PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNIK DAN INFORMATIKA
UNIVERSITAS MULTIMEDIA NUSANTARA
SEMESTER GENAP TAHUN AJARAN 2024/2025

IF420 – ANALISIS NUMERIK

Pertemuan ke 11 – Numerical Integration

Dr. Ivransa Zuhdi Pane, M.Eng., B.CS.
Marlinda Vasty Overbeek, S.Kom., M.Kom.
Seng Hansun, S.Si., M.Cs.

Capaian Pembelajaran Mingguan Mata Kuliah (Sub-CPMK):

Sub-CPMK 11: Mahasiswa mampu memahami dan menerapkan teknik integrasi numerik – C3

Reviews

- Numerical Differentiation Problem Statement
- Finite Difference Approximating Derivatives
- Approximation of Higher Order Derivatives
- Numerical Differentiation with Noise

UMN
UNIVERSITAS MULTIMEDIA NUSANTARA

Outlines

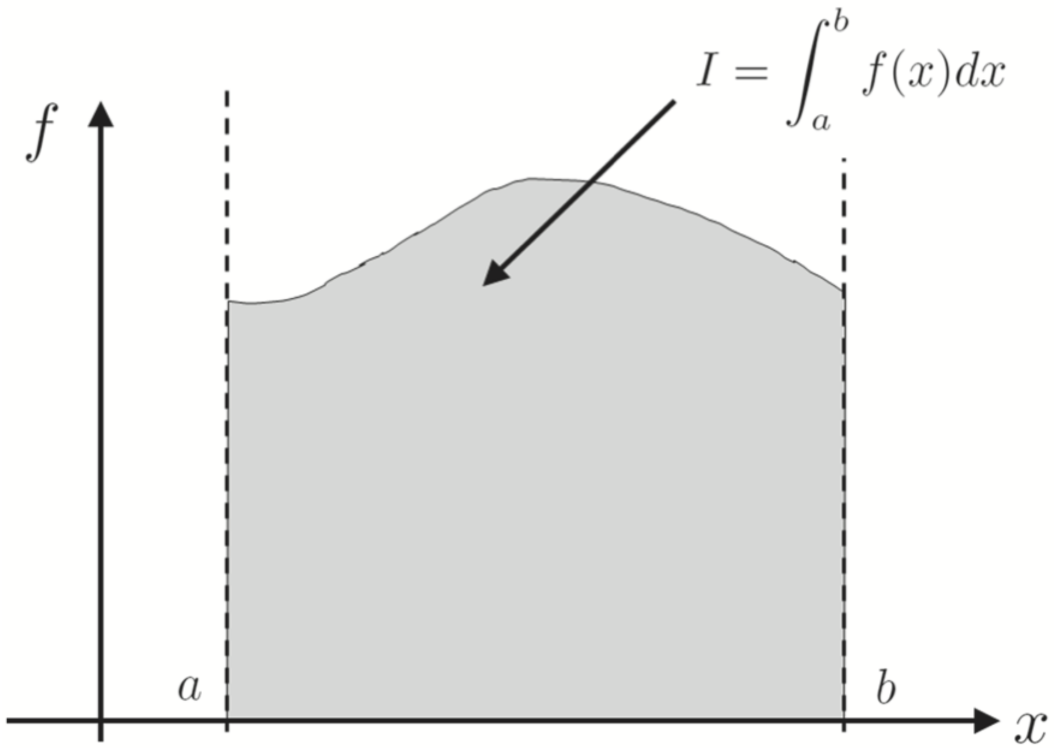
- Numerical Integration Problem Statement
- Riemann's Integral
- Trapezoid Rule
- Simpson's Rule
- Computing Integrals in Python

Motivation

- The **integral** of a function is normally described as the "**area under the curve**."
- In engineering and science, the **integral** has many applications for **modeling**, **predicting**, and **understanding** physical systems.
- However, in practice, finding an **exact solution** for the integral of a function is **difficult** or **impossible**.

Numerical Integration Problem Statement

- Given a function $f(x)$, we want to approximate the **integral** of $f(x)$ over the **total interval**, $[a, b]$.
- The following figure illustrates this area.



