

**Solving ordinary differential equations
part 1**

Outline

- ODE and PDE in physics
- 1-st order ODE
- finite difference method
- Forward Euler method:
 - derivation of difference scheme
 - local and global errors, order of method
 - stability
 - examples of numerical solution
- Backward Euler method
 - derivation of implicit formula
 - Picard iteration and Secant/Newton methods combined with implicit methods
 - errors, order, stability
- trapezoidal formula
 - derivation of difference formula
 - errors, order, stability
- n-th order ODE solved as a system of 1-st order ODE

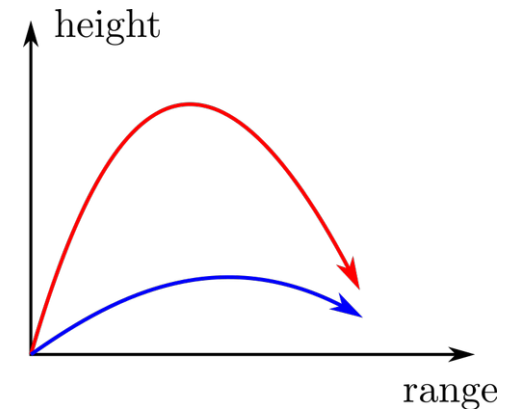
Wide class of problems in physics are defined in language of differential equations, these can be further divided into two groups

- **ODE – ordinary differential equations**, which depends on single variable (univariate problem), in many cases this variable is time then ODE defines/governs the dynamic of the system

example: dynamic simulation using Newton's equation of motion like projectile missile

$$\frac{d^2 \vec{r}}{dt^2} = \vec{F}_{gravity} + \vec{F}_{drag} + \vec{F}_{friction} + \dots \implies \vec{r}(t) = ?$$

here we are interested in finding the trajectory of missile that gives the all information about the dynamic of the object



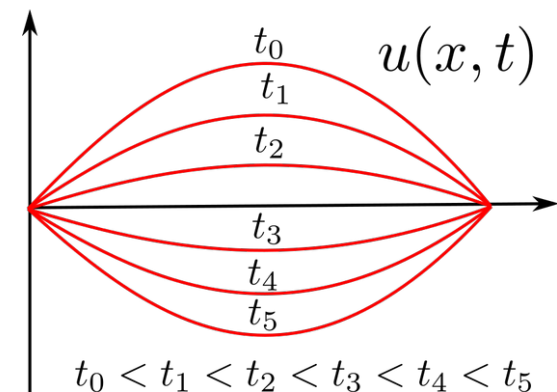
- **PDE – partial differential equations** contain two or more independent variables (multivariate problem)

example: wave equation governing the dynamic of spatial-time oscillations of guitar string

$$\frac{\partial^2 u(x, t)}{\partial t^2} = -\frac{1}{v^2} \frac{\partial^2 u(x, t)}{\partial x^2} \implies u(x, t) = ?$$

- **Here arise the question: how to find the solution?**

Only a small number (class) of differential problems can be solved analytically or semi-analytically, the other ones must be solved by means of numerical methods.



1-st order ODE for initial value problem

Assume that $u=u(t)$ is unknown function which fulfills differential equation

$$\frac{du}{dt} = f(t, u), \quad t \in \mathbb{R}, \quad f(t, u) \text{ - known function}$$

After separating the variables (dependent u and independent t) and integrating the equation one gets an alternative formulation of the problem – the integral equation

$$du = dt f(t, u) \implies u(t) = u(t_0) + \int_{t_0}^t dt' f(t', u(t'))$$

- this formulation is useful if $\mathbf{f=f(t)}$ so the integral can be easily (?) computed numerically
- because integral equation provides the solution, the numerical schemes used to solve the ODE are sometimes called the integration schemes (but rarely)
- $u(t_0)$ – it is an initial condition, must be specified if we want to find the unique solution
- our task is to find/derive the method which allows solving the ODE but avoiding the explicit integration

Finite difference method

All numerical schemes used to solve ODE are based on finite difference method. So, by first, the mesh of n (equidistant) nodes at time axis must be introduced, solution will be calculated at these nodes

$$\frac{du}{dt} = f(t, u)$$

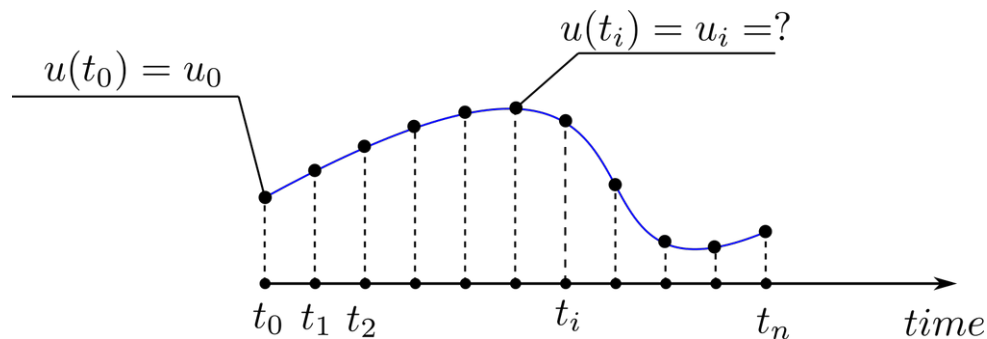
$$\Delta t = \frac{t_{max} - t_0}{n}$$

$$t \rightarrow t_i = t_0 + \Delta t \cdot i, \quad i = 0, 1, 2, \dots, n$$

$$u(t) \rightarrow u(t_i) \equiv u_i$$

Required parameters:

- t_0 – beginnig of simulation, usually $t_0=0$
- t_{max} – the end of simulation
- n – number of time steps
- Δt – width of time step
- u_0 – initial condition

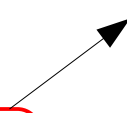


Forward (explicit) Euler method

If we know the solution at time t_i (u_i) we may try to estimate the solution at next node u_{i+1} using **Taylor series** expansion

$$u(t_i + \Delta t) = u(t_i) + \left. \frac{du}{dt} \right|_{t_i} \Delta t + \underbrace{\frac{1}{2!} \left. \frac{d^2 u}{dt^2} \right|_{t_i} \Delta t^2 + \frac{1}{3!} \left. \frac{d^3 u}{dt^3} \right|_{t_i} \Delta t^3 + \dots}_{\text{truncated terms}}$$

