Computer Methods (MAE 3403)

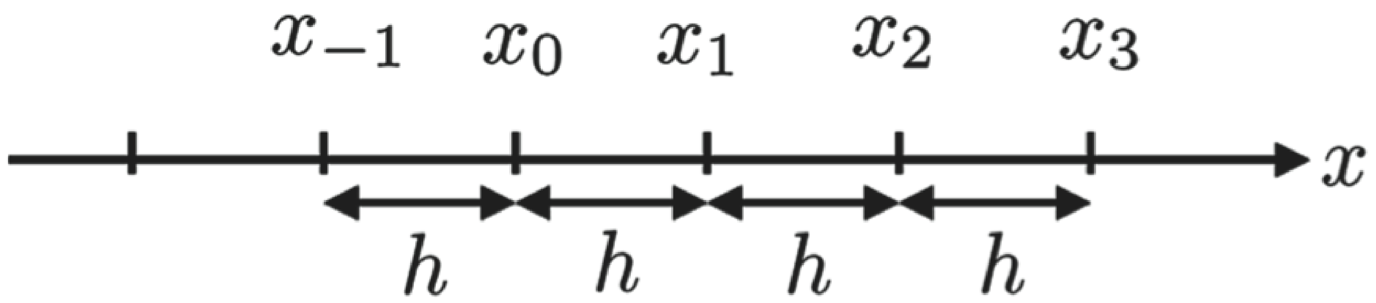
Numerical differentiation and integration

Motivation

- Many systems change over time, space and other dimensions of interest. Such changes are modelled as function derivatives.
- In practice, the function may not be known or the function may be implicitly known by data points.
- Can we compute derivatives numerically rather than analytically?

Numerical differentiation

- Compute df(x)/dx numerically
- Numerical grids: linspace in 1-D

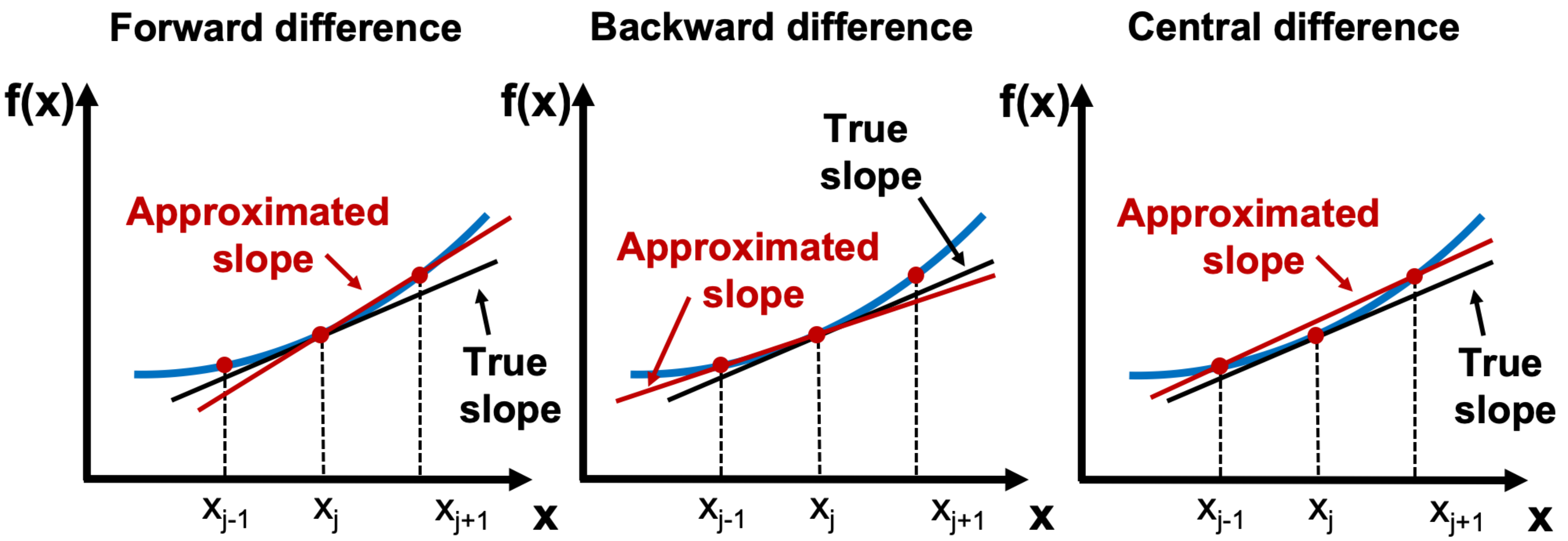


- Although a function may be continuous, its discretized version is more useful for differentiation and integration.