- To accomplish this goal, we assume that the interval has been **discretized** into a numerical **grid**, x, consisting of $n+1$ points with **spacing**, $h = \frac{b-a}{n}$.
- Here, we denote each **point** in x by x_i, where $x_0 = a$ and $x_n = b$.
- **Note**: There are $n+1$ grid points because the count starts at x_0.
- We also assume we have a **function**, $f(x)$, that can be computed for any of the grid points, or that we have been given the function **implicitly** as $f(x_i)$.
- The **interval** $[x_i, x_{i+1}]$ is referred to as a **subinterval**.
- The following sections give some of the most **common methods** of **approximating** $\int_a^b f(x)dx$.
- Each method **approximates** the **area under** $f(x)$ for each **subinterval** by a shape for which it is easy to compute the exact area, and then **sums** the **area** contributions of every **subinterval**.

Riemann's Integral

- The **simplest** method for **approximating integrals** is by **summing the area of rectangles** that are defined for each **subinterval**.
- The **width** of the **rectangle** is $x_{i+1} - x_i = h$, and the **height** is defined by a function value $f(x)$ for some x in the subinterval.
- An obvious choice for the **height** is the function value at the **left endpoint**, x_i, or the **right endpoint**, x_{i+1}, because these values can be used even if the function itself is not known.
- This method gives the **Riemann Integral approximation**, which is

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

depending on whether the **left** or **right endpoint** is chosen.

Riemann's Integral

- As with **numerical differentiation**, we want to characterize how the **accuracy improves** as h gets **small**.
- To determine this characterizing, we first rewrite the integral of $f(x)$ over an **arbitrary subinterval** in terms of the **Taylor series**.
- The **Taylor series** of $f(x)$ around $a = x_i$ is

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \cdots$$

- Thus

$$\int_{x_i}^{x_{i+1}} f(x)dx = \int_{x_i}^{x_{i+1}} (f(x_i) + f'(x_i)(x - x_i) + \cdots)dx$$

by **substitution** of the Taylor series for the function.

Riemann's Integral

- Since the integral **distributes**, we can rearrange the **right side** into the following form:

$$\int_{x_i}^{x_{i+1}} f(x_i)dx + \int_{x_i}^{x_{i+1}} f'(x_i)(x - x_i)dx + \cdots$$

- Solving each integral separately results in the approximation

$$\int_{x_i}^{x_{i+1}} f(x)dx = hf(x_i) + \frac{h^2}{2} f'(x_i) + O(h^3)$$

which is just

$$\int_{x_i}^{x_{i+1}} f(x)dx = hf(x_i) + O\left(h^2\right).$$

- Since the $hf(x_i)$ term is our Riemann integral **approximation** for a **single subinterval**, the Riemann integral approximation over a single interval is $O(h^2)$.

- If we **sum** the $O(h^2)$ **error** over the **entire Riemann sum**, we get $nO(h^2)$.
- The **relationship** between n and h is

$$h = \frac{b-a}{n}$$

and so, our total error becomes $\frac{b-a}{h}O\left(h^2\right) = O(h)$ over the whole interval. Thus, the **overall accuracy** is $O(h)$.

- The **Midpoint Rule** takes the **rectangle height** of the rectangle at each **subinterval** to be the function value at the **midpoint** between x_i and x_{i+1}, which for compactness we denote by $y_i = \frac{x_{i+1}+x_i}{2}$.
- The **Midpoint Rule** says

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

- Similarly to the **Riemann integral**, we take the **Taylor series** of $f(x)$ around y_i, which is

$$f(x) = f(y_i) + f'(y_i)(x - y_i) + \frac{f''(y_i)(x - y_i)^2}{2!} + \cdots$$

- Then the **integral** over a **subinterval** is

$$\int_{x_i}^{x_{i+1}} f(x)dx = \int_{x_i}^{x_{i+1}} \left(f(y_i) + f'(y_i)(x - y_i) + \frac{f''(y_i)(x - y_i)^2}{2!} + \cdots \right) dx$$

which **distributes** to

$$\int_{x_i}^{x_{i+1}} f(x)dx = \int_{x_i}^{x_{i+1}} f(y_i)dx + \int_{x_i}^{x_{i+1}} f'(y_i)(x - y_i)dx + \int_{x_i}^{x_{i+1}} \frac{f''(y_i)(x - y_i)^2}{2!} dx + \cdots$$

- Recognizing that since x_i and x_{i+1} are **symmetric** around y_i, then

$$\int_{x_i}^{x_{i+1}} f'(y_i)(x - y_i)dx = 0.$$

- This is **true** for the integral of $(x - y_i)^p$ for any **odd** p.

