



TMA4125 Matematikk 4N

Numerics for IVPs – Stepsize & Stability

Ronny Bergmann

Department of Mathematical Sciences, NTNU.

March 30, 2023

Notation & Summary from last week

Notation / Definitions.

- ▶ the ODE: $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$
- ▶ the exact solution through (t^*, \mathbf{y}^*) : $\mathbf{y}(t, t^*, \mathbf{y}^*)$
- ▶ the exact solution of the ODE with initial values t_0, \mathbf{y}_0 :

$$\mathbf{y}(t) = \mathbf{y}(t; t_0, \mathbf{y}_0)$$
- ▶ one step of the (explicit) method (increment function Φ):

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\Phi(t_n, \mathbf{y}_n, h)$$

Error Concepts.

Let Φ represent a method of order p and $\hat{\Phi}$ a method of order $p + 1$.

- ▶ local truncation error (LTE)

(comparing exact solution to one step "from there"):

$$\mathbf{d}_{n+1} = \mathbf{y}(t_n + h; t_n, \mathbf{y}(t_n)) - (\mathbf{y}(t_n) + h\hat{\Phi}(t_n, \mathbf{y}(t_n), h))$$
- ▶ local error: $\mathbf{l}_{n+1} = \mathbf{y}(t_n + h; t_n, \mathbf{y}_n) - (\mathbf{y}_n + h\Phi(t_n, \mathbf{y}_n, h))$
- ▶ the global error: $\mathbf{e}_n = \mathbf{y}(t_n) - \mathbf{y}_n$



NTNU

Norwegian University of Science and Technology

Error Estimates & Adaptivity

Motivation

In our numerical tests we saw that the step size h has to be chosen

- ▶ if it is too large \Rightarrow too inexact results in \mathbf{y}_n
- ▶ if it is too small \Rightarrow too much computational time or memory usage

Approaches.

1. control the **global error** $\mathbf{e}_n = \mathbf{y}(t_n) - \mathbf{y}_n$
this is quite difficult and beyond the scope of this lecture
2. control the **local error** $\mathbf{l}_{n+1} := \mathbf{y}(t_{n+1}; t_n, \mathbf{y}_n) - \mathbf{y}_{n+1}$
where
 - ▶ we “start from” (t_n, \mathbf{y}_n) : We consider the exact solution $\mathbf{y}(t; t_n, \mathbf{y}_n)$ but running through (t_n, \mathbf{y}_n) .
 - ▶ compared to \mathbf{d}_{n+1} we have hence a “different starting point” \mathbf{y}_n and not $\mathbf{y}(t_n)$!

Approximating the local error

Now since we do not have the actual solution $\mathbf{y}(t)$, let's take two methods, where one is more exact. For both we start from our previous point (t_n, \mathbf{y}_n) :

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\Phi(t_n, \mathbf{y}_n, h)$$

$$\hat{\mathbf{y}}_{n+1} = \mathbf{y}_n + h\hat{\Phi}(t_n, \mathbf{y}_n, h)$$

where

- ▶ Φ is an increment function for a method of order p
- ▶ $\hat{\Phi}$ is an increment function for a method of order $p + 1$

Approach. We compare the two [convergence order](#), or more precisely their [power serieses](#) in h .

Goal. Approximate the local error

$$\mathbf{l}_{n+1} = \mathbf{y}(t_n + h; t_n, \mathbf{y}_n) - (\mathbf{y}_n + h\Phi(t_n, \mathbf{y}_n, h)) \approx \hat{\mathbf{y}}_{n+1} - \mathbf{y}_{n+1} = \mathbf{l}_{e_{n+1}}$$

Comparing local error terms

Advantage. Given the two numerical schemes, we can compute \mathbf{le}_{n+1} .

But even more. Since we know their orders p and $p + 1$ we can estimate how their errors behave

$$\begin{aligned}\mathbf{y}(t_{n+1}; t_n, \mathbf{y}_n) - \mathbf{y}_{n+1} &= \Psi(t_n, \mathbf{y}_n)h^{p+1} + \dots \text{higher order terms in } h \\ \mathbf{y}(t_{n+1}; t_n, \mathbf{y}_n) - \hat{\mathbf{y}}_{n+1} &= \dots \text{only higher order terms in } h\end{aligned}$$

where $\Psi(t_n, \mathbf{y}_n)h^{p+1}$ is the **principal error term** that dominates the first error.

