}(\Delta x^p)$ if $\dfrac{E(\Delta x)}{\Delta x^p} \to \text{const.}$ when $\Delta x \to 0$

The finite difference (FD) method

Philosophy

- Replace a continuous partial differential operator (in this case, only ∂_x) with a **finite difference** operator:

$$\partial_x u\big|_{(x_i,t^n)} \approx \frac{u_{i+1}^n - u_i^n}{x_{i+1} - x_i} = \frac{u_{i+1}^n - u_i^n}{\Delta x} \qquad \text{(forward difference)}$$

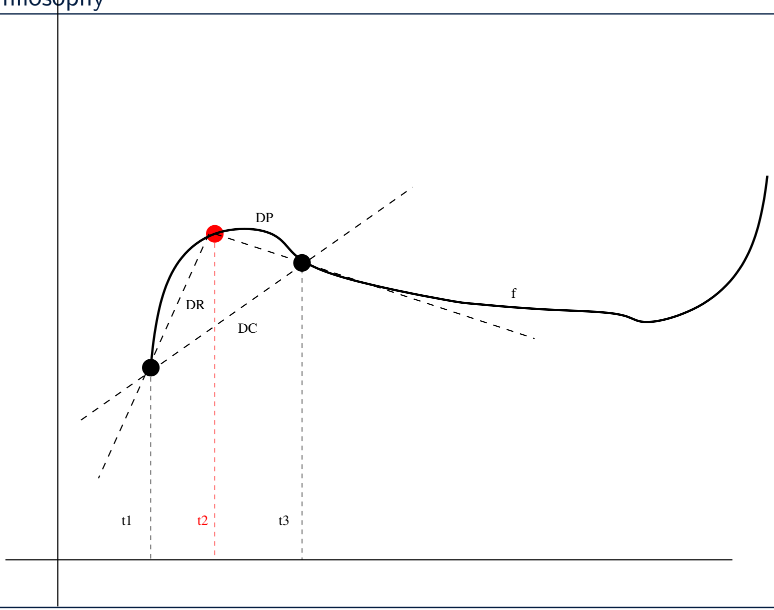
$$\partial_x u\big|_{(x_i,t^n)} \approx \frac{u_i^n - u_{i-1}^n}{x_i - x_{i-1}} = \frac{u_i^n - u_{i-1}^n}{\Delta x} \qquad \text{(backward difference)}$$

$$\partial_x u\big|_{(x_i,t^n)} \approx \frac{u_{i+1}^n - u_{i-1}^n}{x_{i+1} - x_{i-1}} = \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} \qquad \text{(central difference)}$$

- Leading error term in the first two formulae is proportional to Δx while in the third formula is proportional to $(\Delta x)^2$
- Useful derivation: interpolate sample points with lowest-degree polynomial $p(x_i) = u(x_i)$, and take $p'(x)$ (check all above in this form)

The finite difference (FD) method

Philosophy



The finite difference (FD) method

Philosophy

- From now on, $\Delta x = h$
- Central differences not only on one point:
- For each $x_2, \ldots, x_m$ can approximate $u'(x_i)$ in the same way, so that

$$u' \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_m \\ x_{m+1} \end{pmatrix} \approx \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_m \\ v_{m+1} \end{pmatrix} = \frac{1}{h} \begin{pmatrix} & & & & & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{-1}{2} & & & \\ & \frac{1}{2} & 0 & \frac{-1}{2} & & \\ & -1 & \ddots & \ddots & \ddots & -1 \\ & & & \frac{1}{2} & 0 & \frac{-1}{2} \\ \frac{-1}{2} & & & & & \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_m \\ u_{m+1} \end{pmatrix}$$

- But what about v_1 and v_{m+1}?
  - Use one-sided difference (backwards / forwards).
  - For periodic problems, $u_0 = u_{m+1}$ and $u_{m+2} = u_1$.
  - Often boundary conditions remove this problem anyway.

The finite difference (FD) method

Philosophy

- From now on, $\Delta x = h$
- Central differences not only on one point:
- For each $x_2, \ldots, x_m$ can approximate $u'(x_i)$ in the same way, so that

$$
u'\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_m \\ x_{m+1} \end{pmatrix} \approx \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_m \\ v_{m+1} \end{pmatrix} = \frac{1}{h}\begin{pmatrix} 1 & -1 & & & & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{-1}{2} & & & \\ & \frac{1}{2} & 0 & \frac{-1}{2} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \frac{1}{2} & 0 & \frac{-1}{2} \\ \frac{-1}{2} & & & & 1 & -1 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_m \\ u_{m+1} \end{pmatrix}
$$

- But what about v_1 and v_{m+1}?
  - Use one-sided difference (backwards / forwards).
  - For periodic problems, $u_0 = u_{m+1}$ and $u_{m+2} = u_1$.
  - Often boundary conditions remove this problem anyway.