- For the integral of $(x - y_i)^p$ and with p **even**, it suffices to say that

  $\int_{x_i}^{x_{i+1}} (x - y_i)^p dx = \int_{-\frac{h}{2}}^{\frac{h}{2}} x^p dx$, which will result in some **multiple** of h^{p+1} with **no lower order powers** of h.

- Utilizing these facts reduces the expression for the integral of $f(x)$ to

$$\int_{x_i}^{x_{i+1}} f(x) dx = hf(y_i) + O(h^3).$$

- Since $hf(y_i)$ is the approximation of the integral over the subinterval, the **Midpoint Rule** is $O(h^3)$ for one subinterval, and using similar arguments as for the **Riemann Integral**, is $O(h^2)$ over the **whole interval**.

- Since the **Midpoint Rule** requires the same number of calculations as the Riemann Integral, we essentially get an **extra order** of **accuracy** for free!

- However, if $f(x)$ is given in the form of data points, then we will **not be able** to compute $f(y_i)$ for this integration scheme.

- **Example**: Use the **left Riemann** Integral, **right Riemann** Integral, and **Midpoint Rule** to approximate $\int_0^\pi \sin(x)\,dx$ with 11 evenly spaced grid points over the whole interval. Compare this value to the exact value of 2.

```
import numpy as np

a = 0
b = np.pi
n = 11
h = (b - a) / (n - 1)
x = np.linspace(a, b, n)
f = np.sin(x)

I_riemannL = h * sum(f[:n-1])
err_riemannL = 2 - I_riemannL

I_riemannR = h * sum(f[1:])
err_riemannR = 2 - I_riemannR

I_mid = h * sum(np.sin((x[:n-1] + x[1:])/2))
err_mid = 2 - I_mid
```

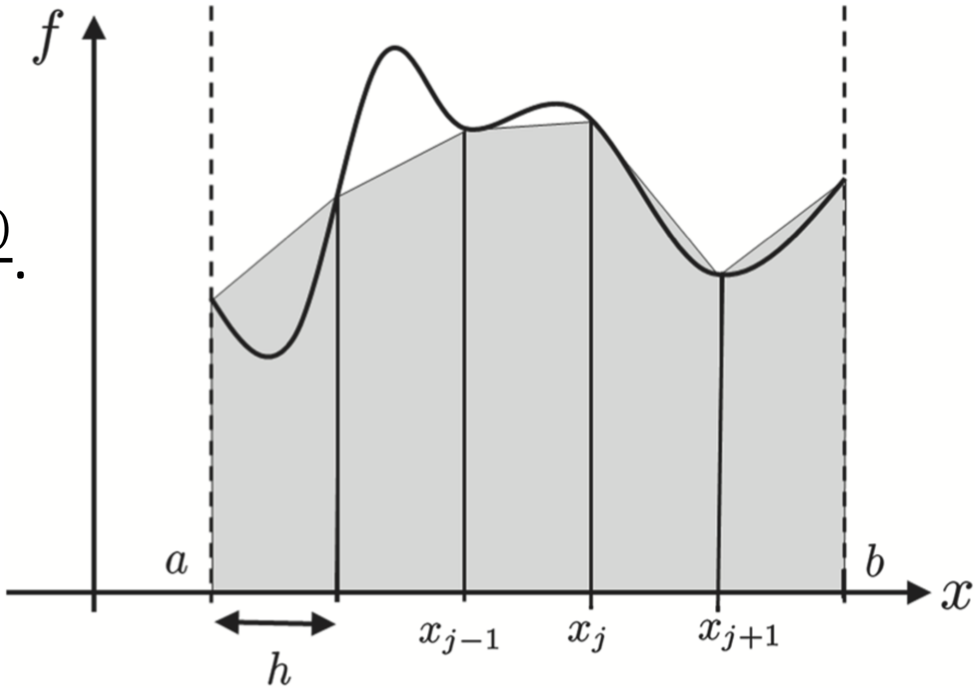
```
print(I_riemannL)
print(err_riemannL)

print(I_riemannR)
print(err_riemannR)

print(I_mid)
print(err_mid)
```

```
1.9835235375094546
0.01647646249054535
1.9835235375094546
0.01647646249054535
2.0082484079079745
-0.008248407907974542
```

