Numerical Methods

Bart Vorselaars

February 2, 2026

Contents

1 Numerical Methods - MTH3007

Lecturer second term: B. Vorselaars

- 1.5 hours/week
- 15 credit module: 150 learning hours. Approx. **6 hours/week**.
- If anything is unclear, please ask questions (during or after the lab session)

1.1 Assignments

The assignments follows a similar structure as in the first term.

- Weekly exercises. They don't count towards your mark for this module, but are *essential* in the learning process and for succesfully completing the final in-class test. These need to be completed since the developed code can be re-used during the in-class test.
- One of them will be marked, but it counts for 0%. It is expected that this is completed before the next session, and a pass/fail mark will be given to them during the lab session. Please raise your hand during the lab session and me or a demonstrator will look whether the question has been answered to a good standard. If not, you have another opportunity.
  - Deadline this term: TBC
- Note: the students that passed the last one had a factor 8 less chance of failing the final-in-class test!
- **Final in-class test**: date is *12/5/2026* (TBC). This counts 50% towards the final mark. You are allowed to access Blackboard and your own set of notes and programme codes (e.g., as part of your logbook).

1.2 Outline syllabus second term

- Numerical solution of *ordinary differential equations*
- Numerical solution of *partial differential equations*
- Elements of *stochastic methods*

These require more background knowledge, hence lecturing part may typically be longer than during the first term.

1.3 Learning outcomes

1. Apply numerical methods for curve fitting and solving various equations
2. Critically analyze applied mathematical problems, choose and apply appropriate numerical methods for their solution
3. Implement numerical solutions in efficient computer codes using a high level computer language

1.4 Literature

A large part of the syllabus is covered in:

- *Numerical Methods for Engineers*, 7th edition, by S. C. **Chapra** and R. P. **Canale**, McGraw-Hill Education, New York, USA (2015)

Some elements are covered in:

- *An Introduction to Partial Differential Equations*, by Y. **Pinchover** and J. **Rubinstein**, Cambridge University Press, Cambridge, UK (2005)
- *Numerical Methods for Engineers and Scientists*, 2nd edition, by J. D. **Hoffman**, CRC Press, Boca Raton, USA (2001)

- *Numerical Recipes: The Art of Scientific Computing*, 3rd edition, by W. H. **Press**, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, Cambridge University Press, Cambridge, UK (2007)
- *Advanced Engineering Mathematics*, by E. **Kreyszig**, John Wiley and Sons, USA (2006 or 2011)

Furthermore, some images are taken from Wikipedia.

1.5 Lecture notes

In the lecture notes some concepts are **bold**. Apart from the author names in the previous slide these ideally have to be learned by heart. They also appear in the index of the lecture notes (at the end).

2 Algorithms and programming languages

We will be mostly using Python (either pure, or within a Jupyter notebook) to present the algoritms.

For the algorithms that I present in this term, a fast programming language is important (for practical calculations), but not needed for learning the algorithms. Therefore I strongly recommend to use a higher-level programming language such as Python or Matlab, since these languages also allow you analyze the algorithms more straightforwardly as opposed to, e.g., C or C++.

2.1 Introduction to Python

Several websites provide basic introductions to Python - https://www.w3schools.com/python/default.asp - https://www.codecademy.com/learn/learn-python-3 - https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-1-python-basics?ex=1

An introduction to Python using a game is https://codecombat.com/

A slightly more advanced python introduction is the tutorial on http://codingbat.com/python

https://nbviewer.jupyter.org/github/barbagroup/CFDPython/blob/master/lessons/00_Quick_Python_Intro.ipynb is an introduction to Python with numerical methods in mind.

2.2 Accessing Python

The following popular options are available: - Sypder (IDE for Python, allows plotting too) can be downloaded and installed on your own computer via https://www.spyder-ide.org/. - Visual Studio Code (IDE for many programming languages, including Python). See https://code.visualstudio.com/.

There are also the following *online* options: - Ipython via https://jupyterlite.readthedocs.io/en/stable/try/tree (no sign-up required) - Ipython via https://cocalc.com/ (sign-up required) - Python with plotting using https://trinket.io/embed/python3/a5bd54189b - Python without plotting https://repl.it/languages/python3

3 Approximations and errors

In Numerical Methods it is important to know the accuracy of your method.

Some concepts that play a role are the following

3.1 Order of magnitude

Definition: A function $f(h)$ is said to be the **order of magnitude** $g(h)$ *as* $h \to 0$, where $g(h)$ is a nonnegative function, if

$$\lim_{h\to 0} \frac{f(h)}{g(h)} = \text{constant}$$

where the constant is a finite number (which could also be zero).

This is written in the O (aka '**big O**') notation as $f(h) = O(g(h))$ $(h \to 0)$.

Typically $g(h) = h^n$.

Often the limit value for h (in this case zero, since $h \to 0$) is not mentioned explicitly but is clear from the context. Note that the limit value could also be a non-zero number $(h \to a)$ or even ∞ $(h \to \infty)$.

3.1.1 Example

The Taylor series of the cosine function $\cos(x)$ around $x = 0$ is

$$1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 + \ldots + \frac{(-1)^n}{(2n)!}x^{2n} + \ldots$$

Hence the difference between $\cos(x)$ and $1 - \frac{1}{2}x^2$ around $x = 0$ is $\frac{1}{24}x^4 + \ldots$. In the limit of $x \to 0$ the leading order term is $\frac{1}{24}x^4$ so one can write

$$\cos(x) = 1 - \frac{1}{2}x^2 + O(x^4)$$

since

$$\lim_{x\to 0} \frac{\cos(x) - (1 - \frac{1}{2}x^2)}{x^4} = \frac{1}{24}$$

and $1/24$ is indeed a constant.

The benefit of using the O notation is that one actually doesn't have to determine the constant (which is often not so important anyway), just the fact that there is a constant.

One could even write

$$\cos(x) = 1 - \frac{1}{2}x^2 + O(x^3)$$

since

$$\lim_{x\to 0} \frac{\cos(x) - (1 - \frac{1}{2}x^2)}{x^3} = 0$$

and zero is also a constant.

Note that

$$\begin{aligned}\lim_{x\to 0}\frac{\cos(x)-(1-\frac{1}{2}x^2)}{x^5} &= \lim_{x\to 0}\frac{\frac{1}{24}x^4}{x^5}\\ &= \lim_{x\to 0}\frac{1}{24x}\\ &= \infty\end{aligned}$$

This limit is therefore not a constant but diverges, and hence the remaining term is not $O(x^5)$.

3.2 Absolute and relative error

Given some approximation v_{approx} of a quantity v, the **absolute error** is defined as

