Motion raw sensor data visualization

Suppose having the integral

f(x) = Integral 0 1 e^-x^2

imagining the f(x) represnset the motion sensor output data based in time:

Min	Sensor
0	2
1	8
2	22
3	12

Here is example raw data we have from the sensor, but here we need to solve the integral first and we do not have any data avaliable. I will use trapezoidal and simsons method to solve the integral.

given n = 4 as the intervall increase the more accurate function we recieve.

h = b - a / h => 1 - 0 / 4 = 0.25

then we have the following steps :

Here we define a function return the function formula.

In [74]:
```
import numpy as np
import matplotlib.pyplot as plt

#Integral from-to
```

```python
A , B = 0 , 1
#Function return our function example which the exponent represent the sensor data with time
def f(x):
    return np.exp(-x**2)
```

Trapezoidal Function

In [73]:
```python
#Calculating the trapezoidal method. The function expected a h value and list of noes representing the inte
#loop range start from the second element of array and end with left last element. Therfoe having {(arr_noc
#intern_trapezoidal the value we add together inside the trapezoidal step 2 * f(xn) where a < n < b
#cal_trap_step multiplty the f(xn) where a < n < b with (2) following trapezoidal formula
#Finally adding all together with res_trap [0] return first element , [-1] return last element of array.
#y_values_y holding the trapezoidal step result for plotting, defin it as golbal to be modify in function
y_values_trapezoidal = []
def trapezoidal(h, arr_nodes):
    intern_trapezoidal = 0
    list_intern = []
    for nodes in range(1, len(arr_nodes)-1):
        cal_trap_step = (2*arr_nodes[nodes])
        list_intern.append(float(cal_trap_step))
        intern_trapezoidal += cal_trap_step

    res_trap = h/2 * (arr_nodes[0] + intern_trapezoidal + arr_nodes[-1])

    global y_values_trapezoidal
    y_values_trapezoidal = [float(arr_nodes[0])] + list_intern + [float(arr_nodes[-1])]

    return float(res_trap)
```

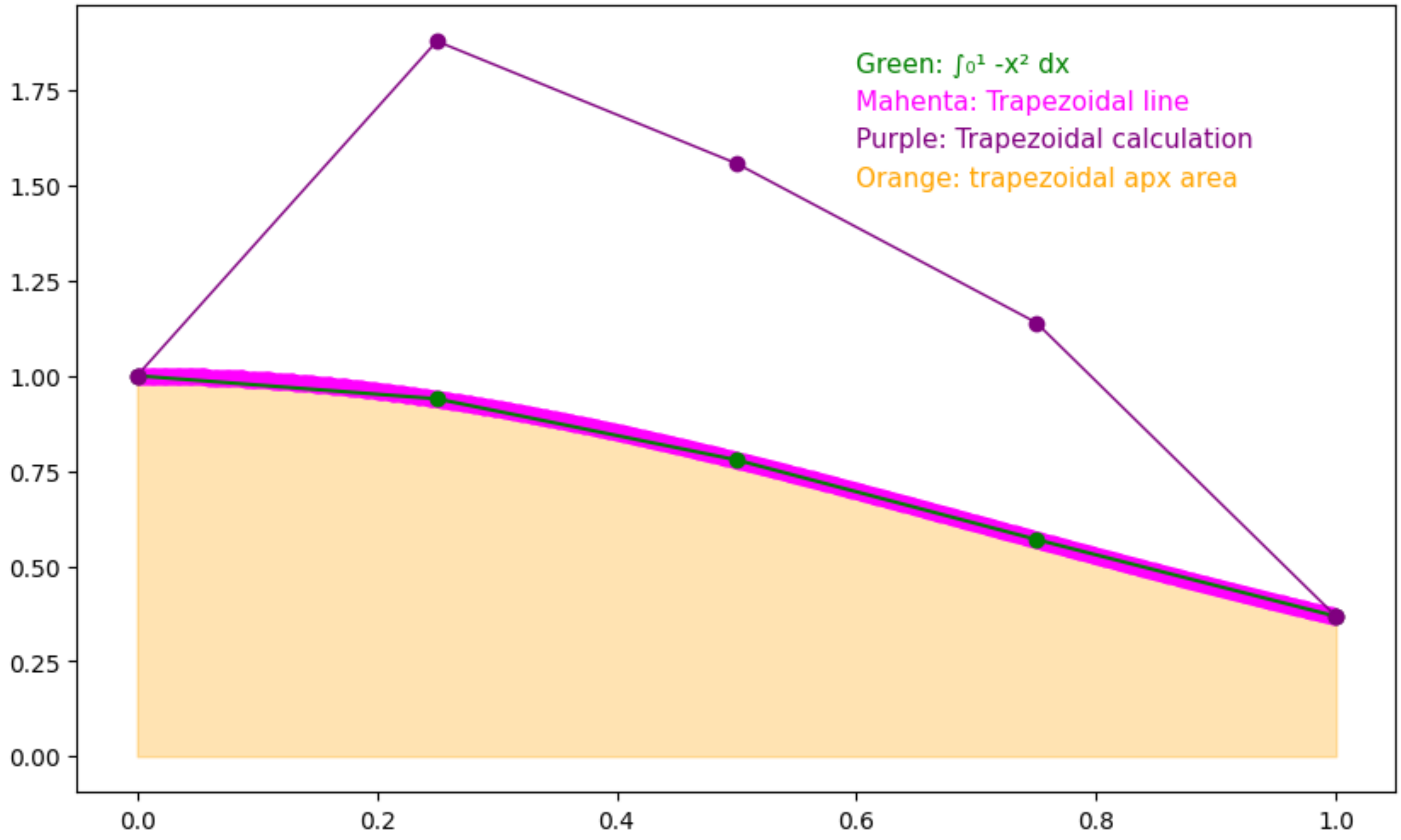
Calling Trapezoidal Method and plotting the function