Trapezoid Rule

- The **Trapezoid Rule** fits a **trapezoid** into each **subinterval** and **sums** the **areas** of the **trapezoid** to approximate the **total integral**.
- This approximation for the integral to an arbitrary function is shown in the following figure.
- For each subinterval, the Trapezoid Rule computes the area of a trapezoid with corners at $(x_i, 0)$, $(x_{i+1}, 0)$, $(x_i, f(x_i))$, and $(x_{i+1}, f(x_{i+1}))$, which is $h\frac{f(x_i)+f(x_{i+1})}{2}$.
- Thus, the **Trapezoid Rule** approximates **integrals** according to the expression

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



Trapezoid Rule

- **Example**: You may notice that the Trapezoid Rule "**double-counts**" most of the terms in the series. To illustrate this fact, consider the expansion of the Trapezoid Rule:

$$\sum_{i=0}^{n-1} h\frac{f(x_i)+f(x_{i+1})}{2}$$
$$=\frac{h}{2}[(f(x_0)+f(x_1))+(f(x_1)+f(x_2))+\cdots+(f(x_{n-1})+f(x_n))]$$

- Computationally, this is many **extra additions** and **calls** to $f(x)$ than is really necessary.
- We can be more computationally **efficient** using the following **expression**.

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

- To determine the **accuracy** of the Trapezoid Rule approximation, we first take **Taylor series expansion** of $f(x)$ around $y_i = \frac{x_{i+1}+x_i}{2}$, which is the **midpoint** between x_i and x_{i+1}.
- This **Taylor series expansion** is

$$f(x) = f(y_i) + f'(y_i)(x - y_i) + \frac{f''(y_i)\,(x - y_i)^2}{2!} + \cdots$$

- Computing the **Taylor series** at x_i and x_{i+1} and noting that $x_i - y_i = -\frac{h}{2}$ and $x_{i+1} - y_i = \frac{h}{2}$, results in the following expressions:

$$f(x_i) = f(y_i) - \frac{hf'(y_i)}{2} + \frac{h^2 f''(y_i)}{8} - \cdots$$

and

$$f(x_{i+1}) = f(y_i) + \frac{hf'(y_i)}{2} + \frac{h^2 f''(y_i)}{8} + \cdots$$

- Taking the **average** of these two expressions results in the new expression,

$$\frac{f(x_{i+1}) + f(x_i)}{2} = f(y_i) + O(h^2)$$

- Solving this expression for $f(y_i)$ yields

$$f(y_i) = \frac{f(x_{i+1}) + f(x_i)}{2} + O(h^2)$$

- Now returning to the **Taylor expansion** for $f(x)$, the integral of $f(x)$ over a subinterval is

$$\int_{x_i}^{x_{i+1}} f(x)dx = \int_{x_i}^{x_{i+1}} \left(f(y_i) + f'(y_i)(x - y_i) + \frac{f''(y_i)(x - y_i)^2}{2!} + \cdots \right) dx$$

- **Distributing** the integral results in the expression

$$\int_{x_i}^{x_{i+1}} f(x)dx = \int_{x_i}^{x_{i+1}} f(y_i)dx + \int_{x_i}^{x_{i+1}} f'(y_i)(x - y_i)dx + \int_{x_i}^{x_{i+1}} \frac{f''(y_i)(x - y_i)^2}{2!} dx + \cdots$$

- Now since x_i and x_{i+1} are **symmetric** around y_i, the integrals of the **odd powers** of $(x - y_i)^p$ **disappear** and the **even powers** resolve to a **multiple** h^{p+1}.

$$\int_{x_i}^{x_{i+1}} f(x)dx = hf(y_i) + O(h^3)$$

- Now if we **substitute** $f(y_i)$ with the expression derived explicitly in terms of $f(x_i)$ and $f(x_{i+1})$, we get

$$\int_{x_i}^{x_{i+1}} f(x)dx = h\left(\frac{f(x_{i+1}) + f(x_i)}{2} + O(h^2)\right) + O(h^3)$$

which is equivalent to

$$h\left(\frac{f(x_{i+1}) + f(x_i)}{2}\right) + hO(h^2) + O(h^3)$$

and therefore,

$$\int_{x_i}^{x_{i+1}} f(x)dx = h\left(\frac{f(x_{i+1}) + f(x_i)}{2}\right) + O(h^3)$$

- Since $\frac{h}{2}\left(f(x_{i+1}) + f(x_i)\right)$ is the **Trapezoid Rule approximation** for the integral over the subinterval, it is $O(h^3)$ for a **single subinterval** and $O(h^2)$ over the **whole interval**.