Finite difference approximating derivatives

$$f'(a) = \lim_{x \to a} \frac{f(x) - f(a)}{x - a}$$

- We use values in the neighborhood of a to approximate the derivative



Three types of differences as derivative

- Forward difference

$$f'(x_j) \approx \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j}$$

- Backward difference

$$f'(x_j) \approx \frac{f(x_j) - f(x_{j-1})}{x_j - x_{j-1}}$$

- Central difference (better accuracy)

$$f'(x_j) \approx \frac{f(x_{j+1}) - f(x_{j-1})}{x_{j+1} - x_{j-1}}$$

Python

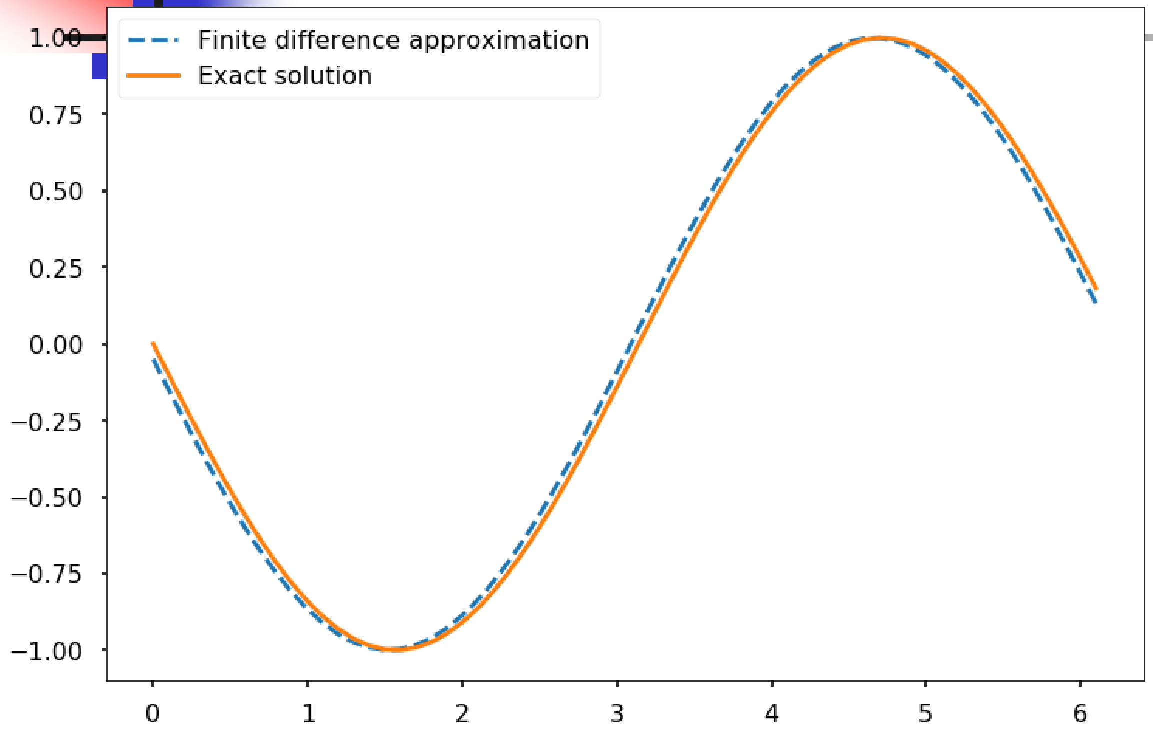
- Finite difference computation: d=np.diff(f)
  - $d(i) = f(i+1)-f(i)$

