SF2520 — Applied numerical methods

Lecture 2

Numerical methods for ODE
Error analysis, Adaptivity

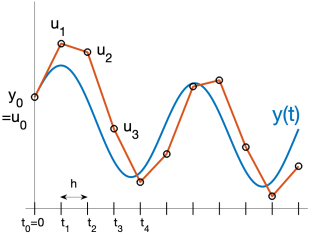
Olof Runborg
Numerical analysis
Department of Mathematics, KTH

2023-08-31

Today's lecture

- Numerical methods for ODEs
  - Local truncation error
  - Error analysis one-step methods
  - Adaptive methods

Numerical methods for ODE



Introduce discrete points in time,

$$t_n = nh, \quad h \ll 1,$$

where h is a (small) time step, and approximations

$$u_n \approx y(t_n).$$

Different time stepping methods for $y' = f(t, y)$,

1. **Explicit Euler**: $u_{n+1} = u_n + hf(t_n, u_n)$,
2. **Implicit Euler**: $u_{n+1} = u_n + hf(t_{n+1}, u_{n+1})$,
3. **Trapezoidal method**: $u_{n+1} = u_n + \frac{1}{2}h\Big(f(t_n, u_n) + f(t_{n+1}, u_{n+1})\Big)$
4. **Heun's method**: $u_{n+1} = u_n + \frac{1}{2}h\Big(f(t_n, u_n) + f(t_{n+1}, u_n + hf(t_n, u_n))\Big)$
5. **Midpoint method**: $u_{n+1} = u_{n-1} + 2hf(t_n, u_n)$,

- Methods (1,4,5) explicit and (2,3) implicit.
- (1,2,3,4) one-step methods and (5) a multistep method.

ODE – implicit methods

- In an implicit method u_{n+1} is an argument of f, and an equation (in general nonlinear) must be solved in each step.
- **Example:** Implicit Euler for linear system $\boldsymbol{y}' = A\boldsymbol{y} + \boldsymbol{g}(t)$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + h\Big[A\boldsymbol{u}_{n+1} + \boldsymbol{g}(t_{n+1})\Big] \quad \Rightarrow \quad (I - hA)\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + h\boldsymbol{g}(t_{n+1})$$

Solve linear system of eqs. in each step. (I is the identity matrix.)

- **Example:** Implicit Euler for general nonlinear ODE $\boldsymbol{y}' = \boldsymbol{F}(t, \boldsymbol{y})$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + h\boldsymbol{F}(t_{n+1}, \boldsymbol{u}_{n+1})$$

Solve $G(\boldsymbol{u}) = 0$ where

$$G(\boldsymbol{u}) = \boldsymbol{u} - h\boldsymbol{F}(t_{n+1}, \boldsymbol{u}) - \boldsymbol{u}_n.$$

Set $\boldsymbol{u}_{n+1} = \boldsymbol{u}^*$ = solution.

- Use e.g. Newton's method to solve $G(\boldsymbol{u}) = 0$, with start value $\boldsymbol{u}_n$.

Gives two levels of iterations: time stepping (outer) and Newton (inner).

One-step methods

The general form of a one-step method is

$$u_{n+1} = u_n + h\phi(h, t_n, u_n, u_{n+1}), \qquad u_0 = y_0,$$

where ϕ depends on f.

- If ϕ does not depend on u_{n+1} the method is explicit.
- Examples:
  - **Explicit Euler**:
  $$\phi(h, t_n, u_n, u_{n+1}) = f(t_n, u_n)$$
  - **Implicit Euler**:
  $$\phi(h, t_n, u_n, u_{n+1}) = f(t_n + h, u_{n+1})$$
  - **Heun's method**:
  $$\phi(h, t_n, u_n, u_{n+1}) = \frac{1}{2}\Big(f(t_n, u_n) + f(t_n + h, u_n + hf(t_n, u_n))\Big)$$

Numerical errors

- Introduce the (global) error e_n in step n

$$e_n := u_n - y(t_n).$$

- For convergence we want $\lim_{h\to 0} e_n \to 0$. (Note: e_n, u_n and t_n depend on h.)
- More precisely: Consider the solution in a fixed interval $t \in [0, T]$. If we use $h = T/N$, i.e. N time steps, then we want to bound

$$\text{maximum global error} = \max_{0\le n\le N} |e_n| \le Ch^p,$$

  such that C does not depend on h and $p \ge 1$.