In [75]:
```python
from colorama import Fore, Style, init
init(autoreset=True)
#subintervals (must be even for simsons method)
N = 4
H = (B - A) / N
#the numpy.linspace will return value between a and b with interval of (n)
```

```
x_nodes = np.linspace(A, B, N + 1)
#setting the step value into the function
y_nodes = f(x_nodes)

#calulcating with trapezoidal method
res = trapezoidal(h, y_nodes)
print(Fore.CYAN + "╔══════════════════════════════════════════════════════════════╗")
print(Fore.CYAN + "║             Trapezoidal Rule Integration Results             ║")
print(Fore.CYAN + "╚══════════════════════════════════════════════════════════════╝\n")

print(Fore.YELLOW + f"➤ Trapezoidal estimate value is: {res:.3f}\n")

x_values_trapezoidal = x_nodes
print(Fore.GREEN + "➤ The integral steps (x values):")
print("  " + ", ".join([f"{x:.3f}" for x in x_values_trapezoidal]) + "\n")

print(Fore.MAGENTA + "➤ Function evaluations at each step:")
print(Fore.MAGENTA + "  " + "-" * 50)

for i, x in enumerate(x_values_trapezoidal):
    print(f"  f(x{i}) = {f(x):.3f} at x = {x:.3f}")

print(Fore.CYAN + "\n" + "*" * 65)
# Plot function
x_dense = np.linspace(A, B, 800)
y_dense = f(x_dense)

plt.figure(figsize=(10, 6))
plt.plot(x_dense, y_dense, 'o-', label='∫ e^(-x^2) dx', linewidth=4, color='magenta')
plt.plot(x_nodes, y_nodes, 'o-', label='Sample points', color='green')
plt.plot(x_values_trapezoidal, y_values_trapezoidal, 'o-', label='f(x) values', linewidth=1, color='purple'
plt.fill_between(x_nodes, y_nodes, alpha=0.3, color='orange', label=f'Trapezoidal area ≈ {res}')

plt.text(x=0.6, y=1.8, s='Green: ∫₀¹ -x² dx', color='green', fontsize=11)
plt.text(x=0.6, y=1.7, s='Mahenta: Trapezoidal line', fontsize=11 , color='magenta')
plt.text(x=0.6, y=1.6, s='Purple: Trapezoidal calculation', fontsize=11, color='purple')
plt.text(x=0.6, y=1.5, s='Orange: trapezoidal apx area', fontsize=11, color='orange')
```

```python
plt.show()
```

```
╔══════════════════════════════════════════════════════════════╗
║              Trapezoidal Rule Integration Results            ║
╚══════════════════════════════════════════════════════════════╝

➤ Trapezoidal estimate value is: 0.743

➤ The integral steps (x values):
  0.000, 0.250, 0.500, 0.750, 1.000

➤ Function evaluations at each step:
  ──────────────────────────────────────────────────
  f(x0) = 1.000 at x = 0.000
  f(x1) = 0.939 at x = 0.250
  f(x2) = 0.779 at x = 0.500
  f(x3) = 0.570 at x = 0.750
  f(x4) = 0.368 at x = 1.000

*****************************************************************
```