- The size of the output is one less than the size of the input

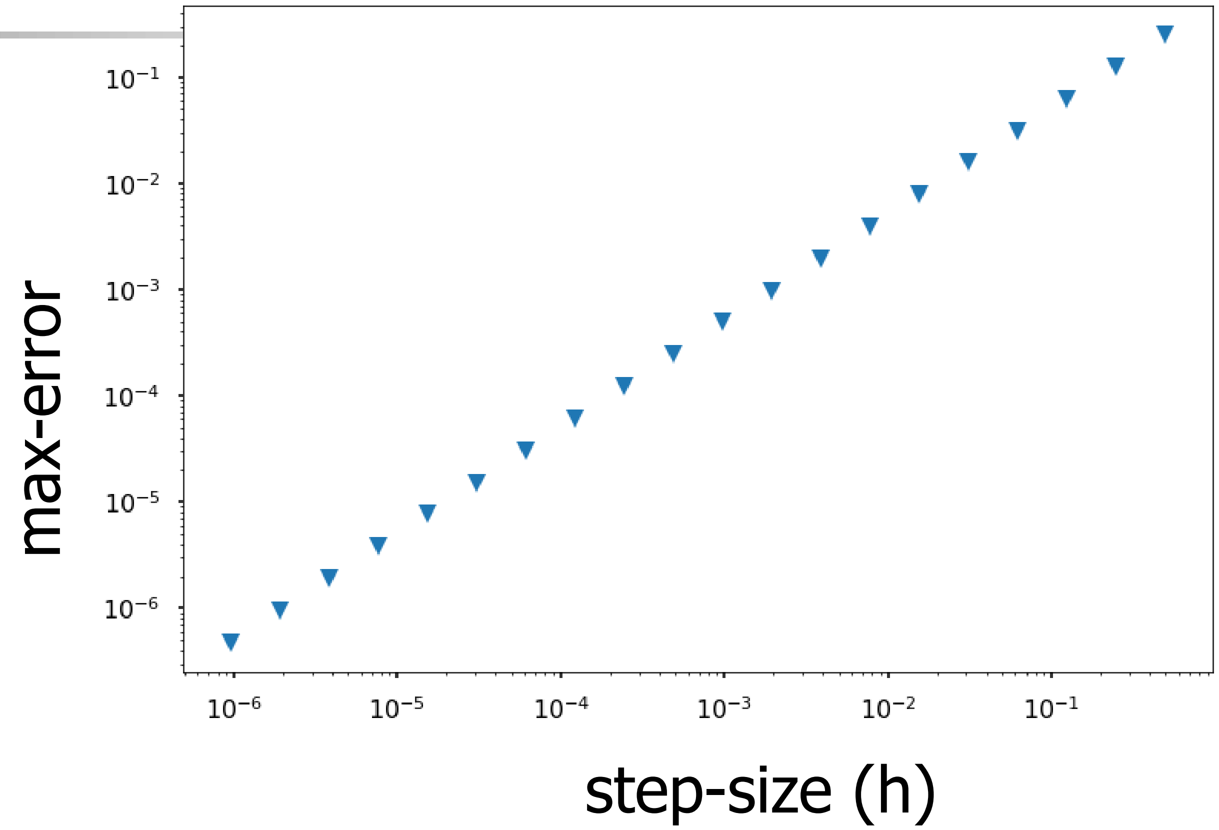
Example

```
import numpy as np
import matplotlib.pyplot as plt

# step size
h = 0.1
# define grid
x = np.arange(0, 2*np.pi, h)
# compute function
y = np.cos(x)
# compute vector of forward differences
forward_diff = np.diff(y)/h
# compute corresponding grid
x_diff = x[:-1:]
# compute exact solution
exact_solution = -np.sin(x_diff)
# Plot solution
plt.figure(figsize = (12, 8))
plt.plot(x_diff, forward_diff, '--', \ label =
'Finite difference approximation')
plt.plot(x_diff, exact_solution, \ label =
'Exact solution’)
plt.legend()
plt.show()
# compute max error
max_error = max(abs(exact_solution -
forward_diff))
print(max_error)
```



Finite difference approximation
Exact solution
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5
6



max-error
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-6}
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
step-size (h)

Higher-order derivatives

- Use Taylor series

$$f(x_{j-1}) = f(x_j) - hf'(x_j) + \frac{h^2 f''(x_j)}{2} - \frac{h^3 f'''(x_j)}{6} + \cdots$$

$$f(x_{j+1}) = f(x_j) + hf'(x_j) + \frac{h^2 f''(x_j)}{2} + \frac{h^3 f'''(x_j)}{6} + \cdots .$$

$$f(x_{j-1}) + f(x_{j+1}) = 2f(x_j) + h^2 f''(x_j) + \frac{h^4 f''''(x_j)}{24} + \cdots ,$$