$\frac{du}{dt} = f(t, u)$



after substituting first derivative by r.h.s. from ODE and truncating the series after first order term we get

$u_{i+1} = u_i + \Delta t \cdot f(t_i, u_i) + O(\Delta t^2)$

- forward Euler formula

$O(\Delta t^2)$ - local (truncation) error

Remark: truncation makes sense provided that Δt is small, namely under condition

$$\Delta t \ll 1$$

under such condition, the largest contribution to local error comes from the truncated term of lowest order in Δt

- forward Euler method is **explicit** one, it means that the values of all terms on its r.h.s. are known, hence allowing one to compute the value of u_{i+1} (l.h.s.) directly
- besides explicit schemes there are implicate ones, which don't have this feature

Local error vs. global error, order of method

All difference schemes are characterized with their own **local (truncation) errors**

$$e_{loc} \sim O(\Delta t^{p+1})$$

Solving ODE with any of finite difference based method is an iterative process

$$\text{START: } u_0 \xrightarrow{\Delta t} u_1 + O(\Delta t^{p+1}) \xrightarrow{\Delta t} u_2 + O(\Delta t^{p+1}) \xrightarrow{\Delta t} \dots \xrightarrow{\Delta t} u_n + O(\Delta t^{p+1}) \quad \text{:STOP}$$

and during this process we shall expect the local errors will accumulate at least partially increasing the global error.

Global error is the difference between (usually) unknown exact solution $u(t_n)$ and that obtained numerically u_n

$$E_{glob} = u(t_n) - u_n$$

$u(t_n)$ - exact solution

u_n - numerical solution

By assuming the worst-case scenario, the local errors would accumulate instead partially cancelling themselves, the **global error** can be easily estimated

$$E_{glob} \approx n \cdot e_{loc} = \frac{t_{max}}{\Delta t} O(\Delta t^{p+1}) \approx O(\Delta t^p)$$

p – order of difference method

- factor **p** defines the order of the method,
the higher the value of p is, the better, i.e. more accurate, is the difference method

Remark: the order **p** of difference method is always smaller than the local error (**p+1**)

Order of method vs. global error

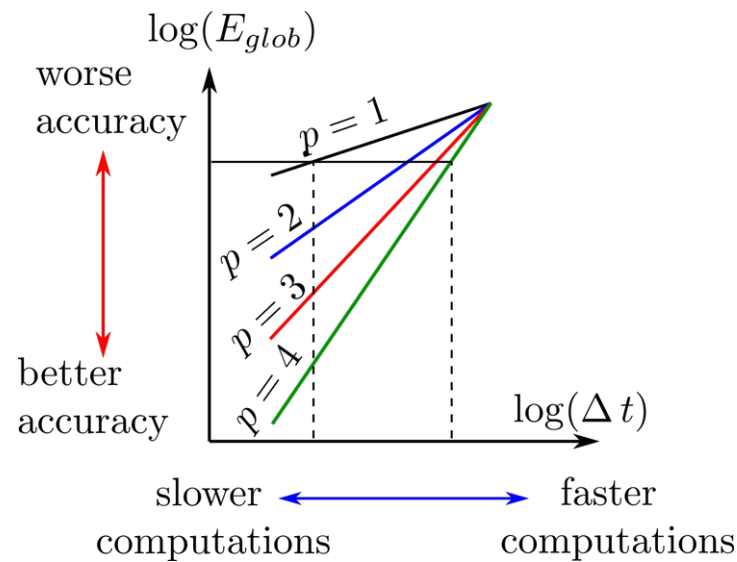
$$E_{glob} = C_p \cdot \Delta t^p, \quad C_p - \text{constant}$$

$$/ \cdot \log(\dots) \quad \frac{E_{glob}}{C_p} = \Delta t^p$$

$$\log\left(\frac{E_{glob}}{C_p}\right) = p \cdot \log(\Delta t)$$

$$\log(E_{glob}) = \log(C_p) + p \cdot \log(\Delta t)$$

In the log-log plot above expression is linear with slope depending on factor p



Remark: due to much better computational efficiency the use of the higher order methods is recommended

- the global error defines also the rate of convergence of numerical data to the exact solution when Δt decreases
- Euler difference scheme is the 1-st order method i.e. the lowest order method among those used to solve ODE
- the time step Δt plays crucial role in the iterative process of solving ODE, how to choose its optimal value?

We shall equilibrate two important factors:

- (i) smaller Δt gives better results
- (ii) smaller Δt means more steps need to be taken to get the $t_{\max}$ point

There are no general rules which allows to estimate an optimal value.

As a rule of thumb, the first guess of time step would be

$$\Delta t_{guess} = \frac{t_{max}}{100}$$

next one must experiment with smaller values

$$\frac{\Delta t_{guess}}{2}, \frac{\Delta t_{guess}}{5}, \frac{\Delta t_{guess}}{10}, \frac{\Delta t_{guess}}{20}, \frac{\Delta t_{guess}}{50}, \frac{\Delta t_{guess}}{100}, \dots$$

until for two consecutive time steps the solutions converge at some acceptable level of accuracy.

- pseudocode of **Forward Euler method**

```
input parameters:
    n, t0, tmax, u0
initialization:
     $\Delta t \leftarrow \frac{t_{max} - t_0}{n}$ 
    u  $\leftarrow$  u0
    t  $\leftarrow$  t0
iteration:
    for i=1 to n do
        u  $\leftarrow$  u +  $\Delta t \cdot f(t, u)$ 
        t  $\leftarrow$  t +  $\Delta t$ 
    end do
```

Numerical stability

Stability is the basic property we required from the difference scheme.

Stable method generates the bounded numerical results irrespective of the number of time steps

$$\exists \Delta t \in \mathbb{R} \implies \lim_{n \rightarrow \infty} u_n < \infty$$

Obviously stability can not depend on the method itself solely but also on the r.h.s. of ODE.