The finite difference (FD) method

Philosophy

- From now on, $\Delta x = h$
- Central differences not only on one point:
- For each $x_2, \ldots, x_m$ can approximate $u'(x_i)$ in the same way, so that

$$u' \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_m \\ x_{m+1} \end{pmatrix} \approx \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_m \\ v_{m+1} \end{pmatrix} = \frac{1}{h} \begin{pmatrix} & \frac{-1}{2} & & & & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{-1}{2} & & & \\ & \frac{1}{2} & 0 & \frac{-1}{2} & & \\ & -1 & \ddots & \ddots & \ddots & -1 \\ & & & \frac{1}{2} & 0 & \frac{-1}{2} \\ \frac{-1}{2} & & & & \frac{1}{2} & \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_m \\ u_{m+1} \end{pmatrix}$$

- But what about v_1 and v_{m+1}?
  - Use one-sided difference (backwards / forwards).
  - For periodic problems, $u_0 = u_{m+1}$ and $u_{m+2} = u_1$.
  - Often boundary conditions remove this problem anyway.

The finite difference (FD) method

Philosophy

- From now on, $\Delta x = h$
- Central differences not only on one point:
- For each $x_2, \ldots, x_m$ can approximate $u'(x_i)$ in the same way, so that

$$
u' \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_m \\ x_{m+1} \end{pmatrix} \approx \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_m \\ v_{m+1} \end{pmatrix} = \frac{1}{h} \begin{pmatrix} h & & & & & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{-1}{2} & & & \\ & \frac{1}{2} & 0 & \frac{-1}{2} & & \\ & -1 & \ddots & \ddots & \ddots & -1 \\ & & & \frac{1}{2} & 0 & \frac{-1}{2} \\ \frac{-1}{2} & & & & & h \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_m \\ u_{m+1} \end{pmatrix}
$$

- But what about v_1 and v_{m+1}?
  - Use one-sided difference (backwards / forwards).
  - For periodic problems, $u_0 = u_{m+1}$ and $u_{m+2} = u_1$.
  - Often boundary conditions remove this problem anyway.

- The derivative of the derivative: use e.g. **backward difference of the forward difference**

$$u_{xx}(x_i, t^n) \approx \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2}$$

- Using the previous "$O(\cdot)$"-stuff

$$u_{xx}(x_i, t^n) = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} + O(h^2)$$

- The approximation is second-order accurate: as we go down with h, the error goes down quadratically
- Replace h with $h/2$, error should divide by four (for smooth enough u)
- As before, interpolate $p(x_i) = f(x_k)$ and take $p''(x)$

Drop the time-dependence for a sec

- If $x \in \Omega = [a, b]$ and $u(a) = u_1 = \alpha$ and $u(b) = u_{m+1} = \beta$
- The classic "1 -2 1" rule at each point can be implemented as

```
>> h=(b-a)/m; x=(a:h:b)';
>> u=sin(x); u(1)=alpha; u(m+1)=beta;

>> for i=2:m
>>     uxx(i)=(u(i+1)-2*u(i)+u(i-1))/h^2;
>> end
```

The finite difference (FD) method

Second derivatives

- In **matrix form**:
- matrix-vector multiplication $\leftrightarrow$ evaluation of differential operator (1D Laplace)

```
uxx =

      |-2    1                 |  | u(2)|     | alpha/h^2 |
      | 1   -2    1            |  | u(3)|     |     0     |
      |   .    .    .          |  |  .  |     |     .     |
1/h^2 |     .    .    .        |  |  .  |  +  |     .     |
      |       .    .    .      |  |  .  |     |     .     |
      |          1   -2    1   |  |     |     |     0     |
      |               1   -2   |  | u(m)|     | beta/h^2  |
```

- implemented as

```
>> e=ones(m-1,1);
>> A=spdiags([e,-2*e,e],(-1:1),m-1,m-1);
>> bc=[ ... ]
>> uxx(2:m)=1.0/h^2*A*u(2:m)+bc;
```

The diffusion equation

A finite difference discretization

One-D steady problem: find u such that

$$-\kappa \partial_{xx} u = f$$

- with boundary conditions as appropriate
- f is a forcing term
- again the matrix structure

```
        |-2    1                |  | u(2)|     | f(2) + Kalpha/h^2|
        | 1   -2     1          |  | u(3)|     | f(3)             |
        |    .     .     .      |  |  .  |     |  .               |
- K/h^2 |      .     .     .    |  |  .  |  =  |  .               |
        |        .     .     .  |  |  .  |     |  .               |
        |          1    -2     1|  |     |     | f(m-1)           |
        |               1    -2 |  | u(m)|     | f(m) + Kbeta/h^2 |
```

i.e., $A\vec{u} = \vec{f}$, with **unknown** $\vec{u}$ and datum $\vec{f}$

$\rightarrow$ Tridiagonal linear system$\rightarrow$ $O(n)$ time solution

The diffusion equation

Steady case - Experimental error

- The "1 -2 1" rule was $O(h^2)$ for evaluating the 2nd derivative. Will it still be so after solving $A\vec{u} = \vec{f}$?
- Example: $-\partial_{xx} u = f$ in $[a, b]$

```
>> h=(b-a)/m; x=(a:h:b)';
```