$$\boxed{f''(x_j) \approx \frac{f(x_{j+1}) - 2f(x_j) + f(x_{j-1})}{h^2}}$$

Sensitivity w.r.t. noise

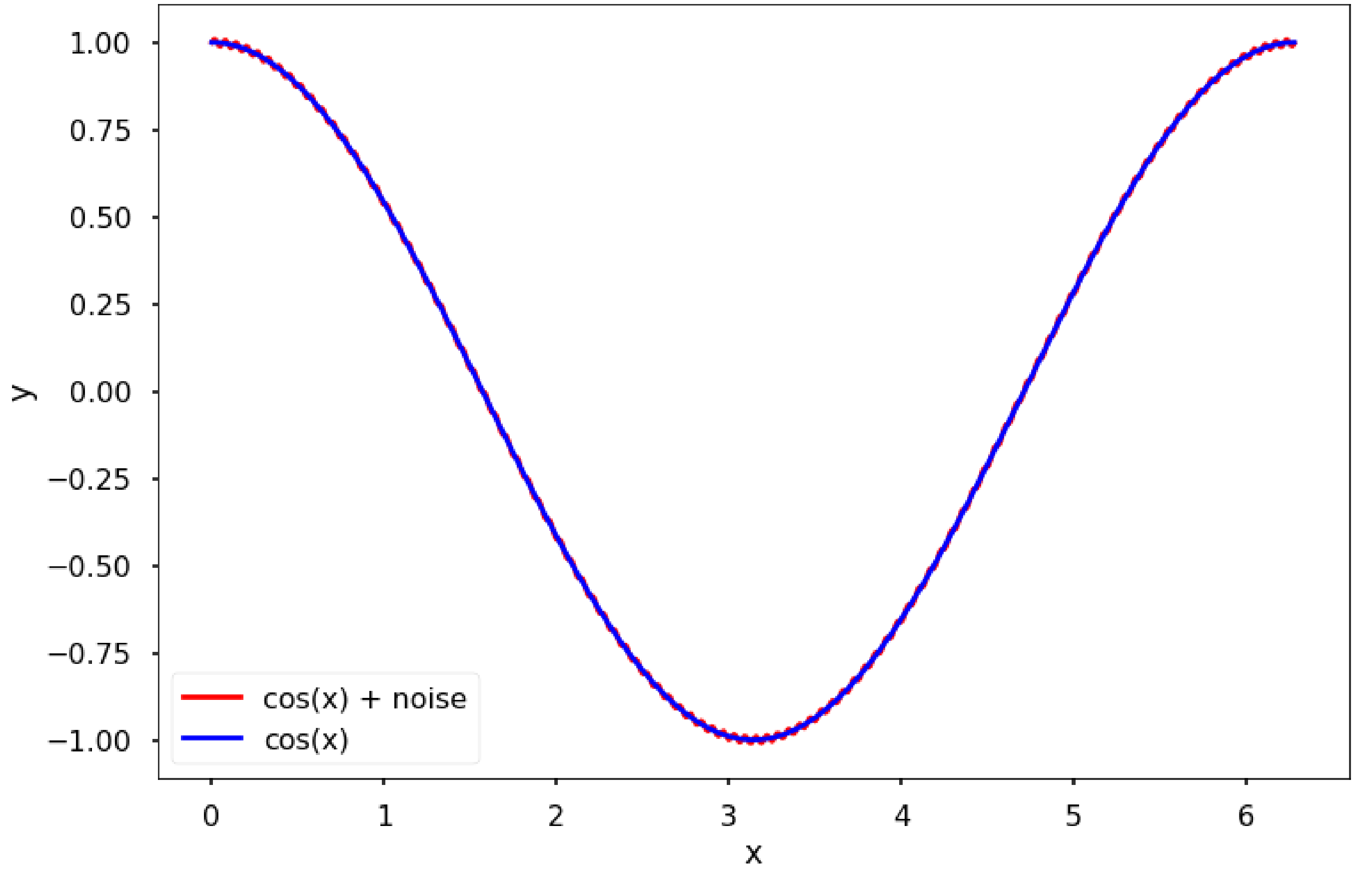
- Sometimes data are contaminated with noise, i.e., data = theoretical value + random offset

- Differentiation is sensitive to noise
  - Even if noise is small, its derivative may be significant

- Consider f(x) = cos(x), and f_noise = cos(x) + e*sin(wx), where e = 0.01 and w = 100.