Therefore in order to standardize the considerations on stability, it is often considered for **the linear autonomous ODE problem** which has simple analytical solution

$$\frac{du}{dt} = f(u) \quad \text{- autonomous problem (no time dependence), r.h.s. depends solely on the solution}$$

linear autonomous problem

$$\frac{du}{dt} = \lambda u, \quad \lambda \in \mathbb{C}$$

has exact solution

$$u(t) = e^{\lambda t}$$

Forward Euler method applied to autonomous problem gives

$$u_{i+1} = u_i + \Delta t \cdot f(t_i, u_i) = u_i + \lambda \Delta t \cdot u_i, \quad (f(t, u) = \lambda u)$$

and rewriting this expression in more compact form

$$u_{i+1} = (1 + \lambda \Delta t) u_i$$

let's introduce new variable z

$$z = \lambda \Delta t, \quad z \in \mathbb{C}$$

and define the stability function $R(z)$ for forward Euler

$$R(z) = 1 + z$$

$$u_{i+1} = R(z) u_i$$

- $R(z)$ amplifies the solution from previous time

Because the value of $R(z)$ is fixed for given Δt and λ , one may expand the solution by rewriting it so as to make it dependent on the initial value u_0

$$u_{i+1} = R(z) u_i = R^2(z) u_{i-1} = \dots = R^i(z) u_0$$

The last expression implies that the stability is guaranteed, provided that the following condition is met

$$\left| \frac{u_{i+1}}{u_i} \right| = |R(z)| \leq 1$$

- stability condition

We get the stability condition. Let's analyze it for the Forward Euler method

$$|R(z)| = |1 + z| \leq 1, \quad z \in \mathbb{C}, \quad z = \operatorname{Re}(z) + i \cdot \operatorname{Im}(z)$$

$$|1 + z| = \sqrt{(1 + \operatorname{Re}(z))^2 + (\operatorname{Im}(z))^2} \leq 1$$

$$(\operatorname{Re}(z) + 1)^2 + (\operatorname{Im}(z))^2 \leq 1$$

let's substitute new variables

$$x = \operatorname{Re}(z), \quad x_0 = -1$$

$$y = \operatorname{Im}(z), \quad y_0 = 0$$

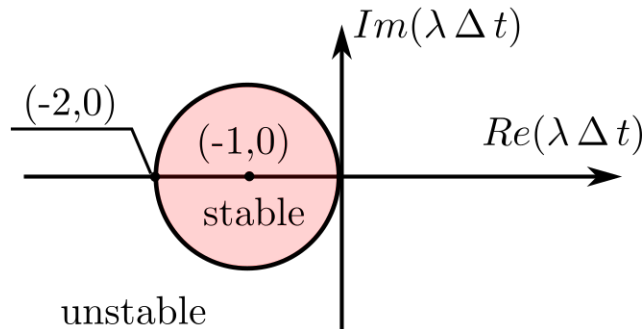
circle

$$(x - x_0)^2 + (y - y_0)^2 \leq 1$$

and rewrite this condition with new variables

$$(x - x_0)^2 + (y - y_0)^2 \leq 1 \quad - \text{it's an inner part of circle which has the radius 1 and is centered at } (-1, 0)$$

stability region of Forward Euler



because: $\Delta t > 0$

stability requires: $\operatorname{Re}(\lambda) < 0$

Example: autonomous problem solved with forward Euler method

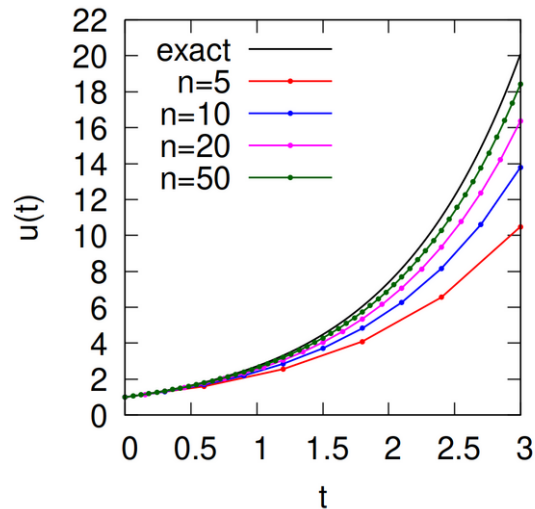
$$\frac{du}{dt} = \lambda \cdot u$$

- $\lambda > 0$, conditionally stable solution, only for finite number of iterations

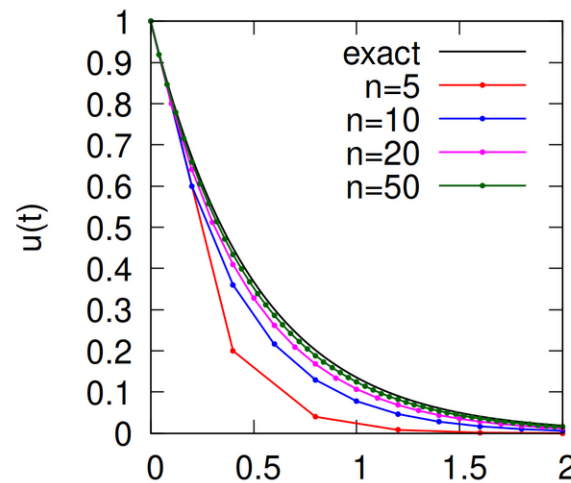
- $\lambda < 0$, unconditionally stable solution if $-2 < z < 0$