- with BC (matrix is now $m+1 \times m+1$ and has a "-" in front)

```
>> e=ones(m+1,1);
>> A=spdiags([-e,2*e,-e],(-1:1),m+1,m+1);
>> A(1,:)=zeros(1,m+1); A(1,1)=1; A(m+1,:)=zeros(1,m+1); A(m+1,m+1)=1;
>> f=@(x)[...]; b=h^2*f(x); b(1)=alpha; b(m+1)=beta;
>> u=A\b; plot(x,u);
```

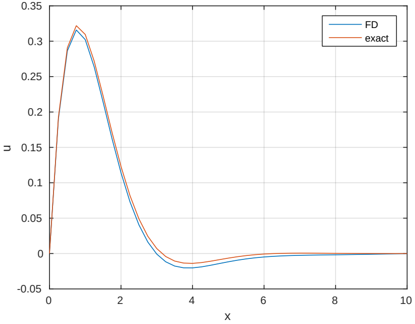
- checking the error (if you know the exact solution)

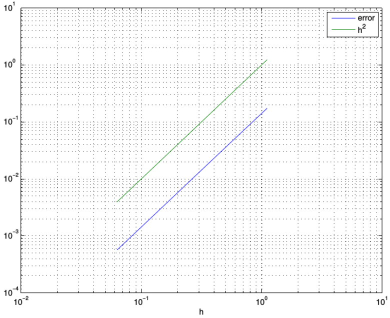
```
>> uex=@(x)[...]; UEX=uex(x);
>> err=max(abs(UEX-u));
```

The diffusion equation

Steady case - Experimental error

- Let's try!! $\Omega = [0, 10]$, $f(x) = 2\cos(x)/\exp(x)$. BC: $u = g(x) = \sin(x)/\exp(x)$ on $\partial\Omega$, exact sol: $u_{ex}(x) = \sin(x)/\exp(x)$
- with $m = 40$ we should get `err=0.0089` (or so)





take $m = 10, 20, 40, 80, 160$, solve the Poisson problem and plot errors vs h, to check that they decay as h^2

The diffusion equation

Error splitting

- OK! The "1 -2 1" rule was $O(h^2)$ also for solving $A\vec{u} = \vec{f}$
- But, what kind of "error" are we measuring?
- See diagram!
- How do we ensure that these decrease when $h \to 0$ (**convergence**)?
- First we need consistency and stability
- Waving hands:
  - Substitute the true solution in the discrete problem. The remainder is called (local) truncation error. The method is consistent if the truncation error goes to zero as $h \to 0$
  - Stability: small perturbation of the data implies perturbed solutions independently of h (e.g. $A_h\vec{u}_h = \vec{f}$ is stable if $\|A_h^{-1}\| \leq C$)
  - e.g. in previous diffusion eqn, stability holds
- Consistency + Stability $\Rightarrow$ Convergence (more to come)

Time-stepping methods for ODEs

A few basic examples for $\partial_t u = f(t,u)$, $u(0) = y_0$

- Grid in time $t_n = n\Delta t$, with a (fixed) time step Δt

```
|-------|-------|-------|-------|-------|-----
t0=0    t1=dt    t2=2dt   ...
```

- Forward Euler

$$\begin{cases} u_{n+1} = u_n + \Delta t\, f(t_n, u_n) & n \geq 0, \\ u_0 = y_0. \end{cases} \tag{1}$$

- Backward Euler

$$\begin{cases} u_{n+1} = u_n + \Delta t\, f(t_{n+1}, u_{n+1}) & n \geq 0, \\ u_0 = y_0. \end{cases} \tag{2}$$

$\rightarrow$Implicit method as updating of u_{n+1} involves itself, requires linear system

- Midpoint rule

$$\begin{cases} u_{n+1} = u_{n-1} + 2\Delta t\, f(t_n, u_n). & n \geq 1, \\ u_0 = y_0,\ u_1 = y_1, & \text{(data or approx.)}. \end{cases} \tag{3}$$