Simsons Method

In [76]:
```
y_values_simsons = []
def simSons(h, arr_nodes):
    intern_simsons = 0
    list_intern = []
```

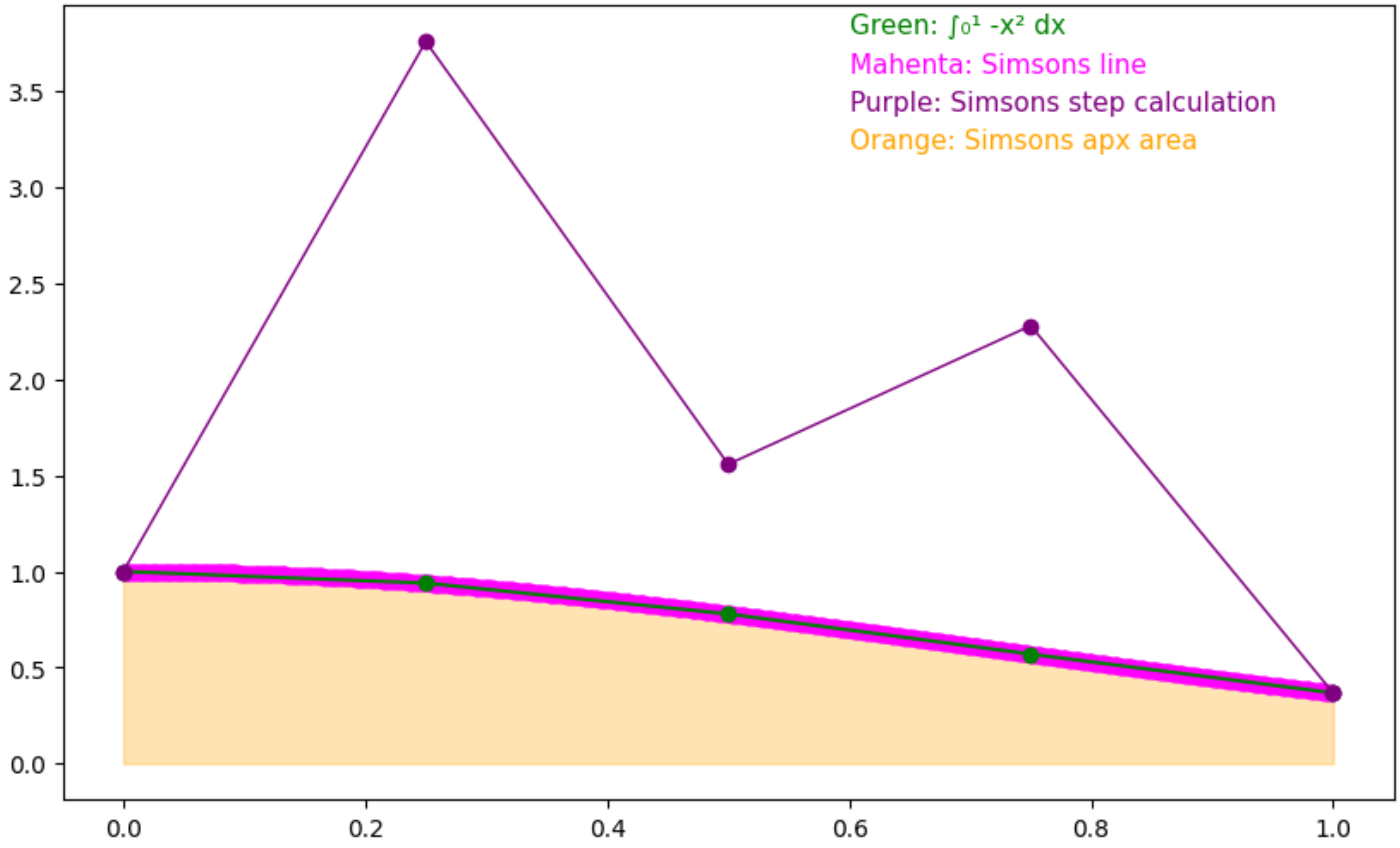
```
    for nodes in range(1, len(arr_nodes)-1):
        if(nodes % 2 != 0):
            cal_trap_step = (4*arr_nodes[nodes])
            list_intern.append(float(cal_trap_step))
            intern_simsons += cal_trap_step
        elif(nodes%2 ==0):
            cal_trap_step = (2*arr_nodes[nodes])
            list_intern.append(float(cal_trap_step))
            intern_simsons += cal_trap_step


    res_simsons = h/3 * (arr_nodes[0] + intern_simsons + arr_nodes[-1])

    global y_values_simsons
    y_values_simsons = [float(arr_nodes[0])] + list_intern + [float(arr_nodes[-1])]

    return float(res_simsons)
```

