**Завдання 1**  
  
import numpy as np

from scipy.misc import derivative

def f(x):

return 6 \* x\*\*4 + 4 \* x\*\*3 - x\*\*2 - x - 10

def newton(a, b, eps):

xi = (a + b) / 2

while True:

df = derivative(f, xi, dx=1e-6, n=1)

d2f = derivative(f, xi, dx=1e-6, n=2)

if d2f == 0:

print("Друга похідна дорівнює нулю. Метод Ньютона не працює.")

return

xi\_1 = xi - df / d2f

if abs(xi\_1 - xi) < eps:

break

xi = xi\_1

print("Розв'язок рівняння методом Ньютона x =", xi\_1)

def komb(a, b, eps):

fa = f(a)

fb = f(b)

if fa \* derivative(f, a, n=2) > 0:

a0, b0 = a, b

else:

a0, b0 = b, a

ai = a0

bi = b0

while abs(ai - bi) > eps:

ai = ai - f(ai) \* (bi - ai) / (f(bi) - f(ai))

bi = bi - f(bi) / derivative(f, bi, n=1)

x = (ai + bi) / 2

return print('Комбінований метод =', x)

a = -2.0

b = -0.5

eps = 0.0001

newton(a, b, eps)

komb(a, b, eps)  
  
  
**Завдання 2**

import numpy as np

import matplotlib.pyplot as plt

from scipy.interpolate import lagrange

def lagrange\_interpolation(x\_values, y\_values, x):

result = 0

for i in range(len(y\_values)):

term = y\_values[i]

for j in range(len(x\_values)):

if j != i:

term = term \* (x - x\_values[j]) / (x\_values[i] - x\_values[j])

result += term

return result

x\_values = np.array([-3, -2, 0, 2, -2.5, -1, 1.5, 3], dtype=float)

y\_values = np.array([-13, 3, 5, 7, -5, 3, 6, 9], dtype=float)

x\_test = 1.2

f\_interp = lagrange\_interpolation(x\_values, y\_values, x\_test)

print("Значення функції у точці x\_test =", f\_interp.round(4))

x\_new = np.linspace(np.min(x\_values), np.max(x\_values), 100)

y\_new = [lagrange\_interpolation(x\_values, y\_values, i) for i in x\_new]

plt.plot(x\_values, y\_values, 'o', label='Вихідні точки')

plt.plot(x\_new, y\_new, label='Інтерполяційний багаточлен Лагранжа')

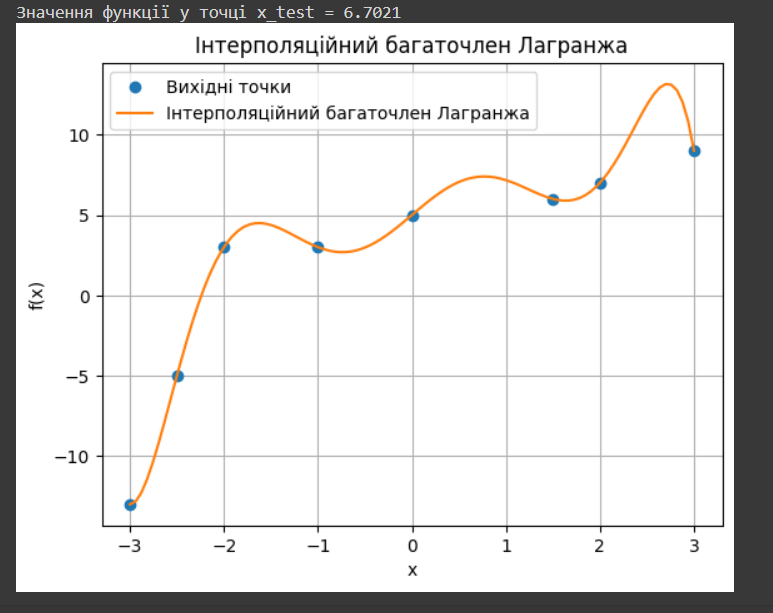
plt.title('Інтерполяційний багаточлен Лагранжа')

plt.xlabel('x')

plt.ylabel('f(x)')

plt.legend()

plt.grid(True)

plt.show()  
  
  
**Завдання 3**  
import numpy as np

from scipy.interpolate import CubicSpline

import matplotlib.pyplot as plt

x = np.array([0, 1, 2, 3, 4])

y = np.array([2.45, 1.63, 0.95, 0.73, 1.95])

cs = CubicSpline(x, y)

x\_new = np.linspace(np.min(x), np.max(x), 100)

y\_new = cs(x\_new)

for i in range(len(x) - 1):

print(f"Відрізок {i + 1}:")

print(f"a\_{i} = {cs.c[i, 3]:.4f}, b\_{i} = {cs.c[i, 2]:.4f}, c\_{i} = {cs.c[i, 1]:.4f}, d\_{i} = {cs.c[i, 0]:.4f}")

plt.figure(figsize=(8, 6))

plt.plot(x, y, 'o', label='Точки')

plt.plot(x\_new, y\_new, label='Кубічний сплайн')