- $\partial_t u \approx$ by forward (1), backward (2), and centred (3) FDs

Time-stepping methods for ODEs

A few more

- Crank-Nicolson (coming from trapezoid rule applied to the integral form)

$$\begin{cases} u_{n+1} = u_n + \dfrac{\Delta t}{2}\left[f(t_n, u_n) + f(t_{n+1}, u_{n+1})\right], \quad n \geq 0, \\ u_0 = y_0. \end{cases}$$

- Heun (obtained through forward Euler of u_{n+1} in the arg of f)

$$\begin{cases} u_{n+1} = u_n + \dfrac{\Delta t}{2}\left[f(t_n, u_n) + f\left(t_{n+1}, u_n + \Delta t f(t_n, u_n)\right)\right], \quad n \geq 0, \\ u_0 = y_0. \end{cases}$$

- These can be **explicit** (if u_{n+1} can be computed directly from u_k, $k \leq n$) or **implicit**. The way we treat f is key (semi-implicit or IMEX methods also available)
- These can be one or multi-**step**

Time-stepping methods for ODEs

Δt and the choice of method do matter!

- Neat! But how do we choose Δt?
- **Not arbitrarily!**
- Consider

$$\begin{cases} u'(t) = -2u(t) & \text{for } t \in \mathbb{R}_+ \\ u(0) = 1, \end{cases}$$

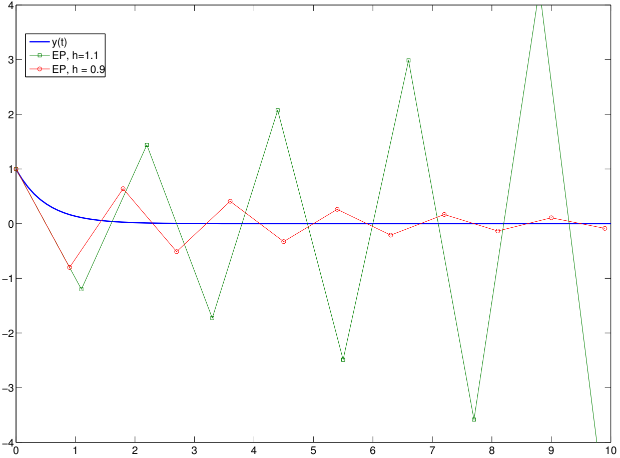
- exact solution

$$u(t) = e^{-2t}$$

- How do the first two methods behave?

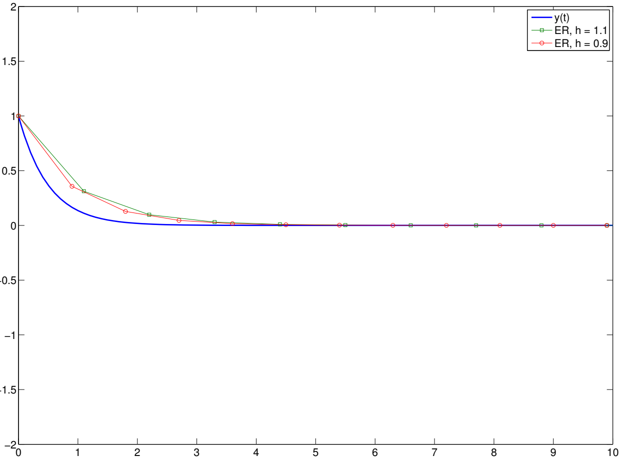
Time-stepping methods for ODEs: Forward Euler

Δt and the choice of method do matter!



Time-stepping methods for ODEs: Back Euler

Δt and the choice of method do matter!



Time-stepping methods for ODEs

Mathematical Institute

Definition

A method for the Dahlquist problem $\partial_t u = \lambda u(t)$, $u(0) = 1$, with $\lambda \in \mathbb{C}$, is *absolutely stable* if $\exists$ $(\Delta t)_0 > 0$ s.t. for all $\Delta t \leq (\Delta t)_0$

$$|u_n| \longrightarrow 0 \quad \text{whenever} \quad t_n \longrightarrow +\infty.$$

If true for all $\Delta t > 0$, the method is unconditionally stable (or $\mathscr{A}$-stable)

- Exact sol $u(t) = e^{\lambda t} \to 0$. Take $\lambda < 0$.
- Forward Euler:

$$u_0 = 1, \quad u_{n+1} = u_n(1 + \lambda \Delta t) = (1 + \lambda \Delta t)^{n+1}, \quad n \geq 0. \tag{4}$$

$\Rightarrow \lim_{n\to\infty} u_n = 0$ iff

$$-1 < 1 + \Delta t \lambda < 1, \quad i.e. \quad \underbrace{\Delta t < 2/|\lambda|}_{\text{Stab. condition}} \tag{5}$$

For fixed Δt, u_n behaves like $u(t_n)$ when $t_n \to \infty$.

