Theme 4: Rocket launches and initial value problems for ordinary differential equations

Martin Berggren

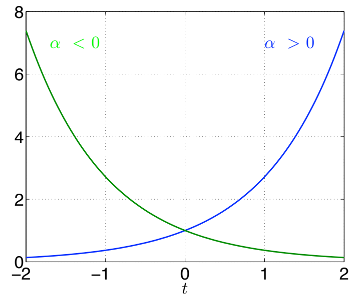
December 20, 2009

Initial value problems, examples

Example 1:

$y\colon \mathbb{R} \to \mathbb{R}$, $\alpha \in \mathbb{R}$,

$$\begin{aligned} y' &= \alpha y \qquad t > 0, \\ y(0) &= y_0 \end{aligned} \tag{1}$$



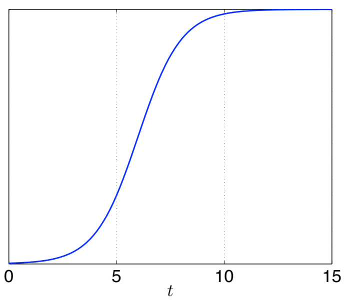
- The solution is $y(t) = e^{\alpha t} y_0$. Numerical solution not needed!
- Models e.g. microbial growth ($\alpha > 0$), radioactive radiation ($\alpha < 0$), chemical reactions

Initial value problems, examples

Example 2: More realistic microbial growth

The logistic equation (Theme 1) in continuous time:

$$\begin{aligned} y' &= \alpha \left(1 - \frac{y}{M}\right) y \qquad t > 0, \\ y(0) &= y_0 \end{aligned}$$



- The growth rate decreases as y increases
- The growth rate vanishes at $y = M$, due to nutritional depletion e.g.
- A **nonlinear** equation. "Linear", "nonlinear" refers to functions y, y' (not t e.g.). Example 1 **linear**.
- The equation can be solved "analytically" (it is separable)

Initial value problems, examples

Example 3: Population modeling in continuous time

$$\begin{cases} h' = \left[c_1 \left(1 - \dfrac{h}{M}\right) - d_1 r\right] h, & t > 0 \\ r' = (-c_2 + d_2 h)\, r, & t > 0 \end{cases}$$

$$\begin{cases} h(0) = h_0 \\ r(0) = r_0 \end{cases}$$

- h: hares. Growth rate inhibited by nutritional depletion and by being preyed on by foxes
- r: foxes. Growth rate increasing with hare population. Population shrinking by natural death
- A **system** of **nonlinear** equations
- Cannot be solved "analytically"!

Initial value problems, examples

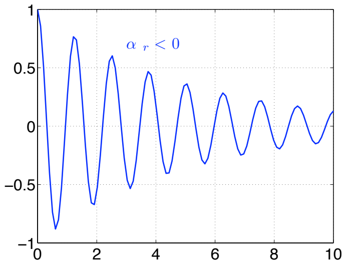
Example 4: Oscillating phenomena, modeled by equation (1), but with $\alpha \in \mathbb{C}$.

$y : \mathbb{R} \to \mathbb{R}$, $\alpha \in \mathbb{C}$,

$$\begin{aligned} y' &= \alpha y \qquad t > 0, \\ y(0) &= y_0 \end{aligned}$$

Solution:

$$\begin{aligned} y(t) &= e^{\alpha t} y_0 = e^{(\alpha_r + i\alpha_i)t} y_0 = e^{\alpha_r t} e^{i\alpha_i t} \\ &= e^{\alpha_r t} (\cos \alpha_i t + i \sin \alpha_i t) \end{aligned}$$



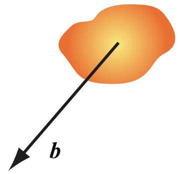
- α_r: exponential growth/decay of amplitude
- α_i: angular frequency

Initial value problems, examples

Example 5: Rigid body mechanics. Newton's second law for the center of mass:

$$\begin{aligned} mx'' &= b_x(x, y, z, x', y', z') \\ my'' &= b_y(x, y, z, x', y', z') \qquad t > 0 \\ mz'' &= b_z(x, y, z, x', y', z') \end{aligned}$$

$$x(0) = 0, \quad y(0) = 0, \quad z(0) = 0$$
$$x'(0) = 0, \quad y'(0) = 0, \quad z'(0) = 0$$