Calling Simsons Method and plotting the function

In [77]:
```
#calulcating with trapezoidal method
res = simSons(H, y_nodes)
print(Fore.CYAN + "╔════════════════════════════════════════════════════════════════╗")
print(Fore.CYAN + "║               Simpson's Rule Integration Results               ║")
print(Fore.CYAN + "╚════════════════════════════════════════════════════════════════╝\n")
x_values_simsons = x_nodes
print(Fore.YELLOW + f"➤ Simpson's estimate value is: {res:.3f}\n")

print(Fore.GREEN + "➤ The integral steps (x values):")
print("  " + ", ".join([f"{x:.3f}" for x in x_values_simsons]) + "\n")

print(Fore.MAGENTA + "➤ Function evaluations at each step:")
print(Fore.MAGENTA + "  " + "─" * 50)

for i, x in enumerate(x_values_simsons):
    print(f"  f(x{i}) = {f(x):.3f} at x = {x:.3f}")
```

```python
print(Fore.CYAN + "\n" + "*" * 65)
# Plot function
x_dense = np.linspace(A, B, 800)
y_dense = f(x_dense)

plt.figure(figsize=(10, 6))
plt.plot(x_dense, y_dense, 'o-', label='∫ e^(-x^2) dx', linewidth=4, color='magenta')
plt.plot(x_nodes, y_nodes, 'o-', label='Sample points', color='green')
plt.plot(x_values_simsons, y_values_simsons, 'o-', label='f(x) values', linewidth=1, color='purple')
plt.fill_between(x_nodes, y_nodes, alpha=0.3, color='orange', label=f'Trapezoidal area ≈ {res}')

plt.text(x=0.6, y=3.8, s='Green: ∫₀¹ -x² dx', color='green', fontsize=11)
plt.text(x=0.6, y=3.6, s='Mahenta: Simsons line', fontsize=11 , color='magenta')
plt.text(x=0.6, y=3.4, s='Purple: Simsons step calculation', fontsize=11, color='purple')
plt.text(x=0.6, y=3.2, s='Orange: Simsons apx area', fontsize=11, color='orange')


plt.show()
```

```
╔════════════════════════════════════════════════════════════════╗
║              Simpson's Rule Integration Results                ║
╚════════════════════════════════════════════════════════════════╝

➤ Simpson's estimate value is: 0.747

➤ The integral steps (x values):
  0.000, 0.250, 0.500, 0.750, 1.000

➤ Function evaluations at each step:
  ──────────────────────────────────────────────────
  f(x0) = 1.000 at x = 0.000
  f(x1) = 0.939 at x = 0.250
  f(x2) = 0.779 at x = 0.500
  f(x3) = 0.570 at x = 0.750
  f(x4) = 0.368 at x = 1.000

*****************************************************************
```



