Lecture 7:

Initial value problem ODEs: The Euler method

- **Numerical integration of ODE initial value problems**

$$\frac{dy}{dt} = f(y,t), \qquad y(0) = y_0$$

Want to find $y(t)$ for all $t > 0$

Example (relaxation equation):

$$\frac{dy}{dt} = -\lambda y, \qquad y(0) = 1$$

- **More generally, need to solve a set of coupled ODEs:**

$$\frac{d\vec{y}}{dt} = \vec{f}(\vec{y},t), \qquad \vec{y}(0) = \vec{y}_0$$

Example (second harmonic generation):

$$\frac{dS}{dt} = \gamma F^2, \qquad \frac{dF}{dt} = \gamma S F^*, \qquad S(0) = 0, F(0) = 1$$

- **Higher-order ODEs can always be reduced to coupled first-order ODEs**

Example (oscillator equation):

$$\frac{d^2x}{dt} = -\omega_0^2 x, \qquad x(0) = 1, \dot{x}(0) = 0$$

Remember! *For initial value problem with n-th order ODE need to specify the values of the function itself + all derivatives up to $(n-1)$th order at $t = 0$.*

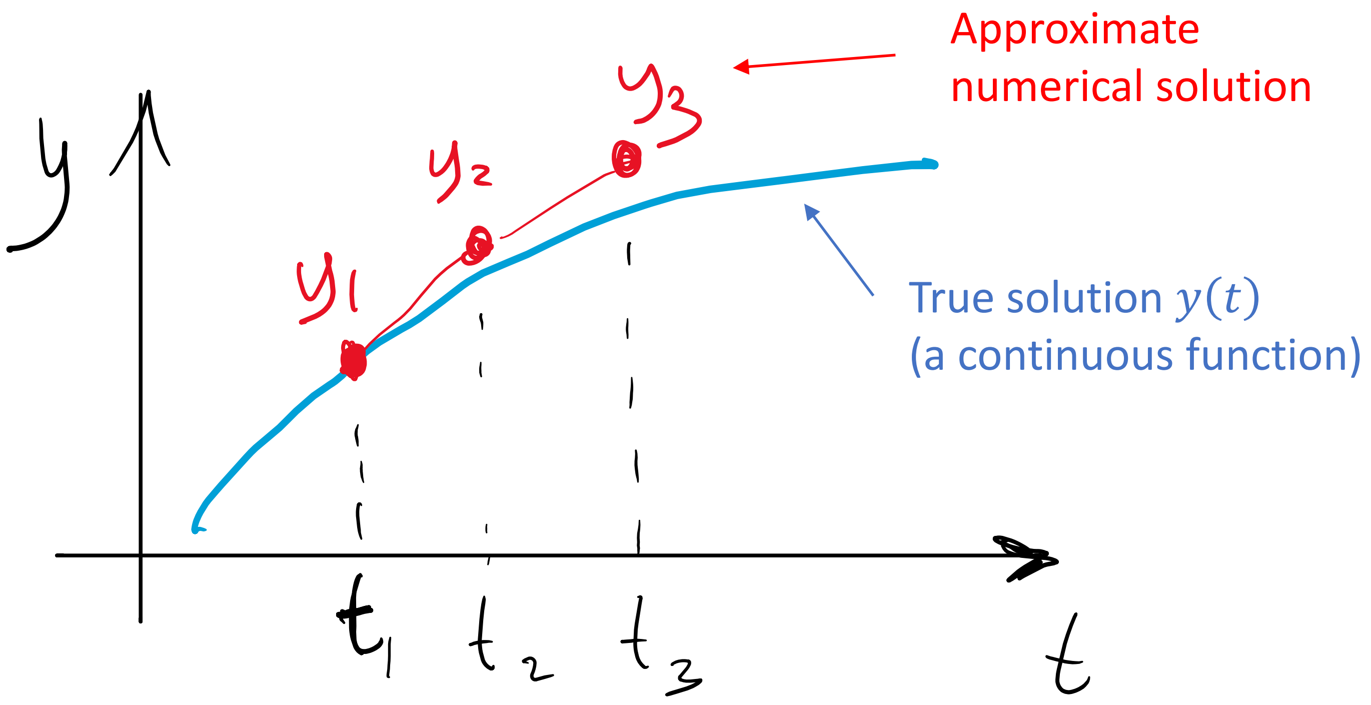
Make substitution:

$$\frac{dx}{dt} = y$$

Hence obtain:

$$\begin{cases} \dfrac{dx}{dt} = y, \\ \dfrac{dy}{dt} = -\omega_0^2 x, \end{cases} \qquad x(0) = 1, y(0) = 0$$

- **Discrete time steps**


Approximate numerical solution
True solution $y(t)$ (a continuous function)
y
t
y_1
y_2
y_3
t_1
t_2
t_3

The general idea is to calculate $y_n = y(t_n)$ using the value at the previous time step (y_{n-1}) and possibly some earlier steps ($y_{n-2}, y_{n-3}, \ldots$)

Earlier we derived approximation for derivatives via finite differences:

Type	Formula	Error
y', FDA	$y_i' \approx \frac{1}{a}\{y_{i+1} - y_i\}$	$O(a)$
y', BDA	$y_i' \approx \frac{1}{a}\{y_i - y_{i-1}\}$	$O(a)$
y', CDA	$y_i' \approx \frac{1}{2a}\{y_{i+1} - y_{i-1}\}$	$O(a^2)$
y'', CDA	$y''(x_i) \approx \frac{1}{a^2}[y_{i+1} + y_{i-1} - 2y_i]$	$O(a^2)$

The easiest approach is to adopt the FDA formula

- **Using FDA for time derivative:**

$$\frac{dy}{dt}(t_n) \approx \frac{1}{a}\{y_{n+1} - y_n\}$$

$$\Rightarrow \quad y_{n+1} \approx y_n + a \cdot \frac{dy}{dt}(t_n)$$

The value for derivative is given by the ODE:

$$\frac{dy}{dt}(t_n) = f(y_n, t_n)$$

$$\Rightarrow \quad \boxed{y_{n+1} \approx y_n + a \cdot f(y_n, t_n)}$$

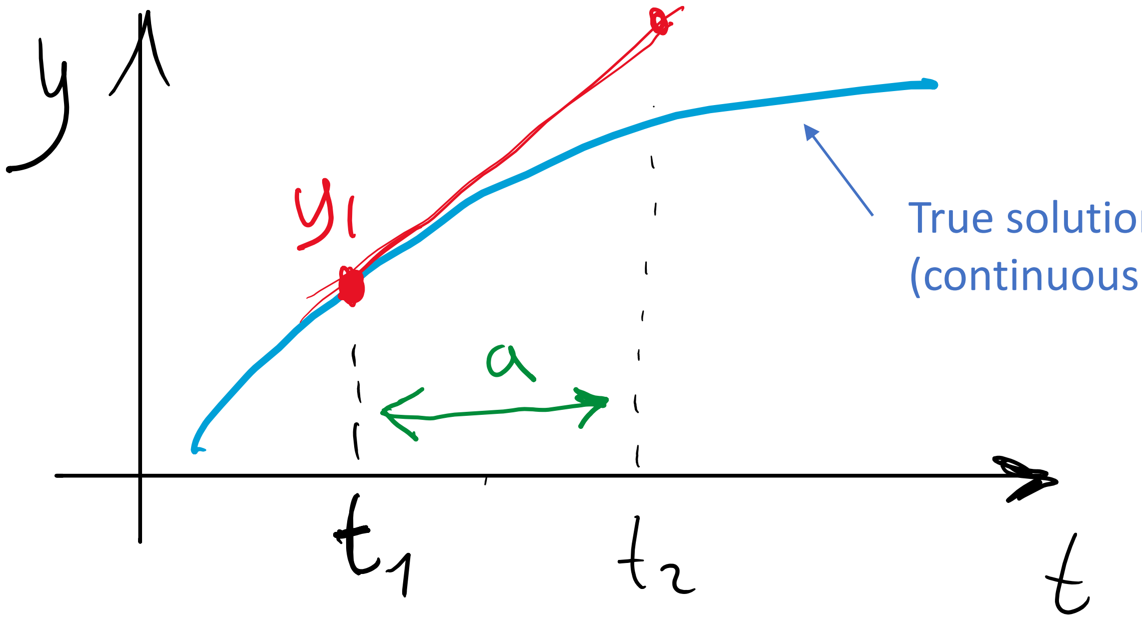
This is known as the Euler method

Leonhard Euler
1707-1783
https://en.wikipedia.org/wiki/Leonhard_Euler

- **The Euler method**

$$\boxed{y_{n+1} \approx y_n + a \cdot f(y_n, t_n)}$$

$$y_2 = y_1 + a \cdot f(y_1, t_1)$$



Error due to non-constant gradient, gets better for smaller steps

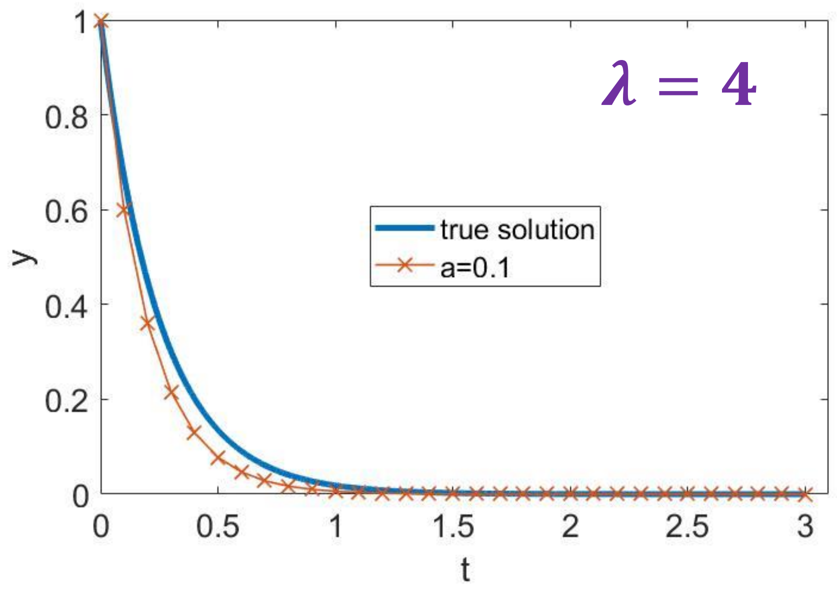
But that is not the only source of error…

- **Consider an example:**

$$\frac{dy}{dt} = -\lambda y, \qquad y(0) = 1$$

Euler method:

$$y_{n+1} = y_n + a \cdot (-\lambda y_n) = (1 - \lambda a) y_n$$

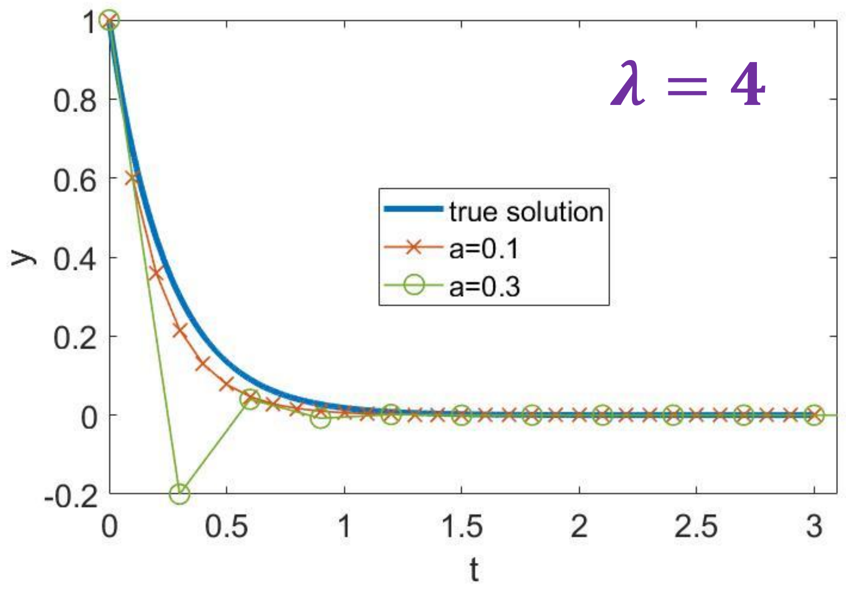


- **Consider an example:**

$$\frac{dy}{dt} = -\lambda y, \qquad y(0) = 1$$

Euler method:

$$y_{n+1} = y_n + a \cdot (-\lambda y_n) = (1 - \lambda a) y_n$$

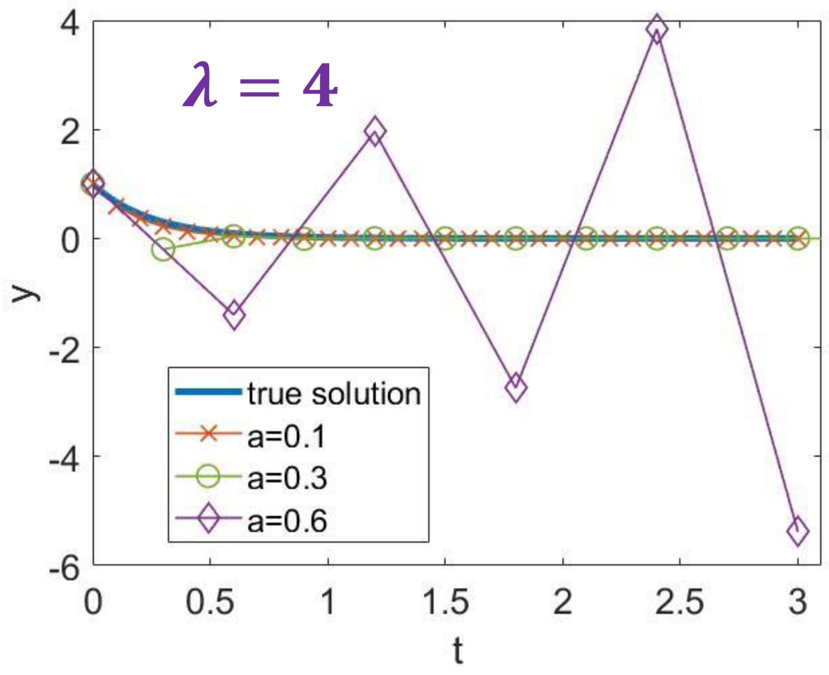


- **Consider an example:**

$$\frac{dy}{dt} = -\lambda y, \qquad y(0) = 1$$

Euler method:

$$y_{n+1} = y_n + a \cdot (-\lambda y_n) = (1 - \lambda a) y_n$$



Growth instead of decay??