- $\boldsymbol{b} = (b_x, b_y, b_z)$ represents the forces on body (gravitation, air resistance)
- System of ODEs of **second order**
- **Nonlinear** if $\boldsymbol{b}$ depends nonlinearly on x, y, z, x', y', z'. **Linear** otherwise

Initial value problems, standard form

- Plenty of "canned" software for solving initial-value problems for ODEs
- Matlab: `ode23, ode45, ode113, ode15s, ode23s, ode23t, ode23tb`
- Need to write all problems in a uniform way to use standard software.
- The standard form for initial value problems:

$$\begin{aligned} \mathbf{u}' &= \mathbf{f}(t, \mathbf{u}) \quad t > 0 \\ \mathbf{u}(0) &= \mathbf{u}^{(0)} \end{aligned} \tag{2}$$

- *Note:* $\mathbf{u}$, $\mathbf{f}$ are **vectors**!
- $\mathbf{u} : \mathbb{R} \to \mathbb{R}^n$; a function from time into n-vectors
- $\mathbf{f} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$; a function of time and of the "state" $\mathbf{u}$ (an n-vector)
- For a **linear** ODE: $\mathbf{f} = \mathbf{A}\mathbf{u} - \mathbf{b}$, where $\mathbf{A}$ (matrix), $\mathbf{b}$ (vector) independent of $\mathbf{u}$

Initial value problems, standard form

Examples 1, 2, and 4 already in standard form.

Example 3:

$$\begin{pmatrix} h \\ r \end{pmatrix}' = \begin{pmatrix} \left[c_1\left(1 - \frac{h}{M}\right) - d_1 r\right] h \\ (-c_2 + d_2 h)\, r \end{pmatrix} \qquad t > 0$$

$$\begin{pmatrix} h(0) \\ r(0) \end{pmatrix} = \begin{pmatrix} h_0 \\ r_0 \end{pmatrix}$$

In standard form (2) for

$$\mathbf{u} = \begin{pmatrix} h \\ r \end{pmatrix}, \qquad \mathbf{f} = \begin{pmatrix} \left[c_1\left(1 - \frac{h}{M}\right) - d_1 r\right] h \\ (-c_2 + d_2 h)\, r \end{pmatrix}$$

Initial value problems, standard form

Example 5:

First, the x-component equation $mx'' = b_x$. Let $p = mx'$ (component of momentum, *rörelsemängd* in x direction). Then

$$\begin{pmatrix} x \\ p \end{pmatrix}' = \begin{pmatrix} p/m \\ b_x \end{pmatrix} = \begin{pmatrix} 0 & 1/m \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ p \end{pmatrix} + \begin{pmatrix} 0 \\ b_x \end{pmatrix}$$

For all three components:

$$\underbrace{\begin{pmatrix} x \\ y \\ z \\ p \\ q \\ r \end{pmatrix}'}_{\mathbf{u}'} = \underbrace{\begin{pmatrix} 0 & 0 & 0 & 1/m & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/m & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/m \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} x \\ y \\ z \\ p \\ q \\ r \end{pmatrix}}_{\mathbf{u}} + \underbrace{\begin{pmatrix} 0 \\ 0 \\ 0 \\ b_x \\ b_y \\ b_z \end{pmatrix}}_{\mathbf{b}}$$

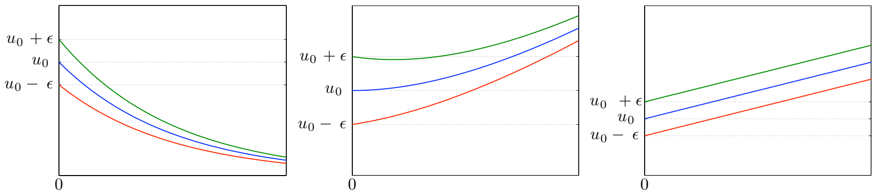
Stability with respect to initial values

Introduce **disturbance $\boldsymbol{\epsilon}$** of initial values $\mathbf{u}^{(0)}$

$$\mathbf{u}'_\epsilon = \mathbf{f}(t, \mathbf{u}_\epsilon) \qquad t > 0$$
$$\mathbf{u}_\epsilon(0) = \mathbf{u}^{(0)} + \boldsymbol{\epsilon}$$

What happens when $t \to \infty$?

