# Computational practicum: Q1 – Lecture 4 Numerical integration and differentiation

Zoë J.G. Gromotka, Ferdinand Grozema, Artur M. Schweidtmann, Tanuj Karia

With support from Giacomo Lastrucci and Sophia Rupprecht

**Computational Practicum**
Dept. Chemical Engineering
Delft University of Technology

TU Delft

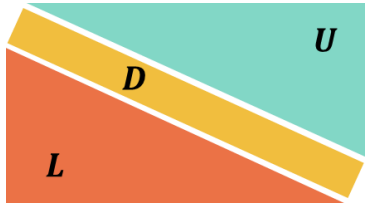
Process Intelligence RESEARCH

Delft Institute of Applied Mathematics

Wednesday, September 24th, 2025

# Recap last lecture (1/2) – Solution of linear systems


- Numerical techniques:
  - Jacobi method $\boldsymbol{x}^{(k+1)} = \boldsymbol{D}^{-1}(\boldsymbol{b} - (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x}^{k})$
  - Gauss-Seidel method $\boldsymbol{x}^{(k+1)} = (\boldsymbol{D} + \boldsymbol{L})^{-1}\boldsymbol{b} - (\boldsymbol{D} + \boldsymbol{L})^{-1}\boldsymbol{U}\boldsymbol{x}^{(k)}$




- Python built-in functions:
  - numpy.linalg.solve()

# Recap last lecture (2/2) - Solution of nonlinear systems

- Numerical technique:
  - Newton-Raphson method $\boldsymbol{f}(\boldsymbol{x}) = 0 \xrightarrow{iterate} \boldsymbol{x}^{k+1} = \boldsymbol{f}(\boldsymbol{x}^k) - \boldsymbol{J}^{-1}\boldsymbol{f}(\boldsymbol{x}^k)$

**HINT: connect the dots**

*Matrix inversion is expensive! Consider to use methods for solving linear systems.*

**RECAP**

*The Jacobian of a system of multivariate functions is the matrix of the partial derivatives (See. Lecture 2)*

- Python built-in functions: (different advanced methods used in the background)
  - scipy.optimize.fsolve
  - scipy.optimize.root

Yes! The same functions can handle both single nonlinear equations and systems :D

# Learning goals of this lecture

After successfully completing this lecture, you are able to…

- explain methods for numerical integration and differentiation
- implement numerical methods for integration and differentiation in Python from scratch
- use Python libraries' built-in functions for numerical integration and differentiation

# Agenda

**Numerical integration**

- Quadrature rules
  - Introduction and overview
  - Rectangle method
  - Newton-Cotes rules

  a. Midpoint method

  b. (Composite) trapezoid method

  c. (Composite) Simpson 1/3 rule
  - Built-in Python functions

**Numerical differentiation**

- Introduction and recap
- Finite difference method
  - First order difference schemes
  - Second order difference schemes
- Built-in Python functions

# Agenda

**Numerical integration**

- Quadrature rules
  - Introduction and overview
  - Rectangle method
  - Newton-Cotes rules

  a. Midpoint method

  b. (Composite) trapezoid method

  c. (Composite) Simpson 1/3 rule
  - Built-in Python functions

**Numerical differentiation**

- Introduction and recap
- Finite difference method
  - First order difference schemes
  - Second order difference schemes
- Built-in Python functions

# Numerical integration: Reaction engineering example

- In a plug-flow reactor (PFR) with reaction $A \rightarrow B$, the component balance is given by:

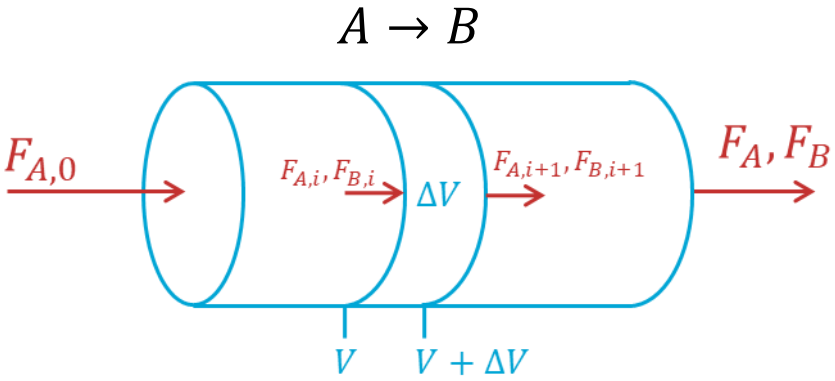
$$\frac{dF_A}{dV} = r_A(c_A)$$

- The balance can be used to derive a design equation for the reactor volume V in $m^3$ to achieve a certain conversion $\chi$ $[-]$

$$\chi = \frac{F_{A,0} - F_A}{F_{A,0}} = \frac{c_{A,0} - c_A}{c_{A,0}}$$

- Assuming constant volumetric flowrate $\dot{V}$[1]:

$$V = F_{A,0} \int_0^{\bar{\chi}} \frac{1}{-r_A} d\chi$$




$F_A$: molar flowrate of A in $mol/s$.
$r_A(c_A)$ : consumption rate of A in $mol/s\ /m^3$.

You can check the full derivation of the design equation in [1]

[1] O. Levenspiel – Chemical Reaction Engineering (Third Edition), pages. 101-102

# Numerical integration: Reaction engineering example

- How to solve $V = F_{A,0} \int_0^{\bar{\chi}} \frac{1}{-r_A} d\chi$ for a given $r_A(c_A)$
  - Elementary first order reaction ($r_A = -k \cdot c_A$):

Express $c_A$ as a function of $\chi$ *(see previous slide)*

$$V = F_{A,0} \int_0^{\bar{\chi}} \frac{1}{kc_A} d\chi$$

You can solve it analytically!

$$c_A = c_{A,0}(1 - \chi)$$

**RECAP**

$$\int \frac{f'(x)}{f(x)} dx = \ln f(x) + C$$

$$V = F_{A,0} \int_0^{\bar{\chi}} \frac{1}{kc_{A,0}(1-\chi)} d\chi = -\frac{F_{A,0}}{kc_{A,0}} \int_0^{\bar{\chi}} \frac{-1}{(1-\chi)} d\chi$$

$$V = -\frac{F_{A,0}}{kc_{A,0}} \cdot \ln(1-\chi) \Big|_0^{\bar{\chi}}$$

$$\boxed{V = -\frac{F_{A,0}}{kc_{A,0}} \cdot \ln(1-\bar{\chi})}$$

# Numerical integration: Reaction engineering example

However, the kinetic expression can become far more complicate:

- Langmuir-Hinshelwood-Hougen-Watson (LHHW) kinetics:

$$V = F_A \int_0^{\bar{\chi}} \frac{1}{-\frac{k_A c_A}{(1 + k_A c_A + k_B c_B)}} d\chi$$

Analytical solution can become inconvenient/unfeasible soon!

Numerical integration allows you to approximate complex definite integrals efficiently.

# Integration: Riemann sum

Let $f: [a, b] \to \mathbb{R}$ be a function over $[a, b]$ with $a, b \in \mathbb{R}$.

The interval $[a, b]$ is partitioned into a finite sequence of N sub-intervals $[x_i, x_{i+1}]$ such that

$$a = x_0 < x_1 < x_2 < \cdots < x_{N-1} = b.$$

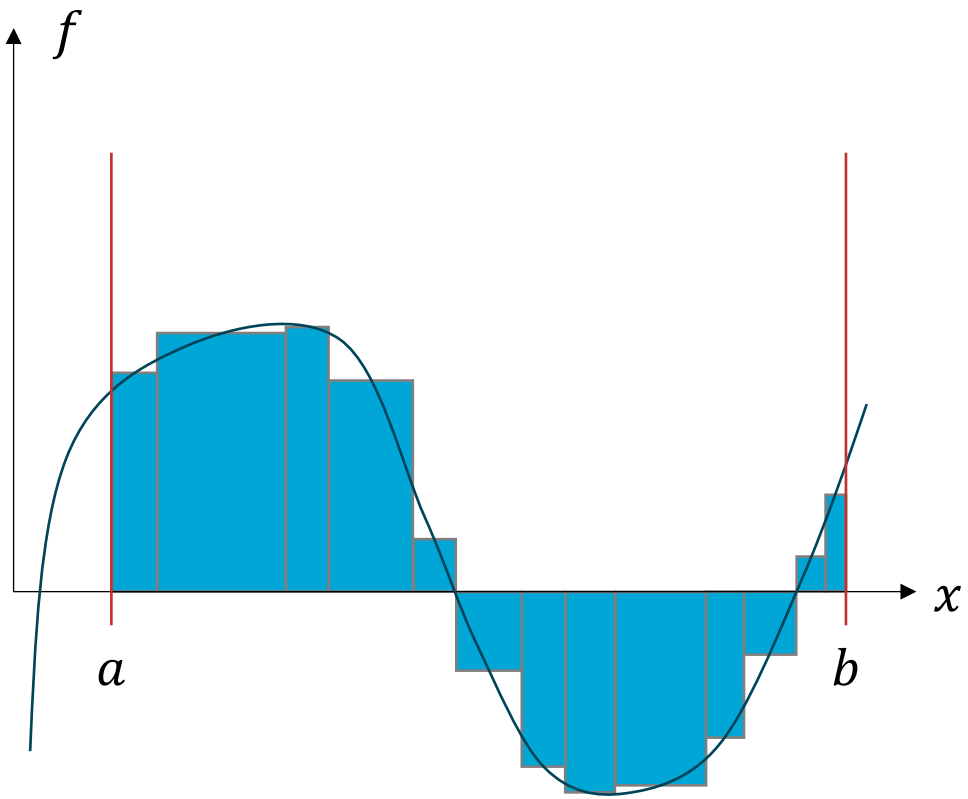
The **Riemann sum** $S$ over $f$ is defined as

$$S = \sum_{i=0}^{N-1} f(x_i^*)(\Delta x)_i$$

with $\Delta x_i = x_{i+1} - x_i$.