  (Note: N, e_n and t_n depend on h.)
- When this holds the method has order of accuracy p. Higher p means faster convergence.

  (Order of accuracy is a central general concept. See notes if you need to catch up on this.)
- Error estimates typically also hold pointwise for the methods:

$$e_n \approx C(t_n)h^p,$$

  where C depends on the (fixed) time $t_n = nh$.

Numerical errors – examples

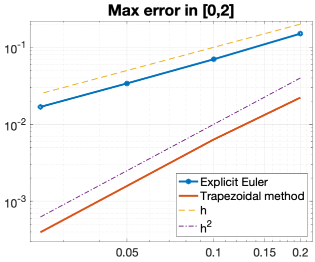
Example

We solve the ODE

$$y' = \sin(y) - y^2 + \cos(2\pi t), \qquad y(0) = 1,$$

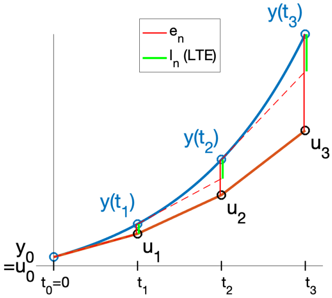
in the interval $t \in [0, 2]$ using Explicit Euler and the Trapezoidal method with time steps $h = 0.2, 0.1, 0.05, 0.025$, i.e. N=10, 20, 40, 80.

- Matlab examples.
- Empirically we get order of accuracy $p = 1$ for Explicit Euler and $p = 2$ for the Trapezoidal method.



Error analysis

- Consider the result of a numerical ODE method.
- Want to analyze how the global error $e_n = u_n - y(t_n)$ depends on h for $0 \le t_n \le T$.
- Each step produces new errors which accumulate. In general e_n increases with n.



- Define the local truncation error (LTE) ℓ_n as the residual when the exact solution is entered into the method. For one-step methods:

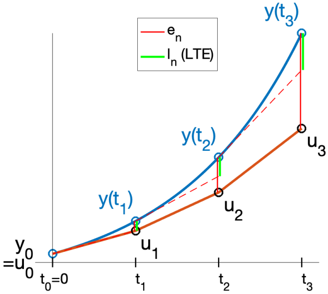
$$y(t_{n+1}) = \underbrace{y(t_n) + h\phi\Big(h, t_n, y(t_n), y(t_{n+1})\Big)}_{\text{Method with exact solution}} + \underbrace{\ell_{n+1}}_{\text{LTE}}$$

- LTE approximates the new error made in a step.
- Note: $e_0 = 0$ and $\ell_1 = e_1$.

Local truncation error

Local truncation error (LTE) $\ell_n(h)$ defined as

$$y(t_{n+1}) = y(t_n) + h\phi\Big(h, t_n, y(t_n), y(t_{n+1})\Big) + \ell_{n+1}(h)$$



Convenient concept since:

1. Fairly straightforward to derive an expression and bound for $\ell_n(h)$, using Taylor expansion of exact solution y around $t = t_n$.
2. One can show that the sum of $\ell_n(h)$ is of the same order (in h) as global error e_n.

Local truncation error, estimate

1. Fairly straightforward to derive an expression and bound for $\ell_n(h)$, using Taylor expansion of exact solution y around $t = t_n$.

Example (Explicit Euler)

Definition

$$y(t_{n+1}) = y(t_n) + hf(t_n, y(t_n)) + \ell_{n+1}(h).$$

Since $y' = f$ for exact solution,

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \ell_{n+1}(h).$$

Hence, $\ell_{n+1}(h)$ is the remainder term in one step Taylor expansion,

$$\ell_{n+1}(h) = \frac{1}{2}h^2 y''(\xi), \qquad \xi \in (t_n, t_{n+1}).$$

Therefore, with $M := \max_{0 \le t \le T} |y''(t)|/2$ (independent of h)

$$|\ell_n(h)| \le Mh^2, \qquad 0 \le nh \le T.$$

- Similarly one can derive $|\ell_n(h)| \le M'h^3$ for the trapezoidal method.