⇒ From these error terms we get that the difference

$$\mathbf{le}_{n+1} = \hat{\mathbf{y}}_{n+1} - \mathbf{y}_{n+1} = \Psi(t_n, \mathbf{y}_n)h^{p+1} + \dots \text{higher order terms in } h$$

which therefore deserves the name **local error estimate**.

Step size Control

Goal. based on $\|\mathbf{le}_{n+1}\|$ we want to adopt the step size h_n in every step.

Approach. Given a tolerance T_{o1}
starting from some t_n, \mathbf{y}_n and a (current) step size h_n

1. Perform a step with your method(s) of choice
That is: Compute \mathbf{y}_{n+1} and $\hat{\mathbf{y}}_{n+1}$
 \Rightarrow find the error estimate \mathbf{le}_{n+1}
2. If $\|\mathbf{le}_{n+1}\| < T_{o1}$
Then accept the step and continue with $t_{n+1} = t_n + h_n, \mathbf{y}_{n+1}$
Maybe / If possible Increase step size for next step.
3. If we are not below T_{o1}
Reduce the step h_n and repeat the step

Question. How to determine the new (increased/reduced) h_n ?

Obtaining a new Stepsize h_n

We know $\|\mathbf{le}_{n+1}\| \approx Dh_n^{p+1}$

...ok we do not know the t D , but let's assume that is a quantity that stays constant in our small steps

We want That the error is approximately $\text{To1} \approx Dh_{\text{new}}^{p+1}$

$$\frac{\text{To1}}{\|\mathbf{le}_{n+1}\|} \approx \left(\frac{h_{\text{new}}}{h_n}\right)^{p+1} \Rightarrow h_{\text{new}} \approx \left(\frac{\text{To1}}{\|\mathbf{le}_{n+1}\|}\right)^{\frac{1}{p+1}} h_n$$

To avoid too many rejection steps, we stay (a bit) pessimistic

$$h_{\text{new}} = P \left(\frac{\text{To1}}{\|\mathbf{le}_{n+1}\|}\right)^{\frac{1}{p+1}} h_n \quad \text{with pessimist factor } P < 1.$$

usually one chooses $\frac{1}{2} \leq P \leq 0.95$.

More general – Embedded Runge-Kutta

Runge-Kutta methods with an error estimate are called **embedded Runge-Kutta methods** or **Runge-Kutta pairs**. They are written as

c_1	a_{11}	a_{12}	\cdots	a_{1s}	
c_2	a_{21}	a_{22}	\cdots	a_{2s}	
\vdots	\vdots	\vdots	\cdots	\vdots	
c_s	a_{s1}	a_{s2}	\cdots	a_{ss}	
	b_1	b_2	\cdots	b_s	(order p)
	\hat{b}_1	\hat{b}_2	\cdots	\hat{b}_s	(order $p + 1$)

Where we can write the local error estimate as

$$\mathbf{le}_{n+1} = h \sum_{i=1}^s (\hat{b}_i - b_i) \mathbf{k}_i$$

Practical notes on the Implementation

1. Even if we use the lower order method to derive the error estimate (that is $le_n + 1$)
We know that the higher order method is more exact.
 \Rightarrow it's common to take \hat{y}_{n+1} as the next step, called local extrapolation.
2. We might only end close to t_{end}
 \Rightarrow take that close by value as adjusted t_{end}
3. To avoid infinite loops (with arbitrary small step sizes)
 \Rightarrow introduce further stopping criteria, e.g. a maximal number of steps
(rejected, accepted or both)

Stiff ODEs – An example

Let's consider the following system of 2 ODEs

$$\begin{aligned}y_1' &= -2y_1 + y_2 + 2\sin(t), \\y_2' &= (a - 1)y_1 - ay_2 + a(\cos(t) - \sin(t)),\end{aligned}$$

with initial values $y_1(0) = 2$, $y_2(0) = 3$ and some parameter $a > 0$. Closed form solution is even independent of a

$$y_1(t) = 2e^{-t} + \sin(t), \quad y_2(t) = 2e^{-t} + \cos(t).$$

For large a this yields a **Stiff ODE**, where

- ▶ we have a slow time scale dominating the evolution
 - ▶ a fast time scale with small perturbation
- ⇒ We have to choose a very small step size in **explicit adaptive methods** like Euler-Heun.