Considering the limit $\triangle\, x_i \to 0$, the Riemann sum is equal to the integral of $f$ over $[a, b]$ (also called *definite integral*):

$$\int_a^b f(x)\, dx = \lim_{|\Delta x_i| \to 0 \ \forall i=0,\ldots,N-1} \sum_{i=0}^{N-1} f(x_i^*)(\Delta x)_i\,.$$




**Note**: Different Reimann sums may be produced depending on the choice of $x^*$.

# Methods overview

Numerical integration methods are also known as quadrature rules.

We teach the most elementary quadrature rules in this course:

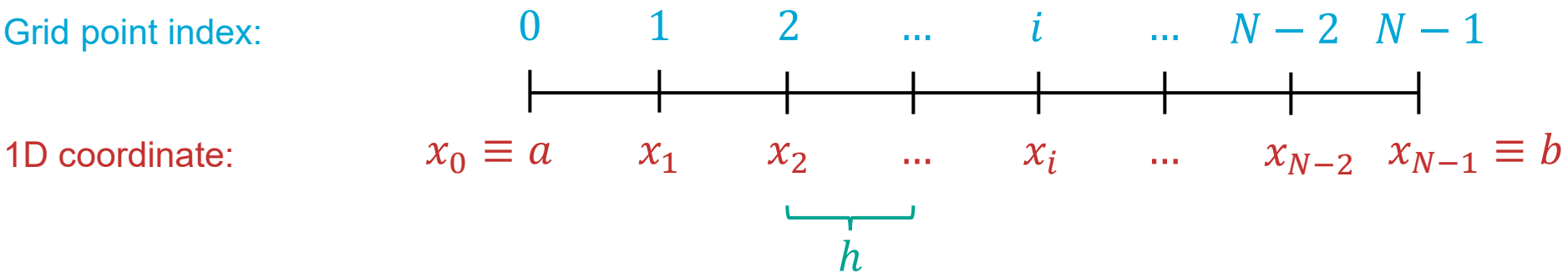
- Rectangle method
- Newton-Cotes rules:
  - Midpoint method
  - Trapezoidal rule → *extended into:* Composite trapezoidal method
  - Simpson's 1/3 Rule → *extended into:* Composite Simpson's 1/3 method

Other advanced methods are also available but are not covered in this course:

- Higher order Newton-Cotes, Gaussian quadrature, adaptive quadrature, etc.

# Grid and indexing

- To solve definite integrals numerically, we discretize the domain.
- In this course, we consider uniform discretization over the 1D domain with constant step size.
- We define indexes ($i$ or sometimes $j$) to indicate both the grid point index (useful for implementation) and the actual 1D coordinate (i.e., $x_i$)
- We start counting from 0 (as Python starts indexing from 0).
- Then, if we consider N discretization points, the last index is N-1.
- N discretization points correspond to N-1 regular intervals of size *h*.



# Agenda

**Numerical integration**

- Quadrature rules
  - Introduction and overview
  - Rectangle method
  - Newton-Cotes rules

  a. Midpoint method

  b. (Composite) trapezoid method

  c. (Composite) Simpson 1/3 rule
  - Built-in Python functions

**Numerical differentiation**

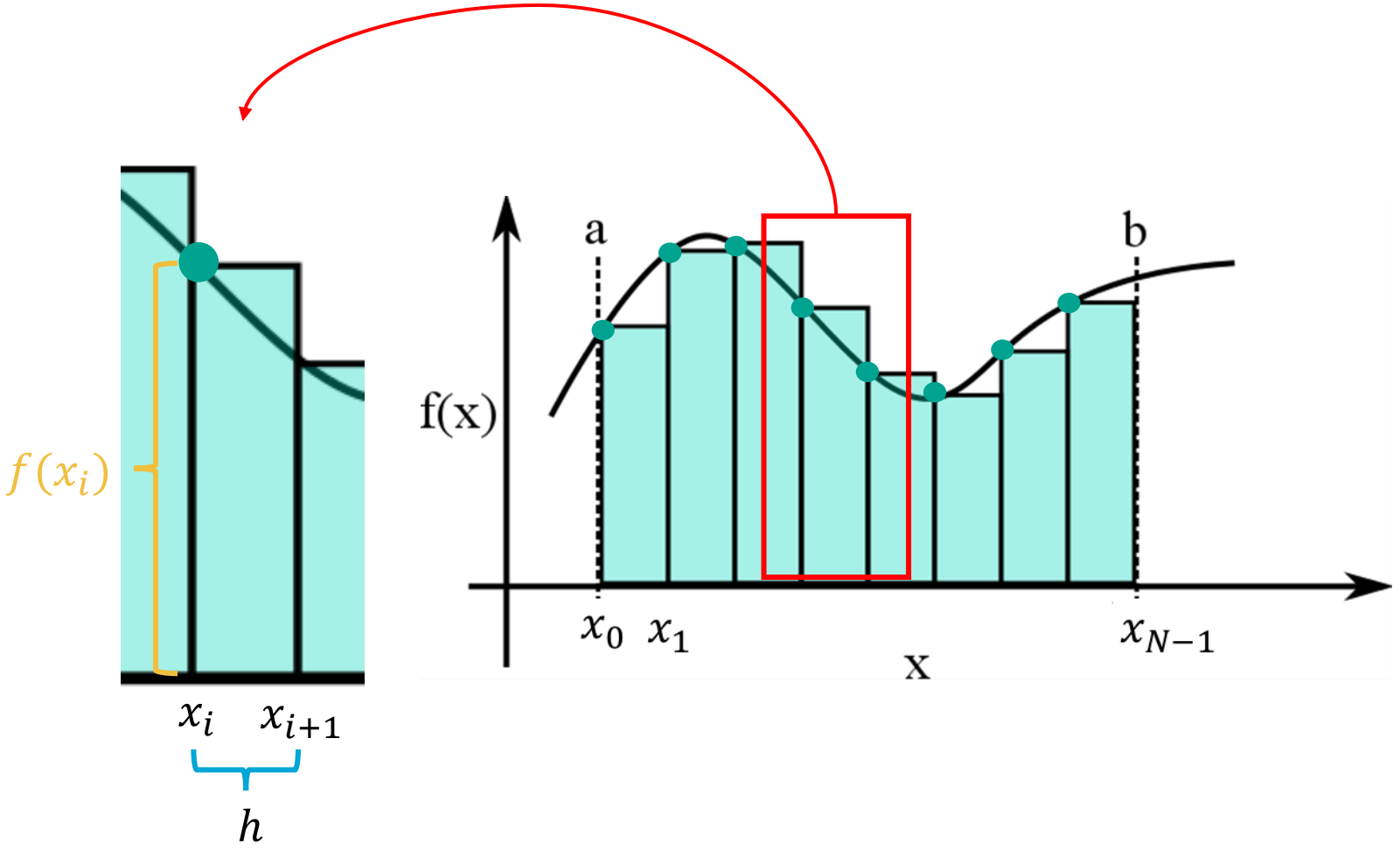
- Introduction and recap
- Finite difference method
  - First order difference schemes
  - Second order difference schemes
- Built-in Python functions

# Rectangle method

Task: compute $I = \int_a^b f(x)dx$

- Number of grid points, $N$
- Number of intervals, $N - 1$
  - Width of interval $h = \frac{b-a}{N-1}$

- $x_i^*$ in left corner
  - $I \approx h \sum_{i=0}^{N-2} f(x_i)$



# Rectangle method (left): Live coding

- Open Colab: Integration - Rectangle method (left)