$$\epsilon = |v - v_{\text{approx}}|$$

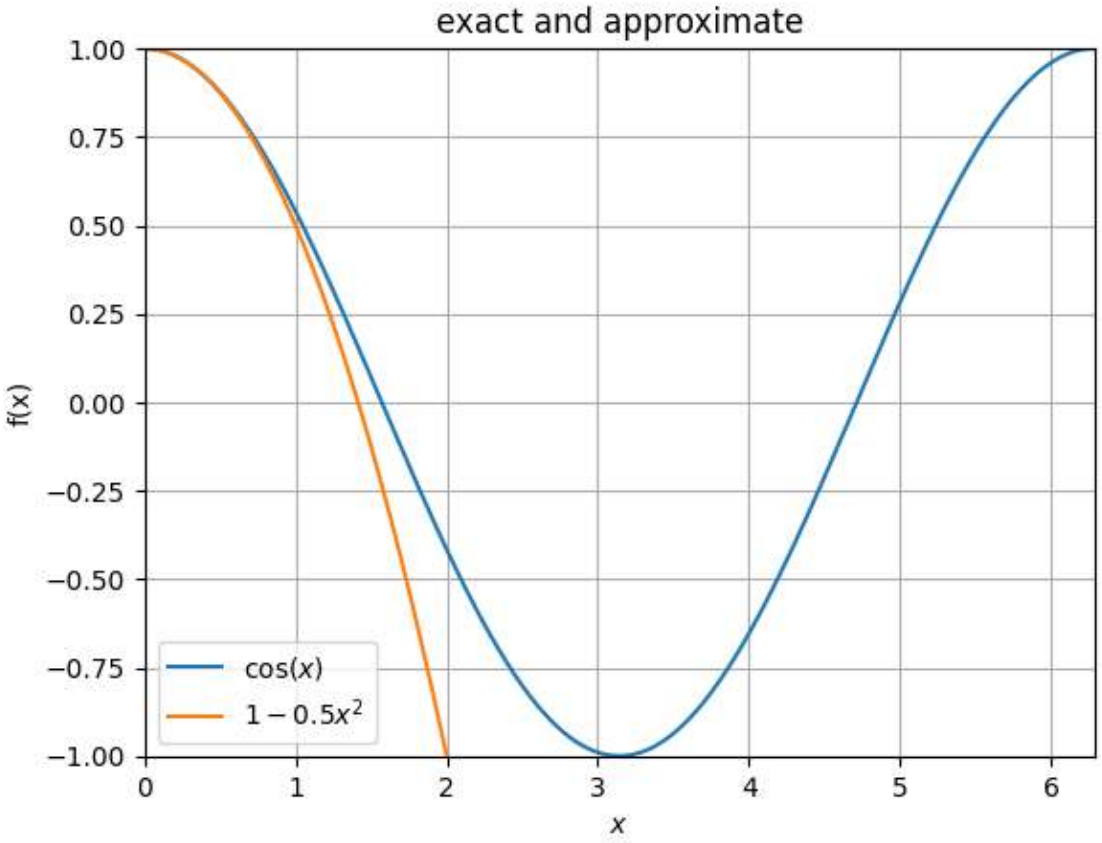
and the **relative error**

$$\eta = \frac{|v - v_{\text{approx}}|}{|v|} = \frac{\epsilon}{|v|}$$

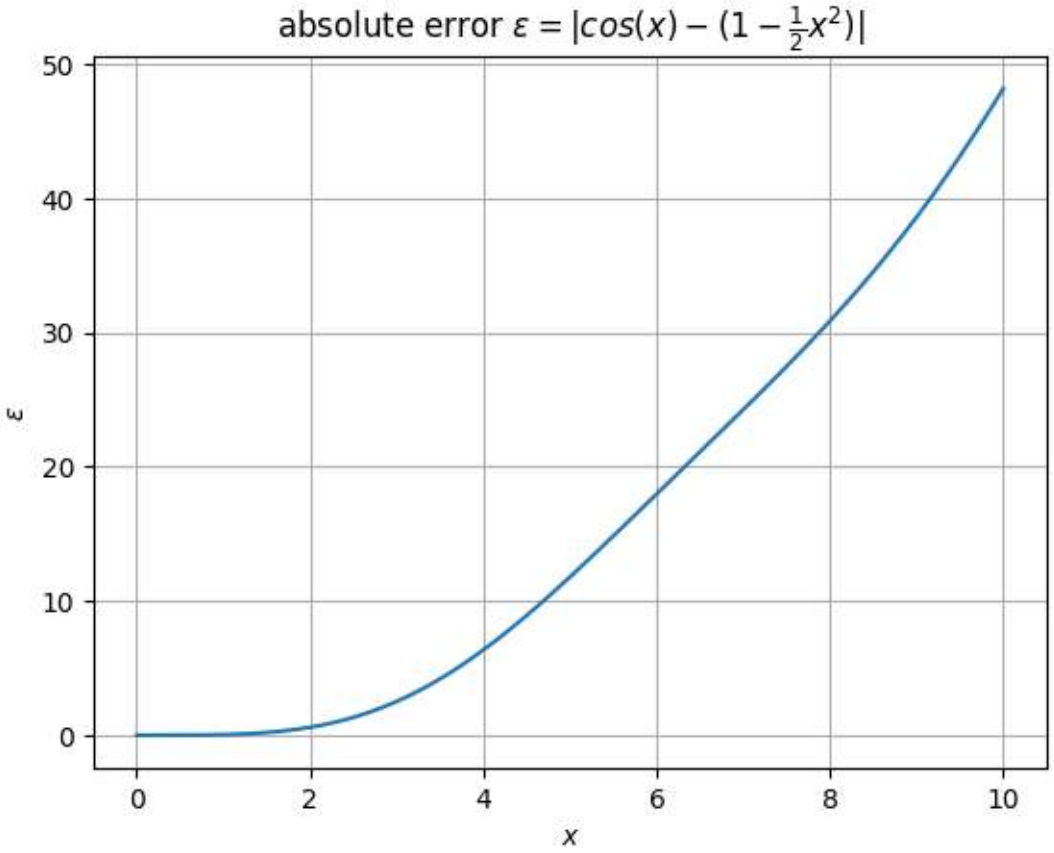
The relative error only works for non-zero values of v.

3.2.1 Example

For the approximation of the $\cos(x)$ by $1-\frac{1}{2}x^2$ the absolute error for $x = 0.3$ is 3.36×10^{-4}. In this example the value is close to 1, so the relative error is almost the same; 3.52×10^{-4}.



The difference between the two curves is then the absolute error



If we plot this on a double-logarithmic scale, we can verify the power 4, in x^4, as follows

The first non-zero term for the error is

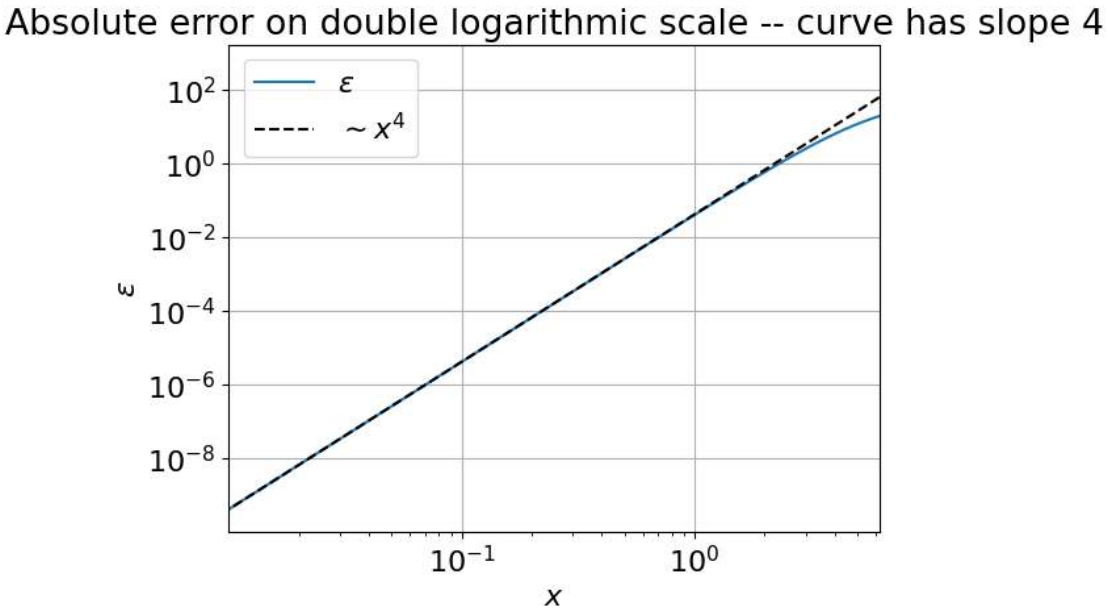
$$\epsilon = \frac{1}{24}x^4$$

and hence

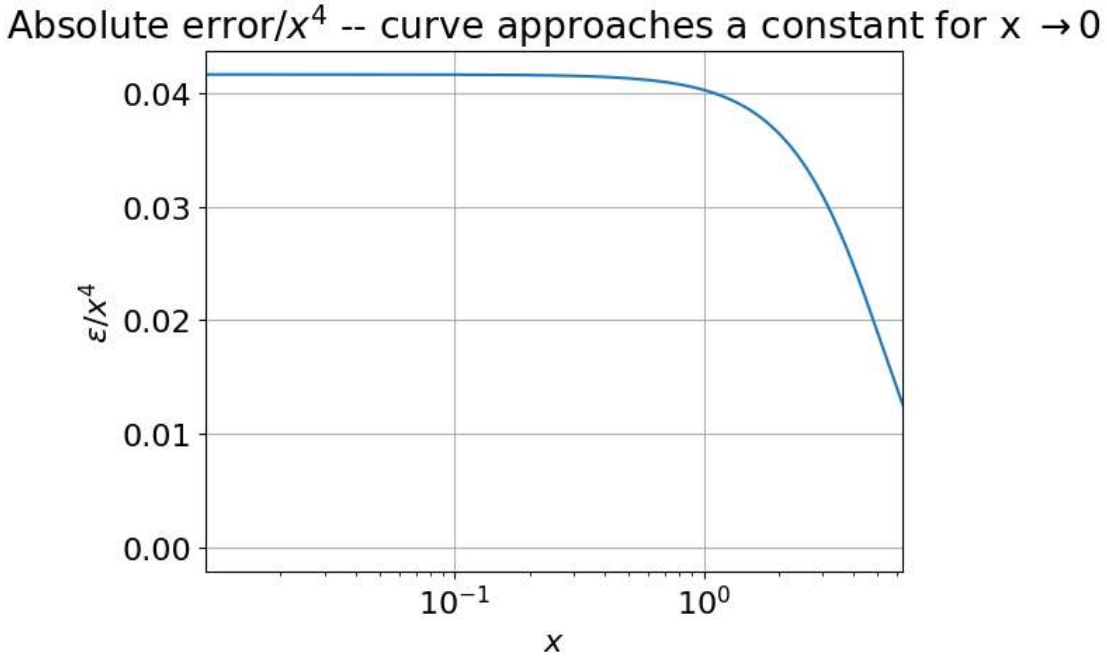
$$\log(\epsilon) = 4\log(x) - \log(24)$$

so if $\log(\epsilon)$ is plotted vs $\log(x)$, the slope is 4, when approaching $x \to 0$.