	Explicit/Implicit	Steps	Stability	Order
FE	E	1	Conditionally	1
BE	I	1	Unconditionally	1
MP	E	2	Unstable	2
CN	I	1	Unconditionally	2
H	E	1	Conditionally	2

Roughly, for linear problems

- (good) implicit methods: stable but requires $Ax = b$
- explicit methods: only needs matrix-vector multiply Ax, but limited stability

- If $\lambda = \lambda(t) < 0$ then $|\lambda| \leftarrow \max_{t\in[0,\infty)} |\lambda(t)|$ in the stability condition
- Generalization I

$$\begin{cases} u'(t) = \lambda(t)u(t) + r(t), \quad t \in (0, +\infty), \\ u(0) = 1, \end{cases} \tag{6}$$

with λ, r continuous and $-\lambda_{max} \leq \lambda(t) \leq -\lambda_{min}$, $0 < \lambda_{min} \leq \lambda_{max} < +\infty$

- Generalization II: $u' = f(t,u)$ if

$$-\lambda_{max} < \partial f/\partial u(t,u) < -\lambda_{min}, \forall t \geq 0,\ \forall u \in (-\infty, \infty),$$

with $\lambda_{min},\ \lambda_{max} \in (0, +\infty)$.

- For generalizations I,II: **a similar stability condition holds**!!

Time-stepping methods for ODEs

Stability regions

- The solution of the Dahlquist problem is $u(t) = \exp(\lambda t)$. If $\mathrm{Re}(\lambda) < 0$ then $\lim_{t\to+\infty} |u(t)| = 0$.
- The *absolute stability region* $\mathscr{A}$ is then $\Delta t\lambda$ s.t. the method produces solutions that tend to zero when $t_n \to \infty$
- The method is *$\mathscr{A}$-stable* (or unconditionally stable) if $\mathscr{A} \cap \mathbb{C}^- = \mathbb{C}^-$

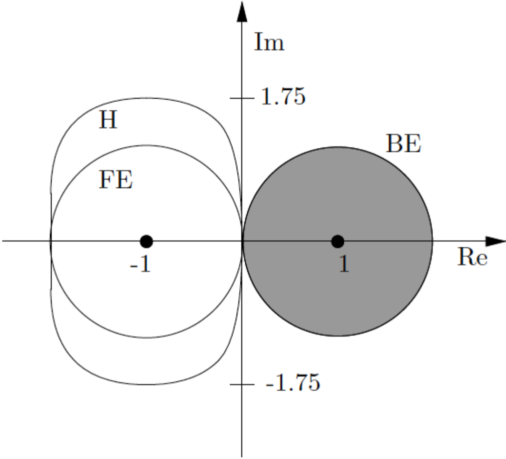


Figure: Stability regions.

Runge-Kutta methods

- Use 'intermediate' steps to obtain higher order
- Most famous is 4th stage, 4th order

$$u_{n+1} = u_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4),$$

where

$$\begin{aligned}
k_1 &= f(t_n, u_n) \\
k_2 &= f(t_n + \frac{1}{2}h, u_n + \frac{1}{2}hk_1), \\
k_3 &= f(t_n + \frac{1}{2}h, u_n + \frac{1}{2}hk_2), \\
k_4 &= f(t_n + h, u_n + hk_3)
\end{aligned}$$

- Accuracy $O(h^4)$ (local accuracy $O(h^5)$)
- Above is explicit; implicit version available (J. Butcher's book)

Other time-stepping methods for ODEs

Not covered, but important

- Stability properties of ODE systems
- Multistep methods (Adams-Bashforth, Adams-Moulton etc): another higher-order method, uses mutliple time-step soln.
  - 2-step Adams-Bashforth

$$u_{n+2} = u_{n+1} + \frac{3}{2} hf(t_{n+1}, u_{n+1}) - \frac{3}{2} hf(t_n, u_n)$$

  - To get started, requires multiple inivial values u_0, u_1 (computed using other methods)
  - Stability analysis: Dahlquist's theorem (consistency+stability=convergence)
- Check LeVeque and/or other sources for more

Now to PDEs

The method of lines

UNIVERSITY OF OXFORD
Mathematical Institute

The 1-D heat equation

$$\partial_t u - \kappa \partial_{xx} u = 0$$

Idea: **semi-discretize** this problem using our friend "1 -2 1"

```
      |         |             |-2    1                 |  |         |
      | u (t)   |             |                        |  | u (t)   |
      |  1      |             | 1   -2    1            |  |  1      |
      |         |             |                        |  |         |
d     |  .      |             |    .    .   .          |  |  .      |
--    |  .      |  =  K/h^2   |     .    .   .         |  |  .      |
dt    |  .      |             |      .    .   .        |  |  .      |
      |         |             |                        |  |         |
      |         |             |          1   -2    1   |  |         |
      | u (t)   |             |                        |  | u (t)   |
      |  m+1    |             |               1   -2   |  |  m+1    |
```

which is an ODE system we can discretize in time (somewhat) independently!!