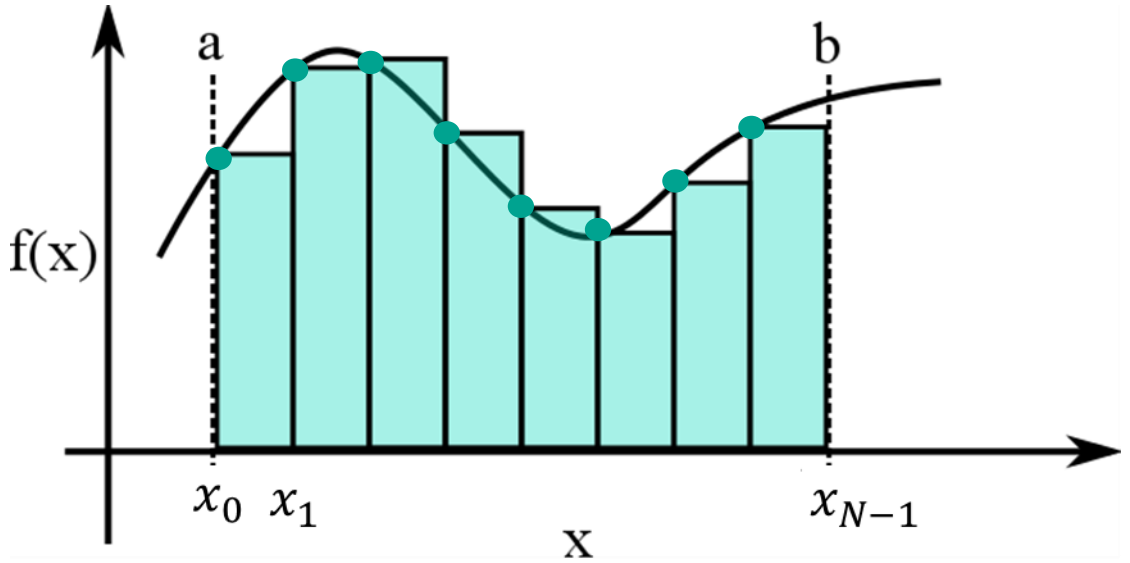
- Find more in the Github repository of the course: https://github.com/process-intelligence-research/computational_practicum_lecture_coding/tree/main

# **Reminder**: Rectangle method

Task: compute $I = \int_a^b f(x)dx$

- Number of grid points, $N$
- Number of intervals, $N - 1$
  - Width of interval $h = \frac{b-a}{N-1}$

- $x_i^*$ in left corner
  - $I \approx h \sum_{i=0}^{N-2} f(x_i)$




What is the error of the rectangle method?

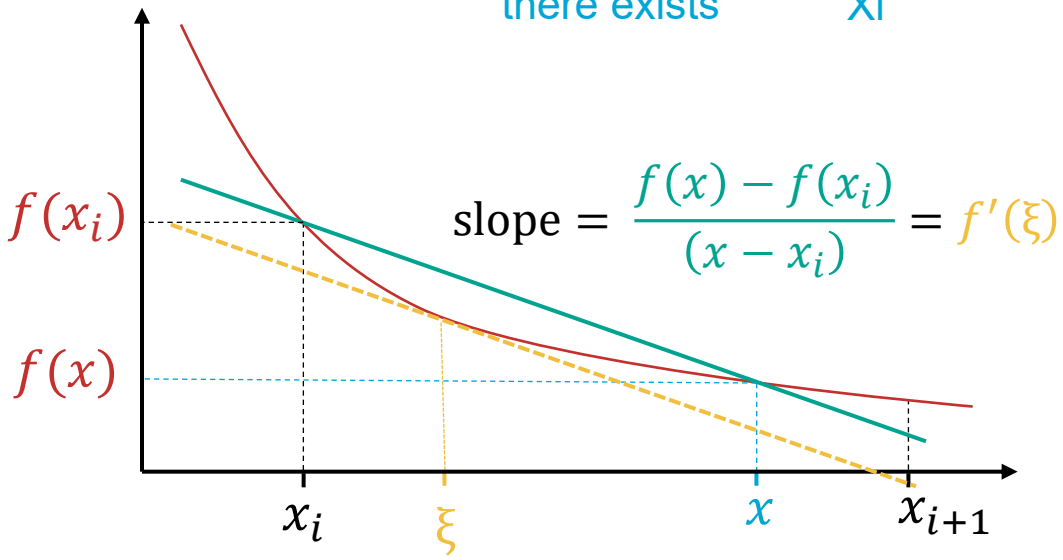
# Rectangle method: Error analysis (1/4)

If you don't remember this, practise it! *here*

- Mean value theorem (or Lagrange theorem):

$$\forall x \in [x_i, x_{i+1}], \quad \exists \xi \in [x_i, x] \text{ with } \frac{f(x) - f(x_i)}{(x - x_i)} = f'(\xi)$$

"there exists" "Xi"




The mean value theorem states that for a given arc between two endpoints $[x_i, x]$, there is a point $\xi$ in the same interval where the tangent to the arc is parallel to the secant through its endpoints.

# Rectangle method: Error analysis (2/4)

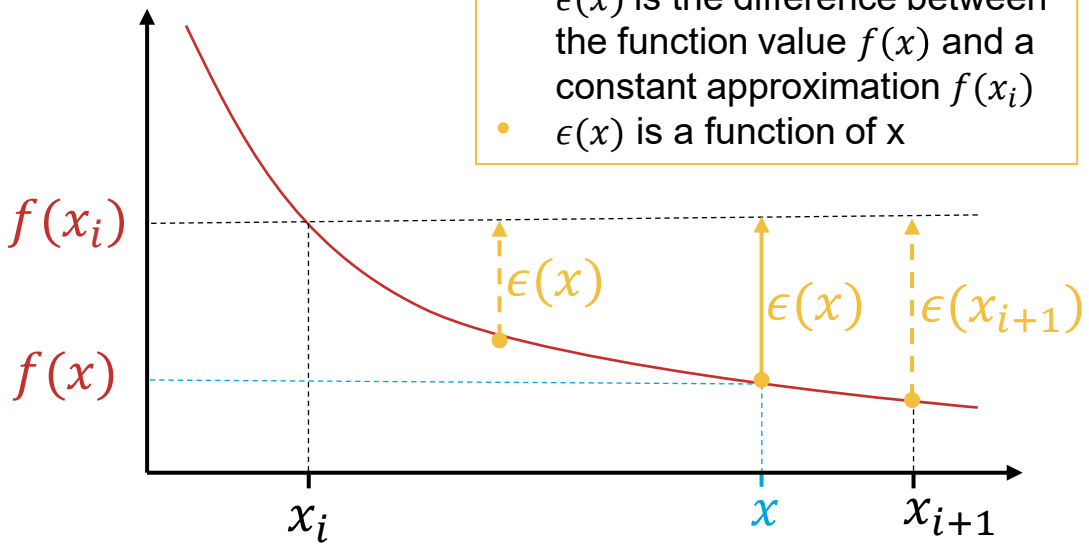
For reference, you can find the derivation here

- We aim to evaluate the integral error of the rectangle method as a function of the step size *h*.
- We firstly estimate the integral error ($E_i$) for one discretization step.
- In the rectangle method you consider the function as constant through the interval.
- Thus, the method does not consider the variation of the function along the interval
- We measure such deviation $\epsilon(x)$ using the mean value theorem:

Mean value theorem: $\frac{f(x)-f(x_i)}{(x-x_i)} = f'(\xi)$

$\rightarrow \epsilon(x) = f(x) - f(x_i) = f'(\xi)(x - x_i)$

- $\epsilon(x)$ is the difference between the function value $f(x)$ and a constant approximation $f(x_i)$
- $\epsilon(x)$ is a function of x



# Rectangle method: Error analysis (3/4)

For reference, you can find the derivation here

- Then, we can estimate the error of the integral over a single rectangle:

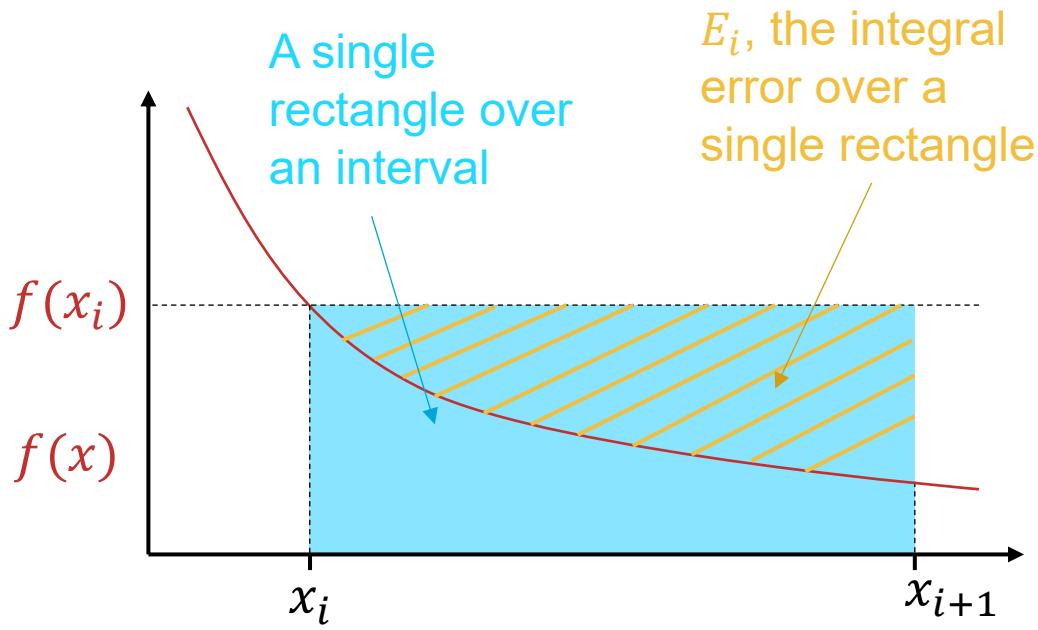
$$E_i = \left| \int_{x_i}^{x_{i+1}} \epsilon_i(x)\, dx \right| = \left| \int_{x_i}^{x_{i+1}} f'(\xi)(x - x_i) dx \right|$$