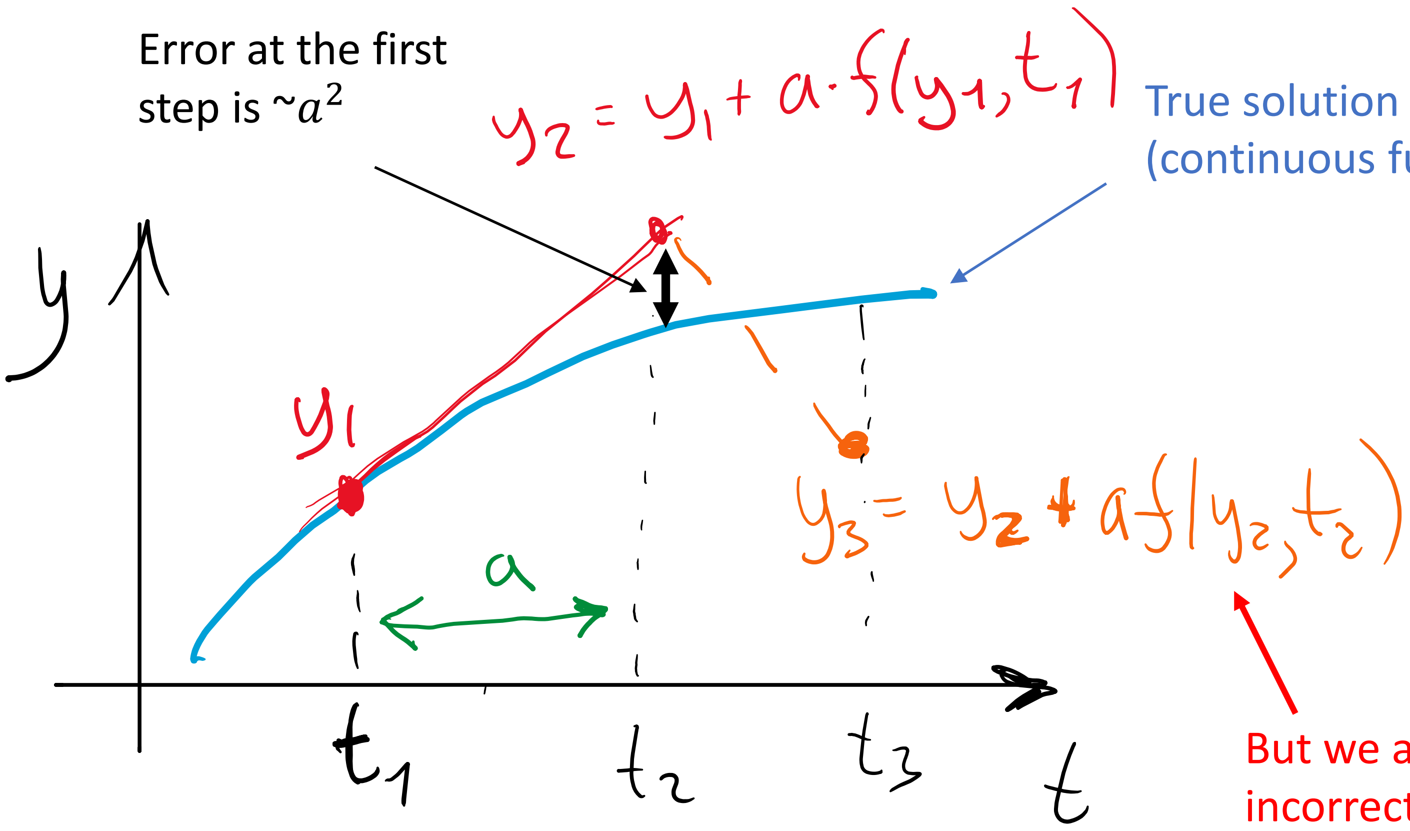
- **Numerical instabilities** is a common problem of various numerical integration algorithms (not only Euler!)

- They cause **rapid divergence** of the numerical solution from the true solution

- Often lead to "blowing up" of the numerical solution: reaching unphysical $\pm\infty$ values when the actual model does not suggest such behaviour

- Associated with **excitation of unphysical modes** of the discretized system, which are not present in the original continuous model

- **Why is this happening?**



Error at the first step is ~a^2
$y_2 = y_1 + a \cdot f(y_1, t_1)$
True solution $y(t)$ (continuous function)
y
y_1
a
$y_3 = y_2 + a f(y_2, t_2)$
t_1
t_2
t_3
t

But we are using an incorrect value y_2 to obtain the gradient!

Error at the second step will also include an incorrect estimation of the gradient!

- **Can we predict/avoid numerical instabilities?**

- Let us split the numerical solution at each iteration step y_n into true exact solution $\mathrm{y}_\mathrm{n}^{(\mathrm{e})}$ (unknown to us) and $\epsilon(t)$ a calculation error

$$y_n = y_n^{(e)} + \epsilon_n$$

- At the $(n+1)$-th iteration step of the Euler method, the solution obtained can thus be written as:

$$y_{n+1} = (1 - \lambda a) y_n$$

$$\Rightarrow \quad y_{n+1}^{(e)} + \epsilon_{n+1} = (1 - \lambda a)\left(y_n^{(e)} + \epsilon_n\right)$$

$$y_{n+1}^{(e)} + \epsilon_{n+1} = (1 - \lambda a)\left(y_n^{(e)} + \epsilon_n\right)$$

- We want ϵ_n to reduce at each subsequent iteration step (ideally), but at least not to grow

- Let us try to establish a relationship between ϵ_{n+1} and ϵ_n

$$y_{n+1}^{(e)} + \epsilon_{n+1} = (1 - \lambda \mathrm{a})y_n^{(e)} + (1 - \lambda \mathrm{a})\epsilon_n$$

- But the exact solution should also satisfy

$$y_{n+1}^{(e)} = (1 - \lambda \mathrm{a})y_n^{(e)}$$

$$\Rightarrow \quad \boxed{\epsilon_{n+1} = (1 - \lambda \mathrm{a})\epsilon_n}$$

$$\boxed{\epsilon_{n+1} = (1 - \lambda \mathrm{a})\epsilon_n}$$

- After each iteration the error gets multiplied by the factor $(1 - \lambda a)$

- After N iterations the initial error ϵ_0 is multiplied by $(1 - \lambda a)^N$:

$$\epsilon_{n+1} = (1 - \lambda \mathrm{a})^N \epsilon_0$$

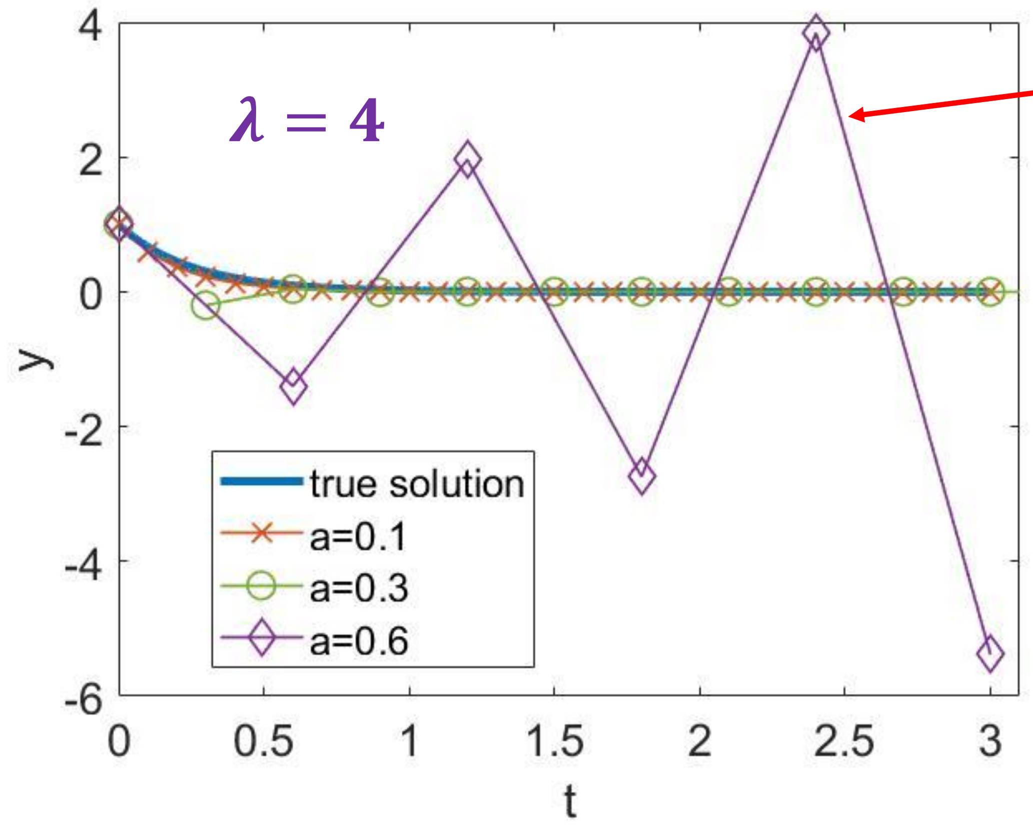
- **If $|1 - \lambda a| > 1$ the error is growing with each iteration!**

- **The stability criterion:**

$$|1 - \lambda a| \leq 1 \quad \Rightarrow \quad a \leq \frac{2}{\lambda}$$

$$\boxed{|1-\lambda a| \leq 1 \quad \Rightarrow \quad a \leq \frac{2}{\lambda}}$$

For $\lambda = 4$ this gives largest possible time step $a_{max} = 0.5$!