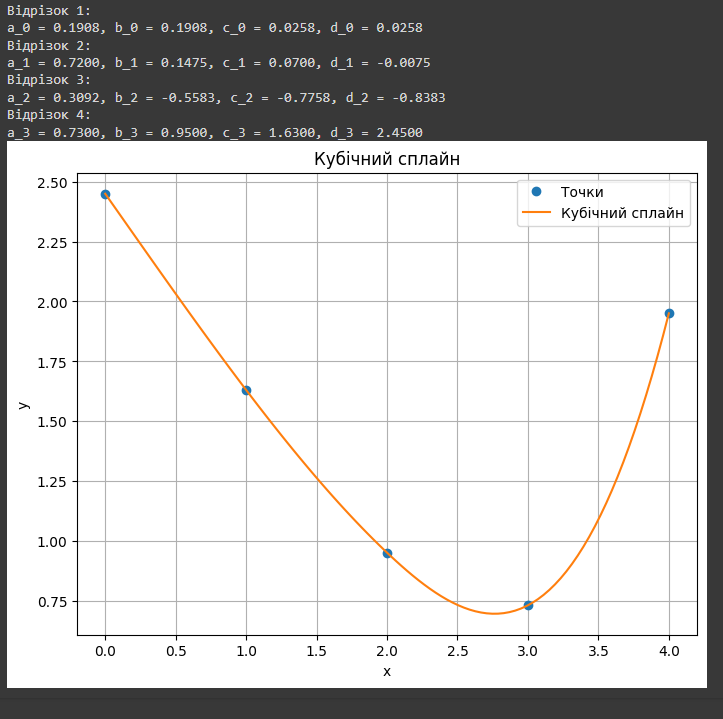
plt.xlabel('x')

plt.ylabel('y')

plt.title('Кубічний сплайн')

plt.legend()

plt.grid(True)

plt.show()  
  
  
**Завдання 4**  
  
import numpy as np

from scipy.optimize import least\_squares

import matplotlib.pyplot as plt

def f(x):

return x \* np.sin(x)

x\_values = np.arange(0.1, 1.1, 0.1)

y\_values = f(x\_values)

def linear\_fit(params, x, y):

a, b = params

return a \* x + b - y

def quadratic\_fit(params, x, y):

a, b, c = params

return a \* x\*\*2 + b \* x + c - y

initial\_params\_linear = [1, 1]

initial\_params\_quadratic = [1, 1, 1]

result\_linear = least\_squares(linear\_fit, initial\_params\_linear, args=(x\_values, y\_values))

params\_linear = result\_linear.x

y\_linear = linear\_fit(params\_linear, x\_values, y\_values) + y\_values

result\_quadratic = least\_squares(quadratic\_fit, initial\_params\_quadratic, args=(x\_values, y\_values))

params\_quadratic = result\_quadratic.x

y\_quadratic = quadratic\_fit(params\_quadratic, x\_values, y\_values) + y\_values

plt.figure(figsize=(8, 6))

plt.plot(x\_values, y\_values, 'o', label='Функція f(x)')

plt.plot(x\_values, y\_linear, label='Наближення прямою')

plt.plot(x\_values, y\_quadratic, label='Наближення параболою')

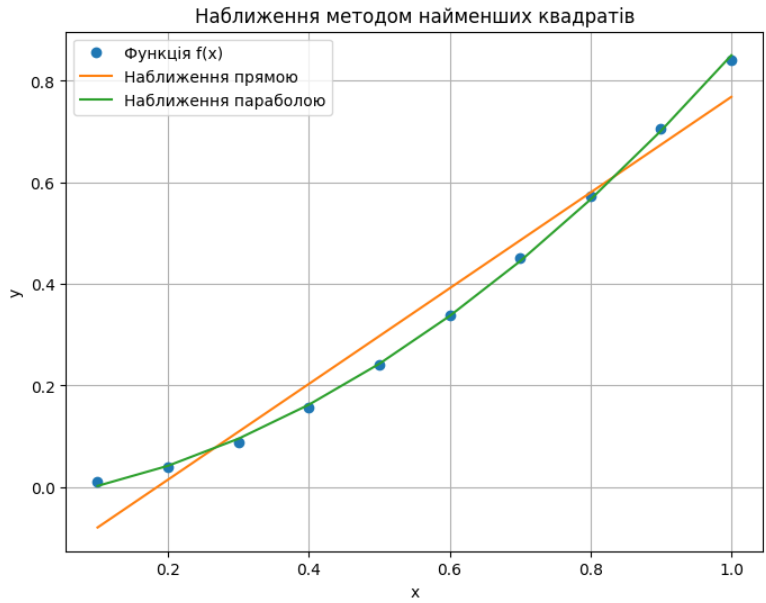
plt.xlabel('x')

plt.ylabel('y')

plt.title('Наближення методом найменших квадратів')

plt.legend()

plt.grid(True)

plt.show()  
  
  
**Завдання 5**  
  
**1) метод прямокутників**  
  
from scipy import integrate

import numpy as np

def f6(x):

return np.sqrt(0.5\*x+2)

a6 = 0.4

b6 = 1.2

integral\_rect = integrate.quad(f6, a6, b6, epsabs=1e-4, limit=10)[0]

print("Метод прямокутників:", round(integral\_rect, 5))  
  
**2) метод Сімпсона**from scipy import integrate

import numpy as np

def f(x):

return np.sin(2\*x) / x\*x

a = 0.8

b = 1.2

n = 8

integral = integrate.simps([f(xi) for xi in np.linspace(a, b, n)], dx=(b - a) / (n - 1))

print("Метод Сімпсона:", round(integral, 4))

v, err = integrate.quad(f, a, b)

print("Перевірка для методу Сімпсона:", round(v, 4))  
  
**3) метод трапеції**  
  
from scipy import integrate

import numpy as np

def f(x):

return np.sqrt(12\*x\*x+0.5)

a = 0.6

b = 1.4

n = 20

integral\_trapz = integrate.trapz([f(xi) for xi in np.linspace(a, b, n)], dx=(b - a) / (n - 1))

print("Метод трапецій:", round(integral\_trapz, 4))

v, err = integrate.quad(f, a, b)

print("Перевірка для методу трапецій:", round(v, 4))