- Computing the integral and bounding the terms:

$$E_i \leq \max_{\xi \in [x_i, x_{i+1}]} |f'(\xi)| \left. \frac{(x - x_i)^2}{2} \right|_{x_i}^{x_{i+1}}$$

- The integral error on a single rectangle is bounded by a term proportional to the square of the interval:

$$E_i \leq \max_{\xi \in [x_i, x_{i+1}]} |f'(\xi)| \frac{h^2}{2}$$



# Rectangle method: Error analysis (4/4)

For reference, you can find the derivation here

- We can now generalize to the total integral error over the discretization grid
- If n is the number of intervals (number of rectangles), i.e., $nh = b - a$, then:

$$E = nE_i \leq \max_{\xi \in [a,b]} |f'(\xi)| n \frac{h^2}{2} = \max_{\xi \in [a,b]} |f'(\xi)| n\, h \frac{h}{2}$$

$$E \leq \max_{\xi \in [a,b]} |f'(\xi)| (b-a) \frac{h}{2}$$

- The total integral error is bounded by a term that is directly proportional to *h*.
- Thus, when the interval step decreases, the error decreases proportionally.
- The error goes to *zero* when the interval step *h* goes to *zero* as well.

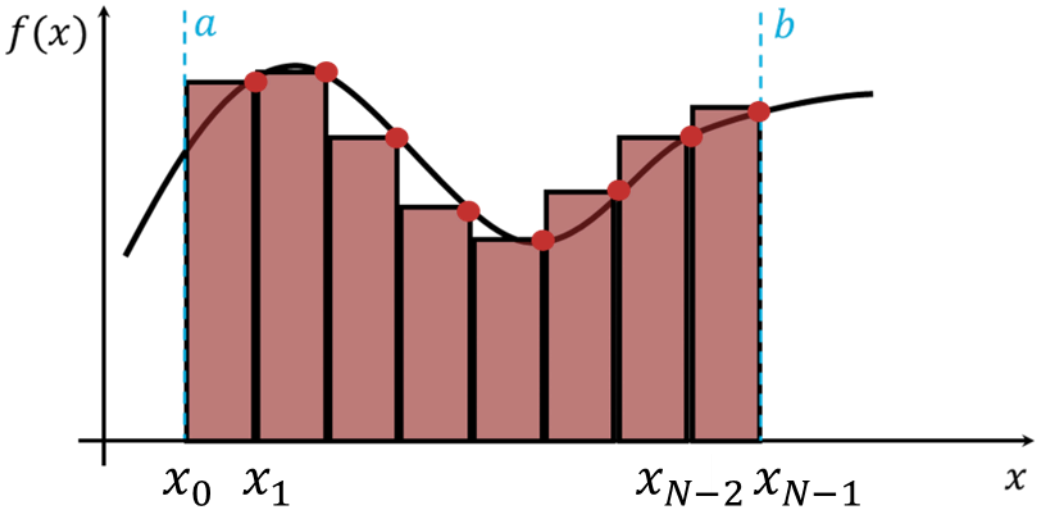
# Rectangle method

Task: compute $I = \int_a^b f(x)dx$

- Number of grid points, $N$
- Number of intervals, $N - 1$
  - Width of interval $h = \frac{b-a}{N-1}$

- $x_i^*$ in right corner
  - $I \approx h \sum_{i=0}^{N-2} f(x_{i+1})$

- Error analysis is equivalent to the rectangle method with left corners.



# Agenda

**Numerical integration**

- Quadrature rules
  - Introduction and overview
  - Rectangle method
  - Newton-Cotes rules

  a. Midpoint method

  b. (Composite) trapezoid method

  c. (Composite) Simpson 1/3 rule
  - Built-in Python functions

**Numerical differentiation**

- Introduction and recap
- Finite difference method
  - First order difference schemes
  - Second order difference schemes
- Built-in Python functions

# Newton-Cotes quadrature

- Newton-Cotes quadrature rules use equally spaced nodes ($x_i, i \in \{0, \dots, N-1\}$) in the integration interval $[a, b]$

$$I = \int_a^b f(x)dx \approx \sum_{i=0}^{N-1} w_i f(x_i)$$

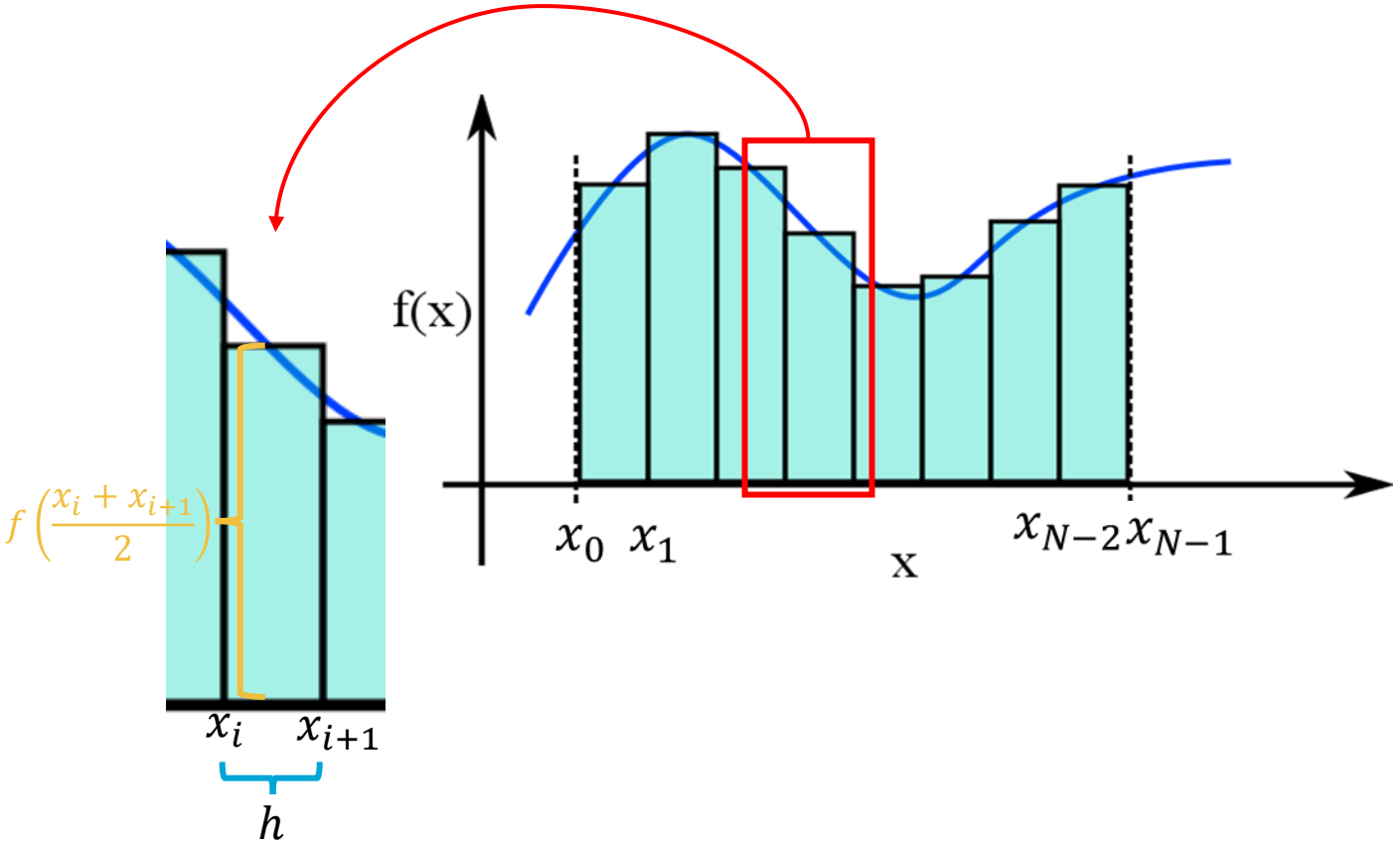
where $x_i$ are called nodes and $w_i$ weights, $i \in [0, \dots, N-1]$.

- Different rules use different nodes and weights.
- Two classes of Newton-Cotes rules:
  - "closed", when they use the function values at the interval endpoints, i.e., $f(x_0 = a)$ and $f(x_0 = a)$ are terms in the summation.
  - "open", when they don't use the function values at the interval endpoints, i.e., $f(x_0 = a)$ and $f(x_0 = a)$ are not terms in the summation.
  - Note that rectangle methods are neither open nor closed. They use only one boundary point. Thus, they do not belong to Newton-Cotes rules!