So $\epsilon = O(x^4)$



So $\epsilon = O(x^4)$



4 Differential equations

See module Differential Equations. Short recap:

Differential equation (DE): An equation involving derivatives of a function, such as $\frac{df(x)}{dx}$

4.1 One vs multiple variables

- **Ordinary differential equation** (ODE): the derivative(s) of the unknown function is with respect to *one* independent variable only.

Example

$$\frac{d^2y(t)}{dt^2} = f(t, y(t))$$

- **Partial differential equation** (PDE): the (partial) derivative(s) of the function is with respect to *multiple* independent variables.

Example

$$\frac{\partial y(t,v)}{\partial t} + \frac{\partial^3 y(t,v)}{\partial v^3} = f(t) + g(v)$$

4.2 Order of a differential equation

The **order** of a differential equation is the highest occuring derivative in the equation.

- Example *first* order ODE:

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

- Examples *second* order ODE:

$$\frac{d^2y(t)}{dt^2} = f\left(t, y(t), \frac{dy(t)}{dt}\right)$$

$$\frac{d^2y(t)}{dt^2} - 27\frac{dy(t)}{dt} = y(t)^3 - 7\exp(t)$$

4.3 Real-world examples

Differential equations appear in many fields:

- *Pure and applied mathematics*: to study their properties, exact solution methods for linear/non-linear equations (Lax pairs), uniqueness, etc. Example: Navier–Stokes existence and smoothness is one of the Millennium Prize Problems by the Clay Mathematics Institute. If you solve it you will win 1 million dollar:

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\nabla p + \nu \Delta \mathbf{v} + \mathbf{f}(\mathbf{x}, t)$$

  with ∇ the gradient and $\Delta \equiv \nabla^2$ the Laplacian.

- *Physics*: Examples are Newton's law, Euler-Lagrange, Hamilton equation, diffusion equation, and the Schrodinger equation:

$$i\hbar \frac{\partial \Psi(\mathbf{r}, t)}{\partial t} = \left(-\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r}, t) \right) \Psi(\mathbf{r}, t)$$

  with $\Psi(\mathbf{r}, t)$ the wave function

- *Chemistry*: Examples are chemical reaction rate equations such as for the simple reaction $A \xrightarrow{k_1} B \xrightarrow{k_2} C$:

$$\begin{aligned} -\frac{dc_A}{dt} &= k_1 c_A \\ \frac{dc_c}{dt} &= k_2 c_B = k_2 (c_A^0 - c_A - c_C) \end{aligned}$$

- *Biology*: Examples are predator-prey (x-y) equations such as

$$\begin{aligned} \frac{dx}{dt} &= ax - bxy \\ \frac{dy}{dt} &= dxy - cy \end{aligned}$$

- *Economics*: Example is the Solow–Swan ODE for modelling capital k

$$\frac{dk(t)}{dt} = f(k(t), t) - \delta\, k(t)$$

- *Finance*: Black-Scholes PDE, for modelling options pricing

$$\frac{\partial V(S,t)}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V(S,t)}{\partial S^2} = rV(S,t) - rS\frac{\partial V(S,t)}{\partial S}$$

  with V the option price and S the stock price.

Main point: more often than not an analytical solution is not possible.

At the end of this term you may be able to solve some or even all of these differential equations numerically (with the help of some extra self-study).

4.4 Side conditions

A differential equation sometimes does not have a unique solution; there could be a family solution with a parameter.

Example

$$\frac{df(x)}{dx} = 0$$

The solution to this DE is $f(x) = c$ with c an arbitrary parameter.

For such DE's often extra **side conditions** are imposed. This will make the solution unique.

Example

$$\begin{aligned} \frac{df(x)}{dx} &= 0 \\ f(0) &= 1 \end{aligned}$$

Now the solution to the first equation is $f(x) = c$. Imposing the extra condition we have $f(0) = 1$, so that the solution is $f(x) = 1$.

4.4.1 Initial value problems

An **initial value problem** (IVP) is a differential equation for which the side conditions are all given *at the same point*. Such side conditions are also known as **initial conditions**.

Example:

$$\begin{aligned} \frac{df(x)}{dx} &= 0 \\ f(0) &= 1 \end{aligned}$$

with solution $f(x) = 1$

Example:

$$\begin{aligned} \frac{d^2f(x)}{dx^2} &= 0 \\ f(0) &= 1 \\ f(0)' &= 1 \end{aligned}$$

with solution $f(x) = c_1x + c_2$ and using the initial conditions this simplifies to $f(x) = 1 + x$. Here the prime, $'$, denotes differentiation.

4.4.2 Classical initial value problems

The classical **initial value problem** is to find a function $y(t)$, which satisfies the differential equation

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

and takes the initial value $y(t_0) = y_0$.

4.4.3 Boundary value problem

The following differential equation is not an intial value problem (IVP), since it has *conditions at two different points*: x_L and x_R

$$\begin{aligned} \frac{d^2 f(x)}{dx^2} &= g(x) \\ f(x_L) &= 0.2 \\ f(x_R)' &= 1.5 \end{aligned}$$

with $x_L \neq x_R$. This is a **boundary value problem**.

5 Finite difference method

Differential equations consist of derivatives, such as the first order derivative:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

The **finite difference method** (FDM) is based on the idea to approximate the *derivatives* by *finite differences*. For example:

$$\frac{df(x)}{dx} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

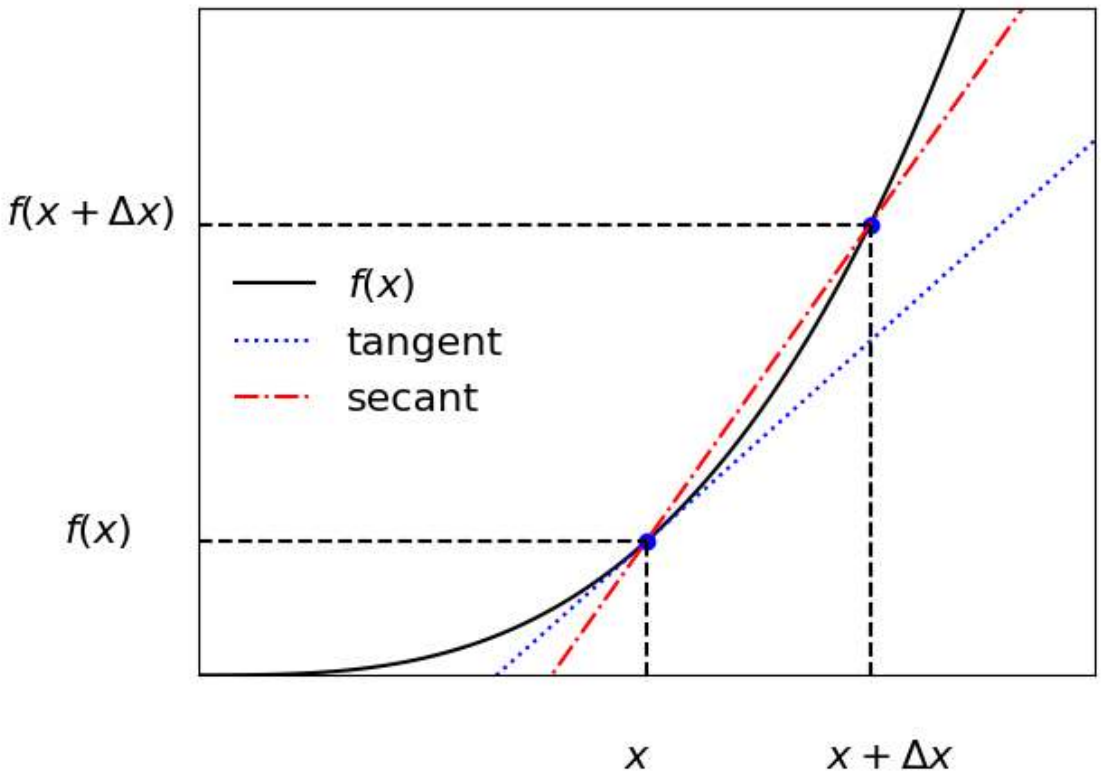
where Δx is a small but not infinitesimal number.

5.1 Comparison derivative vs finite difference

The slope of the tangent is given by the derivative.

The finite difference method is an approximation of the derivative.

We can compare the two by plotting the tangent (exact) with a secant line whose slope is given by the FDM.



A secant is a line that intersects a curve in at least two (distinct) points. A tangent is a line that just touches a point of a curve, so it has the same derivative at that point.