The heat equation $\partial_t u = \kappa \partial_{xx} u$

A finite difference discretization – Wait, but...

Mathematical Institute

- Previously we realized that the choice of Δt is important
- Now, how do we choose h?
- Are they "compatible"?
- How do we assess whether $u_i^n \to u_{ex}(x_i, t^n)$ when $h, \Delta t \to 0$?

The heat equation $\partial_t u = \kappa \partial_{xx} u$

A finite difference discretization – Some properties

Mathematical Institute

Definition

A numerical scheme $\mathscr{L}(u_i^n) = 0$ for a given PDE $\mathscr{P}(u(x,t)) = 0$ is said *consistent* if the truncation error satisfies

$$\tau(h, \Delta t) \equiv \mathscr{P}(u(x,t)) - \mathscr{L}((u_i^n)_{i \leq m+1}) \to 0, \quad \text{when } h, \Delta t \to 0.$$

The scheme is consistent of order (p,q) if $\tau(h,\Delta t) = \mathscr{O}(h^p) + \mathscr{O}(\Delta t^q)$.

Example

The explicit forward Euler method (for the heat eqn.) is consistent of order (2,1) (next slide):

$$\partial_t u - \kappa \partial_{xx} u - \left(\frac{u_i^{n+1} - u_i^n}{\Delta t} - \kappa \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} \right) = \mathscr{O}(h^2) + \mathscr{O}(\Delta t)$$

Explicit forward Euler for heat equation $\partial_t u = \partial_{xx} u$

University of Oxford
Mathematical Institute

$$u_i^{n+1} = u_i^n + \Delta t \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2}$$

- $\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} = \partial_{xx} u + \frac{h^2}{12}\partial_{xxxx} u + O(h^3) = \partial_t u + \frac{h^2}{12}\partial_{tt} u + O(h^3)$
- Hence $u_i^{n+1} = u_i^n + \Delta t \partial_t u + \frac{h^2 \Delta t}{12}\partial_{tt} u + O(h^3)$
- Local error:

$$\begin{aligned} u_i^{n+1} - \left(u_i^n + \Delta t \partial_t u + \frac{(\Delta t)^2}{2}\partial_{tt} u\right) &= \frac{h^2 \Delta t}{12}\partial_{tt} u - \frac{(\Delta t)^2}{2}\partial_{tt} u \\ &= O(h^2 \Delta t + (\Delta t)^2) \end{aligned}$$

- Since $O(1/\Delta t)$ timesteps taken, global error $O(h^2 + \Delta t)$

- Solution for difference eqn. of the form: $u_i^n = \rho^n \exp(2\pi \mathbf{i} \ell x_i)$
- ℓ: discrete Fourier harmonics, ρ: amplification factor

von Neumann criterion

A numerical scheme for an evolution eqn. is stable if and only if its largest amplification factor satisfies

$$|\rho| \leq 1 + \mathcal{O}(\Delta t).$$

Remark

- *The explicit forward Euler method is stable if $\Delta t \leqslant \frac{h^2}{2\kappa}$*
- *The implicit backward Euler scheme is unconditionally stable!*

Lax principle

Mathematical Institute

Definition (Convergence)

A scheme is convergent if $u_i^n \to u(x_i, t^n)$ when $h, \Delta t \to 0$

Theorem (Lax principle)

Consistency $+$ *Stability* $\Rightarrow$ *Convergence*

valid only in the nice linear case... (but appilcable to PDEs, not just ODEs)

Crank-Nicolson method

March in time using two points

$$(1-\frac{r}{2}\delta_x^2)U_m^{n+1} = (1+\frac{r}{2}\delta_x^2)U_m^n$$

where U_m^n is approximation to $u(nk, mh)$ and $r = k/h^2$.

- unconditionally stable, just like backward Euler
- but often exhibit spurious oscillations

The heat equation in multi-D

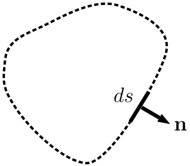
Generalities

Let $\Omega \subset \mathbb{R}^3$ with smooth boundary $\partial\Omega = \Gamma_D \dot{\cup} \Gamma_N$. One seeks to solve

$$
\begin{aligned}
\partial_t u - \operatorname{div}(\kappa \nabla u) &= f & t > 0,\ \boldsymbol{x} \in \Omega, \\
(\kappa \nabla u) \cdot \mathbf{n} &= g & t > 0,\ \boldsymbol{x} \in \Gamma_N, \\
u &= u_D & t > 0,\ \boldsymbol{x} \in \Gamma_D, \\
u(\boldsymbol{x}, 0) &= u_0(\boldsymbol{x}) & \boldsymbol{x} \in \Omega
\end{aligned}
$$