$a = 0.6 > a_{max}$

- Let's derive the **stability criterion for the general case**

$$\frac{dy}{dt} = f(y,t)$$

- Euler's iterations: $\boxed{y_{n+1} = y_n + a \cdot f(y_n, t_n)}$

- Split the numerical solution into exact $\mathrm{y}_{\mathrm{n}}^{(\mathrm{e})}$ and error $\epsilon(t)$:

$$y_n = y_n^{(e)} + \epsilon_n$$

- Substitute into the Euler's iterations:

$$y_{n+1}^{(e)} + \epsilon_{n+1} = y_n^{(e)} + \epsilon_n + a \cdot f(y_n^{(e)} + \epsilon_n, t_n)$$

$$y_{n+1}^{(e)} + \epsilon_{n+1} = y_n^{(e)} + \epsilon_n + a \cdot f(y_n^{(e)} + \epsilon_n, t_n)$$

- Our goal is to obtain a separate equation for the error $\epsilon(t)$

- Assume the error is small, use Taylor expansion:

$$f\left(y_n^{(e)} + \epsilon_n, t_n\right) = f\left(y_n^{(e)}, t_n\right) + \frac{\partial f}{\partial y}\left(y_n^{(e)}, t_n\right) \cdot \epsilon_n + O(\epsilon_n^2)$$

Ignore all small terms

- Hence:

$$y_{n+1}^{(e)} + \epsilon_{n+1} = y_n^{(e)} + a \cdot f\left(y_n^{(e)}, t_n\right) + \left(1 + a\frac{\partial f}{\partial y}\left(y_n^{(e)}, t_n\right)\right) \cdot \epsilon_n$$

$$y_{n+1}^{(e)} + \epsilon_{n+1} = y_n^{(e)} + a \cdot f\left(y_n^{(e)}, t_n\right) + \left(1 + a\frac{\partial f}{\partial y}\left(y_n^{(e)}, t_n\right)\right) \cdot \epsilon_n$$

- The exact solution satisfies:

$$y_{n+1}^{(e)} = y_n^{(e)} + a \cdot f\left(y_n^{(e)}, t_n\right)$$

- Hence:

$$\epsilon_{n+1} = \left(1 + a\frac{\partial f}{\partial y}\left(y_n^{(e)}, t_n\right)\right) \cdot \epsilon_n$$

- And therefore the general stability criterion is:

$$\boxed{\left|1 + a\frac{\partial f}{\partial y}\left(y_n^{(e)}, t_n\right)\right| \leq 1}$$

- **Another example:**

$$\frac{dy}{dt} = -\lambda y^3, \qquad y(0) = 1$$

Euler iterations:

$$y_{n+1} = y_n - a \cdot \lambda y_n^3$$

Stability criterion:

$$\boxed{|1 - a \cdot 3\lambda y_n^2| \leq 1}$$

Note: Generally, the stability criterion will involve values of the function itself (y_n) which are not known in advance!

-> Can implement this into an iteration scheme with variable step size *(Here, the important advantage of the FDA discretization scheme is that it works well with a variable step size)*

Summary:

- Euler method is the simplest and the least accurate.

(consider the balance between desired accuracy and required computational effort)

- Numerical integration schemes can suffer from instabilities.

This is generally true for any integration scheme, not just Euler!

- Stability analysis reveals requirements for the time step size

  - *This requirement will generally involve the function amplitude: may need an adaptive step size*
  - In some cases the scheme can be **unstable for any time step size**! (we will see examples on next lectures)

Lecture 8:

Initial value problem ODEs: Leapfrog method

- **Stability criterion for coupled ODEs**

Example: the oscillator equation

$$\ddot{x} + \omega_0^2 x = 0, \qquad \Rightarrow \begin{cases} \dfrac{dx}{dt} = y, \\ \dfrac{dy}{dt} = -\omega_0^2 x, \end{cases}$$

- Euler's iterations:

$$x_{n+1} = x_n + a y_n, \qquad y_{n+1} = y_n - a\omega_0^2 x_n$$

- Split the numerical solution into exact and error:

$$x_n = x_n^{(e)} + \epsilon_n \qquad y_n = y_n^{(e)} + \eta_n$$

- Substitute into the Euler's iterations equations:

$$x_{n+1}^{(e)} + \epsilon_{n+1} = x_n^{(e)} + \epsilon_n + a \cdot (y_n^{(e)} + \eta_n)$$

$$y_{n+1}^{(e)} + \eta_{n+1} = y_n^{(e)} + \eta_n - a\omega_0^2 \cdot (x_n^{(e)} + \epsilon_n)$$

$$\begin{cases} \frac{dx}{dt} = y, \\ \frac{dy}{dt} = -\omega_0^2 x, \end{cases}$$

- **Stability criterion for coupled ODEs**

$$x_{n+1}^{(e)} + \epsilon_{n+1} = x_n^{(e)} + \epsilon_n + a \cdot (y_n^{(e)} + \eta_n)$$
$$y_{n+1}^{(e)} + \eta_{n+1} = y_n^{(e)} + \eta_n - a\omega_0^2 \cdot (x_n^{(e)} + \epsilon_n)$$

- Exact solution solves Euler's equations:

$$x_{n+1}^{(e)} = x_n^{(e)} + a \cdot y_n^{(e)}$$
$$y_{n+1}^{(e)} = y_n^{(e)} - a\omega_0^2 \cdot x_n^{(e)}$$

- Hence obtain equations for the error terms:

$$\epsilon_{n+1} = \epsilon_n + a\eta_n \qquad \eta_{n+1} = \eta_n - a\omega_0^2\epsilon_n$$

$$\begin{cases} \dfrac{dx}{dt} = y, \\ \dfrac{dy}{dt} = -\omega_0^2 x, \end{cases}$$

- **Stability criterion for coupled ODEs**

$$\epsilon_{n+1} = \epsilon_n + a\eta_n\,, \qquad \eta_{n+1} = \eta_n - a\omega_0^2\epsilon_n$$

- Can re-write this in the matrix form:

$$\begin{bmatrix}\epsilon_{n+1}\\ \eta_{n+1}\end{bmatrix} = \begin{bmatrix}1 & a\\ -a\omega_0^2 & 1\end{bmatrix}\cdot\begin{bmatrix}\epsilon_n\\ \eta_n\end{bmatrix} = \widehat{M}\cdot\begin{bmatrix}\epsilon_n\\ \eta_n\end{bmatrix},$$

- The matrix $\widehat{M}$ is called ***the amplification matrix***. For stable scheme, all eigenvalues of this matrix must not exceed 1 by modulus:

$$\boxed{|\lambda_i| \le 1 \;\; \forall\, i}$$

($i = 1{,}2$ in our case)

- In our case:

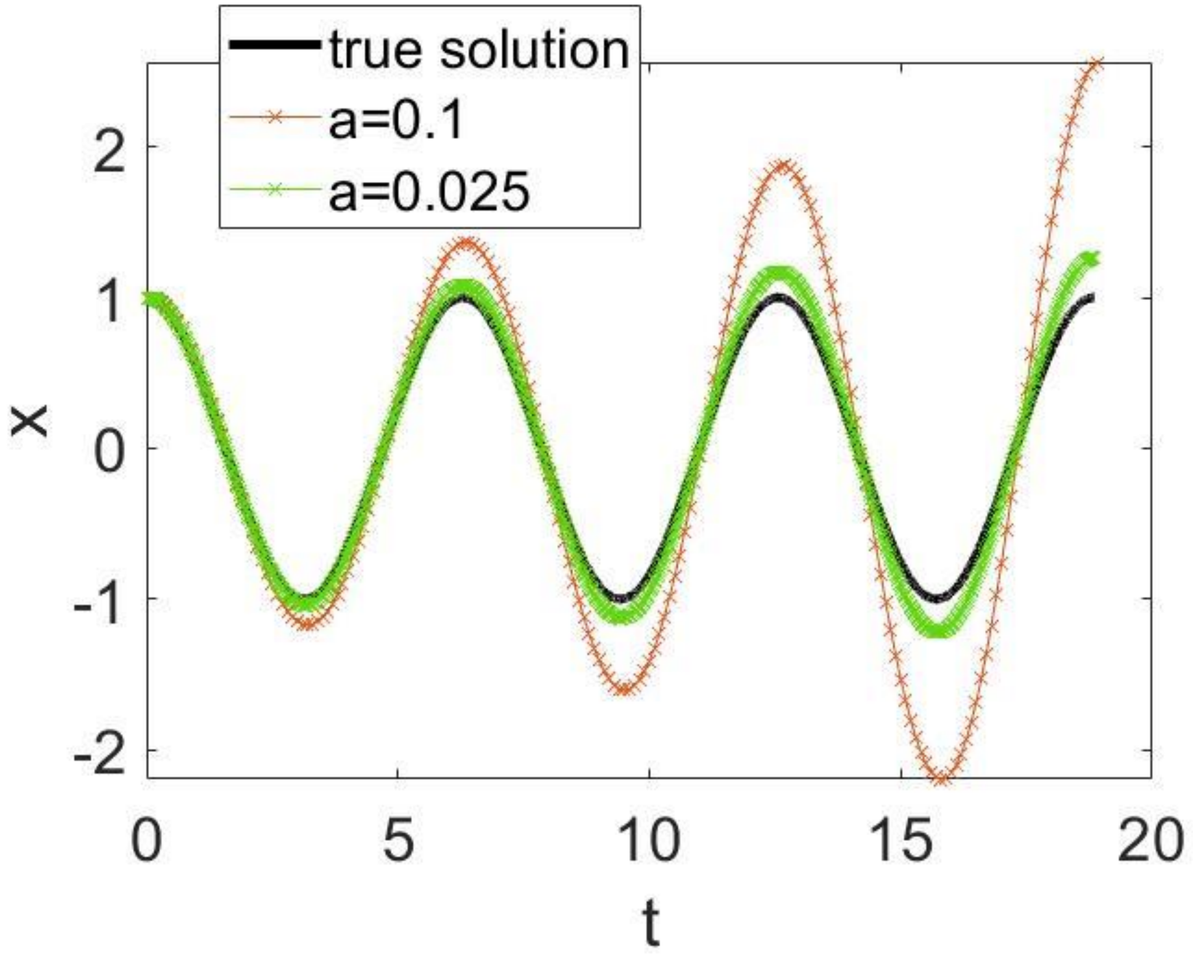
$$\lambda_{1,2} = 1 \pm ia\omega_0 \Rightarrow |\lambda_{1,2}| = \sqrt{1 + a^2\omega_0^2} > 1$$

ALWAYS UNSTABLE!

- **Stability criterion for coupled ODEs**

$$\begin{cases} \dfrac{dx}{dt} = y, \\ \dfrac{dy}{dt} = -\omega_0^2 x, \end{cases}$$

Numerical propagation, $\omega_0 = 1$



- **We need a better algorithm!**

- Euler method is based on FDA approximation $O(a)$

- We could try better approximations $O(a^2)$

Type	Formula	Error
y', FDA	$y_i' \approx \frac{1}{a}\{y_{i+1} - y_i\}$	$O(a)$
y', BDA	$y_i' \approx \frac{1}{a}\{y_i - y_{i-1}\}$	$O(a)$
y', CDA	$y_i' \approx \frac{1}{2a}\{y_{i+1} - y_{i-1}\}$	$O(a^2)$
y'', CDA	$y''(x_i) \approx \frac{1}{a^2}[y_{i+1} + y_{i-1} - 2y_i]$	$O(a^2)$

Could try CDA formula, which gives much better precision?

- **Using CDA for time derivative:**

$$\frac{dy}{dt}(t_n) \approx \frac{1}{2a}\{y_{n+1} - y_{n-1}\}$$

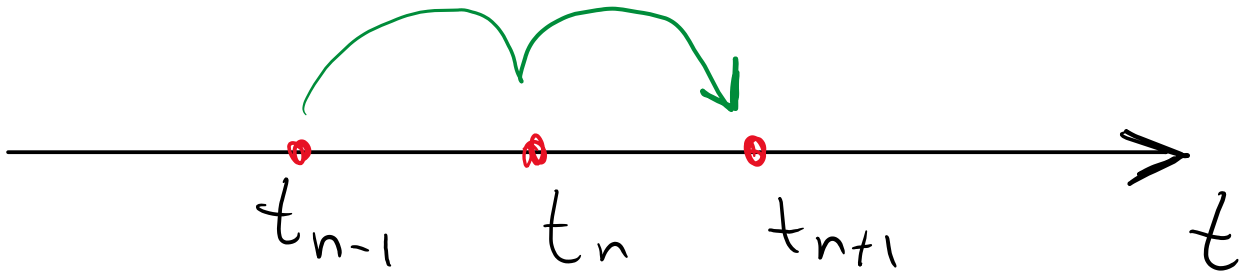
$$\Rightarrow \quad y_{n+1} \approx y_{n-1} + 2a \cdot \frac{dy}{dt}(t_n)$$

The value for derivative is given by the ODE:

$$\frac{dy}{dt}(t_n) = f(y_n, t_n)$$

$$\Rightarrow \quad \boxed{y_{n+1} \approx y_{n-1} + 2a \cdot f(y_n, t_n)}$$

$$\boxed{y_{n+1} \approx y_{n-1} + 2a \cdot f(y_n, t_n)}$$



- Evaluating function at t_{n+1} using its value at t_{n-1} and the derivative at t_n
- This is known as **Leapfrog method**
- Requires two initial conditions: $y(t_0)$ and $y(t_0 - a)$
  That's more than you normally have to start with!
- Also requires regular time grid (fixed step in time)

$y_{n+1} \approx y_{n-1} + 2a \cdot f(y_n, t_n)$

- **Consider an example:**

$$\frac{dy}{dt} = -\lambda y, \qquad y(0) = 1$$

Leapfrog method:

$$y_{n+1} = y_{n-1} + 2a \cdot (-\lambda y_n)$$

To compute y_1 we need to know y_{-1} and y_0

But we only know $y_0 = \mathrm{y}(0)$ at the start!

Generally, we need to make one step using another method (such as Euler). And then switch to Leapfrog.

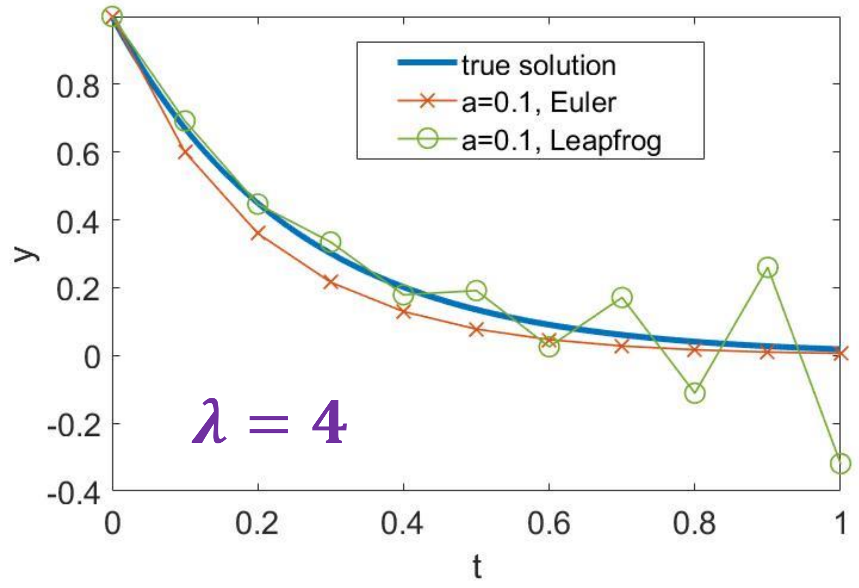
This is an additional source of error in the initial conditions!

- **Consider an example:**

$$\frac{dy}{dt} = -\lambda y, \qquad y(0) = 1$$

Leapfrog method:

$$y_{n+1} = y_{n-1} - 2\lambda a \cdot y_n$$



(in this case I produced y_{-1} from the analytical solution)

Apparently, Leapfrog method also suffers from instabilities!