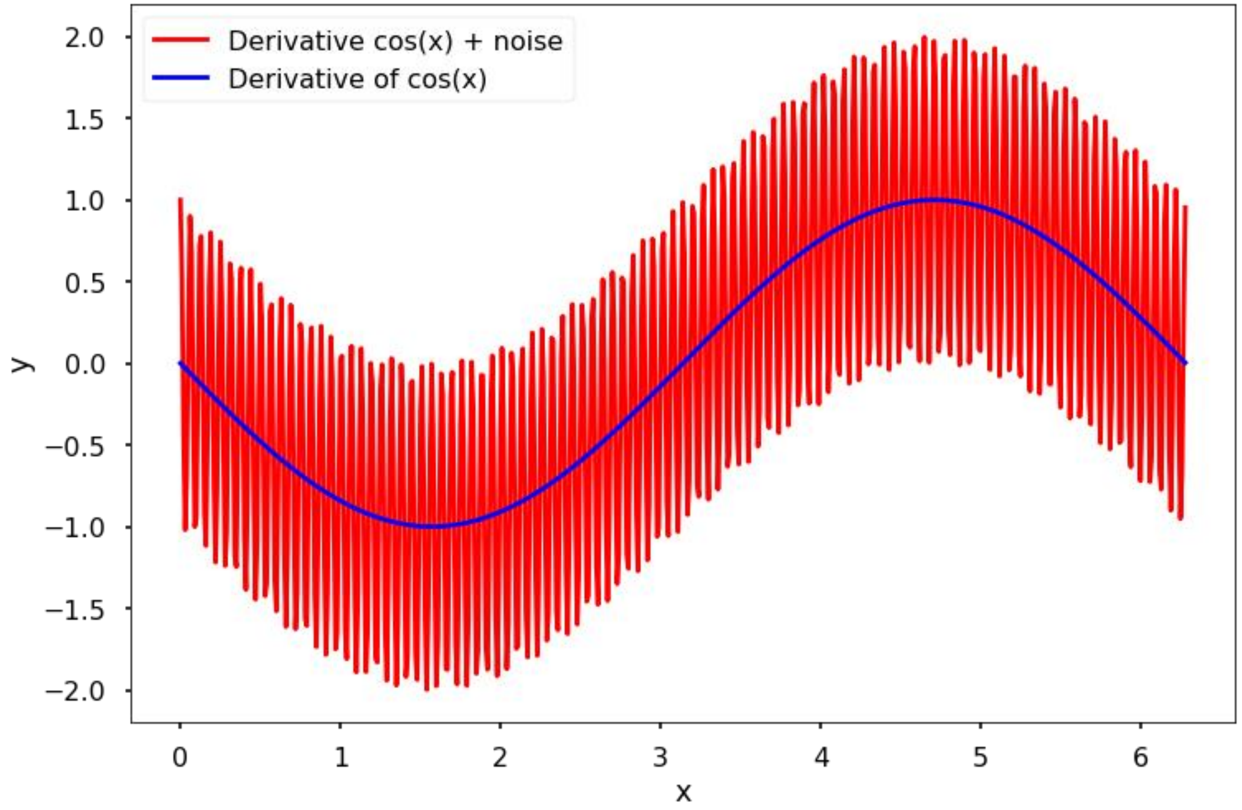
Plot of f vs. f_noise

Can you write code to compare f and f_noise in a figure?



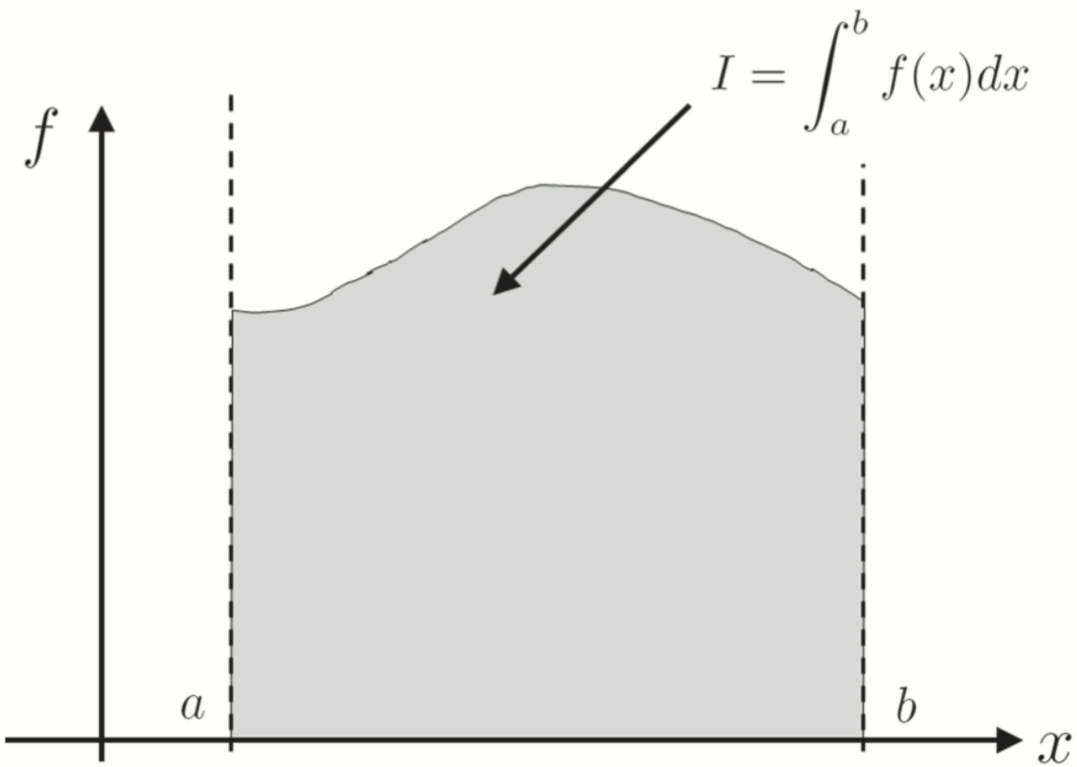
Compare f' and f'_noise

- Can you write code to compare their derivatives?