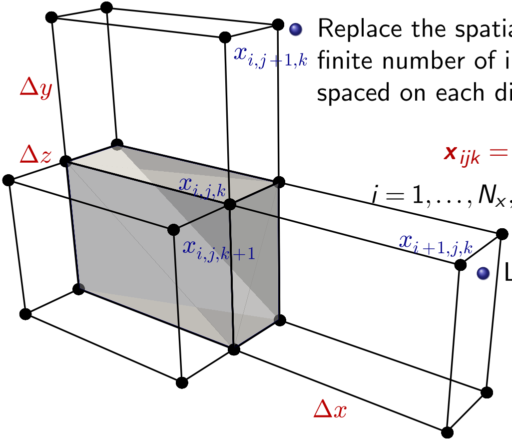
where

- $u = u(\boldsymbol{x}, t)$: temperature field (scalar) at given space-time
- $\kappa = \kappa(\boldsymbol{x}, t)$: heat capacity (known)
- $f = f(\boldsymbol{x}, t)$: internal heat generation (known)
- $g\ ds$: prescribed heat flux through $ds \subset \Gamma_N$
- u_D: prescribed temperature on Γ_D



The finite difference (FD) method

Philosophy



- Replace the spatial domain with a mesh of a finite number of interior nodes (assumed unif. spaced on each direction)

$$\boldsymbol{x}_{ijk} = (i\Delta x, j\Delta y, k\Delta z),$$

$$i = 1, \ldots, N_x, j = 1, \ldots, N_y, k = 1, \ldots, N_z$$

- Likewise for the time domain:

$$t^n = n\Delta t, \quad n = 0, \ldots, N_t$$

- Properly define continuous functions and variables on the grid, e.g.

$$u(\boldsymbol{x}_{ijk}, t^n), \text{ to be approximated by } u^n_{ijk}$$

The heat equation in multi-D

A finite difference discretization

Many alternatives available, depending on how we approximate $\partial_t u$, $\operatorname{div}(\kappa \nabla u)$. We list two (and restrict ourselves to constant κ and $\partial\Omega = \Gamma_D$)

- Explicit forward Euler scheme (forward D in time, centered D in space):

$$\frac{u_{i,j,k}^{n+1} - u_{i,j,k}^{n}}{\Delta t} - \kappa \left(\frac{u_{i-1,j,k}^{n} - 2u_{i,j,k}^{n} + u_{i+1,j,k}^{n}}{(\Delta x)^2} + \frac{u_{i,j-1,k}^{n} - 2u_{i,j,k}^{n} + u_{i,j+1,k}^{n}}{(\Delta y)^2} \right.$$
$$\left. + \frac{u_{i,j,k-1}^{n} - 2u_{i,j,k}^{n} + u_{i,j,k+1}^{n}}{(\Delta z)^2} \right) = f_{i,j,k}^{n},$$

  for $n = 1, \ldots$ and for all $2 \leqslant i \leqslant m_x$, $2 \leqslant j \leqslant m_y$, $2 \leqslant k \leqslant m_z$ (interior)

- $u_{i,j,k}^{0} = u_0(x_{i,j,k})$
- $u_{1,j,k}^{n} = u_D(x_{1,j,k})$, $n = 0, \ldots$. Analogously for the remaining borders
- Only function evaluations are incurred at each time step

The heat equation

A finite difference discretization

- Implicit backward Euler scheme (backward D in time, centered D in space):

$$\frac{u_{i,j,k}^{n+1}-u_{i,j,k}^{n}}{\Delta t}-\kappa\left(\frac{u_{i-1,j,k}^{n+1}-2u_{i,j,k}^{n+1}+u_{i+1,j,k}^{n+1}}{(\Delta x)^2}+\frac{u_{i,j-1,k}^{n+1}-2u_{i,j,k}^{n+1}+u_{i,j+1,k}^{n+1}}{(\Delta y)^2}\right.$$
$$\left.+\frac{u_{i,j,k-1}^{n+1}-2u_{i,j,k}^{n+1}+u_{i,j,k+1}^{n+1}}{(\Delta z)^2}\right)=f_{i,j,k}^{n+1},$$

  for $n=1,\ldots$ and for all $2\leqslant i\leqslant m_x$, $2\leqslant j\leqslant m_y$, $2\leqslant k\leqslant m_z$ (interior)

- $u_{i,j,k}^{0}=u_0(x_{i,j,k})$
- $u_{1,j,k}^{n}=u_D(x_{1,j,k})$, $n=0,\ldots$. Analogously for the remaining borders
- The solution of a system is required at each time step