The (blue) tangent of the (black) curve is approximated by the (red) secant. The approximation becomes better, if the step Δx becomes smaller.

6 Explicit Euler method

The finite difference method can be used to solve ODEs. The Euler method is an example of a FDM.

The first step is replacing the derivatives by finite differences.

So in the ODE

$$\frac{df(x)}{dx} = g(x, f(x))$$

we make the replacement

$$\frac{df(x)}{dx} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

and hence

$$\frac{f(x + \Delta x) - f(x)}{\Delta x} \approx g(x, f(x))$$

6.1 Explicit Euler method for initial value problem

For an *initial value problem* the ODE can be solved as follows. Rewrite

$$\frac{f(x+\Delta x)-f(x)}{\Delta x} \approx g(x, f(x))$$

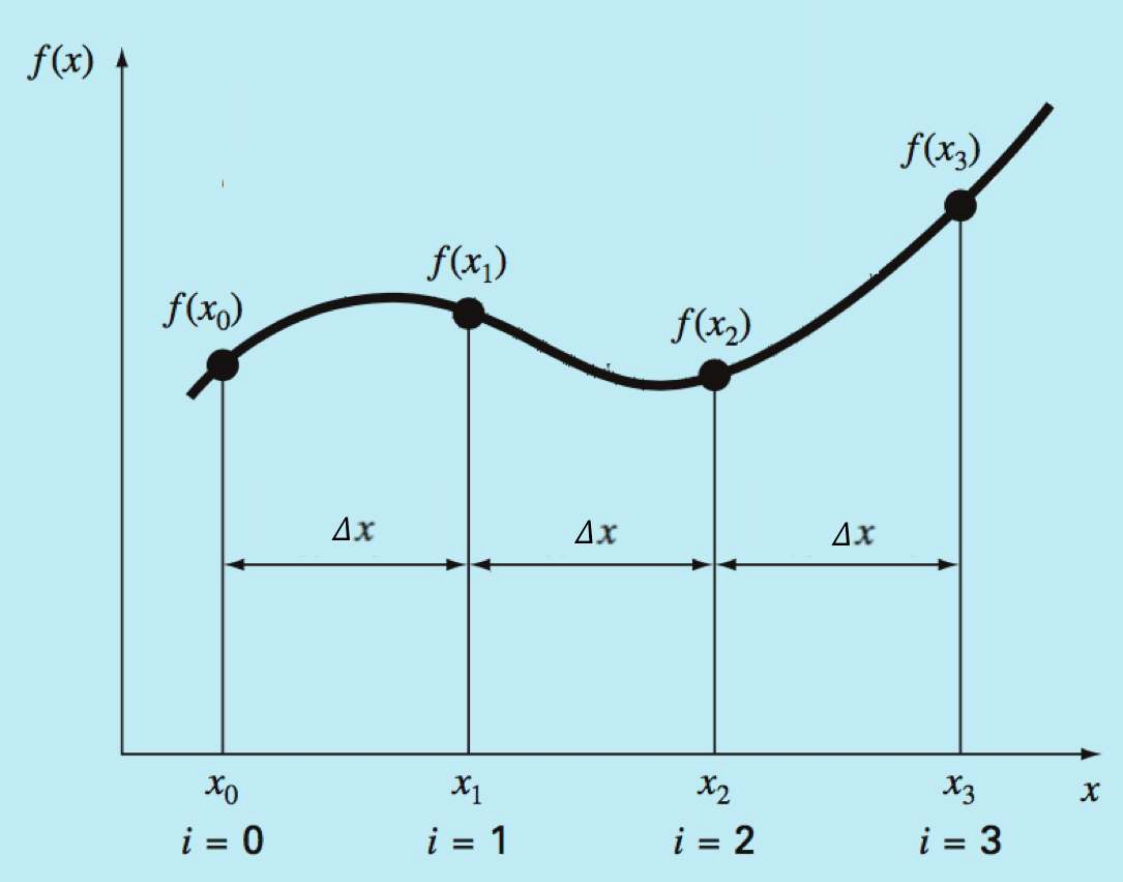
as

$$\boxed{f(x+\Delta x)_{\text{approx}} \approx f(x) + \Delta x \cdot g(x, f(x))}$$

This is called the **explicit Euler method**, aka the **forward Euler method** for the ODE $f(x)' = g(x, f(x))$. This allows us to approximate the value of $f(x+\Delta x)$ by only having information on f at the position x.

Since the initial value is given, we can construct the whole solution, by doing it step-by-step (iteratively applying the equation). Denote the **number of steps so far** by i and assume we start at $x_{\text{start}} = x_0$:

$$\begin{array}{rrrrcrc}
x_0 \to & x_0+\Delta x \to & x_0+2\Delta x \to & x_0+3\Delta x \to & \dots \to & x_0+i\Delta x \to & \dots \\
x_0 \to & x_1 \to & x_2 \to & x_3 \to & \dots \to & x_i \to & \dots \\
f(x_0) \to & f(x_0+\Delta x) \to & f(x_0+2\Delta x) \to & f(x_0+3\Delta x) \to & \dots \to & f(x_0+i\Delta x) \to & \dots \\
f_0 \to & f_1 \to & f_2 \to & f_3 \to & \dots \to & f_i \to & \dots
\end{array}$$



Lets say we want to integrate the ODE from $x_{\rm start}$ to $x_{\rm end}$. The **total number of integration steps** $N_{\rm int}$ is then

$$N_{\rm int} = |x_{\rm end} - x_{\rm start}|/\Delta x$$

and hence

$$x_{\rm end} \equiv x_{N_{\rm int}} = x_0 + N_{\rm int}\Delta x$$

A smaller Δx will improve the accuracy, but at the expense of doing more calculations, $N_{\rm int}$.

6.1.1 Example Euler forward method for 1st order ODE

Let us solve the ODE

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

subject to an initial condition $y(0) = y_0$. Analytical solution is known: $y = y_0 \exp(t)$. This allows us to check the integration error.

Replacing derivative in the ODE by finite differences:

$$\frac{y(t+\Delta t) - y(t)}{\Delta t} \approx y(t)$$

and rewriting this equation by writing $y(t+\Delta t)$ as a function of the rest:

$$y(t+\Delta t) \approx y(t) + \Delta t y(t) = (1+\Delta t)y(t)$$

We can repeat this algorithm to get to larger and larger times:

$$\begin{aligned} y(t+2\Delta t) &\approx (1+\Delta t)y(t+\Delta t) \\ &\approx (1+\Delta t)(1+\Delta t)y(t) \end{aligned}$$

and

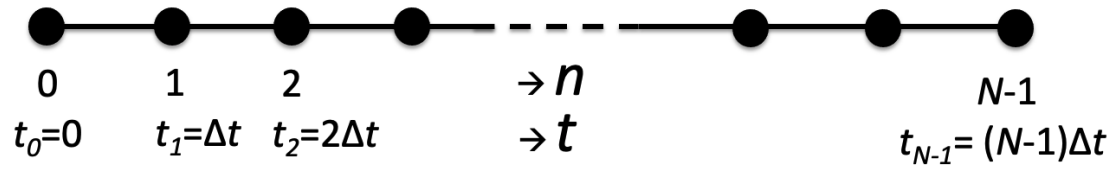
$$\begin{aligned} y(t+3\Delta t) &\approx (1+\Delta t)y(t+2\Delta t) \\ &\approx (1+\Delta t)(1+\Delta t)y(t+\Delta t) \\ &\approx (1+\Delta t)(1+\Delta t)(1+\Delta t)y(t) \end{aligned}$$

etc.

Here the time-dependence for the discrete scheme is obvious and can just be written as

$$y(t+N_{\rm int}\Delta t) = (1+\Delta t)^{N_{\rm int}} y(t)$$

but such simple expressions are not always available for other DEs.



6.1.2 Example code Euler forward method

Hence in order to repeat the step

$$y(t+\Delta t) \approx y(t) + \Delta t y(t) = (1+\Delta t)y(t) \qquad (*)$$

many times we can use a computer. We can convert eq. (*) to variables that can be used in a programming language as

```
ynext=(1+dt)*ynow
```

where `ynext` stands for $y(t+\Delta t)$, and `ynow` for $y(t)$.

For the next time step, $y(t+2\Delta t)$, we update `ynow` to be `ynext` and recalculate `ynext`.