plott the error and try with different n values

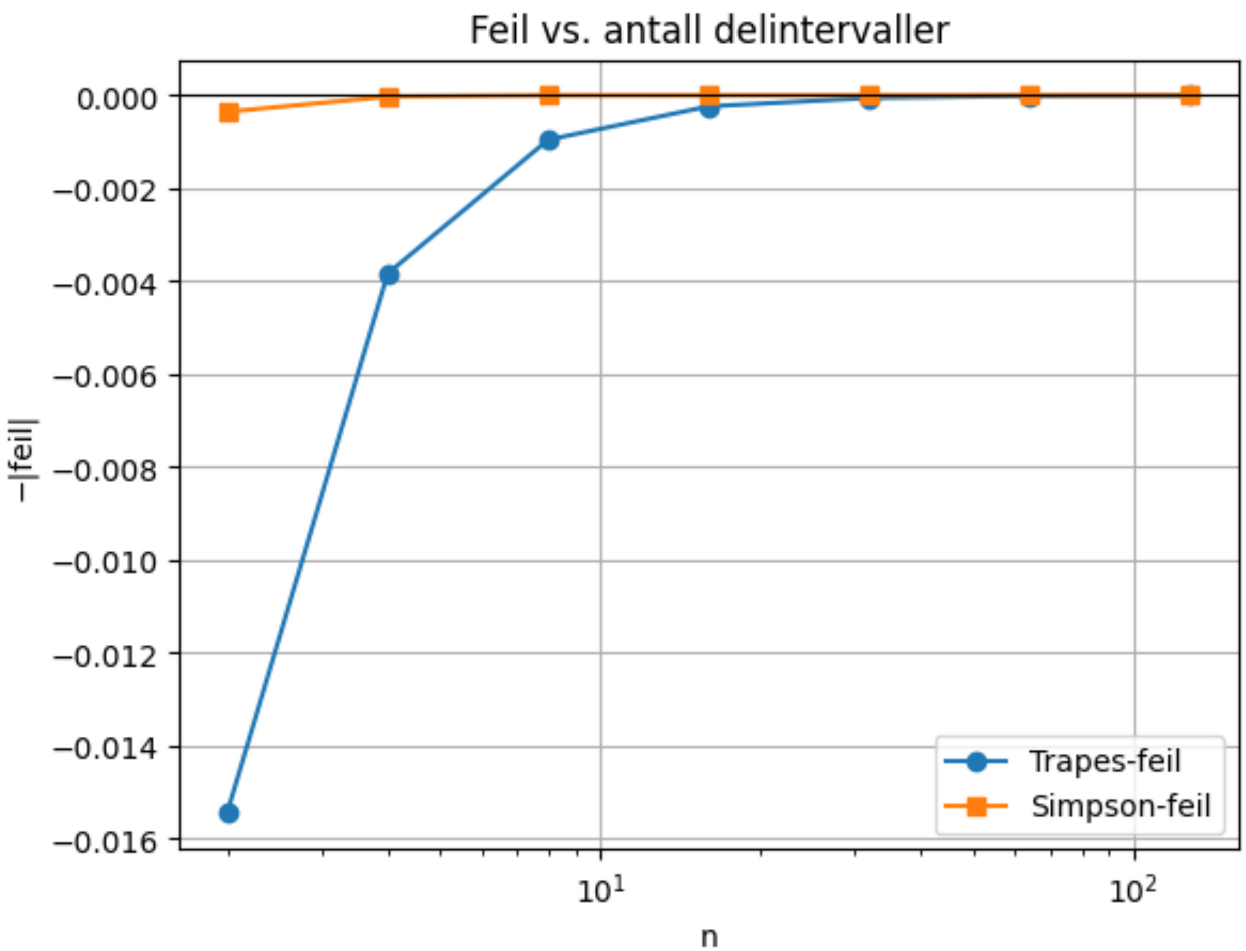
a visualization of the error in the trapezoidal and Simpson's method for the integral:

- x-axis: number of subintervals (n)

- y-axis: the error (difference from the true value)
- The plot should show how the error decreases as n increases.