# Midpoint method

Task: compute $I = \int_a^b f(x)dx$

- Number of grid points, $N$
- Number of intervals, $N - 1$
  - Width of interval $h = \frac{b-a}{N-1}$
- Midpoints → open method
  - $I \approx h \sum_{i=0}^{N-2} f\left(\frac{x_i + x_{i+1}}{2}\right)$



# Midpoint method: Error analysis

- The numerical error arising when using the midpoint method for integration can be estimated similarly to the rectangle method (you can consider a second order Taylor approximation and proceed analogously).
- The midpoint rule provides a more accurate approximation (or faster convergence rate), since the error is bounded by an expression proportional to $h^2$.

$$E \leq \max_{\xi \in [a,b]} |f''(\xi)| (b-a) \frac{h^2}{24}$$

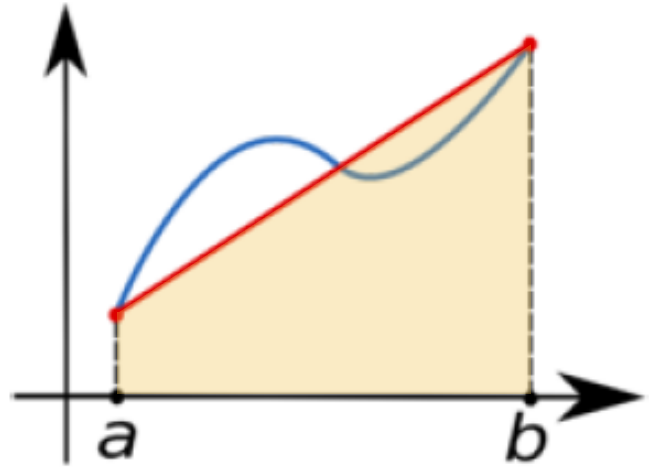
- You can check the full derivation in the reference[1].

[1] https://engcourses-uofa.ca/books/numericalanalysis/numerical-integration/rectangle-method/

# Trapezoid Rule

- On a small interval, a function can be approximated by linear interpolation
- The integral of the function can be approximated by the integral of the interpolant:
  - 2 grid points
  - 1 interval (interval width $h = b - a$)
- Considering the function evaluation at both corners:

$$I_j \approx \frac{h}{2}(f(a) + f(b))$$




**HINT**

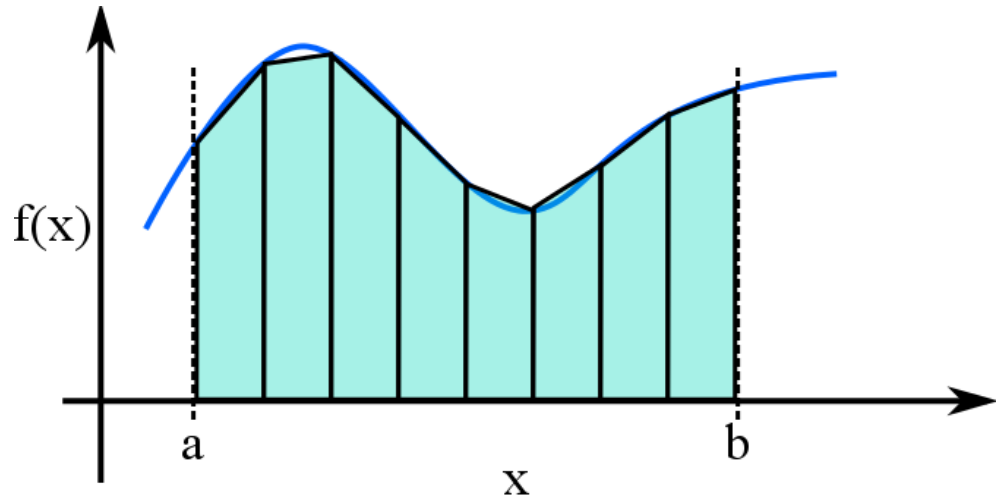
*Note that the integral of the linear interpolation is equal to the area of the trapezoid defined by the linear approximation in the given interval.*

# Composite Trapezoid Method

- You can extend the theory and consider a linear interpolation for each grid interval

Task: compute $I = \int_a^b f(x)dx$

- Number of grid points, $N$
- Number of intervals, $N-1$ (interval width: $h = \frac{b-a}{N-1}$)
- Both corners of the intervals: (closed method)




$$I \approx \sum_{j=0}^{N-2} I_j \text{ , with } I_j = \frac{h}{2}(f(x_j) + f(x_{j+1}))$$

$$I \approx \frac{h}{2}\sum_{j=0}^{N-2}\left(f(x_j) + f(x_{j+1})\right) = \frac{h}{2}\left(f(x_0) + \sum_{j=1}^{N-2}\left(2f(x_j)\right) + f(x_{N-1})\right)$$

**HINT**

*This is computationally more efficient because in the loop (summation) you only have one function evaluation.*

# Composite trapezoid method: Error analysis

- The error introduced by the composite trapezoid method can be computed by considering 4th order Taylor approximation of the function.
- The numerical error committed by using the trapezoid method is of the same order as the midpoint method (~ $h^2$):

$$E \leq \max_{\xi \in [a,b]} |f''(\xi)|(b-a)\frac{h^2}{12}$$

You can check the derivation of error in [1]

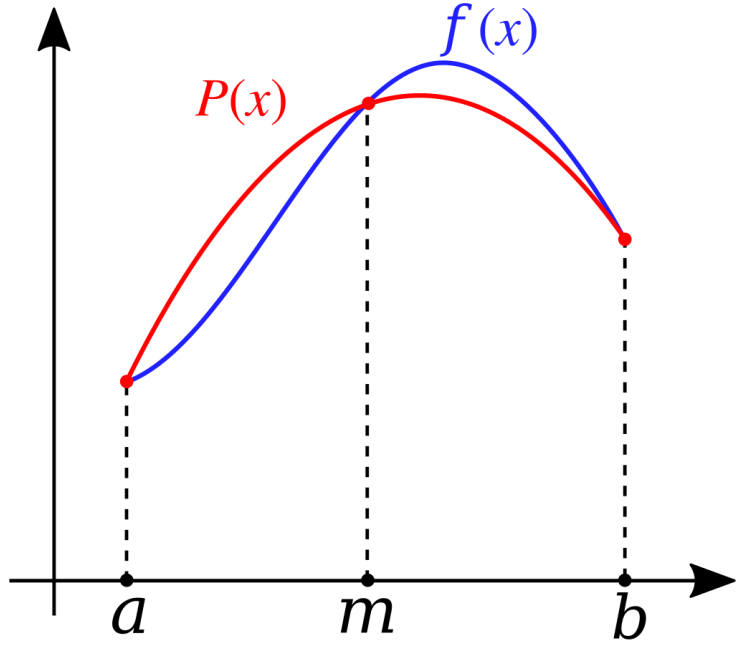
[1] https://engcourses-uofa.ca/books/numericalanalysis/numerical-integration/trapezoidal-rule/

# Simpson's 1/3 Rule

- A better approximation is a quadratic interpolation
- The integral of the function can be approximated by the integral of the interpolant (*more details in Q2 – Lecture 6*):
  - 3 grid points: extremes and halfway point
  - over 2 intervals (interval width $h = \frac{b-a}{2}$)
- Considering the function evaluation in three points:[1]

$$I \approx \frac{h}{3}\left( f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right)$$

- The name "1/3 rule" comes from the normalization coefficient in the formula.




You can check the derivation of the formula in [1]

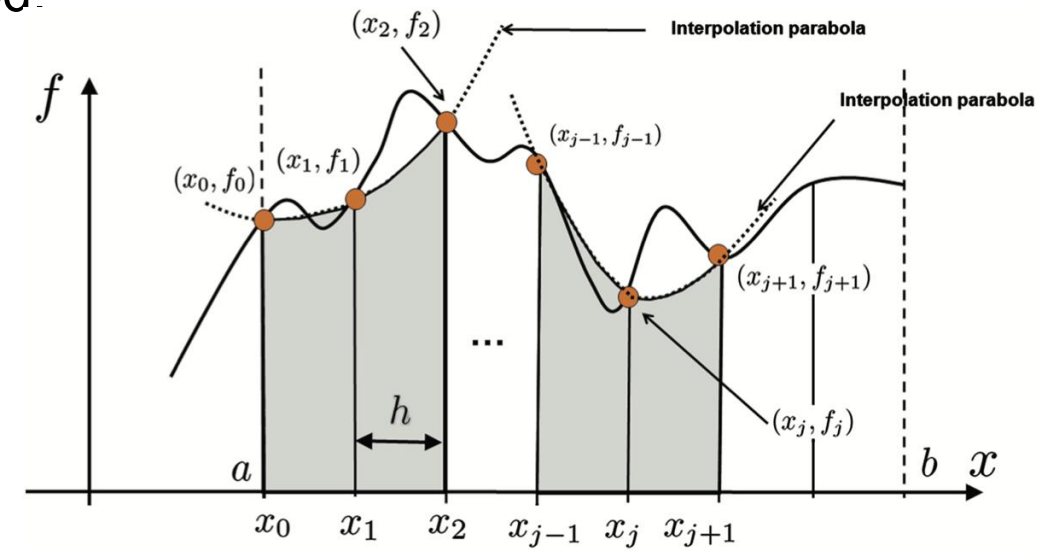
[1] https://engcourses-uofa.ca/books/numericalanalysis/numerical-integration/simpsons-rules/