Numerical Stability

Linear Stability Analysis

We can study this phenomenon we consider the [test equation](#)

$$y' = \lambda y, \quad y(0) = y_0$$

where $\lambda \in \mathbb{C}$ satisfies $\Re\lambda < 0$ (the real part is negative.)

The true solution reads $y(t) = e^{\lambda t} y_0$.

Writing $\lambda = a + ib$, we have $a < 0$ and our solution becomes

$$y(t) = e^{\lambda t} = e^{ibt} e^{at} = (\cos(bt) + i \sin bt) e^{at}.$$

Since $a < 0$ we obtain $\lim_{t \rightarrow \infty} y(t) = 0$.

Question. Does our numerical method match this behaviour?
And if so – for which [step sizes](#) h ?

Example. Euler's method for the test equation.

The Stability Function

In general we aim to write a numerical method — e.g. some of the Runge-Kutta methods — as

$$y_{n+1} = R(z)y_n, \quad z = \lambda h$$

Where $R(z)$ is called the **stability function**.

The stability function $R: \mathbb{C} \rightarrow \mathbb{C}$ is either a polynomial or a rational function of z .

Example 1. Form before. For **Euler's** method we have

$$y_{n+1} = y_n + \lambda h y_n = (1 + \lambda h)y_n \quad \text{i.e. the stability function is} \quad R(z) = 1 + z$$

Example 2. For Heun's method we obtain $R(z) = 1 + 2 + \frac{z^2}{2}$

Example 3. For the trapezoidal rule we obtain $R(z) = \frac{1 + \frac{z}{2}}{1 - \frac{z}{2}}$

Stability Region

From our definition

$$y_{n+1} = R(z)y_n \quad \text{we get} \quad y_n = (R(z))^n y_0$$

- ▶ $|R(z)| < 1$ implies $y_n \rightarrow 0$ for $n \rightarrow \infty$
- ▶ $|R(z)| = 1$ implies $|y_{n+1}| = |y_n|$
- ▶ $|R(z)| > 1$ implies $y_n \rightarrow \infty$ for $n \rightarrow \infty$

This motivates to define the **stability region** of a method

$$\mathcal{S} := \{z \in \mathbb{C} : |R(z)| < 1\}$$

\Rightarrow to have a **stable algorithm**, we have to choose h such that $\lambda h \in \mathcal{S}$.

A-Stability and bad news

Goal. Our method should “work for all step sizes h ”

$\Rightarrow h\lambda \in \mathcal{S}$ for all h !

Since $\Re\lambda < 0$ we define:

A method is called **absolutely stable** or **A-stable** if

$$\mathcal{S} \supset \mathbb{C}^- := \{z \in \mathbb{C} : \Re z < 0\}.$$

The bad news. explicit methods can never be A-stable.

To Implicit Methods then: Implicit Euler

the implicit or backward Euler method reads

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$$

Challenge.

We have to solve a (possibly nonlinear) equation in every step!

Stability function.

$$R(z) = \frac{1}{1-z}$$

We observe for $R(\lambda h)$ that

- ▶ $h > 0$ is our step size
- ▶ $\Re \lambda < 0$ implies that **always** $\Re(1 - \lambda h) > 1$

$$\Rightarrow \left| \frac{1}{1 - \lambda h} \right| < 1 \Rightarrow \text{Implicit Euler is A-stable!}$$

While we have to discuss how to perform a step (in general), if we have that, this method works **for any step size h** !