Trapezoid Rule

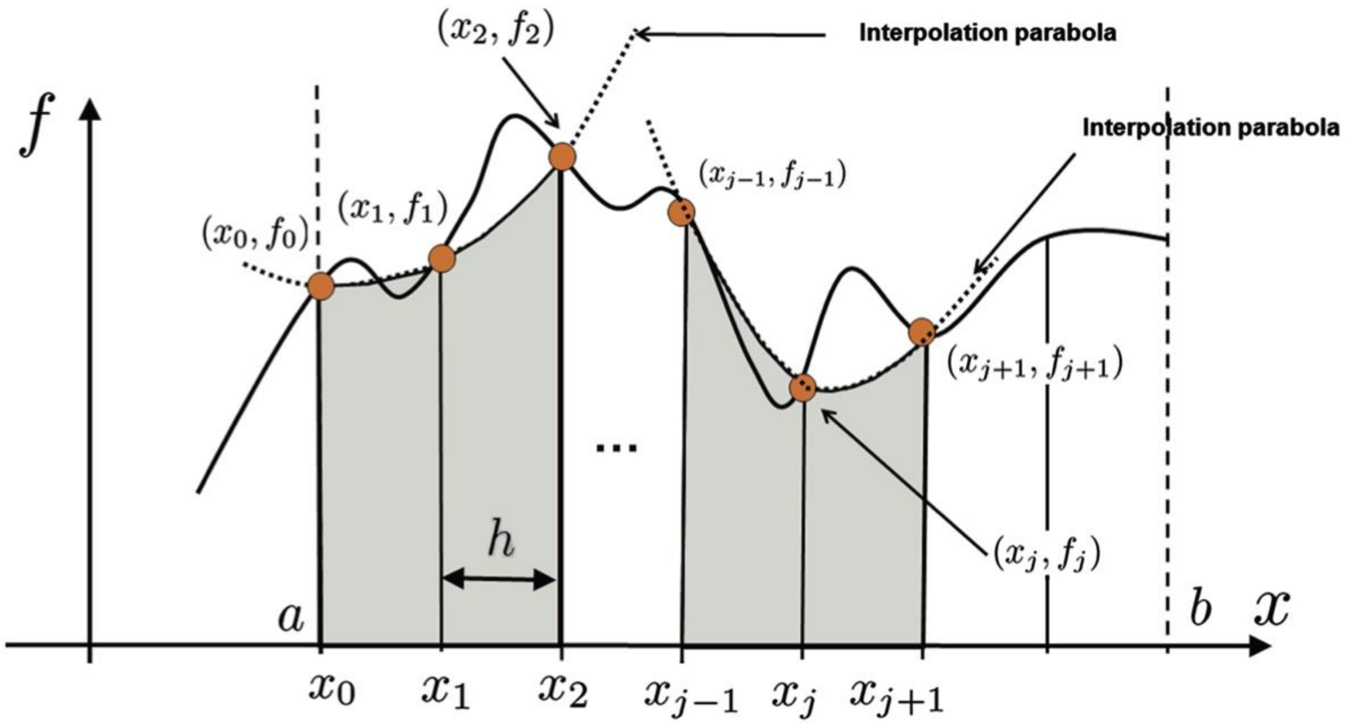
- **Example:** Use the Trapezoid Rule to approximate $\int_0^\pi \sin(x)\,dx$ with 11 evenly spaced grid points over the whole interval. Compare this value to the exact value of 2.

```
import numpy as np

a = 0
b = np.pi
n = 11
h = (b - a) / (n - 1)
x = np.linspace(a, b, n)
f = np.sin(x)

I_trap = (h/2)*(f[0] + 2 * sum(f[1:n-1]) + f[n-1])
err_trap = 2 - I_trap

print(I_trap)
print(err_trap)
```

```
1.9835235375094546
0.01647646249054535
```

Simpson's Rule

- Consider **two consecutive subintervals**, $[x_{i-1}, x_i]$ and $[x_i, x_{i+1}]$.
- **Simpson's Rule** approximates the **area** under $f(x)$ over these two subintervals by fitting a **quadratic polynomial** through the points $\left(x_{i-1}, f(x_{i-1})\right)$, $\left(x_i, f(x_i)\right)$, and $\left(x_{i+1}, f(x_{i+1})\right)$, which is a unique polynomial, and then **integrating** the quadratic exactly.
- The following shows this integral approximation for an arbitrary function.