In Python code this becomes

```
y0=1.0                    # initial condition. Assume y(0)=1.0
t=0.0                     # start time. Use 0.0 for floats
dt=0.01                   # integration time step
tmax=10.0                 # end time
Nint=int(round(tmax/dt)) # number of integration steps
ynow=y0
for n in range(Nint):     # loop from 0 to but not including Nint
   ynext=(1+dt)*ynow      # calculate y(t+dt)

   # next step: ynext becomes ynow, t increases
   ynow=ynext
   t=t+dt
```

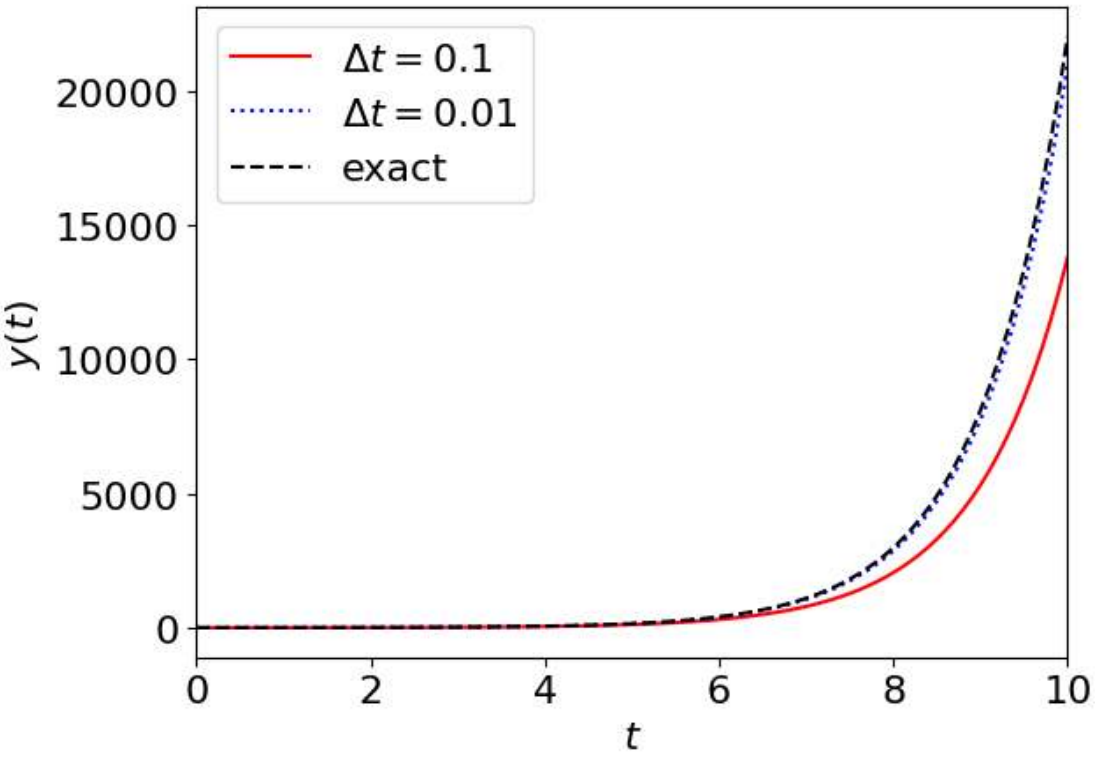
6.1.3 Program for storing the whole function

Our program can be rewritten to store all intermediate values:

```
y0=1.0        # start value for y
dt=0.01       # integration step
t0=0.0        # start time
tmax=10.0     # end time
Nint=int(round(tmax/dt))
y=np.zeros(Nint+1)   # number of points is number of integration steps + 1
t=np.zeros(Nint+1)
y[0]=y0
t[0]=t0
for n in range(Nint):
    t[n+1]=t[n]+dt
    y[n+1]=y[n]*(1+dt)
```

6.1.4 Result whole function

The result for $y(t)$ is then as follows. Note that using $\Delta t = 0.01$ gives a good agreement with the analytical solution.



6.2 Error in approximation due to truncation: local error

General Taylor series

$$f(x+\Delta x)_{\text{exact}} = f(x) + \Delta x f'(x) + \Delta x^2 f''(x)/2! + O(\Delta x^3)$$

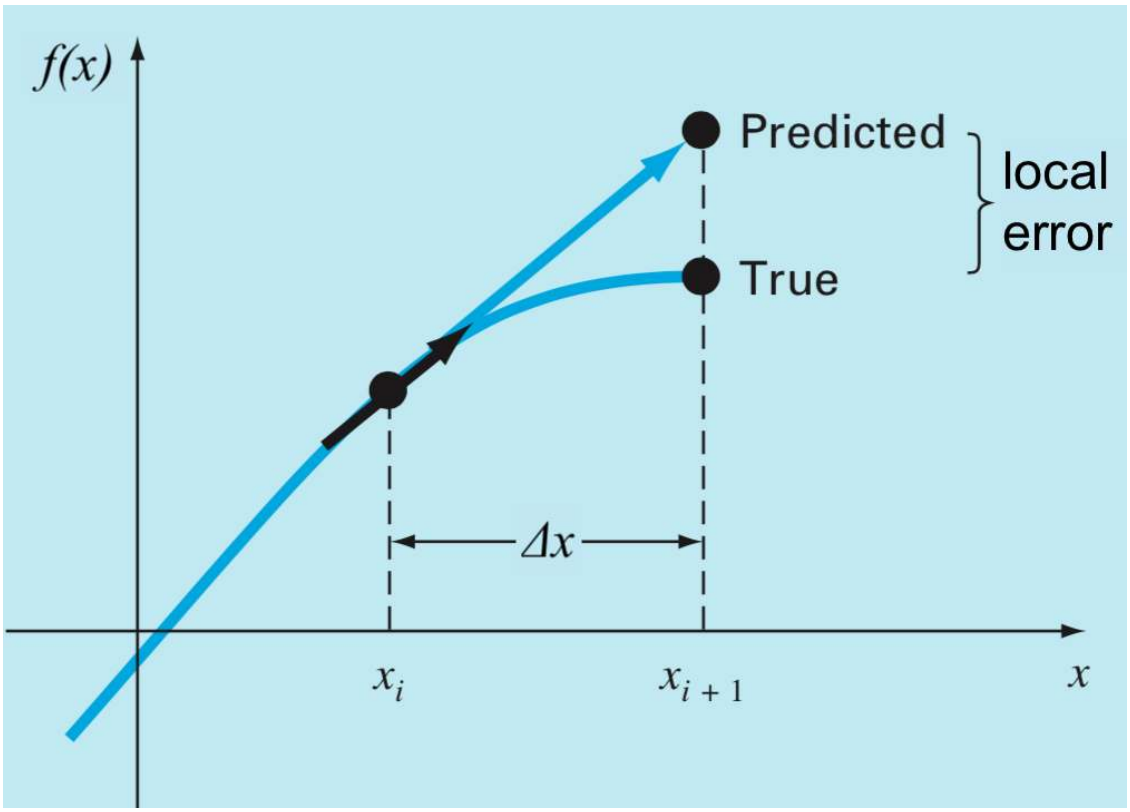
For our ODE we have that the derivative is given by $\frac{df(x)}{dx} = g(x, f(x))$ with g a known function, so for the Euler method this gives

$$f(x+\Delta x)_{\text{approx}} \approx f(x) + \Delta x \cdot g(x, f(x))$$

and the difference is

$$f(x+\Delta x)_{\text{approx}} - f(x+\Delta x)_{\text{exact}} = \Delta x^2 f''(x)/2! + O(\Delta x^3)$$

where we have used $g(x, f(x)) = f'(x)$.



So the error due to the *truncation of the Taylor series* in $f(x + \Delta x)$ is for small Δx proportional to Δx^2 (since Δx^3 is even smaller and can be neglected).

The **local truncation error**, which is the error after *one* integration step due to truncating the Taylor series, is therefore $O(\Delta x^2)$ for the Euler method.

6.3 Error in approximation: global truncation error

Assume we want to integrate the ODE from x_{start} to x_{end} with $N_{\text{int}} = (x_{\text{end}} - x_{\text{start}})/\Delta x$ integration steps.

