Numerical Integration & Differentiation
Lab 1
E/20/037

Numerical Quadrature

Q1) $e(h) \simeq ch^p \quad -(1)$

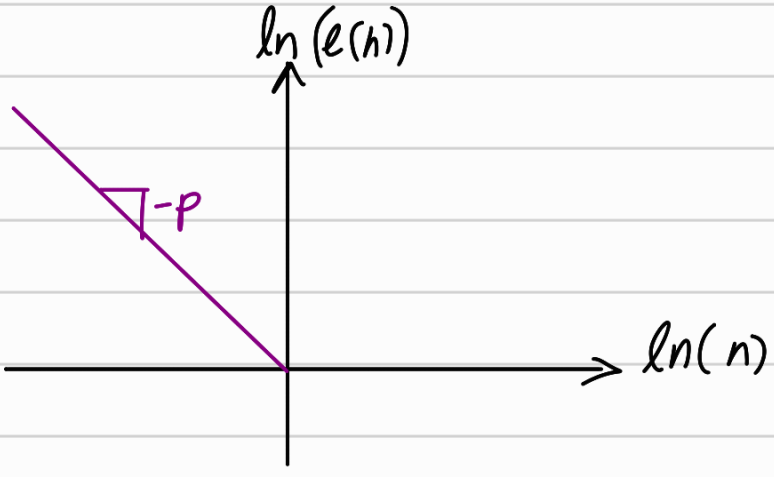
$\ln e(h) \simeq \ln c + p \ln h$

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

$\ln e(h) \simeq \ln c + p \ln\left(\frac{b-a}{n}\right)$

$$\boxed{\ln e(h) \simeq -p \ln(n) + \ln[c(b-a)^p]}$$

$y = m x + c$



$(1) \rightarrow \quad e(h) \simeq ch^p$

$\simeq c\left(\frac{b-a}{n}\right)^p$

$e(h) \propto \frac{1}{n^p}$

By studying the code, it can be seen that

Composite Trapezoidal Rule	Composite Simpson's Rule
$p = 2$	$p = 4$
$e(h) \propto \frac{1}{n^2}$	$e(h) \propto \frac{1}{n^4}$
$\therefore$ decays at the rate of $\frac{1}{n^3}$	decays at the rate of $\frac{1}{n^4}$

(Q2) Composite Simpson's Rule error

$$E(f) = -\frac{(b-a)}{180} h^4 f^4(\xi)$$

$$\boxed{E(f) = -\frac{(b-a)^5}{180 n^4} f^4(\xi)}$$

$$I = \int_1^2 x \log x \, dx$$

$f(x) = x \log x$
$f'(x) = \log x + 1$
$f^2(x) = x^{-1}$
$f^3(x) = -x^{-2}$
$f^4(x) = 2x^{-3}$ → when $\xi = 1$, $f^4(\xi) \to f^4(\xi)_{max}$

$a = 1, b = 2$

$\xi \in [1, 2]$

$$|E(f)| \leq \frac{(b-a)^5}{180\, n^4} \left| f^{(4)}(\xi) \right|_{\xi=1}$$

$$|E(f)| \leq \frac{(2-1)^5}{180\, n^4} f^{(4)}(1)$$

$$|E(f)| \leq \frac{1^5 \times 2}{180\, n^4} = \frac{1}{90\, n^4}$$

To ensure n with an absolute error of at most 10^{-6}

$$E(f)_{max} \leq 10^{-6}$$

$$\frac{1}{90\, n^4} \leq 10^{-6}$$

$$\frac{10^6}{90} \leq n^4$$

$$10.27 \leq n$$

Since n must be even,

$$\underline{\underline{n = 12}}$$

Numerical verification for the reasonability of n=12

```
# function definition

f = lambda x : x*log(x)
left = 1.0 ; right = 2.0
```

✓ 0.0s

```
# Exact integration with scipy.integrate.quad
exact, e = integrate.quad(f, left, right)
print(f"Exact integral: {exact}")
```

✓ 0.0s

```
Exact integral: 0.6362943611198906
```

```
# value calculated from Simpson's method, when n = 12

simp_approximation = simpson(f, 1, 2, 12)
print(f'simpson\'s approximation when n is 12 : {simp_approximation}')
```