- $\lambda < 0$, unstable solution if $z < -2$

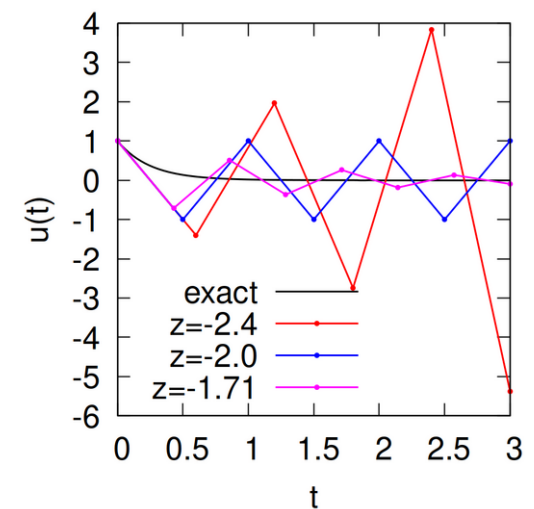
Forward Euler: $\lambda = +1$



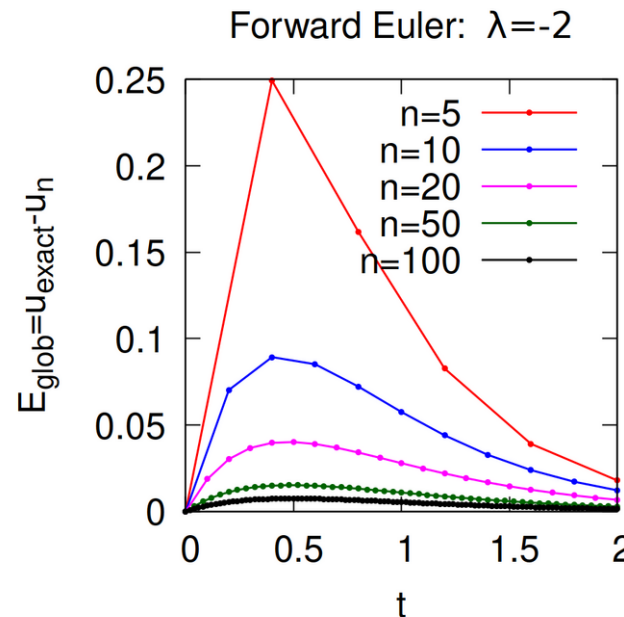
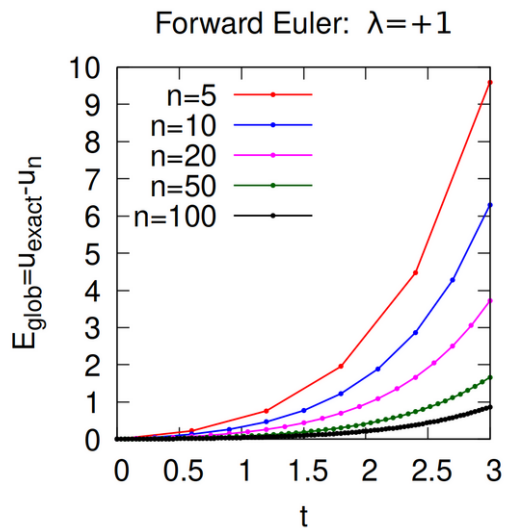
Forward Euler: $\lambda = -2$



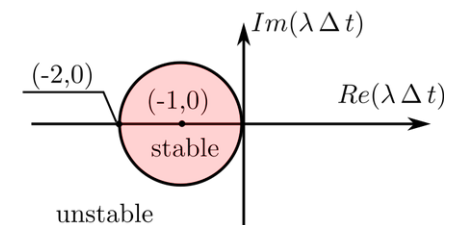
Forward Euler: $\lambda = -4$



Remark: stability doesn't guarantee the results are accurate, these can be completely defective



stability region of Forward Euler



Backward (implicit) Euler method

We have convinced ourselves that the problem of large local error in forward Euler method is a result of fixing the value of first order derivative at the beginning of a next step. Consequently for strongly nonlinear solution $u(t)$, the predicted numerical value at next time step might be over- or under-estimated.

Here we may pose a question:

What will happen if instead of present value of derivative (r.h.s. of ODE) we use its value in the next step?

This question leads us straightforwardly to the definition of the **Backward Euler Method**, which again we derive from the Taylor series but using the **backward** expansion of the solution

$$u(t_i) = u(t_{i+1}) - \underbrace{\frac{du}{dt} \Big|_{t_{i+1}}}_{=f(t_{i+1}, u_{i+1})} \Delta t + \underbrace{\frac{1}{2} \frac{d^2u}{dt^2} \Big|_{t_{i+1}} \Delta t^2 + \dots}_{\text{truncation}}$$

after rearrangement of terms, truncation of higher order terms and replacing derivative with r.h.s. of ODE we get **Backward Euler formula**

$$u_{i+1} = u_i + \Delta t \cdot f(t_{i+1}, u_{i+1}) + O(\Delta t^2)$$

Remarks:

- the local (truncation) error has the same order as forward formula
- the method is **implicit**, in order to find the solution at next time step we must put it into r.h.s.

We can solve this problem with:

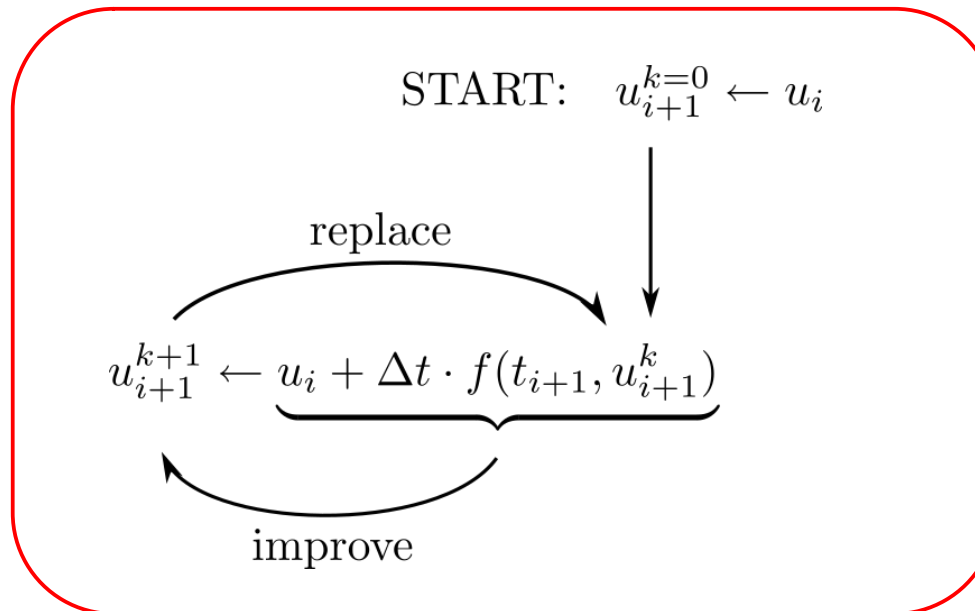
- (1) **Picard iteration**
- (2) transform equation into the nonlinear one and find its solution (u_{i+1}) using the **secant or Newton method**

Picard iteration for implicit method

Picard method relies on assumption that the solution in next step u_{i+1} shouldn't differ much from a present one u_i , therefore we may find the solution by following the steps below