$y_{n+1} = y_{n-1} - 2\lambda a \cdot y_n$

Stability analysis

- As usual, we split the numerical solution at each iteration step y_n into true exact solution $\mathrm{y_n^{(e)}}$ (unknown to us) and $\epsilon(t)$ a calculation error

$$y_n = y_n^{(e)} + \epsilon_n$$

$$\Rightarrow \quad y_{n+1}^{(e)} + \epsilon_{n+1} = y_{n-1}^{(e)} + \epsilon_{n-1} - 2\lambda a \cdot y_n^{(e)} - 2\lambda a \cdot \epsilon_n$$

- The exact solution satisfies

$$y_{n+1}^{(e)} = y_{n-1}^{(e)} - 2\lambda a \cdot y_n^{(e)}$$

$$\Rightarrow \quad \epsilon_{n+1} = \epsilon_{n-1} - 2\lambda a \cdot \epsilon_n$$

$y_{n+1} = y_{n-1} - 2\lambda a \cdot y_n$

Stability analysis

$$\epsilon_{n+1} = \epsilon_{n-1} - 2\lambda a \cdot \epsilon_n$$

- Can re-write this in the matrix form:

$$\begin{bmatrix} \epsilon_{n+1} \\ \epsilon_n \end{bmatrix} = \begin{bmatrix} -2\lambda a & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \epsilon_n \\ \epsilon_{n-1} \end{bmatrix}$$

- The amplification matrix

$$\widehat{M} = \begin{bmatrix} -2\lambda a & 1 \\ 1 & 0 \end{bmatrix}$$

has eigenvalues $\lambda_{1,2} = -\lambda a \pm \sqrt{(\lambda a)^2 + 1}$

$|\lambda_2| = \lambda a + \sqrt{(\lambda a)^2 + 1} > 1$ for any step size a

Hence this scheme is ALWAYS UNSTABLE!

Oscillator equation

$$\ddot{x} + \omega_0^2 x = 0, \qquad \Rightarrow \begin{cases} \dfrac{dx}{dt} = y, \\ \dfrac{dy}{dt} = -\omega_0^2 x, \end{cases}$$

- Can investigate stability of the Leapfrog scheme in a similar way

$$x_n = x_n^{(e)} + \epsilon_n \qquad\qquad y_n = y_n^{(e)} + \eta_n$$

- Will result in this case in a 4x4 amplification matrix

$$\begin{bmatrix} \epsilon_{n+1} \\ \epsilon_n \\ \eta_{n+1} \\ \eta_n \end{bmatrix} = \widehat{M} \begin{bmatrix} \epsilon_n \\ \epsilon_{n-1} \\ \eta_n \\ \eta_{n-1} \end{bmatrix}$$

- All eigenvalues $|\lambda_{1,2,3,4}| = 1$ in this case!

The scheme is MARGINALLY STABLE!

Euler vs Leapfrog

$$\boxed{y_{n+1} \approx y_n + a \cdot f(y_n, t_n)}$$

$$\boxed{y_{n+1} \approx y_{n-1} + 2a \cdot f(y_n, t_n)}$$

- Both schemes compute y_{n+1} in one step => computation effort is similar

Euler

- Discretisation error $O(a)$
- Easy to set up
- Step can be variable
- Can be stabilized (with small enough step) for decay-type problems
- Always unstable for oscillator-type problems

Leapfrog

- Discretisation error $O(a^2)$
- Requires an extra initial condition (generally leads to an additional error)
- Step must be fixed
- Always unstable for decay-type problems
- Always marginally stable for oscillator-type problems

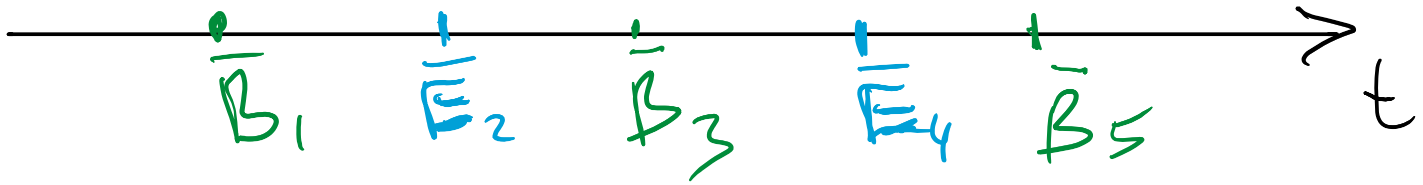
One important area of application of Leapfrog method is electromagnetism (there it is called FDTD method – Finite Difference Time Domain)

- Maxwell equations appear to be particularly suitable for Leapfrog

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \qquad \vec{B}_{n+1} = \vec{B}_{n-1} - 2a \cdot (\nabla \times \vec{E})(t_n)$$

$$\nabla \times \vec{B} = \frac{1}{c^2}\frac{\partial \vec{E}}{\partial t} \qquad \vec{E}_{n+1} = \vec{E}_{n-1} - 2ac^2 \cdot (\nabla \times \vec{B})(t_n)$$

- Can define B-field at odd points of time only, and E-field at even points only



Saves a lot of memory to store all fields!

- Problems in electromagnetism are usually oscillatory-type: the scheme is stable

Summary:

- There is no “golden scheme” to solve all possible problems.
- Each particular problem requires careful consideration. Some methods work well with certain type of problems, but completely fail with other types.
- Always useful to start with literature research.

Lecture 9:

Implicit methods and Runge-Kutta methods

- In Euler and Leapfrog methods we express the unknown value of the function y_{n+1} in terms of already known values from the previous steps y_n, y_{n-1}

$$\boxed{y_{n+1} \approx y_n + a \cdot f(y_n, t_n)}$$

$$\boxed{y_{n+1} \approx y_{n-1} + 2a \cdot f(y_n, t_n)}$$

- Such methods are generally called **explicit methods**

- In particular, to obtain the Euler formula, we used FDA for the time derivative:

$$y'_n = f(y_n, t_n) \approx \frac{1}{a}\{y_{n+1} - y_n\}$$

- But we could use BDA instead:

$$y'_{n+1} = f(y_{n+1}, t_{n+1}) \approx \frac{1}{a}\{y_{n+1} - y_n\}$$

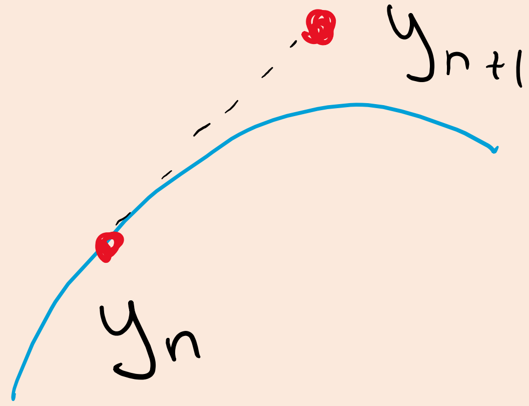
$$y'_{n+1} = f(y_{n+1}, t_{n+1}) \approx \frac{1}{a}\{y_{n+1} - y_n\}$$

- Re-arrange:

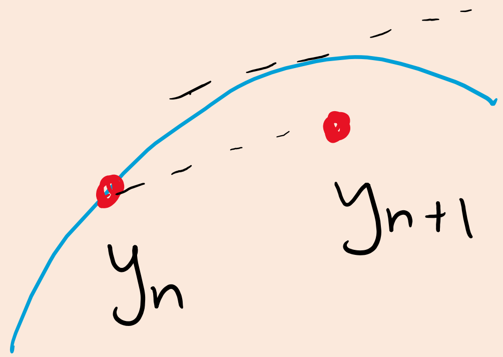
$$y_{n+1} \approx y_n + a \cdot f(y_{n+1}, t_{n+1})$$

This scheme is known as Backward Euler method

FDA (Forward Euler)



BDA (Backward Euler)



$$y_{n+1} \approx y_n + a \cdot f(y_{n+1}, t_{n+1})$$

This is still unknown!