- First, we construct the **quadratic polynomial** approximation of the function over the two subintervals.
- The **easiest** way to do this is to use **Lagrange polynomials**, which was discussed in the Week 7 Lesson.
- By applying the formula for constructing **Lagrange polynomials** we get the polynomial

$$P_i(x) = f(x_{i-1})\frac{(x-x_i)(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})} + f(x_i)\frac{(x-x_{i-1})(x-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})} + f(x_{i+1})\frac{(x-x_{i-1})(x-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)}$$

and with **substitutions** for h results in

$$P_i(x) = \frac{f(x_{i-1})}{2h^2}(x-x_i)(x-x_{i+1}) - \frac{f(x_i)}{h^2}(x-x_{i-1})(x-x_{i+1}) + \frac{f(x_{i+1})}{2h^2}(x-x_{i-1})(x-x_i)$$

Simpson's Rule

- You can confirm that the polynomial **intersects** the **desired points**.
- With some algebra and manipulation, the **integral** of $P_i(x)$ over the **two subintervals** is

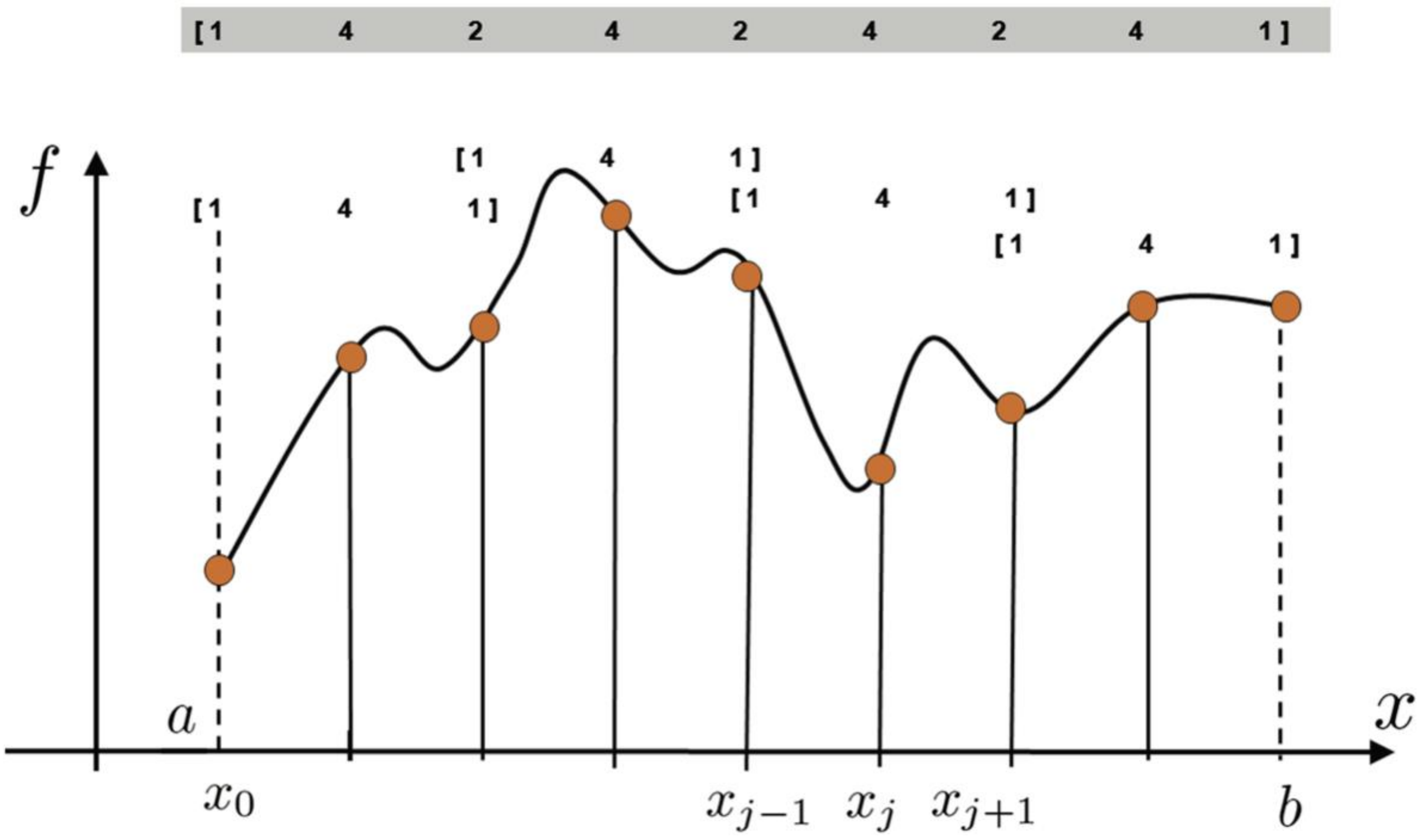
$$\int_{x_{i-1}}^{x_{i+1}} P_i(x)dx = \frac{h}{3}(f(x_{i-1}) + 4f(x_i) + f(x_{i+1}))$$