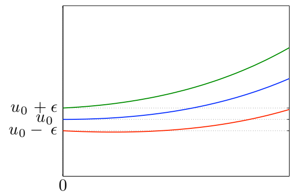
Stability with respect to initial values



(a) (b) (c)

- These are **stable** cases
- The solution curves for different initial values **do not diverge** as $t \to \infty$
- Cases (a) & (b) **asymptotically** stable (the different curves converge towards each other)
- Case (c) stable but not asymptotically stable

Stability with respect to initial values



- **Unstable** with respect to initial values: the solution curves for different initial values diverge from each other as $t \to \infty$
- Nothing "wrong" with the equation!
- Errors in indata grows as t grows
- Needs to be solved on a bounded interval $t \in [0, T]$

Stability with respect to initial values

How to quantify stability?

Start with linear, scalar equations ($\alpha \in \mathbb{C}$):

$$\begin{aligned} y' &= \alpha y + f(t) \qquad t > 0 \\ y(0) &= y_0 \end{aligned}$$

- Stable if $\operatorname{Re}\alpha \le 0$
- Asymptotically stable if $\operatorname{Re}\alpha < 0$
- Unstable if $\operatorname{Re}\alpha > 0$

Stability with respect to initial values

Linear systems of equations

$$\begin{aligned} \mathbf{u}' &= \mathbf{f}(t,\mathbf{u}) = \mathbf{A}\mathbf{u} + \mathbf{b} \qquad t > 0 \\ \mathbf{u}(0) &= \mathbf{u}^{(0)} \end{aligned} \tag{3}$$

- $\mathbf{A}$: n-by-n matrix
- Assume that $\mathbf{A}$ is **diagonalizable**: there are n linearly independent vectors $\mathbf{v}_1, \dots, \mathbf{v}_n$ (in $\mathbb{C}^n$) such that

  $$\mathbf{A}\mathbf{v}_k = \lambda_k \mathbf{v}_k,$$

  where $\lambda_1, \dots, \lambda_n \in \mathbb{C}$ are the eigenvalues of $\mathbf{A}$
- System (3) is
  - Stable if $\operatorname{Re}\lambda_k \le 0 \ \forall k$
  - Asymptotically stable if $\operatorname{Re}\lambda_k < 0 \ \forall k$
  - Unstable if there is a k such that $\operatorname{Re}\lambda_k > 0$

Stability with respect to initial values

- The stability of *linear* systems does not depend on initial data. Stability is a **system property** (depends on the real part of the eigenvalues of the system matrix)
- The concept of stability for nonlinear systems

  $$\begin{aligned} \mathbf{u}' &= \mathbf{f}(t,\mathbf{u}) \qquad t > 0 \\ \mathbf{u}(0) &= \mathbf{u}^{(0)} \end{aligned} \tag{4}$$

  more complicated.
- Look at the disturbed system

  $$\begin{aligned} \mathbf{u}_\epsilon' &= \mathbf{f}(t,\mathbf{u}_\epsilon) \qquad t > 0 \\ \mathbf{u}_\epsilon(0) &= \mathbf{u}^{(0)} + \boldsymbol{\epsilon} \end{aligned}$$

- For stability, want $\mathbf{u} - \mathbf{u}_\epsilon$ not to grow!
- Difficult problem to analyze in general!