# Composite Simpson's 1/3 Method

- Following the same reasoning as for trapezoid method we can "compose" Simpson's 1/3 on a discretization grid
- Note that here we consider 2 intervals and 3 discretization points for each sub-integration
- After some computation you can achieve the final formula (closed method):

$$I \approx \frac{h}{3}\left(\sum_{j=0}^{\frac{N-3}{2}} f\left(x_{2\cdot j}\right) + 4f\left(x_{2\cdot j+1}\right) + f\left(x_{2\cdot j+2}\right)\right)$$

$$I \approx \sum_{j=0}^{\frac{N-3}{2}} Simpson_{1/3}\left(\left[x_{2\cdot j}, x_{2\cdot j+2}\right]\right)$$




**Implementation HINT**

*Constraints for domain discretization:*

- *At least 3 discretization points*
- *Odd number of discretization points*

Figure from: https://pythonnumericalmethods.studentorg.berkeley.edu/notebooks/chapter21.04-Simpsons-Rule.html

# Composite Simpson's 1/3 Method

**Illustrative example:**

Grid = $[x_0, x_1, x_2, x_3, x_4, x_5, x_6]$ → number of grid points = 7

Index j is from 0 to 2

- Iterations:
  - Start: $\mathrm{I} = 0$
  - First iteration: $I = I + \frac{\mathrm{h}}{3}(\mathrm{f}(\mathrm{x}_0) + 4\mathrm{f}(\mathrm{x}_1) + \mathrm{f}(\mathrm{x}_2))$ → j=0
  - Second iteration: $I = I + \frac{\mathrm{h}}{3}(\mathrm{f}(\mathrm{x}_2) + 4\mathrm{f}(\mathrm{x}_3) + \mathrm{f}(\mathrm{x}_4))$ → j=1
  - Third iteration: $I = I + \frac{\mathrm{h}}{3}(\mathrm{f}(\mathrm{x}_4) + 4\mathrm{f}(\mathrm{x}_5) + \mathrm{f}(\mathrm{x}_6))$ → j=2
  - End

# Composite Simpson 1/3 method: Error analysis

- Similarly, the error introduced by the composite Simpson 1/3 method can be computed by considering 4th order Taylor approximation of the function.
- The error results to be bounded by a term proportional $h^4$:

$$E \leq \max_{\xi \in [a,b]} \left| f^{(4)}(\xi) \right| (b-a) \frac{h^4}{180}$$

You can check the derivation of error in [1]

[1] https://engcourses-uofa.ca/books/numericalanalysis/numerical-integration/simpsons-rules/

# Summary of quadrature rules

| Method | Formula | Error |
|---|---|---|
| Rectangle method: | $I \approx h\sum_{i=0}^{N-2} f(x_i)$ $or$ $I \approx h\sum_{i=0}^{N-2} f(x_{i+1})$ | $E \leq \max_{\xi\in[a,b]} \lvert f'(\xi)\rvert (b-a)\frac{h}{2}$ |
| Midpoint method: | $I \approx h\sum_{i=0}^{N-2} f\left(\frac{x_i + x_{i+1}}{2}\right)$ | $E \leq \max_{\xi\in[a,b]} \lvert f''(\xi)\rvert (b-a)\frac{h^2}{24}$ |
| Composite trapezoid method: | $I \approx \frac{h}{2}\left(f(x_0) + \sum_{j=1}^{N-2}\left(2f(x_j)\right) + f(x_{N-1})\right)$ | $E \leq \max_{\xi\in[a,b]} \lvert f''(\xi)\rvert (b-a)\frac{h^2}{12}$ |
| Composite Simpson's 1/3 rule method: | $I \approx \frac{h}{3}\left(\sum_{j=0}^{\frac{N-3}{2}} f(x_{2\cdot j}) + 4f(x_{2\cdot j+1}) + f(x_{2\cdot j+2})\right)$ | $E \leq \max_{\xi\in[a,b]} \lvert f^{(4)}(\xi)\rvert (b-a)\frac{h^4}{180}$ |

# Agenda

**Numerical integration**

- Quadrature rules
  - Introduction and overview
  - Rectangle method
  - Newton-Cotes rules

  a. Midpoint method

  b. (Composite) trapezoid method

  c. (Composite) Simpson 1/3 rule
  - Built-in Python functions

**Numerical differentiation**

- Introduction and recap
- Finite difference method
  - First order difference schemes
  - Second order difference schemes
- Built-in Python functions

# Built-in Python functions for numerical integration

## Numpy

- numpy.trapz → Performs numerical integration using the trapezoidal rule.

## SciPy

- scipy.integrate.quad → Computes the definite integral of a function using adaptive quadrature.
- scipy.integrate.simpson → Performs numerical integration using Simpson's rule.

Check the official documentation (in the links) for further details!

# Agenda

**Numerical integration**

- Quadrature rules
  - Introduction and overview
  - Rectangle method
  - Newton-Cotes rules

a. Midpoint method

b. (Composite) trapezoid method

c. (Composite) Simpson 1/3 rule

  - Built-in Python functions

**Numerical differentiation**

- Introduction and recap
- Finite difference method
  - First order difference schemes
  - Second order difference schemes
- Built-in Python functions

# Numerical differentiation: why?

Differentiation allows to measure the rate of variation of variables in space and time.

Example application:

- Residence Time Distribution (RTD)
  - Fluids in a reactor follow different paths → different exit times.
  - Described by the exit age distribution $E(t)$ with units $\left[\frac{1}{\text{time}}\right]$
- Step experiment: Step input: $C_0$ → Measured outlet concentration $C_{out}(t)$ vs. time
- From measurements to distributions
  - Define cumulative distribution (CDF): $F(t) = \frac{C_{out}(t)}{C_0}$ (fraction of tracer exited by time $t$)
  - Obtain RTD: $E(t) = \frac{dC_{out}(t)}{dt}$ → **Numerical differentiation is required, since we only have discrete data points**

Numerical differentiation is a technique used to estimate the derivative of a function using discrete data points.

# From analytical to numerical derivative

- The derivative is defined as the limit of the incremental ratio

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

**Analytical first-order derivative**

- What if we take a discrete interval $\Delta x$ instead of the limit?

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

**Numerical first-order derivative (approximation)**

# Refresh: Taylor Polynomial

- Taylor polynomial allows to express the value of a function $f$ in a generic point $x$ of the domain by expanding around a given point $x_i$:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_i)}{n!}(x - x_i)^n$$

- We can approximate the value of a function by taking a finite number of expansion terms:

$$f(x) = f(x_i) + \frac{df}{dx}\Big|_{x_i}(x - x_i) + \cdots + \frac{d^N f}{dx^N}\Big|_{x_i}(x - x_i)^N + o(\Delta x^N)$$

$$f(x) \cong f(x_i) + \frac{df}{dx}\Big|_{x_i}(x - x_i) + \cdots + \frac{d^N f}{dx^N}\Big|_{x_i}(x - x_i)^N$$

- The approximation error is of the order of the interval to the power of the order of the last expansion term

# Refresh: Taylor Polynomial of second order

- Example: Second order Taylor approximation:

$$f(x) = f(x_i) + \frac{df}{dx}\Big|_{x_i}(x - x_i) + \frac{d^2f}{dx^2}\Big|_{x_i}\frac{(x - x_i)^2}{2} + o(\Delta x^2)$$

$$f(x) \cong f(x_i) + \frac{df}{dx}\Big|_{x_i}(x - x_i) + \frac{d^2f}{dx^2}\Big|_{x_i}\frac{(x - x_i)^2}{2}$$

The approximation error is of the order of the square of the interval:
$(x - x_i)^2 = \Delta x^2$

The values of the derivatives computed at $x_i$ appear in the Taylor expansion

# Agenda

**Numerical integration**

- Quadrature rules
  - Introduction and overview
  - Rectangle method
  - Newton-Cotes rules

a. Midpoint method

b. (Composite) trapezoid method

c. (Composite) Simpson 1/3 rule

  - Built-in Python functions

**Numerical differentiation**

- Introduction and recap
- Finite difference method
  - First order difference schemes
  - Second order difference schemes
- Built-in Python functions

# Finite Difference Method (FDM)

- Given the original function (or discrete values on a grid), it is possible to approximate the value of the derivatives by using Finite Difference Method (FDM)
- The Finite Difference Method approximates the derivatives of a function at discrete points $x_i$ by using differences between function values on a grid ($i = 0, \dots, N-1$).

- We are especially interested in considering numerical strategies for:
  - First-order derivatives:

$$f'(x) = \frac{df}{dx} \rightarrow \left.\frac{df}{dx}\right|_{x_i} \quad i = 0, \dots, N-1$$

  - Second-order derivatives:

$$f''(x) = \frac{d^2f}{dx^2} \rightarrow \left.\frac{d^2f}{dx^2}\right|_{x_i} \quad i = 0, \dots, N-1$$

# Finite Difference Method (FDM)

- Given the original function (or discrete values on a grid), it is possible to approximate the value of the derivatives by using Finite Difference Method (FDM)
- The Finite Difference Method approximates the derivatives of a function at discrete points $x_i$ by using differences between function values on a grid ($i = 1, \dots, N$).