The heat equation

A finite difference discretization – Doing the actual solving in 3-D

- A node located at (x_i, y_j, z_k) will be stored in the position s, in U

```
s = (k - 1)*mx*my + (j - 1)*mx + i ;
```

- Backward Euler on uniform grid: $(\mathbb{I} - \kappa\frac{\Delta t}{h^2}\mathbb{A})U^{n+1} = U^n + F^{n+1}$
- $D_\ell \in \mathbb{R}^{m-1\times m-1}$, $E_\ell \in \mathbb{R}^{(m-1)^2\times(m-1)^2}$, $G_\ell \in \mathbb{R}^{(m-1)^3\times(m-1)^3}$, $I \in \mathbb{R}^{m-1\times m-1}$, $J \in \mathbb{R}^{(m-1)^2\times(m-1)^2}$
- $D_\ell = \begin{pmatrix} -\ell & 1 & 0 & 0 & \cdots & 0 \\ 1 & -\ell & 1 & 0 & \cdots & 0 \\ & & \ddots & & & \\ 0 & 0 & 0 & 1 & -\ell & 1 \end{pmatrix}$, $E_\ell = \begin{pmatrix} D_\ell & I & 0 & 0 & \cdots & 0 \\ I & D_\ell & I & 0 & \cdots & 0 \\ & & \ddots & & & \\ 0 & 0 & 0 & I & D_\ell & I \end{pmatrix}$

$G_\ell = \begin{pmatrix} E_\ell & J & 0 & 0 & \cdots & 0 \\ J & E_\ell & J & 0 & \cdots & 0 \\ & & \ddots & & & \\ 0 & 0 & 0 & J & E_\ell & J \end{pmatrix}$. For 1-D: $\mathbb{A} = D_2$, for 2-D: $\mathbb{A} = E_4$, for 3-D: $\mathbb{A} = G_6$

- Reshape the solution vector into a 3-D grid

- Using Kronecker products

$$\mathtt{kron}(A,B) = A \otimes B = \begin{pmatrix} a_{2,2}B & \dots & a_{2,m}B \\ \vdots & & \vdots \\ a_{m,2}B & \dots & a_{m,m}B \end{pmatrix}$$

- Therefore $\quad \mathtt{kron}(\mathtt{eye}(\mathtt{m}-1), D_\ell)\underline{u} = \begin{pmatrix} D_\ell & & \\ & \ddots & \\ & & D_\ell \end{pmatrix} \begin{pmatrix} u_{:,2} \\ \vdots \\ u_{:,m} \end{pmatrix}$

- Good. Now we need something to put "I-blocks" on the sub-diagonals. $\mathtt{kron}(K, \mathtt{eye}(\mathtt{m}-1))$, with K being the sub-diagonals does the trick

- Then, the kronecker sum

$$D_\ell \oplus K := \mathtt{kron}(K, \mathtt{eye}(\mathtt{m}-1)) + \mathtt{kron}(\mathtt{eye}(\mathtt{m}-1), D_\ell)$$

represents our discrete (plus) Laplace operator

Let's try this:

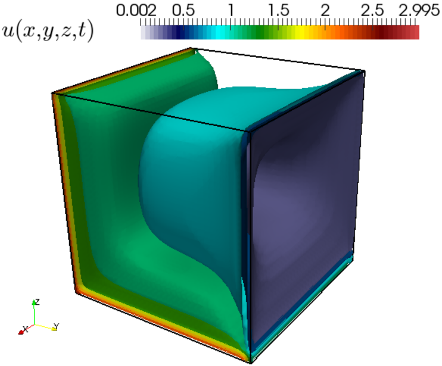
- $\Omega = (0,1)^2$, $\kappa = 1$, $f = \pi^2(x^2+y^2)\sin(\pi xy)$
- $u = \sin(\pi xy)$ on $\partial\Omega$
- m internal points on each direction
- available from
  `http://people.maths.ox.ac.uk/nakatsukasa/InFoMM/lapl_df.m`
- play with it! (change BCs, compute error history, add time-dependence, L-shaped domain, etc)

The heat equation

Numerical example

Another one:

- $\Omega = (0,1)^3$, $\kappa =$1e-3, $f = 0$, $\partial\Omega = \Gamma_D$
- $u_D = \begin{cases} 1 & \text{on back, front and top} \\ 3 & \text{on left and down sides} \\ 0 & \text{on the right lid} \end{cases}$
- Equidistant grid points
  $h = \Delta x = \Delta y = \Delta z = \frac{1}{16}$, $\Delta t = \frac{h^2}{6\kappa}$
- Explicit forward Euler method
- $u_0 = 0$, run until $t = 50$



How far can you go with mesh refinement before burning your RAM?