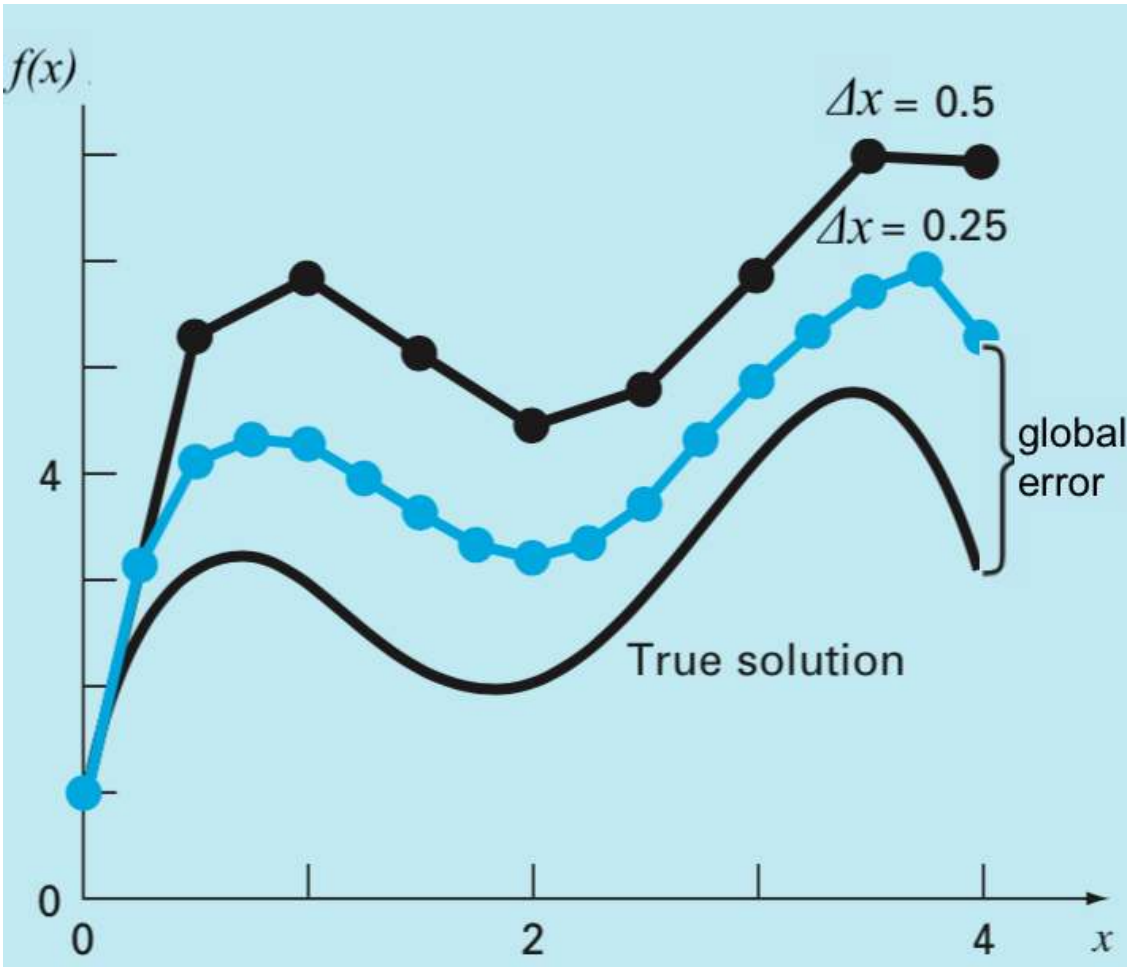
Intuitively the **global truncation error**, which is the error due to truncation after integrating over the whole interval, is therefore in this case N_{int} times the *local truncation error* (the error after one step).

For the Euler method we have that the local error is $O(\Delta x^2)$, and hence the global error is $N_{\text{int}} O(\Delta x^2) = \frac{|x_{\text{end}} - x_{\text{start}}|}{\Delta x} \cdot O(\Delta x^2) = |x_{\text{end}} - x_{\text{start}}| O(\Delta x)$, so proportional to Δx, or $O(\Delta x)$.

The **order of a method** is given by *how the global truncation error varies with the integration step*.

For the Euler method the error is therefore first order in the integration step Δx: the *Euler method* is a **first order algorithm**.

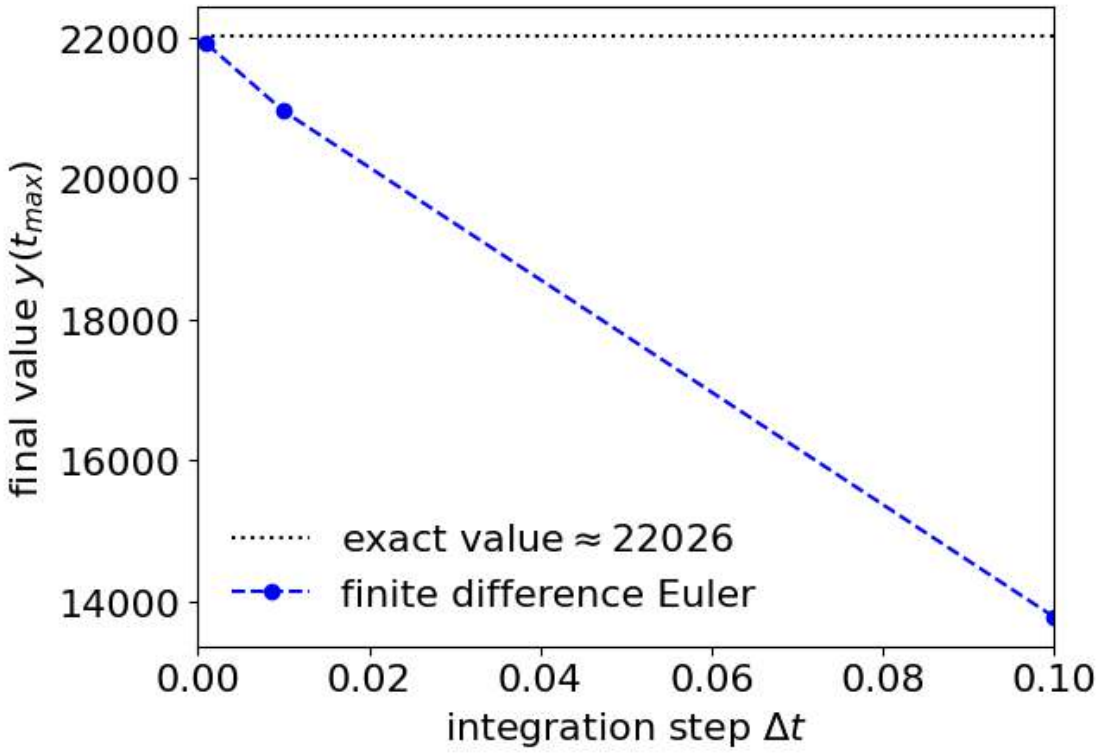
Argument can be made more rigorous, but result for the order stays the same.



6.4 Result error explicit Euler method

We can estimate the error in the algorithm by comparing the exact value for $y(t_{\max}) = \exp(t_{\max})$ (which equals approximately 22026 for $t_{\max} = 10$) with the approximate value as a function of the integration time step.

Repeating the whole integration for 3 different integration steps (0.1, 0.01 and 0.001) gives



The error between the finite difference scheme and the Euler method is the difference between the blue and black curve.

We observe that the error decreases linearly with decreasing Δt: the Euler method is indeed a *first order method*.

In this case we can also calculate the error analytically.

The analytical result for $y(t)$ is

$$y(t)_{\text{exact}} = \exp(t)$$

while the forward Euler method gives

$$\begin{aligned} y(t)_{\text{approx}} &= (1+\Delta t)^n \\ &= (1+\Delta t)^{\frac{t}{\Delta t}} \end{aligned}$$

The error in the approximation, the difference, is therefore

$$\begin{aligned} y(t)_{\text{exact}} - y(t)_{\text{approx}} &= \exp(t) - (1+\Delta t)^{\frac{t}{\Delta t}} \\ &= \ldots = \frac{t}{2}\exp(t)\Delta t + O(\Delta t^2) \end{aligned}$$

which is indeed proportional to Δt in leading order at $t = t_{\max}$.

6.5 That's it?

- No problems for any differential equation?
- Can we do better than $O(\Delta t)$ for the global truncation error?

6.6 Example explicit Euler

Let us solve the ODE

$$\frac{dy(t)}{dt} = -ay(t)$$

subject to an initial condition $y(0) = y_0$ and with $a > 0$.

Analytical solution is known, $y = y_0 \exp(-at)$, so this allows us to check the integration error.