- 1) set $u_{i+1} \leftarrow u_i$
- 2) modify u_{i+1} by computing the l.h.s. of the backward formula
- 3) put the modified u_{i+1} to the l.h.s.
- 4) repeat steps (2) and (3) iteratively until u_{i+1} will stop changing
(~5-20 Picard iterations are usually needed)

convergence criterion: $|u_{i+1}^{k+1} - u_{i+1}^k| < \varepsilon$ ε – small number
 k – index of Picard iteration



- pseudocode for the **Backward Euler** method combined with **Picard iteration**

```

input parameters:
    n, t0, tmax, u0, ε, kmax
initialization:
    Δt ←  $\frac{t_{max} - t_0}{n}$ 
    u ← u0
    t ← t0
iteration:
    for i=1 to n do
        t ← t + Δt
        uold ← u
        for k=1 to kmax do
            unew ← u + Δt · f(t, uold)
            if (|unew - uold| < ε) then
                u ← unew
                break
            end if
            uold ← unew
        end do
        send to output: t, u
    end do

```

Backward Euler scheme combined with the Secant/Newton method

First, the difference scheme shall be rearranged to form a function of unknown variable u_{i+1}

$$u_{i+1} = u_i + \Delta t \cdot f(t_{i+1}, u_{i+1}) + O(\Delta t^2)$$

nonlinear function:

$$F(u_{i+1}) = u_{i+1} - u_i - \Delta t \cdot f(t_{i+1}, u_{i+1})$$

We look for such value of u_{i+1} which gives $F(u_{i+1})=0$. Making one step in Backward Euler method requires employing one of the methods devised to finding the roots of a nonlinear function: **Newton** or **Secant** method.

The choice of one of these two methods depends on that, whether derivative of f w.r.t the variable u_{i+1} can be calculated analytically or not:

- $f(t,u)$ is given as an analytical expression, then **Newton iterative method** is recommended (order: $p=2$)

- general formula

$$u_{i+1}^{k+1} = u_{i+1}^k - \frac{F(u_{i+1}^k)}{\left. \frac{\partial F(u_{i+1})}{\partial u_{i+1}} \right|_{u_{i+1}^k}}$$

- working formula

$$u_{i+1}^{k+1} = u_{i+1}^k - \frac{u_{i+1} - u_i - \Delta t \cdot f(t_{i+1}, u_{i+1})}{1 - \Delta t \left. \frac{\partial f(t,u)}{\partial u} \right|_{u=u_{i+1}^k}}$$

- when the derivative of $f(t,u)$ is not known (e.g. the expression for f is not known) or its calculation is difficult, then the **Secant method** is used (order: $p=1.618$). This iterative process requires two starting points

- general formula

$$u_{i+1}^{k+1} = u_{i+1}^k - \frac{F(u_{i+1}^k) (u_{i+1}^k - u_{i+1}^{k-1})}{F(u_{i+1}^k) - F(u_{i+1}^{k-1})}$$

- example starting points

$$\begin{aligned} u_{i+1}^{k=0} &\leftarrow u_i \\ u_{i+1}^{k=1} &\leftarrow 1.01 \cdot u_i \end{aligned}$$

pseudocode for Backward Euler method with embedded Secant method

```

input  parameters:
    n, t0, tmax, u0, ε, kmax
initialization:
    Δt ←  $\frac{t_{max}-t_0}{n}$ 
    u ← u0
    t ← t0
iteration:
    for i=1 to n do
        t ← t + Δt
        starting points for Secant method:
            ui+10 ← ui
            ui+11 ← ui + Δt f(ti+1, ui+10)
        Secant iteration:
            for k=1 to kmax do
                F(ui+1k-1) ← ui+1k-1 - ui - Δt f(ti+1, ui+1k-1)
                F(ui+1k) ← ui+1k - ui - Δt f(ti+1, ui+1k)
                ui+1k+1 ← ui+1k -  $\frac{F(u_{i+1}^k)(u_{i+1}^k - u_{i+1}^{k-1})}{F(u_{i+1}^k) - F(u_{i+1}^{k-1})}$ 
                if (|ui+1k+1 - ui+1k| < ε) then
                    u ← ui+1k+1
                    break
                end if
            end do
        end do
        send to output: t, u
    end do

```

Stability of Backward Euler Method

Stability analysis proceeds the same way as for the forward Euler formula for the linear autonomous ODE problem

$$\frac{du}{dt} = \lambda u, \quad \lambda \in \mathbb{C}$$

$$u_{i+1} = u_i + \lambda \Delta t u_{i+1}$$

$$(1 - \lambda \Delta t) u_{i+1} = u_i$$

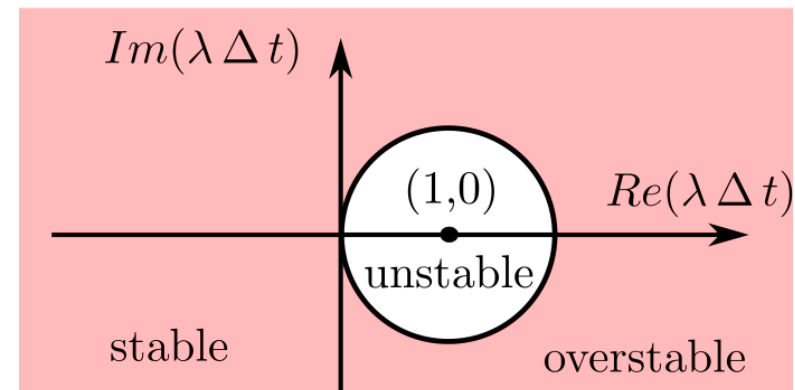
$$u_{i+1} = \frac{1}{(1 - \lambda \Delta t)} u_i$$

$$R(z) = \frac{1}{1 - z}, \quad z = \lambda \Delta t$$

stability condition:

$$|R(z)| = \frac{1}{|1 - z|} \leq 1 \quad \Rightarrow \quad 1 \leq |1 - z|$$

stability region of Backward Euler



- stability region extends outside the circle of unit radius centered at $z=(1,0)$