✓ 0.0s

```
simpson's approximation when n is 12 : 0.6362945608313058
```

```
# absolute error

e = abs(exact - simp_approximation)
print(f'absolute error when n is 12 : {e}')
```

✓ 0.0s

```
absolute error when n is 12 : 1.9971141518304592e-07
```

↓

error is less than 10^{-6}

```python
# Simpson's rule for different number of quadrature points
N = linspace(2, 100, 50)  # Ensure N is even
res = array([simpson(f, left, right, int(n)) for n in N])
err = abs(res - exact)


for n, e in zip(N, err):
    print(f"n = {int(n)}    |    e < 10-6 : {e<1e-6}")
```

✓ 0.0s

```
n = 2    |    e < 10-6 : False
n = 4    |    e < 10-6 : False
n = 6    |    e < 10-6 : False
n = 8    |    e < 10-6 : False
n = 10    |    e < 10-6 : True
n = 12    |    e < 10-6 : True
n = 14    |    e < 10-6 : True
n = 16    |    e < 10-6 : True
n = 18    |    e < 10-6 : True
```

n=12 is reasonable

Q3)

```
from scipy import integrate
import numpy as np
from numpy.linalg import eigh
import matplotlib.pyplot as plt
import pandas as pd
```

✓ 2.5s

```
f = lambda x : (2/np.sqrt(np.pi))*np.e**(-x**2)
```

✓ 0.0s

a)

a) Find the value of the integral for a = 1.5 with scipy.integrate.quad and use this as the exact value of the integral.

```
exact, e = integrate.quad(f, 0, 1.5)
print(f"Exact integral: {exact}")
```

✓ 0.0s

```
Exact integral: 0.966105146475311
```

b)

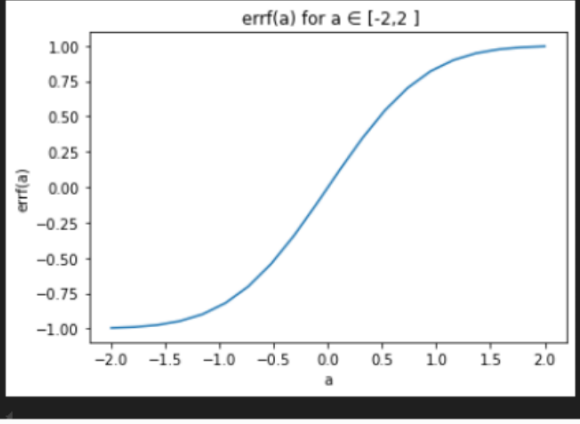
b) Plot $errf(a)$ for a ∈ [−2,2].

```
errf = lambda a: integrate.quad(f, 0, a)[0]

x = linspace(-2,2,20)
y = list(map(errf, x))

plt.plot(x, y)
plt.xlabel('a')
plt.ylabel('errf(a)')
plt.title(' errf(a) for a ∈ [-2,2 ]')
plt.show()
```

✓ 0.1s



c)

```
nodes = [2, 3, 4, 5]

results = [quad_gauss(f, 0, 1.5, n) for n in nodes]

relative_error = [abs(exact - result)/exact for result in results]

df = pd.DataFrame()
df['number of nodes'] = nodes
df['approximation'] = results
df['relative error'] = relative_error

df.style.hide(axis='index')
```

✓ 0.0s

number of nodes	approximation	relative error
2	0.974173	0.008351
3	0.965502	0.000624
4	0.966130	0.000025
5	0.966104	0.000001

d)

d) Calculate the number of trapeziums required by the trapezoidal rule to produce an error similar to that produced by the 3-point Gauss integration scheme, then evaluate the integral using the trapezoidal rule with the obtained number of trapeziums.

```
def trapezoidal(func, a, b, N):
    """
    Numerical quadrature based on the trapezoidal rule.

    Parameters:
    func : function
        The function to integrate, handle to y = f(x).
    a : float
        The lower bound of the integration interval.
    b : float
        The upper bound of the integration interval.
    N : int
        The number of subintervals (N+1 points).
    """
    from numpy import linspace, sum

    # Quadrature nodes
    x = np.linspace(a, b, N + 1)
    h = x[1] - x[0]

    # Quadrature weights: internal nodes: w = 1, boundary nodes: w = 0.5
    I = sum(func(x[1:-1])) + 0.5 * (func(x[0]) + func(x[-1]))

    return I * h
```