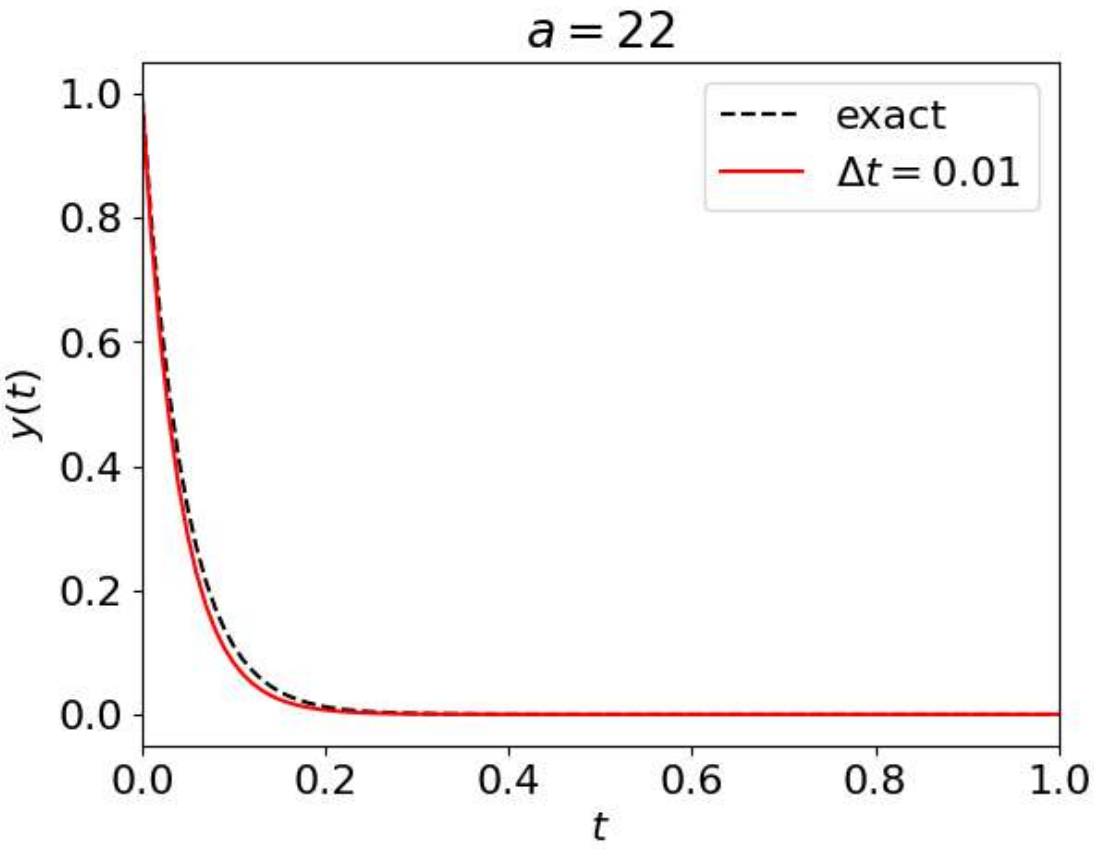
6.7 Program

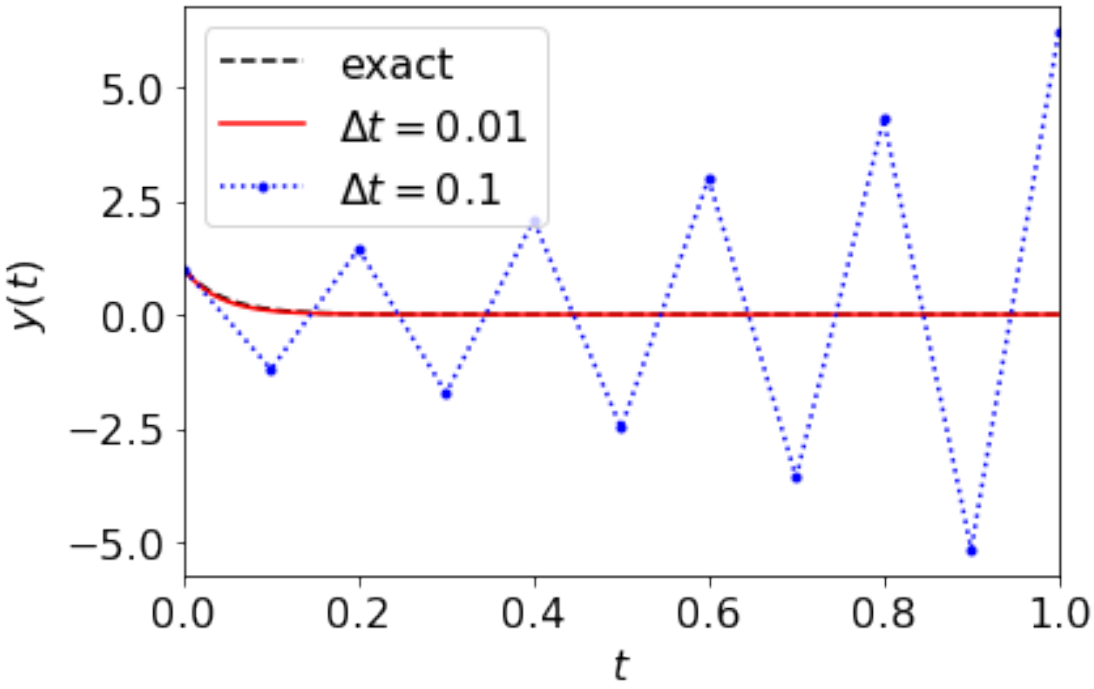
```
y0=1.0
a=20.0  # Extra variable a
dt=0.01
t=0.0
tmax=10.0
Nint=int(round(tmax/dt))
ynow=y0
for n in range(Nint):
    # Difference to previous ODE: -a instead of +1
    ynext=(1-a*dt)*ynow

    ynow=ynext
    t=t+dt
```

6.8 Result

Solution $y(t)$ of the ODE $\dot{y} = -ay$

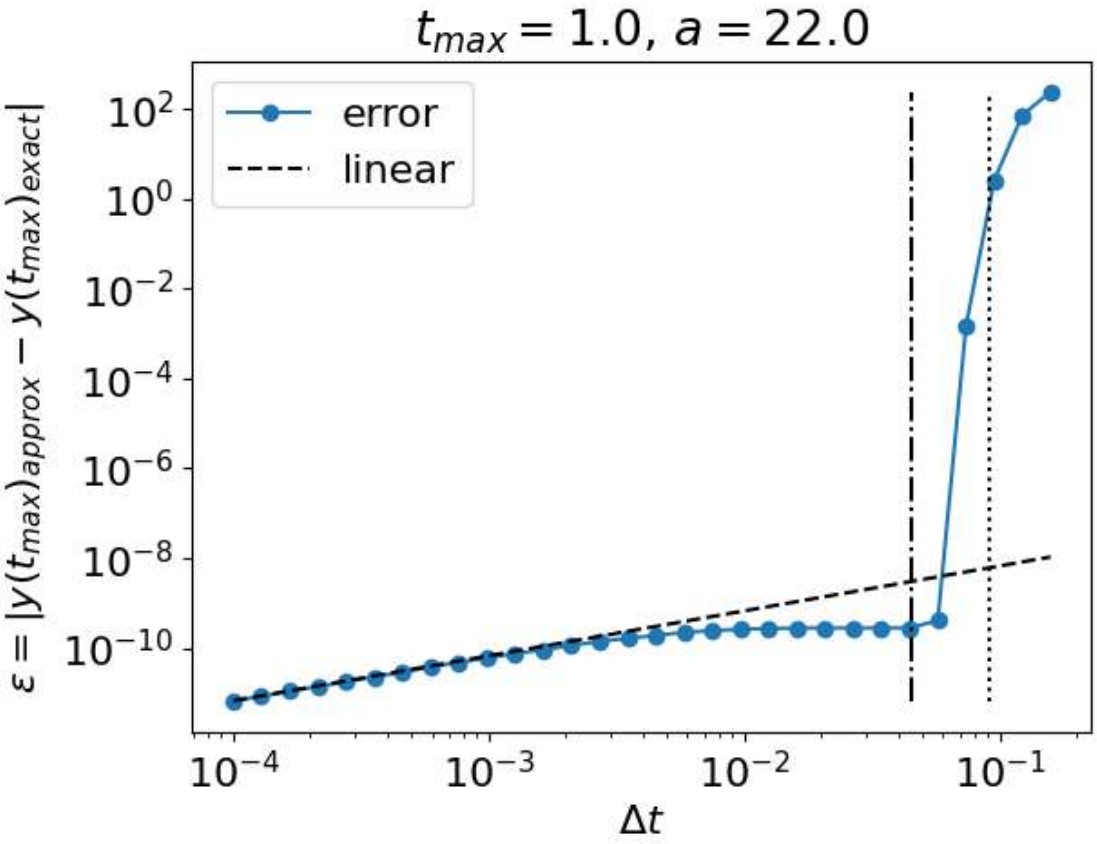

$a = 22$
exact
$\Delta t = 0.01$
$y(t)$
t
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


exact
$\Delta t = 0.01$
$\Delta t = 0.1$
$y(t)$
t
5.0
2.5
0.0
−2.5
−5.0
0.0
0.2
0.4
0.6
0.8
1.0

The explicit Euler algorithm becomes unstable for large time step!

6.9 Result vs timestep

Let us now plot the resulting error at $t = t_{\max}$, again repeating the same calculation for various values of Δt



The explict Euler algorithm becomes *unstable* for $a\Delta t > 2$ (second, dotted, vertical line). It shows *oscillatory behaviour* for $a\Delta t > 1$ (first vertical line).

7 Implicit Euler method

7.1 implicit vs explicit

An **implicit relation** is a relation where a dependent variable is not isolated on one side of the equation. An **explicit relation** is a relation where *a dependent variable is isolated on one side*. For example

$$x^2 + xy - y^2 = 1$$

is an implicit relation for y, while

$$y = x^2 + 2x - 4$$

is an explicit relation for y.