Trapezoid method

Derivation of difference formula again is based on Taylor series, but now we combine forward and backward expansions

$$(1) \quad u(t_i + \Delta t) = u(t_i) + \left. \frac{du}{dt} \right|_{t_i} \Delta t + \frac{1}{2!} \left. \frac{d^2 u}{dt^2} \right|_{t_i} \Delta t^2 + \frac{1}{3!} \left. \frac{d^3 u}{dt^3} \right|_{t_i} \Delta t^3 + \dots$$

$$(2) \quad u(t_i) = u(t_{i+1}) - \left. \frac{du}{dt} \right|_{t_{i+1}} \Delta t + \frac{1}{2} \left. \frac{d^2 u}{dt^2} \right|_{t_{i+1}} \Delta t^2 + \dots$$

$$(2a) \quad u(t_{i+1}) = u(t_i) + \left. \frac{du}{dt} \right|_{t_{i+1}} \Delta t - \frac{1}{2} \left. \frac{d^2 u}{dt^2} \right|_{t_{i+1}} \Delta t^2 + \dots$$

before we add (1) and (2a) we must replace the second order derivative for time step $t=t_{i+1}$

$$\left. \frac{d^2 u}{dt^2} \right|_{t_{i+1}} = \left. \frac{d^2 u}{dt^2} \right|_{t_i} + \left. \frac{d^3 u}{dt^3} \right|_{t_i} \Delta t + O(\Delta t^2)$$

$$(2b) \quad u(t_{i+1}) = u(t_i) + \left. \frac{du}{dt} \right|_{t_{i+1}} \Delta t - \frac{1}{2} \Delta t^2 \left[\left. \frac{d^2 u}{dt^2} \right|_{t_i} + \left. \frac{d^3 u}{dt^3} \right|_{t_i} \Delta t + O(\Delta t^3) \right] + \frac{1}{3!} \left. \frac{d^3 u}{dt^3} \right|_{t_{i+1}} \Delta t^3 + \dots$$

difference scheme for trapezoid method

$$[(1)+(2b)]/2 \Rightarrow u(t_i + \Delta t) = u(t_i) + \frac{\Delta t}{2} [f(t_i, u_i) + f(t_{i+1}, u_{i+1})] + O(\Delta t^3)$$

- implicit as backward Euler but it is 2-order method (higher accuracy)

Stability of trapezoid method

Let's consider linear autonomous problem

$$\frac{du}{dt} = \lambda u$$

$$u_{i+1} = u_i + \frac{\Delta t \lambda}{2} [u_i + u_{i+1}]$$

$$\left(1 - \frac{\Delta t \lambda}{2}\right) u_{i+1} = \left(1 + \frac{\Delta t \lambda}{2}\right) u_i$$

$$z = \lambda \Delta t$$

- stability function

$$R(z) = \frac{u_{i+1}}{u_i} = \frac{1 + \frac{z}{2}}{1 - \frac{z}{2}}$$

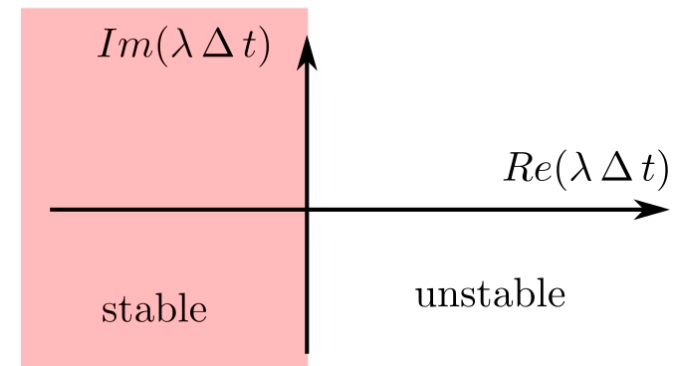
- stability condition

$$|R(z)| \leq 1 \implies |2 + z| \leq |2 - z|$$

- trapezoid difference scheme

$$u(t_i + \Delta t) = u(t_i) + \frac{\Delta t}{2} [f(t_i, u_i) + f(t_{i+1}, u_{i+1})]$$

stability region of trapezoid method

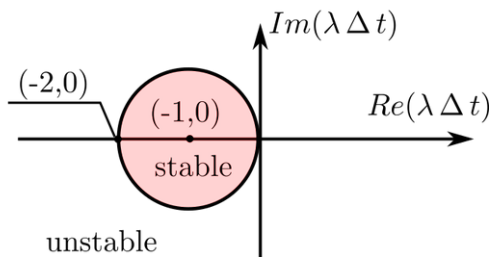


Remark: region of stability is the left half of the complex plane

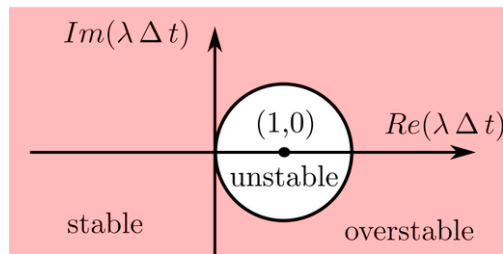
- summary – simple Euler based methods

<i>method</i>	<i>formula</i>	<i>order</i>	$R(z)$	<i>type</i>	<i>step</i>
<i>Forward Euler</i>	$u_{i+1} = u_i + \Delta t f(t_i, u_i)$	1	$1 + z$	<i>explicit</i>	<i>simple substitution</i>
<i>Backward Euler</i>	$u_{i+1} = u_i + \Delta t f(t_{i+1}, u_{i+1})$	1	$\frac{1}{1-z}$	<i>implicit</i>	<i>Picard Newton Secant</i>
<i>Trapezoid</i>	$u_{i+1} = u_i + \frac{\Delta t}{2} [f(t_i, u_i) + f(t_{i+1}, u_{i+1})]$	2	$\frac{1+\frac{z}{2}}{1-\frac{z}{2}}$	<i>implicit</i>	<i>Picard Newton Secant</i>