Numerical Integration

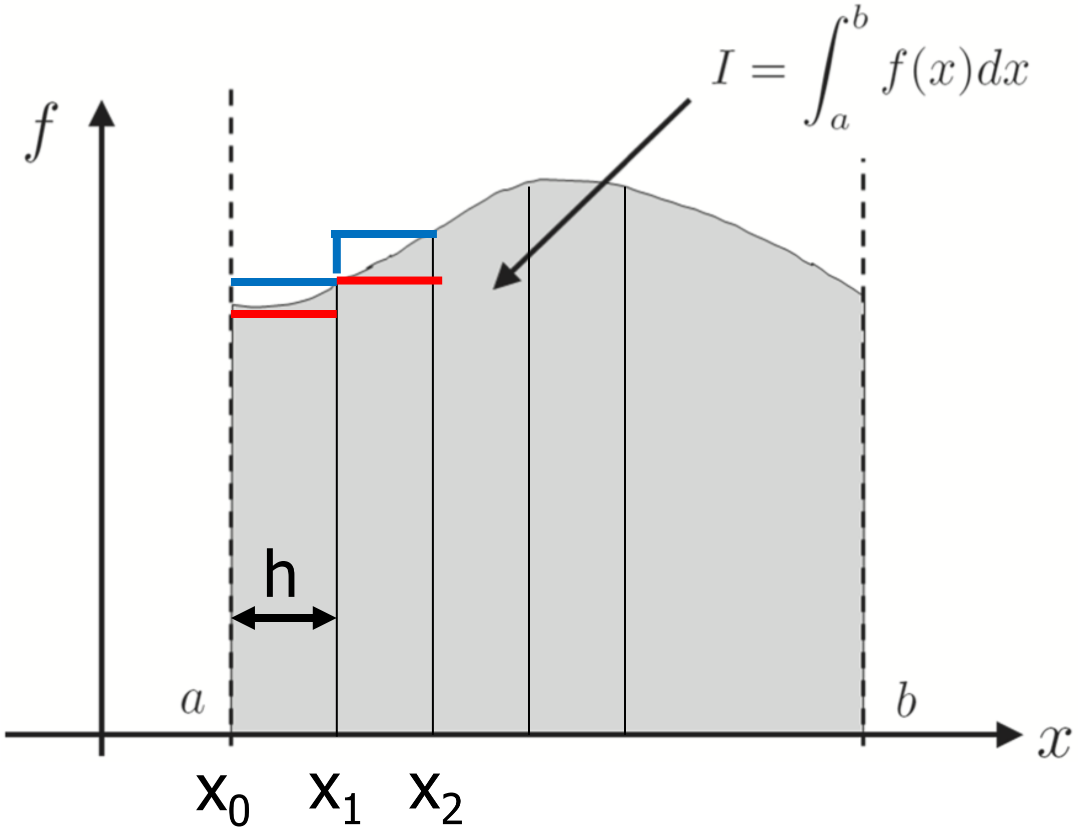
- Compute the integral of f(x) given f(x)
- Such an integral typically described as *area under the curve.*
- Many applications in modeling, predicting, and understanding of physics.



Typical process to approximate integral

- Discretize the interval [a,b] into a numerical grid, consisting of n+1 points with spacing h = (b-1)/n.
- Let x_i be the corresponding ith grid points (x_0=a, x_n=b).
- We can compute $f(x_i)$, i=0,…,n.
- The integral is typically approximated as the sum of the areas for each subinterval $[x_i, x_{i+1})$.

Riemanns integral



- Summing the area of *rectangles*
  - Left

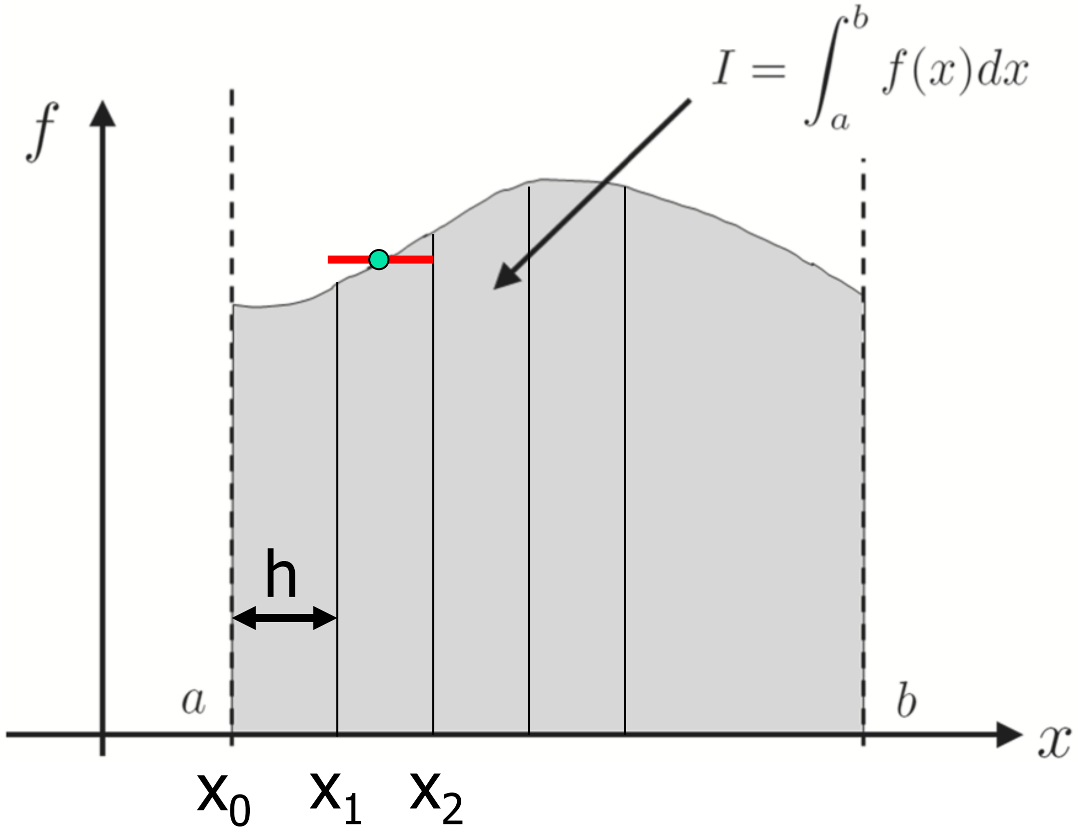
    $$\boxed{\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} hf(x_i),}$$