Sometimes it is rather easy to convert an implicit relation into an explicit one

$$x + y = 1$$

is implicit in y. Bringing x to the RHS turns this into an explicit relation. Hence

$$y = 1 - x$$

is explicit in y.

A slightly more difficult example is the following implicit relation

$$x^2 + y^2 = 1$$

The accompanying explicit relation is

$$y = \pm\sqrt{1 - x^2}$$

It is not always possible to convert an implicit relation into an explicit using elementary functions.

7.2 Precursor

For the explicit **Euler method** we saw that we replaced the derivative by

$$\frac{df(x)}{dx} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \qquad (*)$$

This was inspired by the limit for the derivative,

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

Eq. (*) is called a **forward difference approximation** (FDA), since the derivative is approximated by the function value at a later/forward point minus the value at the current point.

However, the derivative can also be written in other limits. These other limits give rise to the *same* derivative. E.g., replace h by $-h$:

$$\begin{aligned} \frac{df(x)}{dx} &= \lim_{-h \to 0} \frac{f(x - h) - f(x)}{-h} \\ &= \lim_{h \to 0} \frac{f(x) - f(x - h)}{h} \end{aligned}$$

and the finite difference for this equation is

$$\frac{df(x)}{dx} \approx \frac{f(x) - f(x - \Delta x)}{\Delta x} \qquad (**)$$

where again Δx is a small but not infinitesimal number. Eq. (**) is called a **backward difference approximation** (BDA), since now the derivative is approximated by the current function value minus the value at a previous/backward point.

Notice that eq. (**) is not the same as the FDA, $\frac{df(x)}{dx} \approx \frac{f(x+\Delta x)-f(x)}{\Delta x}$.

Only in the limit $\Delta x \to 0$ they become equal to each other. Hence they generally give rise to different numerical schemes.

7.3 Algorithm implicit Euler method for IVP

Given the ODE

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

The explicit method is

$$\frac{y(t+\Delta t) - y(t)}{\Delta t} \approx g(t, y(t))$$

The **implicit Euler method** (also known as **backward Euler method**) is

$$\frac{y(t+\Delta t) - y(t)}{\Delta t} \approx g(t+\Delta t, y(t+\Delta t)) \qquad (*)$$

Notice that now $y(t+\Delta t)$ also occurs at the right hand side of eq. (*). This means that the equation is not any more an explicit equation, but an implicit one.

Rewriting gives

$$\boxed{y(t+\Delta t) \approx y(t) + \Delta t \cdot g(t+\Delta t, y(t+\Delta t))}$$

In order to determine y at the next time step, $y(t+\Delta t)$ (LHS), we already have to know $y(t+\Delta t)$.

If g is linear with y, then it is easy to transform the implicit relation into a explicit one.

If g is non-linear, then it is a bit more tricky. It is an equation containing $y(t+\Delta t)$ and could be solved using a root-finder such as Newton's method.

7.3.1 Why is it called backward?

The implicit Euler method

$$\frac{y(t+\Delta t) - y(t)}{\Delta t} \approx g(t+\Delta t, y(t+\Delta t))$$

can be rewritten as

$$\frac{y(t) - y(t-\Delta t)}{\Delta t} \approx g(t, y(t)) \qquad (*)$$

where we just shifted the time by Δt backwards. This equation can also be derived by starting from the *backward difference approximation*

$$\frac{df(x)}{dx} \approx \frac{f(x) - f(x-\Delta x)}{\Delta x} \qquad (1)$$

Notice that now $y(t+\Delta t)$ also occurs at the right hand side of eq. (*). This means that the equation is not any more an explicit equation, but an implicit one.

7.4 Example implicit Euler

Let us solve the ODE

$$\frac{dy(t)}{dt} = -ay(t)$$

with $a > 0$ subject to an initial condition $y(0) = y_0$. Analytical solution is known, $y = y_0 \exp(-at)$, so this allows us to check integration error.

Replacing by finite differences:

$$\frac{y(t + \Delta t) - y(t)}{\Delta t} \approx -ay(t + \Delta t)$$

Note that this is an implicit relation in $y(t + \Delta t)$. However, we can rewrite it in an explicit form by expressing $y(t + \Delta t)$ as a function of the rest:

$$y(t + \Delta t)(1 + a\Delta t) \approx y(t)$$

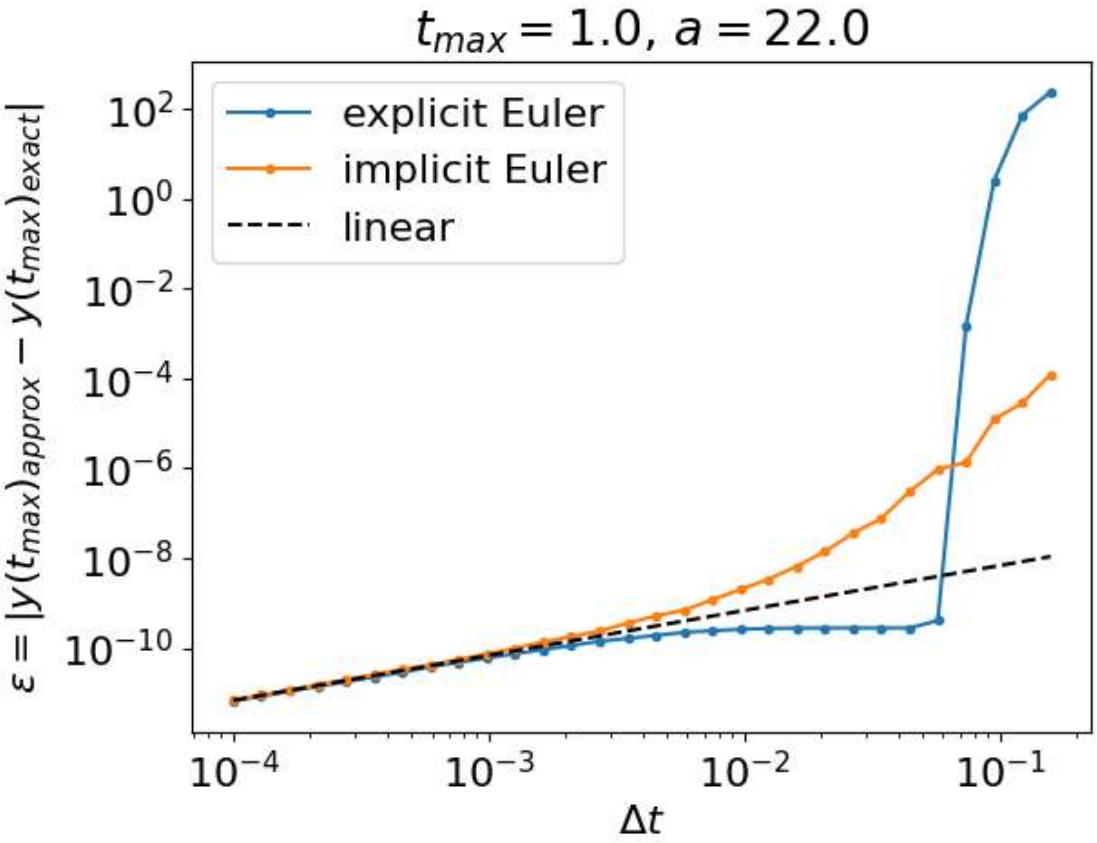
or

$$y(t + \Delta t) \approx \frac{y(t)}{1 + a\Delta t}$$

In this example no root-finding is needed, since $y(t + \Delta t)$ can be isolated analytically.

Implicit Euler method is simple to implement for this differential equation. Instead of multiplying $y(t)$ by $(1-a\Delta t)$ for the explicit Euler method, we have to divide $y(t)$ by $(1+a\Delta t)$. In our aforementioned program this implies replacing `(1-a*dt)*ynow` by `ynow/(1+a*dt)`. In general the implicit scheme is more difficult.

7.5 Result error in solution vs timestep



8 Exercises session 1

1. What is meant by the following?
   - local truncation error
   - global truncation error
   - order of an algorithm
   - finite difference method
2. What is the difference between an implicit and explicit relation?
3. Implement the explicit Euler method in a programming language (preferably Python), and with it solve the ODE
$$\frac{dy(t)}{dt} = bt - ay(t)$$
The coefficients should be implemented as variables, but take $b = 1$, $a = 22$, $y(0) = 1$, $t_{\max} = 1$, and integrate using both $\Delta t = 0.01$ and $\Delta t = 0.1$
4. Implement the implicit Euler method and solve the same ODE.

5. The analytical solution is

$$y(t) = e^{-at}\left(y_0 + \frac{b}{a^2}\right) + \frac{bt}{a} - \frac{b}{a^2}$$

Compare the errors for the two algorithms and plot the solution for each of them and the analytical solution vs time, $y(t)$.

Index