Local to global error

2. One can show that the sum of $\ell_n(h)$ is of the same order (in h) as global error e_n.

   Implies that global error is one order less than local error.

   Intuition:

   - Suppose $|\ell_n| = O(h^{p+1})$.
   - ℓ_n is (approximately) the error in one step.
   - Then
   $$|e_n| \sim \sum_{k=0}^{n} |\ell_k| \sim \sum_{k=0}^{t_n/h} h^{p+1} \sim \frac{1}{h} h^{p+1} = O(h^p)$$
   - I.e. we take $O(1/h)$ steps where, in the worst case, $O(h^{p+1})$ errors accumulate, to $O(h^p)$.

Error estimates for one-step methods

Theorem (Global error, one-step methods)

Suppose the differential equation is approximated by the one-step method

$$u_{n+1} = u_n + h\phi(h, t_n, u_n, u_{n+1}), \qquad u_0 = y_0, \qquad 0 \le n \le N_h,$$

where $N_h h = T$. If

1. *ϕ is Lipschitz in both u_n and u_{n+1}, for $h \in [0, h_0]$ and $t_n \in [0, T]$, uniformly,*
$$|\phi(h, t_n, u_n, u_{n+1}) - \phi(h, t_n, v_n, v_{n+1})| \le L\Big(|u_n - v_n| + |u_{n+1} - v_{n+1}|\Big),$$
2. *Local truncation error satisfies*
$$\max_{0 \le n \le N_h} |\ell_n(h)| \le Mh^{p+1}, \qquad \text{(M independent of h)},$$

Then,

$$\max_{0 \le n \le N_h} |e_n| \le Ch^p, \qquad \text{(C independent of h)}.$$

Error estimates for one-step methods

- Global error is one order less than local error in h.
- LTE estimate done by Taylor expansion of y as above.
- One-step methods always convergent if they are *consistent*, i.e. when order $p \geq 1$.
- For multi-step methods this is not true. Additional stability conditions needed to ensure convergence.
- Lipschitz condition on ϕ almost always follows from requirement that f is Lipschitz (to ensure unique solutions of ODE). Recall, e.g. that

  $$\phi(h, t_n, u_n, u_{n+1}) = \frac{1}{2}\Big(f(t_n, u_n) + f(t_n + h, u_n + hf(t_n, u_n))\Big)$$

  for Heun's method. In particular, all one-step methods mentioned in this course are convergent.
- One can also show that a constant C independent of h exists such that

  $$|e_n| \leq C \sum_{k=1}^{n} |\ell_k(h)|.$$

Error estimates for one-step methods

Notes: Proof of theorem for Explicit Euler, i.e. the case $\phi = f$.

- We showed
$$|e_n| \leq Ch, \qquad C = MTe^{LT}.$$
- C is an increasing function of L, M, T.
- As a intermediate step we also showed
$$|e_n| \leq e^{LT} \sum_{k=1}^{n} |\ell_k|.$$
- Here e^{LT} is typically large. However, estimate is pessimistic and not sharp for stable ODEs, and absolutely stable schemes.

Error estimates for stable one-step methods

Better estimates when the ODE is stable, e.g. the scalar ODE

$$\frac{dy}{dt} = f(y), \qquad \frac{\partial f}{\partial y} < 0,$$

or the system

$$\frac{d\boldsymbol{y}}{dt} = A\boldsymbol{y} + \boldsymbol{g}(t), \qquad \text{Real}(\lambda_j) < 0 \text{ for all eigenvalues } \lambda_j \text{ of } A.$$

- Then, if the scheme is *absolutely stable* the constant e^{LT} replaced by 1,

$$|e_n| \leq \sum_{k=1}^{n} |\ell_k|, \qquad \text{(sum of local errors bounds the global error).}$$

- Explicit schemes typically require $h < h_{\text{stab}}$ for some *stability limit* h_{stab}.