✓ 0.0s

```python
# 3-point Gauss integration error
max_error = relative_errors[1]

# Trapezoid rule for different number of quadrature points
n = 1
while True:
    trap_approximation = trapezoidal(f, 0, 1.5, n)
    trap_relative_error = abs(exact - trap_approximation)/exact
    if trap_relative_error <= max_error:
        print(f"Number of trapeziums required = {n}")
        print(f"Trapezoidal approximation for that number of trapeziums = {trap_approximation}")
        break
    n+=1
```

✓ 0.0s

```
Number of trapeziums required = 11
Trapezoidal approximation for that number of trapeziums = 0.9655527850725197
```

Numerical Differentiation

Q1)

Q1. Use the forward difference formula with h = 10 -8 to estimate f'(2).

```python
first_c(f,2,method='forward',h=1e-8)
```

✓ 0.0s

```
3.079641430758784
```

Q2)

Q2. Suppose You have obtained a signal f=cos(x).
But this signal is spoiled by some noise given by a small sine wave:
$f_{\epsilon,\omega}(x)=\cos(x)+\epsilon\sin(\omega x)$. where $0<\epsilon\ll 1$ is a very small number and ω is a large number

For $\epsilon=0.01$ and $\omega=100$, first plot in the same graph the function f=cos(x) and the function with noise $f_{\epsilon,\omega}(x)=\cos(x)+\epsilon\sin(\omega x)$.

```python
import matplotlib.pyplot as plt

epsilon = 0.01
omega = 100


x = np.linspace(0, 2*np.pi, 50)
f = lambda x : np.cos(x)
w = lambda x : np.cos(x) + epsilon * np.sin(omega * x)

plt.plot(x, f(x))
plt.plot(x, w(x))

plt.legend(['f(x)','w(x)'], loc='best')

plt.title('f(x) and w(x)',fontsize=20, pad = 30)
plt.xlabel('x',fontsize=15 )

plt.show()
```

✓ 0.1s

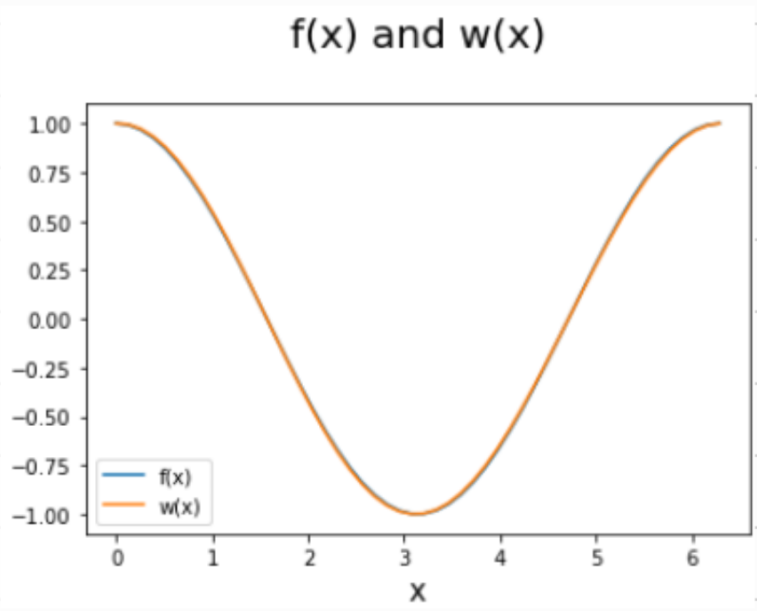


Figure Q2.a

Then in a different graph plot the derivatives f' and f'ϵ,ω and comment on your observations.

```
f_dash = first_c(f,x,method='forward',h=1e-8)
w_dash = first_c(w,x,method='forward',h=1e-8)

plt.plot(x, f_dash)
plt.plot(x, w_dash)

plt.legend(['f\'(x)','w\'(x)'], loc='best')

plt.title('f\'(x) and w\'(x)',fontsize=20, pad = 30)
plt.xlabel('x',fontsize=15 )

plt.show()
```