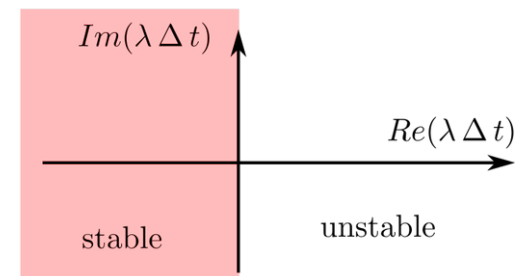
stability region of Forward Euler



stability region of Backward Euler



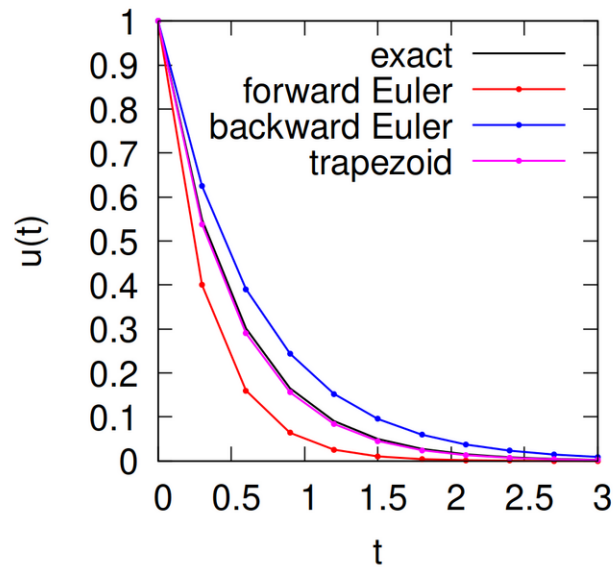
stability region of trapezoid method



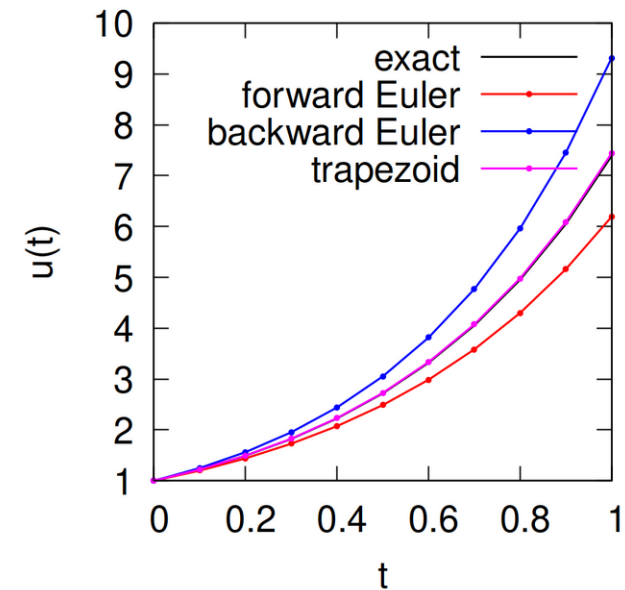
- example: linear autonomous problem

$$\frac{du}{dt} = \lambda u$$

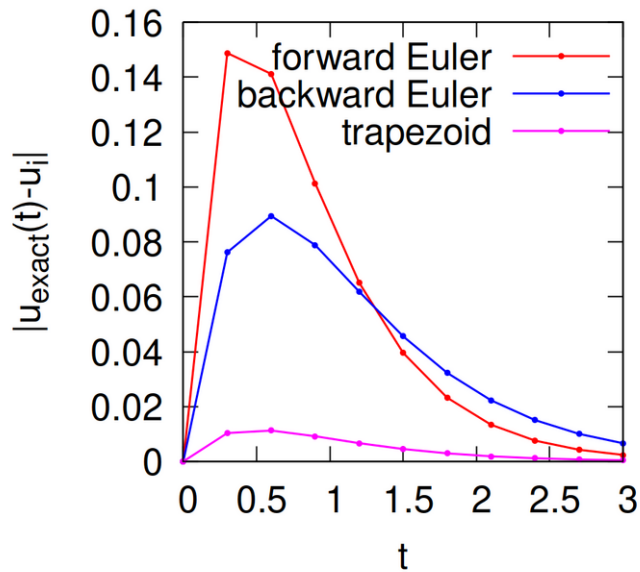
$\lambda = -2, n=10$



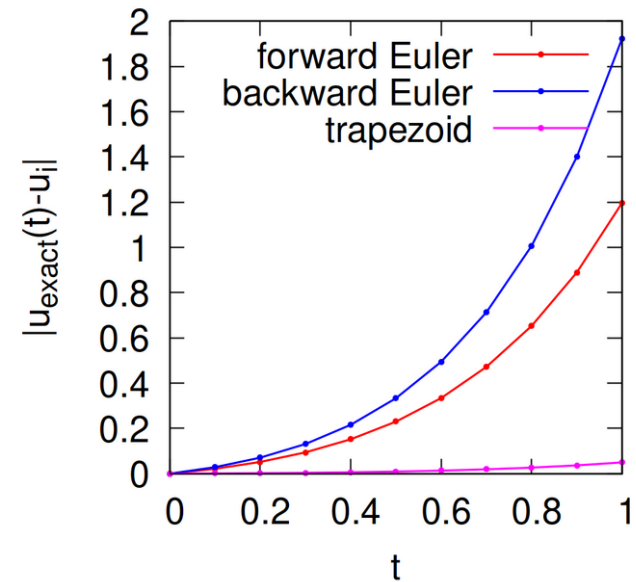
$\lambda = +2, n=10$



$\lambda = -2, n=10$



$\lambda = +2, n=10$



Solving n-th order ODE with numerical methods devised for 1-st order ODE

- first, let's consider simple but nontrivial case, i.e. **2-nd order ODE**

$$\frac{d^2u}{dt^2} + a(t, u) \left(\frac{du}{dt} \right)^p + b(t) (u)^q + c(t) = 0, \quad p, q, \in \mathbb{N}, \quad \mathbb{R} : (u, t) \rightarrow a(t, u), \quad \mathbb{R} : t \rightarrow c(t)$$

- to find unique solution we need 2 **initial conditions**: $u(t_0), u_t(t_0) = \left. \frac{du}{dt} \right|_{t=t_0}$
- in this example ODE is nonlinear i.e. the function u may enter the equation with power higher than 1, this is just a little generalization, it doesn't make any difference for the following algorithm.
- let's rearrange the terms slightly, leaving the highest order derivative on the left side

$$\frac{d^2u}{dt^2} = -a(t, u) \left(\frac{du}{dt} \right)^p - b(t) (u)^q - c(t) \quad \Longrightarrow \quad \begin{cases} \frac{d^2u}{dt^2} = f \left(t, u, \frac{du}{dt} \right) \\ f(t, u, w) = -a(t, u)w^p - b(t) (u)^q - c(t) \end{cases}$$