Example: $y' = -\lambda y + g$ for $\lambda > 0$

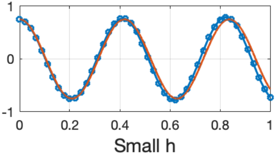
In this case $f_y = -\lambda < 0$ and if $h\lambda < 1$,

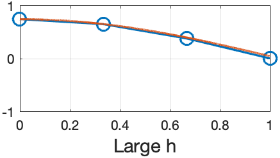
$$|e_n + h[f(u_n) - f(y(t_n))]| = |(1 - h\lambda)e_n| = (1 + hL)|e_n|, \quad L = -\lambda,$$

and effectively, $L < 0$, so $e^{LT} \leq 1$, $\rho < 1$ and $|e_n| \leq \sum_{k=1}^{n} |\ell_k|$. (In fact $h\lambda < 2$ is enough!)

Choosing time step h

- Time step must resolve variations in the solution





- Well resolved $\approx$ small local truncation error (LTE). Ex. (Explicit Euler):

$$\ell_n(h) \approx \frac{h^2}{2} y''(t_n) \quad \Rightarrow \quad h \text{ must be small if } y''(t) \text{ is large.}$$

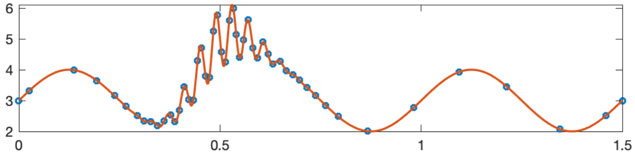
- Small LTE gives small global error for stable schemes, since

$$|e_n| \leq \sum_{k=1}^{n} |\ell_k(h)| \qquad (\ell_k(h) \text{ depends on } h \text{ and exact solution } y)$$

- Global error $\leq$ TOL, will require $h \leq h_{\mathrm{acc}}$ for some h_{acc}, which in addition to TOL, (only) depends on exact solution (via the LTE).
- However, we also need stability, $h < h_{\mathrm{stab}}$. A difficulty for stiff problems, where $h_{\mathrm{stab}} \ll h_{\mathrm{acc}}$. (More later!)

Adaptive methods

- So far we have used a constant time step h. Not always efficient.
- Consider the following solution $y(t)$:



- The fast variations in the middle would require us to use a small h throughout the computation if we had a constant h.
- Increases computational costs, without reducing the error (much).
- In adaptive methods the time step h is changed continuously based on the solution itself, to optimize the cost to achieve a preset error tolerance.
- Much fewer steps and less expensive.

Adaptive methods, general strategy

Let the time step in step n be h_n. General strategy is based on the estimate

$$|e_n| \leq \sum_{k=1}^{n} |\ell_k(h_k)|,$$

i.e. the sum of local errors gives a bound for the global error.

1. Decide on a tolerance TOL for the global error e_n.
2. In each step, estimate local error $\ell_n(h_n)$ for the current h_n.
3. – Decrease h_n if $|\ell_n(h_n)| > TOL \cdot h_n/T$ (and redo the step).

   – Increase h_n if $|\ell_n(h_n)| \leq TOL \cdot h_n/T$. (Here T is final time).

   The goal is to choose h_n such that $|\ell_n(h_n)| \approx TOL \cdot h_n/T$ in each step. Then

$$|e_n| \leq \sum_{k=0}^{n} |\ell_k(h_k)| \approx \frac{TOL}{T} \sum_{k=0}^{n} h_k = \frac{TOL}{T} t_n \leq TOL.$$

- Many methods leave out the T dependence.
- Some methods simply keep local error constant, $|\ell_n| \approx TOL$.

Estimating the local error

2. In each step, estimate local error ℓ_n for the current h_n.

Strategies for estimating LTE:

(1) Compute approximation if $y(t_{n+1})$ using h and $h/2$ (two steps). The difference approximates the local error:

$$|\ell_{n+1}| \sim |u_{n+1,h} - u_{n+1,h/2}|.$$

(Typically requires two function evaluations since two steps.)

(2) Compute approximation of $y(t_{n+1})$ using two different methods with different orders of accuracy p and q. The difference approximates the local error:

$$|\ell_{n+1}| \sim |u_{n+1,p} - u_{n+1,q}|.$$

(Can be done with few extra function evaluations.)

Note: In practice the most accuracte computed value is always used, eventhough the error estimate is done for the least accurate value.

Adjusting the time step

3. – Decrease h_n if $|\ell_n(h_n)| > TOL \cdot h_n/T$ (and redo the step).
   – Increase h_n if $|\ell_n(h_n)| \leq TOL \cdot h_n/T$.