Stability with respect to initial values

- Useful for numerical methods: study stability **locally**:

  $$\begin{aligned} \mathbf{v}' &= \mathbf{J}(\mathbf{u}^{(0)})\mathbf{v} \qquad t > 0 \\ \mathbf{v}(0) &= \boldsymbol{\epsilon} \end{aligned} \tag{5}$$

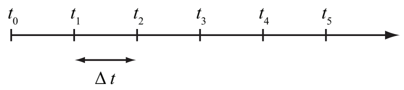
  where $J_{ij} = \partial f_i / \partial u_j$, the Jacobian matrix of $\mathbf{f}$
- We have

  $$\mathbf{v}(t) \approx \mathbf{u}(t) - \mathbf{u}_\epsilon(t)$$

  for $\|\boldsymbol{\epsilon}\|$ small and for small t
- Equation (5) a *linear* system whose stability depends on the eigenvalue of $\mathbf{J}(\mathbf{u}^{(0)})$
- Thus, equation (4) is **locally** stable (with respect to initial conditions $\mathbf{u}^{(0)}$) if all eigenvalues to $\mathbf{J}(\mathbf{u}^{(0)})$ are nonpositive.

Numerical methods for initial value problems

$$\begin{cases} y' = f(t, y) & t > 0 \\ y(0) = y^{(0)} \end{cases} \qquad (6)$$



Method 1: **Forward Euler** (*Euler framåt*).
Introduce the sequence $y_0, y_1, y_2, \ldots$. Approximate

$$y(t_k) \approx y_k, \qquad y'(t_k) \approx \frac{y_{k+1} - y_k}{\Delta t}$$

$$\begin{cases} y_{k+1} = y_k + \Delta t \, f(t_k, y_k) & k = 0, 1, 2, \ldots \\ y_0 = y^{(0)} \end{cases}$$

- Few flops per time step!
- Low accuracy ("1st-order accurate")
- Becomes unstable for large time steps

Numerical methods for initial value problems

Method 2: **Backward Euler** (*Euler bakåt*).

$$\begin{cases} y_{k+1} = y_k + \Delta t \, f(t_{k+1}, y_{k+1}) & k = 0, 1, 2, \ldots \\ y_0 = y^{(0)} \end{cases}$$

- Low accuracy: as inaccurate as Forward Euler ("1st-order accurate")
- **Implicit method**: need to solve a nonlinear equation for y_{k+1} at each time step! (Forward Euler is **explicit**)
- Many, many flops per time step!
- What's the point? (Will come back to that!)

Numerical methods for initial value problems

Method 3: **The trapezoidal method** (*trapetsmetoden*).

$$\begin{cases} y_{k+1} = y_k + \dfrac{\Delta t}{2} \left[f(t_k, y_k) + f(t_{k+1}, y_{k+1}) \right] & k = 0, 1, 2, \ldots \\ y_0 = y^{(0)} \end{cases}$$

- "Compromise" between Forward and Backward Euler!
- More accurate than Forward and Backward Euler ("2nd-order accurate")
- Implicit method that is usually a better choice that Backward Euler

Numerical methods for initial value problems

Method 4: **Heun's method**
Idea: Take the trapezoidal method, replace y_{k+1} in $f(t_{k+1}, y_{k+1})$ with estimate from Forward Euler.

$$\begin{cases} y_{k+1} = y_k + \dfrac{\Delta t}{2} (\kappa_1 + \kappa_2), \text{ where} \\ \kappa_1 = f(t_k, y_k), \\ \kappa_2 = f(t_{k+1}, y_k + \Delta t \, \kappa_1) \end{cases}$$

- Accuracy as the trapezoidal method ("2nd-order accurate")
- Explicit method!
- Becomes unstable for large time steps, similarly as Forward Euler
- The simplest member of the family of **Runge–Kutta methods**
- Runge–Kutta methods (e.g. Matlabs `ode23`, `ode45`) a standard tool for solving initial-value problems

How good are the methods?

Several issues to consider:

- In general, $y_k \neq y(t_k)$; we introduce a **discretization error**
- How accurate is the numerical solution: how small is the error $y_k - y(t_k)$? (We will be able to estimate the error even if we cannot compute the exact solution y.)
- How fast can we compute the solution?
- How robust is the solution? Can something go wrong?

We will analyze the methods with respect to

- Accuracy (“truncation error”)
- Stability (with respect to choice of time step Δt)

Accuracy, truncation error

Question: how to quantify the error introduced by any of methods 1–4?