- We are especially interested in considering numerical strategies for:
  - First-order derivatives:

$$f'(x) = \frac{df}{dx} \rightarrow \frac{df}{dx}\Big|_{x_i} \quad i = 0, \dots, N-1$$

  - Second-order derivatives:

$$f''(x) = \frac{d^2f}{dx^2} \rightarrow \frac{d^2f}{dx^2}\Big|_{x_i} \quad i = 0, \dots, N-1$$

# First-order differentiation schemes: **Forward** difference

$$f'(x) = \frac{df}{dx} \rightarrow \left.\frac{df}{dx}\right|_i \quad \forall i \; in \; grid$$

- You can use Taylor expansion centered at $x_i$ to approximate the value of $f(x_{i+1})$

$$f(x_{i+1}) \cong f(x_i) + \left.\frac{df}{dx}\right|_{x_i} (x_{i+1} - x_i) + \left.\frac{d^2 f}{dx^2}\right|_{x_i} \frac{(x_{i+1} - x_i)^2}{2} \qquad (1)$$

- The first-order derivative can be approximated as:

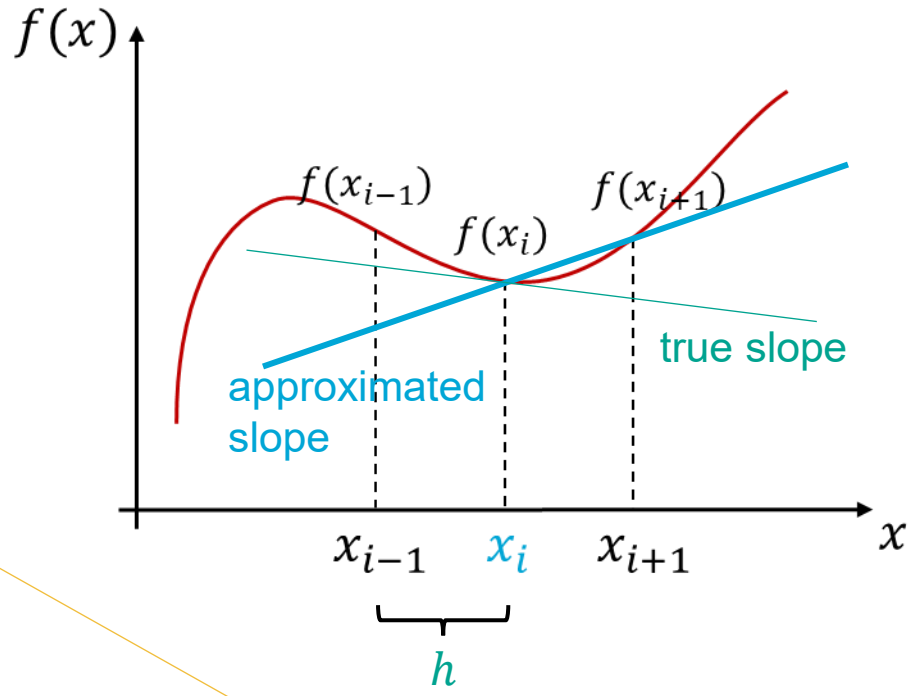
$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_{i+1}) - f(x_i)}{(x_{i+1} - x_i)} - \frac{1}{(x_{i+1} - x_i)} \left.\frac{d^2 f}{dx^2}\right|_{x_i} \frac{(x_{i+1} - x_i)^2}{2}$$

- For small intervals ($h \rightarrow 0$), the second order term is negligible:

$$\left.\frac{df}{dx}\right|_{x_i} \cong \frac{f(x_{i+1}) - f(x_i)}{(x_{i+1} - x_i)}$$

$= h$

First-order forward difference scheme




**Numerical Error**

$$E = \frac{1}{h} \left.\frac{d^2 f}{dx^2}\right|_{x_i} \frac{h^2}{2} \sim \alpha h$$

# First-order differentiation schemes: **Backward** difference

$$f'(x) = \frac{df}{dx} \rightarrow \left.\frac{df}{dx}\right|_i \quad \forall i \; in \; grid$$

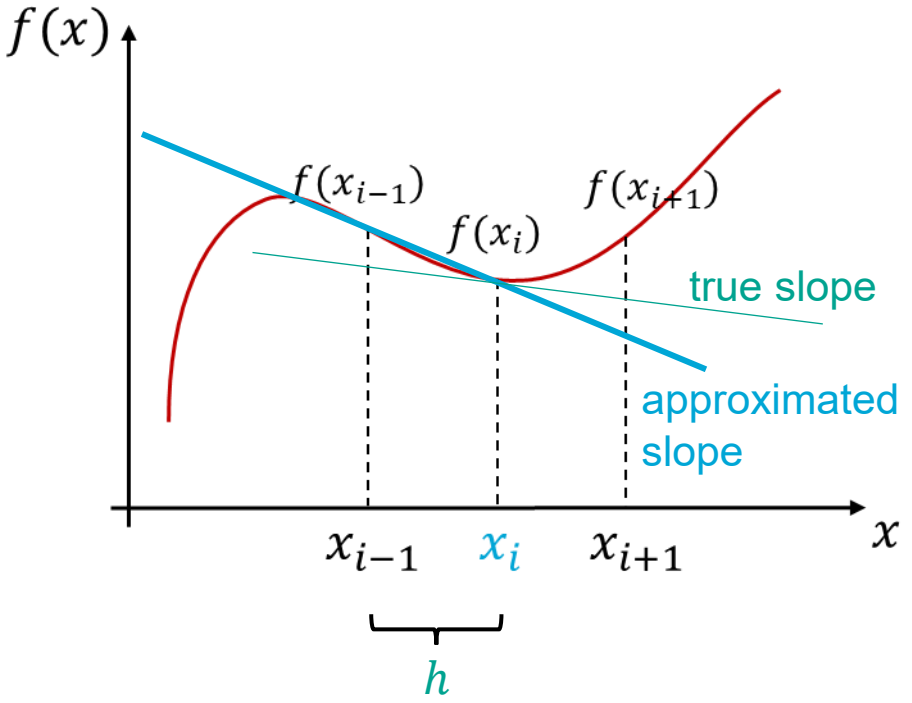
- You can use Taylor expansion centered in $x_i$ to approximate the value of $f(x_{i-1})$

$$f(x_{i-1}) \cong f(x_i) + \left.\frac{df}{dx}\right|_{x_i} (x_{i-1} - x_i) + \left.\frac{d^2 f}{dx^2}\right|_{x_i} \frac{(x_{i-1} - x_i)^2}{2} \qquad (2)$$

- The first order derivative can be approximated as:

$$\left.\frac{df}{dx}\right|_{x_i} \cong \frac{f(x_{i-1}) - f(x_i)}{(x_{i-1} - x_i)} - \frac{1}{h}\left.\frac{d^2 f}{dx^2}\right|_{x_i} \frac{h^2}{2}$$

- For small intervals ($h \rightarrow 0$), the second order term is negligible:




$$\left.\frac{df}{dx}\right|_{x_i} \cong \frac{f(x_i) - f(x_{i-1})}{(x_i - x_{i-1})}$$

$= h$

First-order backward difference scheme

**Numerical Error**

$$E = \frac{1}{h}\left.\frac{d^2 f}{dx^2}\right|_{x_i} \frac{h^2}{2} \sim \alpha h$$

# First-order differentiation schemes: **Central** difference

$$f'(x) = \frac{df}{dx} \rightarrow \left.\frac{df}{dx}\right|_i \quad \forall i \; in \; grid$$

- You can also combine the approximation in $x_{i+1}$ and $x_{i-1}$, by subtracting Eq(2) from Eq(1):

$$f(x_{i+1}) \cong f(x_i) + \left.\frac{df}{dx}\right|_{x_i} (x_{i+1} - x_i) + \left.\frac{d^2f}{dx^2}\right|_{x_i} \frac{(x_{i+1} - x_i)^2}{2} \quad (-)$$

$$f(x_{i-1}) \cong f(x_i) + \left.\frac{df}{dx}\right|_{x_i} (x_{i-1} - x_i) + \left.\frac{d^2f}{dx^2}\right|_{x_i} \frac{(x_{i-1} - x_i)^2}{2} \quad (=)$$

---

$$f(x_{i+1}) - f(x_{i-1}) \cong \cancel{f(x_i)} - \cancel{f(x_i)} + \left.\frac{df}{dx}\right|_{x_i} (x_{i+1} - x_i) - \left.\frac{df}{dx}\right|_{x_i} (x_{i-1} - x_i) + \left.\frac{d^2f}{dx^2}\right|_{x_i} \cancel{\frac{(x_{i+1} - x_i)^2}{2}} - \left.\frac{d^2f}{dx^2}\right|_{x_i} \cancel{\frac{(x_{i-1} - x_i)^2}{2}}$$

**Numerical Error**

Here we only truncate the third order Taylor's term. The second order term just cancels out.