- To approximate the integral over (a, b), we must **sum** the integrals of $P_i(x)$ over every **two subintervals** since $P_i(x)$ spans.
- **Substituting** $\frac{h}{3}(f(x_{i-1}) + 4f(x_i) + f(x_{i+1}))$ for two subintervals the integral of $P_i(x)$ and regrouping the terms for **efficiency** leads to the **formula**

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

Simpson's Rule

- This regrouping is illustrated in the figure below:



- **WARNING**! Note that to use **Simpson's Rule**, you must have an **even number** of **intervals** and, therefore, an **odd number** of **grid points**.

Simpson's Rule

- To compute the **accuracy** of the Simpson's Rule, we take the **Taylor series** approximation of $f(x)$ around x_i, which is

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{f''(x_i)(x - x_i)^2}{2!} + \frac{f'''(x_i)(x - x_i)^3}{3!} + \cdots$$

- Computing the **Taylor series** at x_{i-1} and x_{i+1} and **substituting** for h where appropriate gives the expressions

$$f(x_{i-1}) = f(x_i) - hf'(x_i) + \frac{h^2 f''(x_i)}{2!} - \frac{h^3 f'''(x_i)}{3!} + \frac{h^4 f''''(x_i)}{4!} - \cdots$$

and

$$f(x_{i+1}) = f(x_i) + hf'(x_i) + \frac{h^2 f''(x_i)}{2!} + \frac{h^3 f'''(x_i)}{3!} + \frac{h^4 f''''(x_i)}{4!} + \cdots$$

- Now consider the expression $\frac{f(x_{i-1})+4f(x_i)+f(x_{i+1})}{6}$.
- Substituting the **Taylor series** for the respective numerator values produces the equation

$$\frac{f(x_{i-1})+4f(x_i)+f(x_{i+1})}{6}=f(x_i)+\frac{h^2}{6}f''(x_i)+\frac{h^4}{72}f''''(x_i)+\cdots$$

- Note that the **odd** terms **cancel out**. This implies

$$f(x_i)=\frac{f(x_{i-1})+4f(x_i)+f(x_{i+1})}{6}-\frac{h^2}{6}f''(x_i)+O(h^4)$$

- By substitution of the **Taylor series** for $f(x)$, the **integral** of $f(x)$ over two subintervals is then

$$\int_{x_{i-1}}^{x_{i+1}}f(x)dx$$

$$=\int_{x_{i-1}}^{x_{i+1}}\left(f(x_i)+f'(x_i)(x-x_i)+\frac{f''(x_i)(x-x_i)^2}{2!}+\frac{f'''(x_i)(x-x_i)^3}{3!}+\cdots\right)dx$$

Simpson's Rule

- Again, we **distribute** the integral and without showing it, we **drop** the integrals of terms with **odd powers** because they are **zero**.

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx$$
$$= \int_{x_{i-1}}^{x_{i+1}} f(x_i)dx + \int_{x_{i-1}}^{x_{i+1}} \frac{f''(x_i)(x-x_i)^2}{2!}dx + \int_{x_{i-1}}^{x_{i+1}} \frac{f''''(x_i)(x-x_i)^4}{4!}dx + \cdots$$

- We then carry out the **integrations**. As will soon be clear, it benefits us to compute the integral of the **second term** exactly.