Solving Linear Systems of ODEs

Assume we are given a system of m differential equations of the form

$$\mathbf{y}'(t) = A\mathbf{y}(t) + \mathbf{g}(t)$$

and we assume we can **diagonalize** $A \in \mathbb{C}^{m \times m}$, i.e. we can write

$$AV = V\Lambda, \quad V = [\mathbf{v}_1, \dots, \mathbf{v}_m] \text{ (invertible)}, \quad \Lambda = \text{diag}(\lambda_1, \dots, \lambda_m)$$

where $\lambda_i, \mathbf{v}_i, i = 1, \dots, m$ are the Eigenvalues and -vectors
 Rewrite the system of ODEs to

$$V^{-1}\mathbf{y}'(t) = V^{-1}AVV^{-1}\mathbf{y}(t) + V^{-1}\mathbf{g}(t)$$

or with $\mathbf{z} = V^{-1}\mathbf{y}$ and $\mathbf{q} = V^{-1}\mathbf{g}$ even as

$$\mathbf{z}'(t) = \Lambda\mathbf{z}(t) + \mathbf{q}(t)$$

\Rightarrow these **decouple** to $z'_i(t) = \lambda_i z_i(t) + q_i(t)$ (solution cf. Math 1).

\Rightarrow back to $\mathbf{y} = V\mathbf{z}$.

Stability for Linear Systems of ODEs

To investigate the stability of

$$\mathbf{y}'(t) = A\mathbf{y}(t) + \mathbf{g}(t)$$

The Eivenvalues λ_i are crucial – analogously to λ in $y' = \lambda y$.

More general. In $\mathbf{y}' = \mathbf{f}(t, \mathbf{y}(t))$ this corresponds to Eigenvalues of the Jacobian $\mathbf{f}_{\mathbf{y}}$ of \mathbf{f} w.r.t. \mathbf{y} .

Example. From the introduction on Stiff ODEs we can write

$$\mathbf{y}' = \begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} -2y_1 + y_2 + 2 \sin(t) \\ (a-1)y_1 - ay_2 + a(\cos(t) - \sin(t)) \end{pmatrix} = A\mathbf{y} + \mathbf{g}(t),$$

with

$$A = \begin{pmatrix} -2 & 1 \\ a-1 & -a \end{pmatrix}, \quad \mathbf{g}(t) = \begin{pmatrix} 2 \sin(t) \\ a(\cos(t) - \sin(t)) \end{pmatrix}, \quad \text{and } \mathbf{y}(0) = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

\Rightarrow The eigenvalues $\lambda_1 = -1$ and $\lambda_2 = (a+1)$.

Solution and stability

General solution. From $\lambda_1 = -1$ and $\lambda_2 = -(a + 1)$ together with their Eigenvectors, we obtain

$$\mathbf{y}(t) = c_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} e^{-t} + c_2 \begin{pmatrix} -1 \\ a - 1 \end{pmatrix} + \begin{pmatrix} \sin(t) \\ \cos(t) \end{pmatrix}$$

where c_1, c_2 depend on the initial values.

For our setting $\mathbf{y}(0) = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$: $c_1 = 2, c_2 = 0$

Step size. What about the step size in Euler's method?

We need a step size h that fulfills **both**.

$$|1 + \lambda_1 h| < 1 \quad \text{and} \quad |1 + \lambda_2 h| < 1$$

even if $c_2 = 0$ in the solution, since the numerical method is based on A
 \Rightarrow We need $h < \frac{2}{1+a}$, which is what caused our "numerical trouble"!

Implementing the implicit Euler Method for Linear Systems of ODEs

With

$$\mathbf{y}'(t) = A\mathbf{y} + \mathbf{g}(t) = \mathbf{f}(t, \mathbf{y})$$

We can write down implicit Euler

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}) = \mathbf{y}_n + hA\mathbf{y}_{n+1} + h\mathbf{g}(t_{n+1})$$

Question. How can we solve this?

Rearrange (collect \mathbf{y}_{n+1} s on the right)

$$(I - hA)\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{g}(t_{n+1})$$

⇒ in every step we have to solve a **linear system of equations!**

Outlook

We can actually do the same for the [trapezoidal rule](#)

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{2} \left(\mathbf{f}(t_n, \mathbf{y}_n) + \mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}) \right)$$

and combine implicit Euler (first order) with this (second order) method to
[an adaptive implicit-Euler-Trapezoidal](#) method!

More general. For implicit Euler, if the ODE is [nonlinear](#), we have to solve

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1})$$

⇒ if we have the Jacobian \mathbf{f}_y we can solve this using [Newton's method](#)!