# First-order differentiation schemes: Central difference

$$f'(x) = \frac{df}{dx} \rightarrow \left.\frac{df}{dx}\right|_i \quad \forall i \; in \; grid$$

- Considering the remaining terms:

$$f(x_{i+1}) - f(x_{i-1}) \cong \left.\frac{df}{dx}\right|_{x_i} (x_{i+1} - x_i) - \left.\frac{df}{dx}\right|_{x_i} (x_{i-1} - x_i)$$

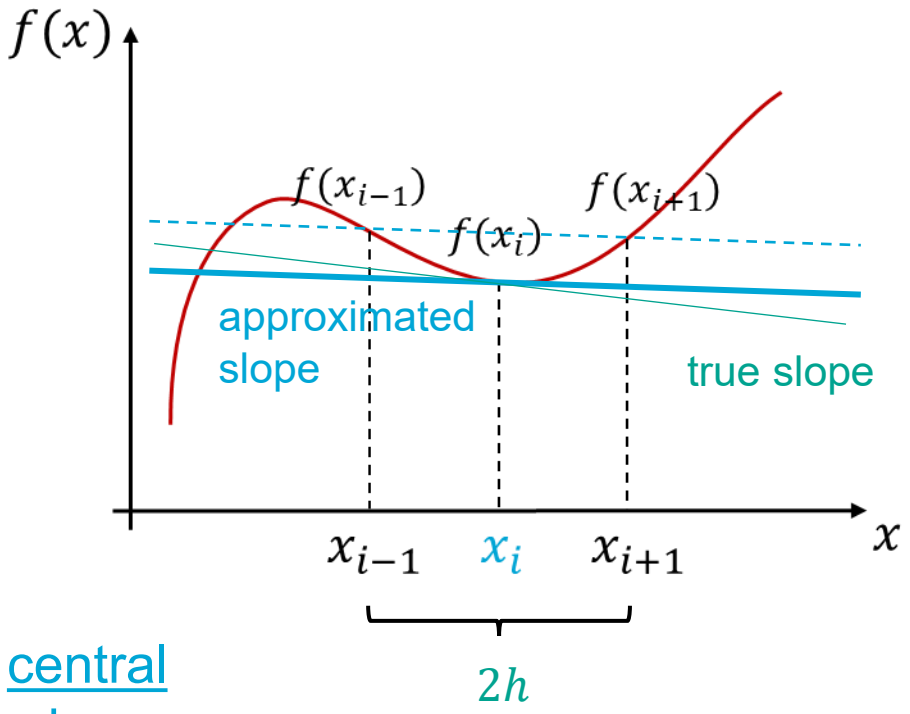
$$f(x_{i+1}) - f(x_{i-1}) \cong \left.\frac{df}{dx}\right|_{x_i} (x_{i+1} - x_i - x_{i-1} + x_i)$$

- Rearranging:

$$\left.\frac{df}{dx}\right|_{x_i} \cong \frac{f(x_{i+1}) - f(x_{i-1})}{(x_{i+1} - x_{i-1})}$$

$= 2h$

First-order central difference scheme




Numerical Error

$$E = \frac{1}{(x_{i+1} - x_{i-1})} \left.\frac{d^3 f}{dx^3}\right|_{x_i} \frac{h^3}{3} \sim \alpha h^2$$

# Recap: First order finite differences

| Method | Formula | Error |
|---|---|---|
| Forward difference | $\frac{df}{dx}\Big\|_{x_i} \cong \frac{f(x_{i+1}) - f(x_i)}{(x_{i+1} - x_i)}$ | $E = \frac{1}{(x_{i+1} - x_i)} \frac{d^2 f}{dx^2}\Big\|_{x_i} \frac{h^2}{2} \sim \alpha h$ |
| Backward difference | $\frac{df}{dx}\Big\|_{x_i} \cong \frac{f(x_i) - f(x_{i-1})}{(x_i - x_{i-1})}$ | $E = \frac{1}{(x_{i+1} - x_i)} \frac{d^2 f}{dx^2}\Big\|_{x_i} \frac{h^2}{2} \sim \alpha h$ |
| Central difference | $\frac{df}{dx}\Big\|_{x_i} \cong \frac{f(x_{i+1}) - f(x_{i-1})}{(x_{i+1} - x_{i-1})}$ | $E = \frac{1}{(x_{i+1} - x_{i-1})} \frac{d^3 f}{dx^3}\Big\|_{x_i} \frac{h^3}{3} \sim \alpha h^2$ |

# Agenda

**Numerical integration**

- Quadrature rules
  - Introduction and overview
  - Rectangle method
  - Newton-Cotes rules

a. Midpoint method

b. (Composite) trapezoid method

c. (Composite) Simpson 1/3 rule

  - Built-in Python functions

**Numerical differentiation**

- Introduction and recap
- Finite difference method
  - First order difference schemes
  - Second order difference schemes
- Built-in Python functions

# Second-order differentiation schemes

$$f''(x) = \frac{d^2f}{dx^2} \rightarrow \left.\frac{d^2f}{dx^2}\right|_i \quad \forall i \; in \; grid$$

- We limit the derivation of numerical approximation of second-order derivatives to the central difference scheme
- Considering again Eq(1) and Eq(2), we can this time **sum** them up:

$$f(x_{i+1}) \cong f(x_i) + \left.\frac{df}{dx}\right|_{x_i}(x_{i+1} - x_i) + \left.\frac{d^2f}{dx^2}\right|_{x_i}\frac{(x_{i+1} - x_i)^2}{2} \quad \oplus$$

$$f(x_{i-1}) \cong f(x_i) + \left.\frac{df}{dx}\right|_{x_i}(x_{i-1} - x_i) + \left.\frac{d^2f}{dx^2}\right|_{x_i}\frac{(x_{i-1} - x_i)^2}{2} \quad =$$

---

$$f(x_{i+1}) + f(x_{i-1}) \cong f(x_i) + f(x_i) + \cancel{\left.\frac{df}{dx}\right|_{x_i}(x_{i+1} - x_i)} + \cancel{\left.\frac{df}{dx}\right|_{x_i}(x_{i-1} - x_i)} + \left.\frac{d^2f}{dx^2}\right|_{x_i}\frac{(x_{i+1}-x_i)^2}{2} + \left.\frac{d^2f}{dx^2}\right|_{x_i}\frac{(x_{i-1}-x_i)^2}{2}$$

$x_{i+1} - x_i = h$, $\quad x_{i-1} - x_i = -h$

$$\left.\frac{df}{dx}\right|_{x_i}(x_{i+1} - x_i) + \left.\frac{df}{dx}\right|_{x_i}(x_{i-1} - x_i) = \left.\frac{df}{dx}\right|_{x_i}(h) + \left.\frac{df}{dx}\right|_{x_i}(-h) = 0$$

# Second-order differentiation schemes

$$f''(x) = \frac{d^2 f}{dx^2} \rightarrow \left.\frac{d^2 f}{dx^2}\right|_i \quad \forall i \; in \; grid$$

- Considering the remaining terms: (*note:* $(x_{i+1} - x_i)^2 = (x_{i-1} - x_i)^2 = \mathrm{h}^2$)

$$f(x_{i+1}) + f(x_{i-1}) \cong 2f(x_i) + 2\left.\frac{d^2 f}{dx^2}\right|_{x_i} \frac{h^2}{2}$$

- Rearranging:

$$\left.\frac{d^2 f}{dx^2}\right|_{x_i} \cong \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2}$$

Second-order central difference scheme

**Numerical Error**

Similarly to 1st order CDS.

$$E = \frac{1}{h^2}\left.\frac{d^4 f}{dx^4}\right|_{x_i} \frac{h}{12}^4 \sim \alpha h^2$$

# Overview: Second order finite differences

| Method | Formula | Error |
|---|---|---|
| Central difference | $\left.\frac{d^2f}{dx^2}\right\|_{x_i} \cong \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2}$ | $E = \frac{1}{(x_{i+1} - x_i)^2}\left.\frac{d^4f}{dx^4}\right\|_{x_i}\frac{h}{24}^4 \sim \alpha h^2$ |
| Forward difference | $\left.\frac{d^2f}{dx^2}\right\|_{x_i} \cong \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2}$ | $E \sim \alpha h$ |
| Backward difference | $\left.\frac{d^2f}{dx^2}\right\|_{x_i} \cong \frac{f(x_i) - 2f(x_{i-1}) + f(x_{i-2})}{h^2}$ | $E \sim \alpha h$ |

# Agenda

**Numerical integration**

- Quadrature rules
  - Introduction and overview
  - Rectangle method
  - Newton-Cotes rules

a. Midpoint method

b. (Composite) trapezoid method

c. (Composite) Simpson 1/3 rule

  - Built-in Python functions

**Numerical differentiation**

- Introduction and recap
- Finite difference method
  - First order difference schemes
  - Second order difference schemes
- Built-in Python functions

# Built-in Python functions for numerical differentiation

- `np.diff` (check documentation)
  - Computes the difference between consecutive elements of an array
  - Given two array: f (dependent variable) and x (independent variable) you can, for instance, obtain the first order forward difference scheme as:
    df_dt = np.diff(f) / np.diff(x)
  - You can play with the parameter `n` in order to achieve other difference schemes. Check it out in the official documentation!

# Numerical differentiation: Live coding

- Open Colab: Numerical differentiation

- Find more in the Github repository of the course: https://github.com/process-intelligence-research/computational_practicum_lecture_coding/tree/main

# Learning goals of this lecture

After successfully completing this lecture, you are able to…

- explain methods for numerical integration and differentiation
- implement numerical methods for integration and differentiation in Python from scratch
- use Python libraries' built-in functions for numerical integration and differentiation

# Thank you very much for your attention!