- Let $y_0, y_1, y_2, \ldots$ be the numerically computed sequence
- Take any y_k and solve *the exact equation* with y_k as initial value
- The difference between y_{k+1} and the above exact solution evaluated at $t = t_{k+1}$ is called the **local truncation error**
- Thus, the local truncation error yields the error after **one step** of the method
- The **global truncation error** (or simply the global error) is the error in the solution after k steps

Accuracy, truncation error

Let

$$\begin{cases} \bar{y}' = f(t, \bar{y}) & t > t_k \\ \bar{y}(t_k) = y_k \end{cases}$$

Def. **Local truncation error:**

$$L_{k+1} = y_{k+1} - \bar{y}(t_{k+1}),$$

the error committed after **one step** with the method

Def. **Global truncation error** (or just “the global error”):

$$E_{k+1} = y_{k+1} - y(t_{k+1}),$$

the error compared with the exact solution to equation (6)

Accuracy, truncation error

Def. A method has the **order of accuracy** p if

$$L_{k+1} = a\Delta t^{p+1} + b\Delta t^{p+2} + \cdots = O(\Delta t^{p+1})$$

Note that $p + 1$ in the exponent corresponds to order p! Why?

In many cases (if the equation is nice enough): the global truncation error is $O(\Delta t^p)$ if the local truncation error is $O(\Delta t^{p+1})$

Thus, two ways to reduce the truncation error $L_k = O(\Delta t^{p+1})$:

- Decrease Δt. Needs more time steps to reach a predefined time
- Keep Δt and switch to a method with higher p. Needs more calculations each time step

Rule of thumb: the higher the demands on accuracy is, the more it pays off to increase p

Accuracy, truncation error

Error analysis example, Forward Euler:

$$y_{k+1} = y_k + \Delta t\; f(t_k, y_k) \tag{7}$$

Let

$$\begin{cases} \bar{y}' = f(t, \bar{y}) & t > t_k \\ \bar{y}(t_k) = y_k \end{cases} \tag{8}$$

Taylor expansion of $\bar{y}$ at $t = t_k$:

$$\begin{aligned} \bar{y}(t_{k+1}) &= \bar{y}(t_k) + \bar{y}'(t_k)\,\Delta t + \frac{1}{2}\bar{y}''(t_k)\,\Delta t^2 + \dots \\ \text{[by eq. (8)]} &= y_k + f(t_k, y_k)\,\Delta t + O(\Delta t^2) \end{aligned} \tag{9}$$

Equations (7)–(9) yields

$$y_{k+1} - \bar{y}(t_{k+1}) = O(\Delta t^2)$$

Conclusion: Forward Euler has the order of accuracy 1. Backward Euler also has the order of accuracy 1.

Stability of numerical schemes

Example: The equation

$$\begin{aligned} y' &= -8ty + t^{3/2} \quad t > 0 \\ y(0) &= 1 \end{aligned}$$

is **stable** with respect to initial values (coefficient in front of y is nonpositive)

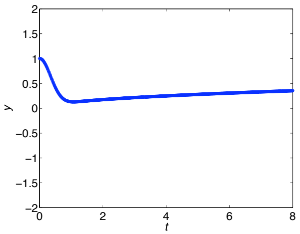
Forward Euler:

$$\begin{aligned} y_{k+1} &= y_k + \Delta t(-8t_k y_k + t_k^{3/2}) \quad k = 0, 1, \dots \\ y_0 &= 1 \end{aligned}$$

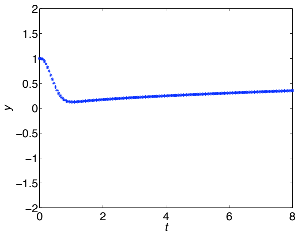
Time steps: $\Delta t = 0.01, 0.05, 0.075, 0.1$

Solving until time $t = 8$, i.e. for 800, 160, 107, and 80 time steps

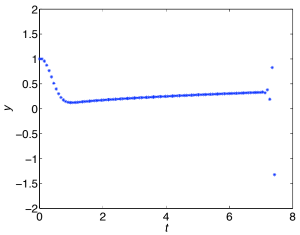
Stability of numerical schemes



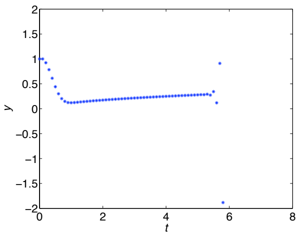
$\Delta t = 0.01$



$\Delta t = 0.05$



$\Delta t = 0.075$: numerically unstable!