In [78]:
```
n_values = [2, 4, 8, 16, 32, 64, 128]
TRUEVALUE = 0.746824
trapezoidal_errors, simpsons_errors = [], []

for k in n_values:
    x_nodes = np.linspace(A, B, k+1)
    y_nodes = f(x_nodes)
    h_test = (B - A) / k
    trap = trapezoidal(h_test, y_nodes)
    simp = simSons(h_test, y_nodes)
    trapezoidal_errors.append(trap - TRUEVALUE)
    simpsons_errors.append(simp - TRUEVALUE)

print("Trapezoidal Errors:")
for n, err in zip(n_values, trapezoidal_errors):
    print(f"  n = {n:>3} → Error = {err:+.8f}")

print("\nSimpson's Errors:")
for n, err in zip(n_values, simpsons_errors):
    print(f"  n = {n:>3} → Error = {err:+.8f}")

# Plotting
plt.figure()
plt.plot(n_values, -np.abs(trapezoidal_errors), 'o-', label='Trapes-feil')
plt.plot(n_values, -np.abs(simpsons_errors), 's-', label='Simpson-feil')
plt.axhline(0, color='black', lw=0.8)
plt.xscale('log'); plt.grid(True)
plt.xlabel('n'); plt.ylabel('-|feil|'); plt.legend()
plt.title('Feil vs. antall delintervaller')
plt.show()
```

```
Trapezoidal Errors:
  n =   2 → Error = -0.01545375
  n =   4 → Error = -0.00383990
  n =   8 → Error = -0.00095839
  n =  16 → Error = -0.00023940
  n =  32 → Error = -0.00005975
  n =  64 → Error = -0.00001484
  n = 128 → Error = -0.00000361

Simpson's Errors:
  n =   2 → Error = +0.00035643
  n =   4 → Error = +0.00003138
  n =   8 → Error = +0.00000212
  n =  16 → Error = +0.00000026
  n =  32 → Error = +0.00000014
  n =  64 → Error = +0.00000013
  n = 128 → Error = +0.00000013
```



Forklaring til feil

Feilen minker raskere for Simpson enn for trapes:

Trapesregelen har globalt feil $\mathcal{O}(h^2)$. Når vi dobler antall delintervaller n (halvering av h), reduseres feilen omtrent med en faktor 4. Fra $n = 2$ til $n = 4$ går feilen fra

$$-1.55 \times 10^{-2}$$

til

$$-3.84 \times 10^{-3}$$

(≈ ¼ av opprinnelig).

Simpsons metode er fjerde ordens, $\mathcal{O}(h^4)$. Ved dobling av n reduseres feilen med omtrent en faktor 16. Eksempelvis minsker feilen fra

$$+3.56 \times 10^{-4}$$

til

$$+3.14 \times 10^{-5}$$

(≈ 11-gangs reduksjon; ideelt 16, men avvik kommer av avrundings- og komponentfordeling).

Begge metodene fungerer bra og gir nesten riktig resultat for funksjonen.

suppose having afunction representing a big-sound sensor data for each second, we do not have araw data at fixed step :

let define the integral by calculating Trapezoidal and simson method - evaluating result accurace and erorr

$$\int_0^1 f(x) = e^{-x^2} dx$$

Define with four step $n = 4$

$$h = \frac{b-a}{4} = \frac{1-0}{4} = 0,25$$

given by formula :

$$x_i = a + ih = b$$

$x_1 = 0 + 1 \cdot 0{,}25 = 0{,}25$

$x_2 = 0 + 2 \cdot 0{,}25 = 0{,}5$

$x_3 = 0 + 3 \cdot 0{,}25 = 0{,}75$

$x_4 = 0 + 4 \; 0{,}25 = 1$

setting the step into function retrive the values:

$f(x_0) = e^{0^2} = 1$

$f(x_1) = e^{-0{,}25^2} = 0{,}9$

$f(x_2) = e^{-0{,}5^2} = 0{,}77$

$f(x_3 = e^{-0{,}75^2} = 0{,}56$

$f(x_4 = e^{-1^2} = e^{-1} = 0{,}36$

Using Trapezoidal method

$$T_n = \frac{h}{2}\{f(x_0) + 2f(x_n) + f(x_4)\}$$

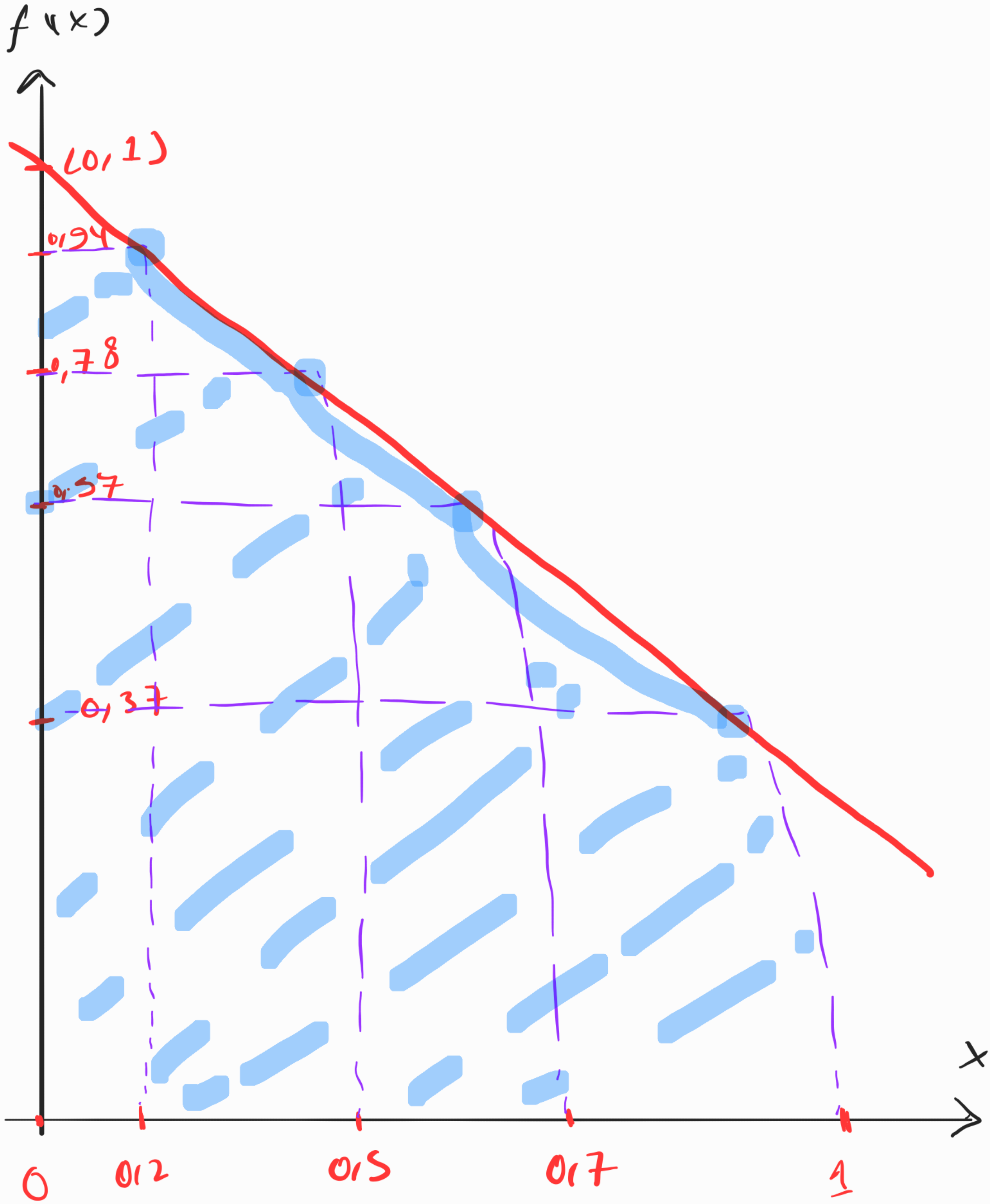
$$\Rightarrow T_n = h\left\{\frac{1}{2}f(x_0) + f(x_n) + \frac{1}{2}f(x_4)\right]$$

setting all together

$$0{,}25\left\{\frac{1}{2}(1) + 0{,}9 + 0{,}7 + 0{,}5 + \frac{1}{2}(0{,}36)\right\}$$

$$T_4 = 0{,}74$$

plotting :



Simson method

$$S_n = \frac{h}{3}\{f(x_0) + 4f(x_1) + 2f\,x_2 + 4f\,x_3 + f\,x_4\}$$

$$S_4 = \frac{0{,}25}{3}\{1 + 3{,}7 + 1{,}5 + 2{,}2 + 0{,}3\} =$$

$$\frac{0{,}25}{3} \cdot 8{,}9 = 0{,}08 \cdot 8{,}96$$

$$= 0{,}746$$

Plotting Simsons

for example if $n = 2$

then we need two subinterval

give by :