- Need to invert $f(y,t)$ in order to find y_{n+1}!
- Backward Euler is an **implicit method**
- Works fine if f is linear in y

e.g. decay equation $$\frac{dy}{dt} = -\lambda y = f(y)$$

gives $$y_{n+1} = y_n + a \cdot (-\lambda \cdot y_{n+1})$$

$$\Rightarrow \boxed{y_{n+1} = \frac{y_n}{1 + a\lambda}}$$

Decay equation: explicit (Forward Euler) vs implicit (Backward Euler)

$$\boxed{y_{n+1} = (1 - \lambda a) y_n}$$

Based on FDA

$$\boxed{y_{n+1} = \frac{y_n}{1 + a\lambda}}$$

Based on BDA

- Both methods have a similar discretization error

  (we will discuss in a minute discretisation error of various schemes in more detail)

- Stability?

FDA Euler:

$$\epsilon_{n+1} = (1 - \lambda a)\epsilon_n$$

$$|1 - \lambda a| \leq 1$$

$$\Rightarrow \quad a \leq \frac{2}{\lambda}$$

BDA Euler:

$$\epsilon_{n+1} = \frac{\epsilon_n}{1 + a\lambda}$$

$$\left|\frac{1}{1 + a\lambda}\right| \leq 1$$

$\Rightarrow$ **Always stable!**

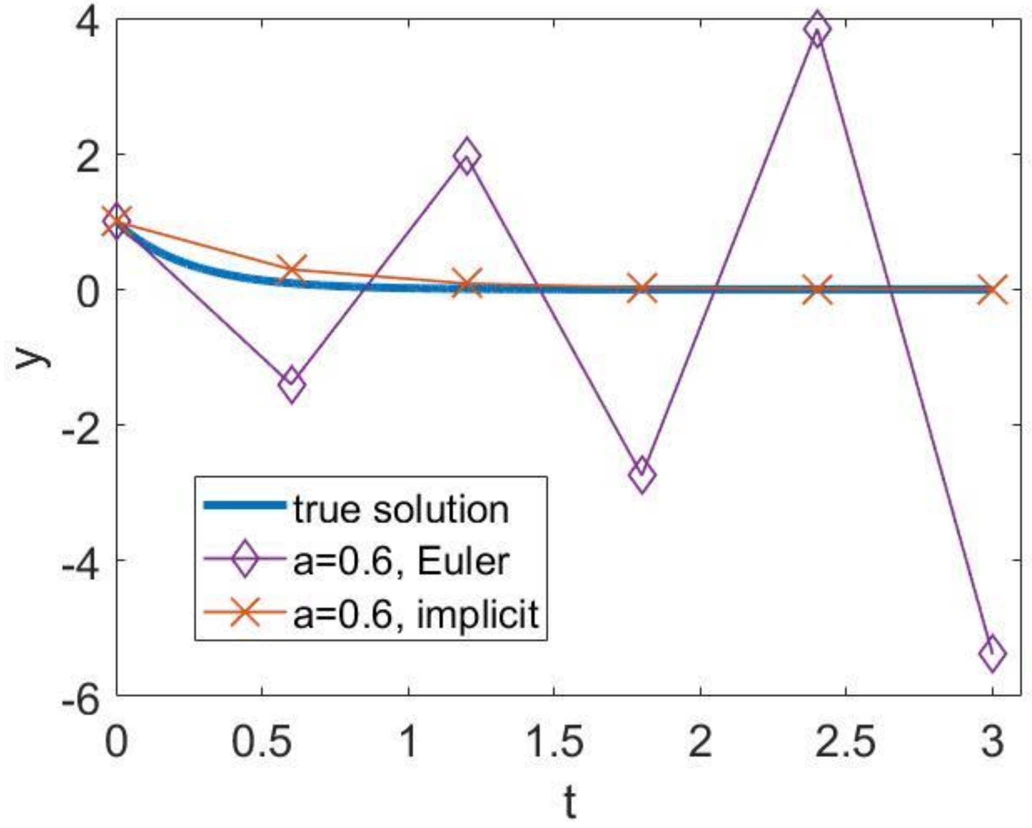
Decay equation: explicit (Forward Euler) vs implicit (Backward Euler)

$$\boxed{y_{n+1} = (1 - \lambda a) y_n}$$

Based on FDA

$$\boxed{y_{n+1} = \frac{y_n}{1 + a\lambda}}$$

Based on BDA



- Generally, **implicit methods have much better stability** than explicit methods

- The need to invert $f(y,t)$ and obtain y_{n+1} is the main difficulty with implicit methods

- Generally deal with nonlinear and/or transcendental equations which you need to solve separately, using various root-finding methods =>

  - extra computational cost
  - an additional source of error
  - need to deal with multiple roots (select the correct one)

e.g. $$\frac{dy}{dt} = -\lambda y^3 \quad \Rightarrow \quad y_{n+1} \approx y_n - \lambda a \cdot y_{n+1}^3$$

need to solve cubic equation for y_{n+1}!

Local vs global error in propagation schemes

- Forward and Backward Euler schemes are based on FDA/BDA

  e.g. Forward Euler:

  $$y_{n+1} \approx y_n + a \cdot f(y_n, t_n) + \boldsymbol{O(a^2)}$$

- Local error (error per step) is not very informative. We need to integrate over a certain time window (e.g. $0 < t < T$). The smaller the step we choose, the smaller the *error per step* is… but we need to make more steps! The error accumulates.

- Global error ~ Local error * Total number of steps

- Number of steps: $N = T/a$ – inversely proportional to step size

➡ *For Euler schemes: Local error ~ $O(a^2)$, Global error ~ $O(a)$*

*Euler schemes are **first order** numerical integration schemes: the error is $\boldsymbol{O(a^1)}$*

Runge–Kutta methods

- General idea is to obtain y_{n+1} in M steps:

$$\boxed{y_{n+1} \approx y_n + a \cdot \sum_{j}^{M} b_j k_j}$$

b_j - numerical coefficients

k_j - $f(y,t)$ evaluated at various extrapolated points between t_n and t_{n+1}

Carl Runge
1856-1927
https://en.wikipedia.org/wiki/Carl_Runge

Martin Kutta
1867-1944
https://en.wikipedia.org/wiki/Martin_Kutta

- An N-th order Runge-Kutta method has a (global) discretisation error $O(a^N)$
- Orders up to $N = 4$ require $M = N$ coefficients. 5th and above orders require more coefficients (usually, $M = N + 2$)

Note:
- *1st order Runge-Kutta requires calculation of 1 coefficients per step and has error $\epsilon \sim O(a)$*
- *Could reduce step $a \to a/2$ and make twice as many calculations to achieve $\epsilon \sim O(a/2)$*
- *Or instead use 2nd order Runge-Kutta, and achieve $\epsilon \sim O(a^2)$ with the same number of calculations per step*

Runge–Kutta methods

$$\boxed{y_{n+1} \approx y_n + a \cdot \sum_{j}^{M} b_j k_j}$$

- The Euler method is a particular case of a Runge-Kutta order N=1:

$$y_{n+1} \approx y_n + a \cdot f(y_n, t_n) \quad \Rightarrow \quad b_1 = 1, \qquad k_1 = f(y_n, t_n)$$

- A 2nd order Runge-Kutta (mid-point method):