Strategies for decreasing/increasing $h_n \to \tilde{h}_n$

(1) Halve/double h_n. I.e. $\tilde{h}_n = h_n/2$ or $\tilde{h}_n = 2h_n$.

(2) Exploit the fact that order of accuracy p is known and that $\ell_n(h) \approx c_n h^{p+1}$. Choose $\tilde{h}_n$ such that

$$TOL \cdot \tilde{h}_n/T = \ell_n(\tilde{h}_n) \approx \underbrace{\frac{\ell_n(h_n)}{h_n^{p+1}}}_{\approx c_n} \tilde{h}_n^{p+1} \quad \Rightarrow \quad \tilde{h}_n = \left(\frac{TOL \cdot h_n^{p+1}}{\ell_n(h_n)T} \right)^{\frac{1}{p}}.$$

(Typically some added constraints on min/max $\tilde{h}_n$ and max change of h_n also included.)

(3) More advanced methods based on control theory.

Matlab

Matlab for systems of equations $\boldsymbol{y}' = \boldsymbol{F}(t, \boldsymbol{y})$)

```
>> [t,Y] = ode45(F, [0 T], Y0);
```

Arguments and output:

- `t=[t0,t1,..., tN]`^{T}
  column vector with discrete time points,
- `Y`
  matrix containing solution
  Component p at time `t(n)` is `Y(n,p)`.
  Alternatively: $\boldsymbol{u}_{n-1}$ is the row `Y(n,:)`.
- `F`
  ODE right hand side $\boldsymbol{F}(t, \boldsymbol{y})$,
  Matlab-function which returns a column vector
- `[0 T]` — time interval
- `Y0` — initial data $\boldsymbol{y}_0$, a column vector

Matlab, cont.

- Other ODE solvers include:
  - `ode23`
  - `ode23s` (for stiff ODE)
  - `ode113` (high order multi-step method)
- Matlab ODE solvers (including `ode45`) are adaptive. To control the error one specifies an absolute and a relative tolerance `AbsTol` and `RelTol`. (I.e., not a step size or the number of steps.)
- The solvers try to reduce the error below `max(AbsTol,RelTol*norm(Y))` if `Y` is the solution.
- Default tolerances are
  - `RelTol` = 10^{-3}
  - `AbsTol` = 10^{-6}
- To adjust the tolerances, use the `odeset` command as follows:

```
>> options = odeset('RelTol',1e-5,'AbsTol',1e-8);
>> [t, Y] = ode45(F, [0 T], Y0, options);
```

Absolute stability

Example

Want to approximate solution to

$$y' = -25y, \qquad y(0) = 1.$$

Exact solution is $y(t) = e^{-25t}$.

- Computer tests: Explicit and Implicit Euler.
- Results:
  - Explicit Euler useless for fixed $h = 0.1$. (Error $\to \infty$ when $n \to \infty$.)
  - Implicit Euler gives an ok solution for the same $h = 0.1$.
  - Both methods are convergent as $h \to 0$.
  - Explicit Euler $\approx$ Implicit Euler for $h = 0.01$.
- Need to distinguish this "good" and "bad" behaviour of convergent methods $\Rightarrow$ absolute stability concept.

Absolute stability and explicit/implicit methods

- **Explicit methods** (Expl. Euler, Heun, ...)
  Stability limit for time step $h \leq h_{\text{stab}}$. Unstable for $h > h_{\text{stab}}$, where h_{stab} depends on both method and problem.
- Computer example.
- **Implicit Methods** (Impl. Euler, Trapezoidal method, ...)
  - No stability limit. Stable for all time steps $h > 0$.
  - More expensive time stepping. In every step an equation must be solved in general, often numerically.
- **Stiff problems**
  - Stability limit $\ll$ accuracy requirement, i.e. $h_{\text{stab}} \ll h_{\text{acc}}$.
  - Explicit methods require excessively small h. Implicit methods accurate enough also for $h \gg h_{\text{stab}}$. Conditions are:

    $$\text{explicit method: } h \leq h_{\text{stab}} \ll h_{\text{acc}}, \qquad \text{implicit method: } h \leq h_{\text{acc}}.$$

  - Implicit methods better: more expensive per time step, but can use fewer steps.
  - Adaptive explicit methods do not work well.
- Computer example.