$[x_0, x_1]$ $[x_1, x_2]$

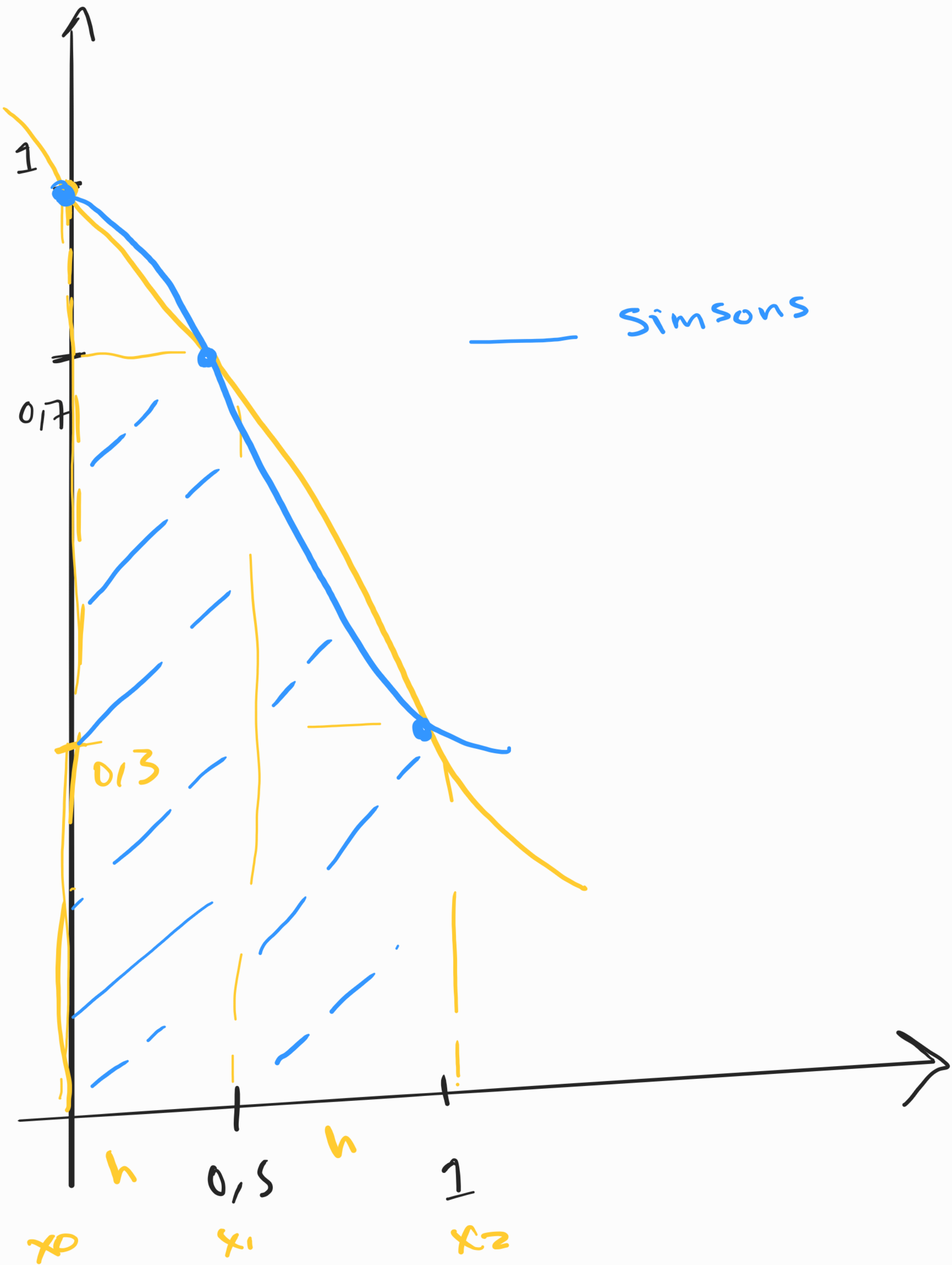
our Integral $[0, 1]$

$x_0 = 0$ - $x_1 = 0{,}5$, $x_2 = 1$

$x_0 = f(0) = e^{0} = 1$

$x_1 = f(0{,}5) = e^{-0{,}5^2} = 0{,}77$

$x_2 = f(1) = e^{-1^2} = e^{-1} = 0{,}36$



1
0,7
0,3
Simsons
h
h
0,5
1
x0
x1
x2