- now we introduce **new auxiliary variable $w(t)$**

$$u = u(t) \quad \leftarrow \text{final solution}$$

2-nd order ODE
transformed to
system of
1-st order ODE

$\longleftrightarrow$

$$\begin{cases} \frac{du}{dt} = w(t) \\ \frac{dw}{dt} = \frac{d^2u}{dt^2} = f(t, u, w) \end{cases}$$

$\leftarrow$ definition of auxiliary variable

 $\leftarrow$ 2-nd order derivative change into
1-st order of auxiliary variable

- 2-nd order ODE was transformed into a system of two coupled 1-st order ODE
- now the problem might be solved with one of the methods devised to solving the 1-st order ODE

As an example we may consider **Forward Euler method**

$$\begin{cases} \frac{du}{dt} = w(t) \\ \frac{dw}{dt} = \left(\frac{d^2u}{dt^2} \right) = f(t, u, w) \end{cases} \implies \begin{cases} u_{i+1} = u_i + \Delta t \cdot w_i \\ w_{i+1} = w_i + \Delta t \cdot f(t_i, u_i, w_i) \end{cases}$$

input parameters:

$n, t_0, t_{max}, u_0, w_0$

initialization:

$\Delta t \leftarrow \frac{t_{max} - t_0}{n}$

$u \leftarrow u_0$

$w \leftarrow w_0$

$t \leftarrow t_0$

Forward Euler iteration:

for $i=1$ to n do

$u_{old} \leftarrow u$

$w_{old} \leftarrow w$

$u \leftarrow u + \Delta t \cdot w_{old}$

$w \leftarrow w + \Delta t \cdot f(t, u_{old}, w_{old})$

$t \leftarrow t + \Delta t$

send to output: t, u, w

end do

- now we may consider the transformation of **n-th order ODE** into a system of coupled 1-st order ODE

$$\frac{d^n u}{dt^n} = f\left(t, u, \frac{du}{dt}, \frac{d^2 u}{dt^2}, \dots, \frac{d^{n-1} u}{dt^{n-1}}\right)$$

+ n initial conditions:

$$u(t_0), \left. \frac{du}{dt} \right|_{t=t_0}, \dots, \left. \frac{d^{n-1} u}{dt^{n-1}} \right|_{t=t_0}$$

- this transformation needs (n-1) new auxiliary variables
- instead of using n different symbols it is better to work with a single symbol with enumerated index being prescribed to particular variable

$$u(t) \equiv u^0(t)$$

$$\left\{ \begin{array}{ll} \frac{du^0}{dt} = u^1(t) & \left(= \frac{du}{dt} \right) \\ \frac{du^1}{dt} = u^2(t) & \left(= \frac{du^2}{dt^2} \right) \\ \frac{du^2}{dt} = u^3(t) & \left(= \frac{d^3 u}{dt^3} \right) \\ \dots & \\ \frac{du^{n-1}}{dt} = f(t, u^0, u^1, \dots, u^{n-1}) & \left(= \frac{d^n u}{dt^n} \right) \end{array} \right.$$

Remarks:

- the upper index k determines the order of derivative (auxiliary functions)
- now we might look for the solution in a vector form (simpler notation)

$$\vec{u} = [u^0, u^1, u^2, \dots, u^{n-1}]$$

Let's rewrite the system of 1-st order ODE in a vector form – the mathematical notation as well as the computer implementation of algorithm will be simpler (especially for implicit methods)

$$\frac{d^n u}{dt^n} = f$$

⇓ (auxiliary variables)

$$\left\{ \begin{array}{l} \frac{du^0}{dt} = u^1(t) = f_0(t, \vec{u}) \\ \frac{du^1}{dt} = u^2(t) = f_1(t, \vec{u}) \\ \frac{du^2}{dt} = u^3(t) = f_2(t, \vec{u}) \\ \dots \\ \frac{du^{n-1}}{dt} = f(t, u^0, u^1, \dots, u^{n-1}) = f_{n-1}(t, \vec{u}) \end{array} \right.$$

⇓ (vector notation)

$$\left\{ \begin{array}{l} \vec{u} = [u^0, u^1, u^2, \dots, u^{n-1}] \\ \vec{f} = [f^0, f^1, f^2, \dots, f^{n-1}] \\ \frac{d\vec{u}}{dt} = \vec{f}(t, \vec{u}) \end{array} \right.$$

Pseudocodes for solving the system of 1-st order ODE with Euler methods, modifications rely on replacing the single variable by 1-d arrays (vectors)

- Forward Euler

```

input parameters:
  n, t0, tmax,  $\vec{u}_0 = \vec{u}(t_0)$ 
initialization:
   $\Delta t \leftarrow \frac{t_{max} - t_0}{n}$ 
   $\vec{u} \leftarrow \vec{u}_0$ 
  t  $\leftarrow$  t0
Forward Euler iteration:
  for i=1 to n do
     $\vec{u} \leftarrow \vec{u} + \Delta t \cdot \vec{f}(t, \vec{u})$ 
    t  $\leftarrow$  t +  $\Delta t$ 
    send to output: t,  $\vec{u}$ 
  end do
  
```

- Backward Euler + Picard iteration

```

input parameters:
  n, t0, tmax,  $\vec{u}_0 = \vec{u}(t_0)$ ,  $\varepsilon$ , kmax
initialization:
   $\Delta t \leftarrow \frac{t_{max} - t_0}{n}$ 
   $\vec{u} \leftarrow \vec{u}_0$ 
  t  $\leftarrow$  t0
Backward Euler iteration:
  for i=1 to n do
    t  $\leftarrow$  t +  $\Delta t$ 
     $\vec{u}_{old} \leftarrow \vec{u}$ 
    for k=1 to kmax do
       $\vec{u}_{new} \leftarrow \vec{u} + \Delta t \cdot \vec{f}(t, \vec{u}_{old})$ 
      if ( $\|\vec{u}_{new} - \vec{u}_{old}\|_2 < \varepsilon$ ) then
         $\vec{u} \leftarrow \vec{u}_{new}$ 
        break
      end if
       $\vec{u}_{old} \leftarrow \vec{u}_{new}$ 
    end do
    send to output: t,  $\vec{u}$ 
  end do
  
```