  - Right

    $$\boxed{\int_a^b f(x)dx \approx \sum_{i=1}^{n} hf(x_i),}$$

- Overall accuracy: *O(h)*

Midpoint rule



$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} hf\left(\frac{x_i + x_{i+1}}{2}\right).$$

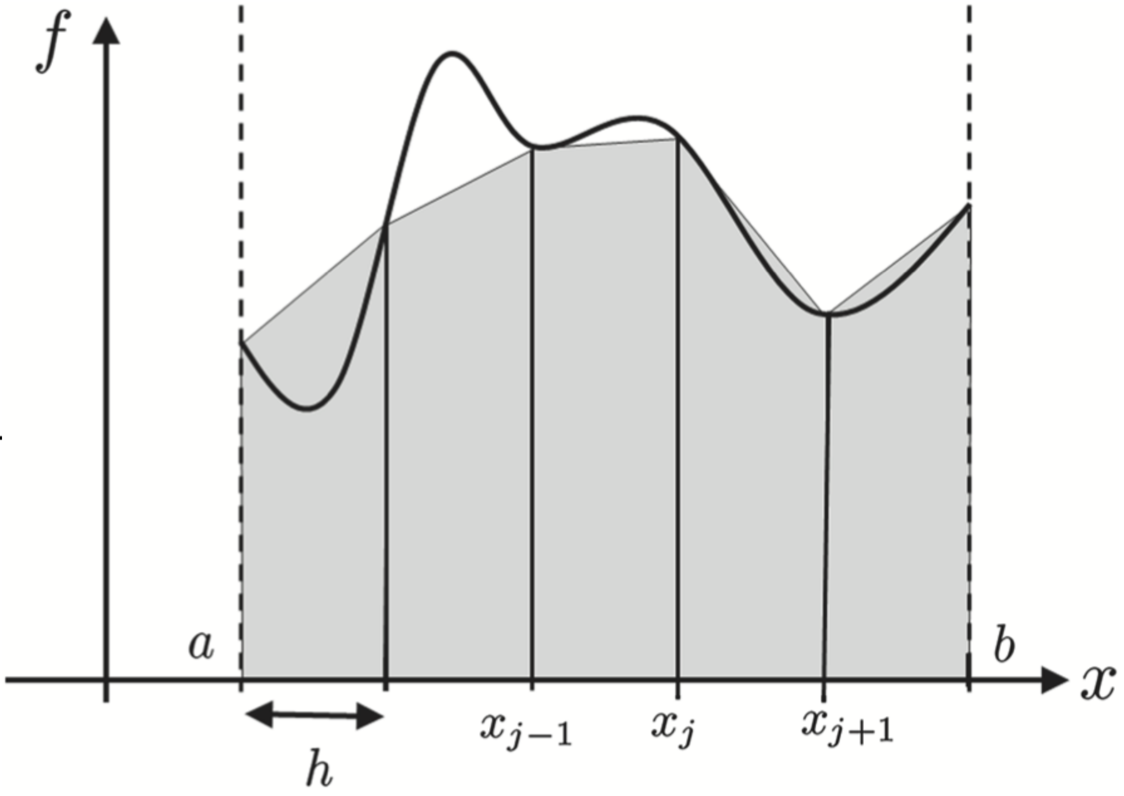
- Overall accuracy: $O(h^2)$, better than the previous two methods
- If $f(x)$ is given as data points, we can't use this rule.

Trapezoid Rule

- Fits a trapezoid into each subinterval and sums the areas of the trapezoids.

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} h\frac{f(x_i)+f(x_{i+1})}{2}$$