- The **resulting** equation is

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx = 2hf(x_i) + \frac{h^3}{3}f''(x_i) + O(h^5).$$

- **Substituting** the expression for $f(x_i)$ derived earlier, the **right-hand** side becomes

$$2h\left(\frac{f(x_{i-1})+4f(x_i)+f(x_{i+1})}{6}-\frac{h^2}{6}f''(x_i)+O(h^4)\right)+\frac{h^3}{3}f''(x_i)+O(h^5)$$

which can be **rearranged** to

$$\left[\frac{h}{3}(f(x_{i-1})+4f(x_i)+f(x_{i+1}))-\frac{h^3}{3}f''(x_i)+O(h^5)\right]+\frac{h^3}{3}f''(x_i)+O(h^5)$$

- **Canceling** and **combining** the appropriate **terms** results in the integral expression

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx=\frac{h}{3}(f(x_{i-1})+4f(x_i)+f(x_{i+1}))+O(h^5).$$

- Recognizing that $\frac{h}{3}(f(x_{i-1})+4f(x_i)+f(x_{i+1}))$ is exactly the **Simpson's Rule** approximation for the integral over this subinterval, this equation implies that Simpson's Rule is $O(h^5)$ over a subinterval and $O(h^4)$ over the whole interval.
- Because the h^3 terms **cancel out** exactly, Simpson's Rule **gains** another **two orders** of **accuracy**!

Simpson’s Rule

- **Example**: Use Simpson’s Rule to approximate $\int_0^{\pi} \sin(x)\, dx$ with 11 evenly spaced grid points over the whole interval. Compare this value to the exact value of 2.

```python
import numpy as np

a = 0
b = np.pi
n = 11
h = (b - a) / (n - 1)
x = np.linspace(a, b, n)
f = np.sin(x)

I_simp = (h/3) * (f[0] + 2*sum(f[2:n-2:2]) \
            + 4*sum(f[1:n-1:2]) + f[n-1])
err_simp = 2 - I_simp

print(I_simp)
print(err_simp)
```

```
2.0001095173150043
-0.00010951731500430384
```

Computing Integrals in Python

- The **scipy.integrate** sub-package has several functions for computing **integrals**.
- The **trapz** takes as **input** arguments an **array** of function values f computed on a numerical grid x.

- **Example**: Use the **trapz** function to approximate $\int_0^{\pi} \sin(x)\, dx$ for 11 equally spaced points over the whole interval. Compare this value to the one computed in the early example using the **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)
```

```
1.9835235375094544
1.9835235375094546
```

Computing Integrals in Python

- The **quad(f,a,b)** function uses a different **numerical differentiation scheme** to approximate integrals.
- **quad integrates** the function defined by the function object, f, from a to b.
- **Example**: Use the **integrate.quad** function to compute $\int_0^{\pi} \sin(x)\,dx$. Compare your answer with the correct answer of 2.

```
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)
```

```
2.0
2.220446049250313e-14 0.0
```

Practice

1. Write a function **my_num_int(f,a,b,n,option)**, where the output I is the numerical integral of f, a function object, computed on a grid of n evenly spaced points starting at a and ending at b. The integration method used should be one of the following strings defined by option: '**rect**', '**trap**', '**simp**'. For the rectangle method, the function value should be taken from the **right endpoint** of the interval. You may assume that n is odd.

```
# Test case
f = lambda x: x**2
my_num_int(f, 0, 1, 3, 'rect')
```

```
]: 0.625
```

```
my_num_int(f, 0, 1, 3, 'trap')
```

```
]: 0.375
```

```
my_num_int(f, 0, 1, 3, 'simp')
```

```
]: 0.3333333333333333
```

Next Week's Outline

- ODE Initial Value Problem Statement
- Reduction of Order
- The Euler Method
- Numerical Error and Instability
- Predictor-Corrector Methods
- Python ODE Solvers
- Advanced Topics

References

- Kong, Qingkai; Siauw, Timmy, and Bayen, Alexandre. 2020. Python Programming and Numerical Methods: A Guide for Engineers and Scientists. Academic Press. https://www.elsevier.com/books/python-programming-and-numerical-methods/kong/978-0-12-819549-9
- Other online and offline references

Visi

Menjadi Program Studi Strata Satu Informatika **unggulan** yang menghasilkan lulusan **berwawasan internasional** yang **kompeten** di bidang Ilmu Komputer (*Computer Science*), **berjiwa wirausaha** dan **berbudi pekerti luhur**.

Misi

1. Menyelenggarakan pembelajaran dengan teknologi dan kurikulum terbaik serta didukung tenaga pengajar profesional.
2. Melaksanakan kegiatan penelitian di bidang Informatika untuk memajukan ilmu dan teknologi Informatika.
3. Melaksanakan kegiatan pengabdian kepada masyarakat berbasis ilmu dan teknologi Informatika dalam rangka mengamalkan ilmu dan teknologi Informatika.