✓ 0.1s

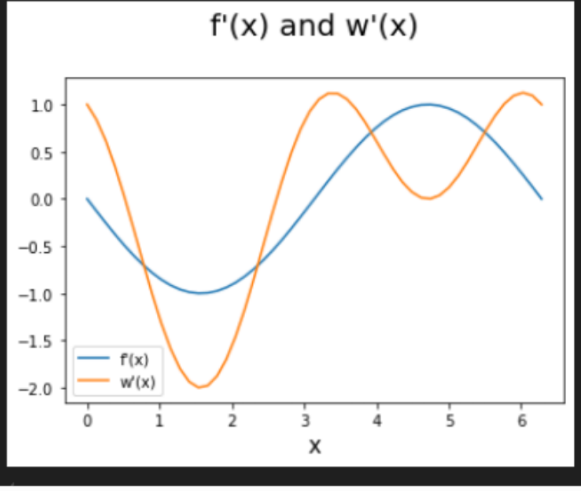


Figure Q2.b

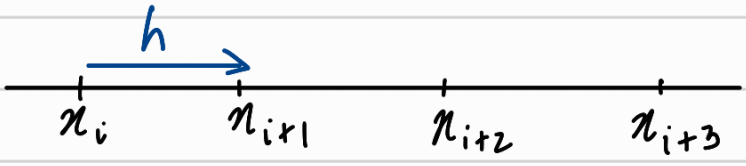
Even though $f(x)$ & $w(x)$ looks almost the same in figure Q2.a, there is a huge difference between $f'(x)$ & $w'(x)$ in figure Q2.b

$$f(x) = \cos x$$
$$f'(x) = -\sin x$$

$$w(x) = \cos x + q\sin wx$$
$$w'(x) = -\sin x + qw\cdot\cos wx$$

even though q is a small value, since w is a large value, qw makes a significant impact

Q3)



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

$$f(x_{i+3}) = f(x_i) + 3\frac{f'(x_i)}{1!}h + 9\frac{f''(x_i)}{2!}h^2 + 27\frac{f'''(x_i)}{3!}h^3 + \dots \quad -(2)$$

$9\times(1) - (2)$

$$9f(x_{i+1}) - f(x_{i+3}) = 8f(x_i) + 6f'(x_i)h - 18\frac{f'''(x_i)}{3!}h^3 + \dots$$

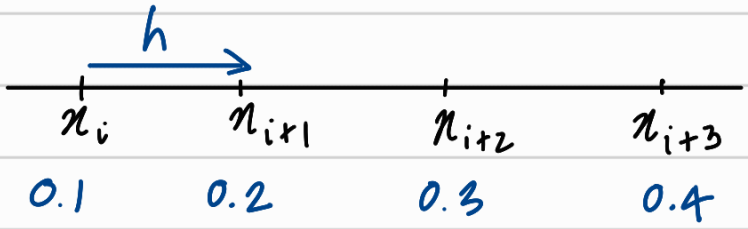
$$f'(x_i) = \frac{-f(x_{i+3}) + 9f(x_{i+1}) - 8f(x_i)}{6h} + \frac{3f^{(3)}(\xi)h^2}{3!}$$

$\xi \in [x_i, x_{i+3}]$

error $\propto h^2$

$\therefore$ order of approximation $= 2$

Q4)



Formula in Q3 can be used for $f'(0.1)$

$$f'(0.1) = \frac{-f(0.4) + 9f(0.2) - 8f(0.1)}{6 \times 0.1}$$

$$= \frac{-0.38942 + 9 \times 0.19861 - 8 \times 0.09983}{0.6}$$

$$\underline{\underline{f'(0.1) = 0.99995}}$$

for $f'(0.3)$

$$f'(x_{i+2}) = \frac{f(x_{i+3}) - f(x_{i+1})}{2h}$$ (centered difference formula)

$$f'(0.3) = \frac{f(0.4) - f(0.2)}{2 \times 0.1}$$

$$= \frac{0.38942 - 0.19867}{0.2}$$

$$f'(0.3) = 0.95375$$