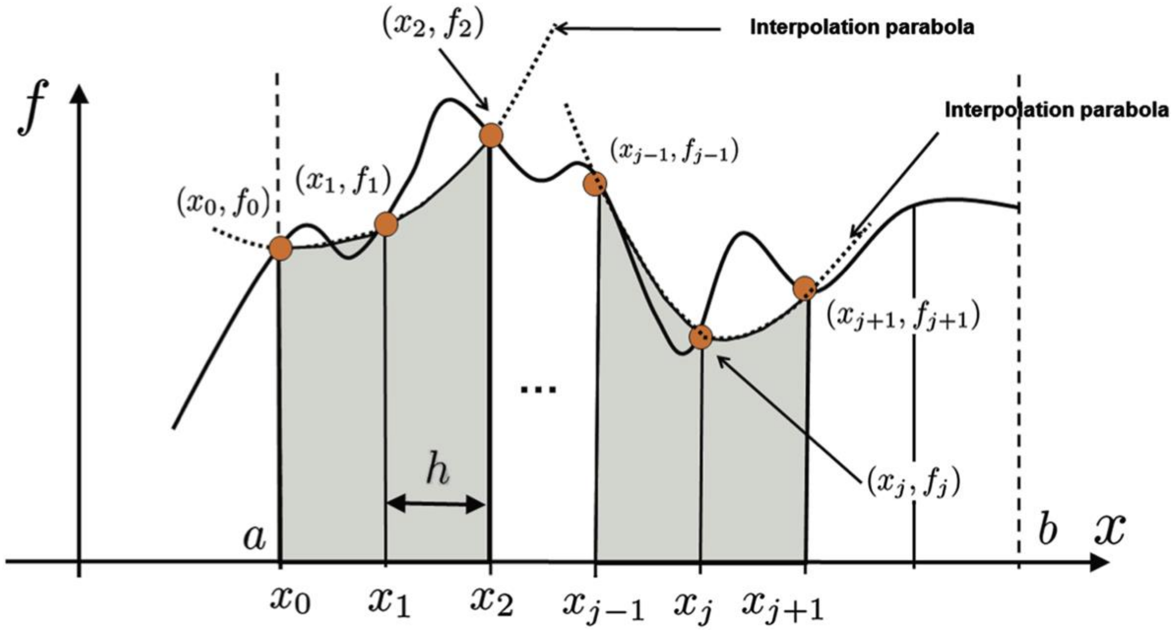
- Overall accuracy: $O(h^2)$
- Simplified form:



$$\int_a^b f(x)dx \approx \frac{h}{2}\left(f(x_0)+2\left(\sum_{i=1}^{n-1} f(x_i)\right)+f(x_n)\right)$$

Simpson's Rule

- Approximates the area under f(x) over two consecutive subintervals by fitting a quadratic polynomial through $(x_{i-1},f(x_{i-1}))$, $(x_i,f(x_i))$, $(x_{i+1},f(x_{i+1}))$

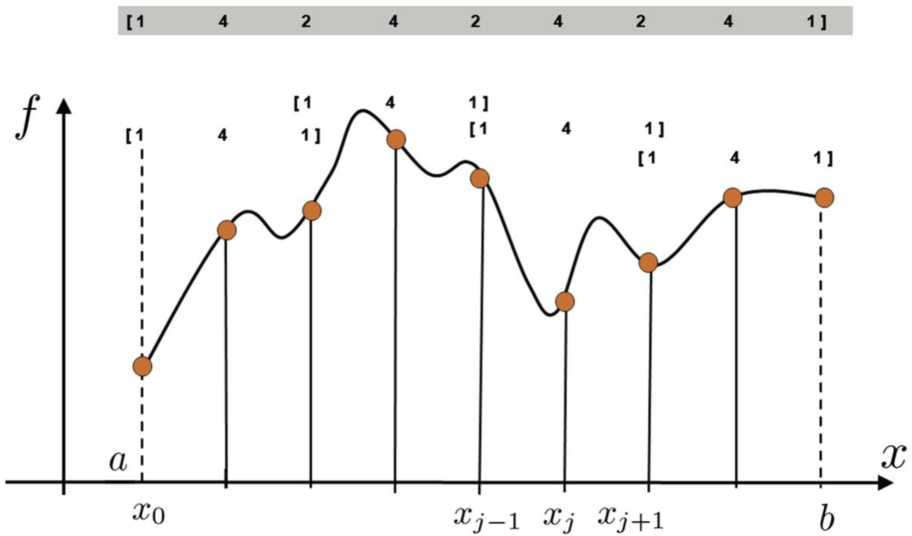


Formula

- Must have an even number of intervals (i.e., an odd number of grid points)

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 4\left(\sum_{i=1, i \text{ odd}}^{n-1} f(x_i) \right) + 2\left(\sum_{i=2, i \text{ even}}^{n-2} f(x_i) \right) + f(x_n) \right].$$

- Overall accuracy: $O(h^4)$



Example

- Use the left Riemann Integral, right Riemann Integral, Midpoint Rule, Trapezoid Rule, Simpson’s Rule to approximate the integration of sin(x) from 0 to pi with 11 evenly spaced grid points over the whole interval. Compare this value to the exact value of 2.

Python implementations

- The scipy.integrate submodule has several functions related to integrals
- trapz takes an array of function values of f on a numerical grid and computes the integral using trapezoid rule

```
import numpy as np
from scipy.integrate import trapz
a = 0
b = np.pi
n = 11
h = (b - a) / (n - 1)
x = np.linspace(a, b, n)
f = np.sin(x)
I_trapz = trapz(f,x)
I_trap = (h/2)*(f[0] + 2 * sum(f[1:n-1]) + f[n-1])
print(I_trapz)
print(I_trap)
```

The quad function

- quad(f, a, b): use different numerical techniques to integrate by function object f from a to b

```
from scipy.integrate import quad
I_quad, est_err_quad = \
                quad(np.sin, 0, np.pi)
print(I_quad)
err_quad = 2 - I_quad
print(est_err_quad, err_quad)
```