$\Delta t = 0.1$: numerically unstable!

Stability of numerical schemes

- Similar effects happen for many schemes
- Typically there is a condition like $\Delta t \le$ *something* to avoid numerical instability
- In order to obtain quantitative information on a numerical methods stability properties, we will analyze it on the **stable model problem**

$$\begin{cases} y' = \lambda y & t > 0 \\ y(0) = y_0 \end{cases} \tag{10}$$

where $\lambda < 0$ (for $\lambda \in \mathbb{R}$); alternatively, $\operatorname{Re}\lambda < 0$ (for $\lambda \in \mathbb{C}$)

- $y(t) = e^{\lambda t}\, y_0$. Since $\operatorname{Re}\lambda < 0$, we have $|y(t)| < |y(0)|$
- We say that the numerical method is **stable** if it holds that $|y_{k+1}| \le |\, y_k|$ **when applied to the above model problem**

Stability of numerical schemes

Example: Forward Euler

$$\begin{aligned} y_{k+1} &= y_k + \Delta t\ f(t_k, y_k) = \text{[for eq. (10)]} \\ &= y_k + \Delta t\ \lambda y_k = \underbrace{(1 + \Delta t\ \lambda)}_{\text{“Growth factor”}}\ y_k \end{aligned}$$

Thus, Forward Euler stable if $|1 + \Delta t\ \lambda| \le 1$. For $\lambda < 0$, we have

$$-1 \le 1 + \Delta t\ \lambda = 1 - \Delta t\ |\lambda| \le 1$$

Conclusion: Forward Euler is stable for

$$\Delta t \le \frac{2}{|\lambda|}$$

Stability of numerical schemes

Example: Backward Euler

$$\begin{aligned} y_{k+1} &= y_k + \Delta t\ f(t_k, y_{k+1}) = \text{[for eq. (10)]} \\ &= y_k + \Delta t\ \lambda y_{k+1}, \end{aligned}$$

that is,

$$(1 - \Delta t\ \lambda) y_{k+1} = y_k,$$

or

$$y_{k+1} = \underbrace{\frac{1}{1 - \Delta t\ \lambda}}_{\text{Growth factor}} y_k,$$

Thus, Backward Euler stable if $1/|1 - \Delta t\ \lambda| \le 1$. For $\lambda < 0$, this is always true!

Conclusion: Backward Euler is **unconditionally stable**.

Stability when solving systems of ODEs

Any of the numerical methods above can be applied to the system

$$\begin{aligned} \mathbf{u}' &= \mathbf{f}(t, \mathbf{u}) \quad t > 0 \\ \mathbf{u}(0) &= \mathbf{u}^{(0)} \end{aligned}$$

We study stability for the linear model problem defined by

$$\mathbf{f}(t, \mathbf{u}) = \mathbf{A}\mathbf{u},$$

where all eigenvalues of $\mathbf{A}$ are real and negative.

For Forward Euler, the stability condition becomes

$$\Delta t \le \frac{2}{|\lambda_i|}$$

for all eigenvalues λ_i.

Thus, the time step will be limited by the eigenvalue of largest magnitude

Stiff systems and implicit methods

- Having eigenvalues of the matrix $\mathbf{A}$ that are vastly different in size corresponds to a system with a huge range in time scales. Fast time scales: $|\lambda_i|$ large; slow time scales: $|\lambda_i|$ small
- Such systems are called **stiff**
- Stiff systems are common in chemistry problems, for instance
- Explicit methods are usually inefficient for stiff methods since the time step is limited by the fastest time scales
- Implicit method typically more efficient for stiff systems, particularly if the interest mostly is in the slow time scales.
- The investment in extra work when solving the implicit equation will be payed back by the possibility of using larger time steps