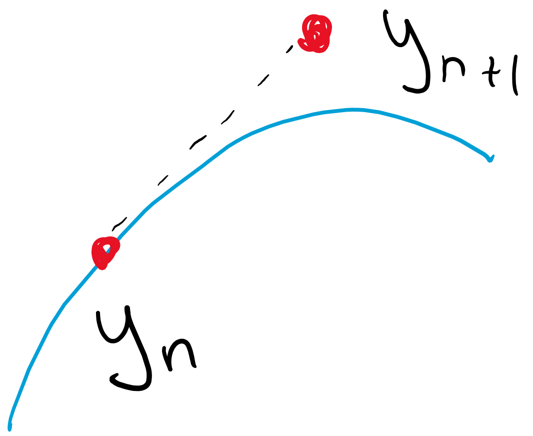
$$k_1 = f(y_n, t_n)$$

$$k_2 = f\left(y_n + \frac{a}{2} k_1, t_n + \frac{a}{2}\right) \qquad b_1 = 0, \qquad b_2 = 1$$

$$y_{n+1} \approx y_n + a \cdot k_2$$

1st order (Euler)

$$y_{n+1} = y_n + a \cdot f(y_n, t_n)$$



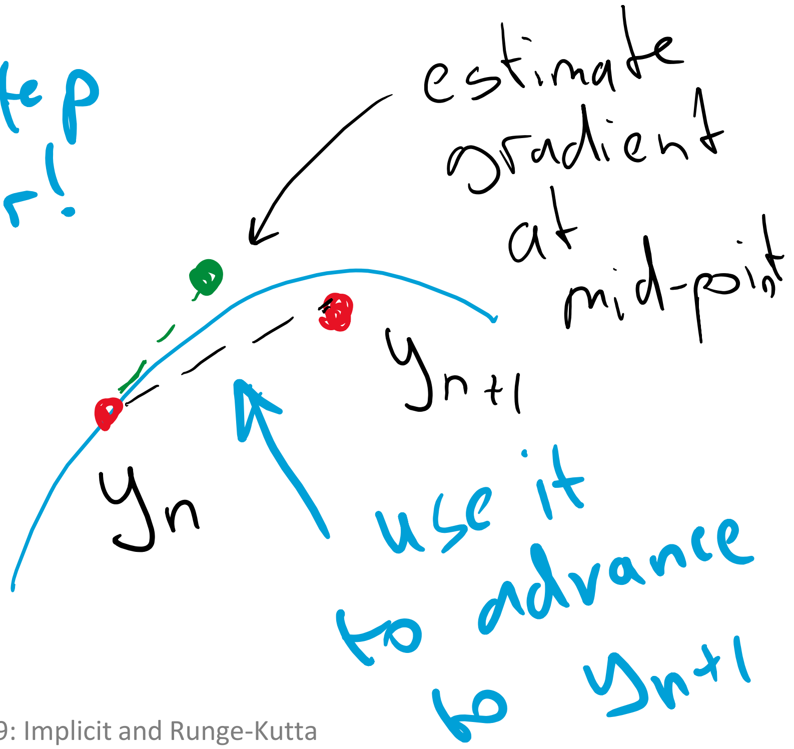
2nd order (mid-point)

$$y_{n+1} = y_n + a \cdot k_2$$

$$k_1 = f(y_n, t_n)$$

$$k_2 = f\left(\underbrace{y_n + \frac{a}{2} k_1}_{}, t_n + \frac{a}{2}\right)$$

This is half-step Euler!



estimate gradient at mid-point
y_{n+1}
y_n
use it to advance to y_{n+1}

- Another variant of a 2nd order Runge-Kutta (modified Euler)

$k_1 = f(y_n, t_n)$ - gradient at the starting point t_n

$k_2 = f(y_n + ak_1, t_n + a)$ - Estimated gradient at next point t_{n+1}

$y_{n+1} \approx y_n + a \cdot \frac{1}{2}(k_1 + k_2)$ - Use the average of the two

$$(b_1 = 1/2,\ b_2 = 1/2)$$

This method also has (global) discretisation error $O(a^2)$
(like any other 2nd order Runge-Kutta)

- The most commonly used method is 4th order Runge-Kutta (RK4)

$$k_1 = f(y_n, t_n)$$

$$k_2 = f\left(y_n + \frac{a}{2}k_1, t_n + \frac{a}{2}\right)$$

$$k_3 = f\left(y_n + \frac{a}{2}k_2, t_n + \frac{a}{2}\right)$$

$$k_4 = f(y_n + ak_3, t_n + a)$$

$$\boxed{y_{n+1} \approx y_n + a \cdot \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)}$$

This method has discretisation error $O(a^4)$

- RK5 and above methods require more than N coefficients

- **Stability of RK methods** is generally analysed along similar lines as previously, but requires a bit more analytical effort

Example: Decay equation $$\frac{dy}{dt} = -\lambda y = f(y)$$

Mid-point 2nd order RK: $$y_{n+1} \approx y_n + a\left[-\lambda y_n + \frac{\lambda^2 a}{2} y_n\right]$$

For the error obtain: $$\epsilon_{n+1} \approx \left[1 - a\lambda + \frac{\lambda^2 a^2}{2}\right]\epsilon_n$$

Require $$\left|1 - a\lambda + \frac{\lambda^2 a^2}{2}\right| \leq 1$$

Gives: $$a \leq \frac{2}{\lambda}$$ *(same as we obtained for Euler)*

- Runge-Kutta methods are often viewed as a compromise between (too simple and inacurate) Euler and (too complicated and resource demanding) implicit schemes.

- For large-size systems (such as many-body problems) RK methods may appear too resource demanding

- For certain type of problems, instabilities of RK methods may still be an issue

Other things to consider when dealing with Initial Value Problems:

- Integrals of motion (such as conservation of energy):

E.g. the oscillator equation

$$\ddot{x} + \omega_0^2 x = 0, \qquad \Rightarrow \begin{cases} \dfrac{dx}{dt} = y, \\ \dfrac{dy}{dt} = -\omega_0^2 x, \end{cases}$$

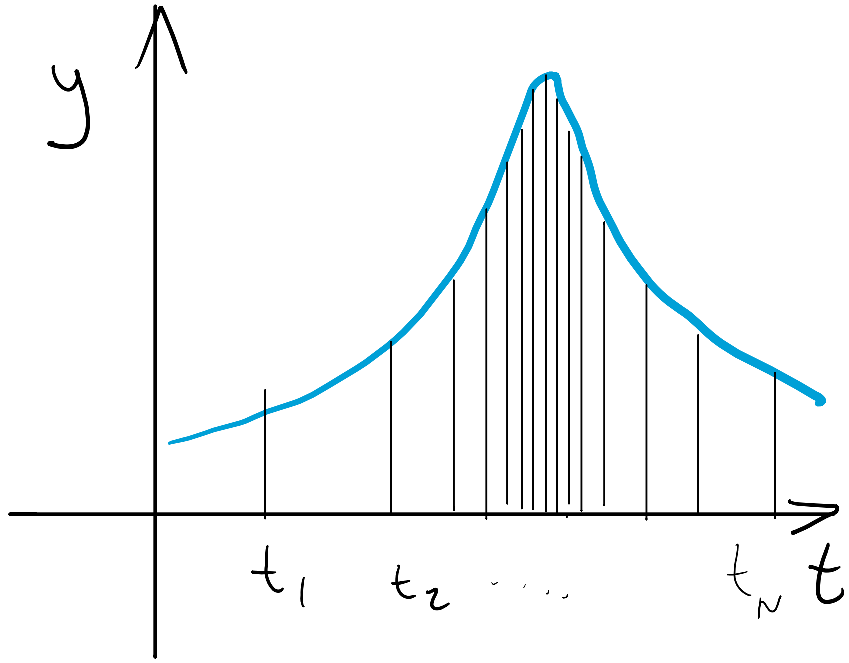
Has integral of motion: $E = \frac{\dot{x}^2}{2} + \frac{\omega_0^2 x^2}{2} = \frac{y^2}{2} + \frac{\omega_0^2 x^2}{2}$

- useful to monitor: could be a clear indicator of some issues with the propagation scheme (or its implementation);
- but should not consider as a “warrant of accuracy”

Other things to consider when dealing with Initial Value Problems:

- Variable step:

Take smaller steps when needed, and larger steps when can afford => save a lot of computational effort!



Two options for monitoring the function behaviour:

- Calculate y_{n+1} simultaneously in one step and two half-steps. A noticeable discrepancy between the two answers would indicate not enough accuracy;

- Calculate y_{n+1} simultaneously by two different orders of accuracy methods (e.g. RK4 and RK5). Estimate the discretization error from the difference.

Many "off-the-shelf" ODE solvers will have adaptive step embedded (such as Matlab's ode45 solver – uses RK4+RK5)

Summary:

- 4th order Runge-Kutta method is often the default choice: offers a good balance between accuracy and computational effort
- Implicit methods are usually most resource-demanding, but also most stable. May need to consider them for those “tricky” problems
- Many-body problems (many coupled ODEs) require minimization of computational effort: RK methods are often not suitable