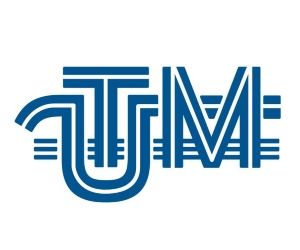
Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Ingineria Software și Automatică

**Raport**

la disciplina **”Programarea Declarativa”**

Tema: **Manipularea datelor folosind limbajul Python**

Efectuat de: studentul/studenta gr.**TI-216 Vlasitchi Stefan**

Verificat de: asis.univ **Viorel Rusu**

Chișinău-2023

**Exercitiul 1.**

Calculați √2 cu 100 de zecimale

import mpmath  
  
# Setăm precizia la 100 de zecimale  
mpmath.mp.dps = 100  
  
# Calculăm rădăcina pătrată a lui 2  
rezultat = mpmath.sqrt(2)  
  
# Afișăm rezultatul  
print(rezultat)

Figura 1. Codul exercitiului 1

Figura 2. Rezultatul exercitiului 1

**Exercitiul 2.**

Calculați + în aritmetica rațională.

from sympy import Rational  
  
# Definim fracțiile  
frac1 = Rational(1, 2)  
frac2 = Rational(1, 3)  
  
# Calculăm suma  
 = frac1 + frac2  
  
# Afișăm rezultatul  
print(suma)

Figura 3. Codul exercitiului 2



Figura 4. Rezultatul exercitiului 2

**Exercitiul 3.**

from sympy import symbols, expand  
  
# Definim simbolurile  
x, y = symbols('x y')  
  
# Definim expresia  
expresie = (x + y)\*\*6  
  
# Calculăm forma extinsă  
forma\_extinsa = expand(expresie)  
  
# Afișăm rezultatul  
print(forma\_extinsa)

Figura 5. Codul exercitiului 3



Figura 6. Rezultatul exercitiului 3

**Exercitiul 4.**

from sympy import symbols, cos, sin, simplify  
  
# Definim simbolul  
x = symbols('x')  
  
# Definim expresia trigonometrică  
expresie\_trig = sin(x)/cos(x)  
  
expresie\_simplificata = simplify(expresie\_trig)  
  
# Afișăm expresia simplificată  
print(expresie\_simplificata)

Figura . Codul exercitiului 4



Figura . Rezultatul exercitiului 4

**Exercitiul 5.**

from sympy import symbols, limit, sin  
  
# Definim simbolul  
x = symbols('x')  
  
# Definim expresia pentru limită  
expresie\_limita = sin(x) / x  
  
# Calculăm limita când x se apropie de 0  
rezultat\_limita = limit(expresie\_limita, x, 0)  
  
# Afișăm rezultatul  
print(rezultat\_limita)

Figura . Codul exercitiului 5



Figura . Rezultatul exercitiului 5

**Exercitiul 6.**

from sympy import symbols, log, diff  
  
# Definim simbolul  
x = symbols('x')  
  
# Definim funcția logaritmică  
functie\_log = log(x)  
  
# Calculăm derivata în raport cu x  
derivata = diff(functie\_log, x)  
  
# Afișăm rezultatul  
print(derivata)

Figura . Codul exercitiului 6



Figura . Rezultatul exercitiului 6

**Exercitiul 7.**

from sympy import symbols, Eq, solve  
  
# Definim simbolurile  
x, y = symbols('x y')  
  
# Definim ecuațiile  
ecuatie1 = Eq(2\*x + 3\*y, 5)  
ecuatie2 = Eq(4\*x - 3\*y, -4)  
  
# Rezolvăm sistemul de ecuații  
sol = solve((ecuatie1, ecuatie2), (x, y))  
  
# Afișăm rezultatul  
print(sol)

Figura . Codul exercitiului 7



Figura . Rezultatul exercitiului 7

**Exercitiul 8.**

from sympy import symbols, Not, Or, And, satisfiable  
  
# Definim simbolurile  
x, y = symbols('x y')  
  
# Definim expresia  
expresie = And(Or(x, Not(y)), Or(y, Not(x)))  
  
# Obținem primul model din generator  
primul\_model = next(satisfiable(expresie, all\_models=True), None)  
  
# Afișăm rezultatul  
print(primul\_model)

Figura . Codul exercitiului 8



Figura . Rezultatul exercitiului 8

**Exercitiul 9.**

from sympy import symbols, Function, Eq, dsolve, bernoulli, dsolve  
  
# Definim simbolul  
x = symbols('x')  
  
# Definim funcția necunoscută  
f = Function('f')(x)  
  
# Definim ecuația diferențială  
ecuatie = x\*f.diff(x) + f - 1/2\*f\*\*2  
  
# Rezolvăm ecuația diferențială direct  
sol\_direct = dsolve(ecuatie)  
  
# Rezolvăm ecuația diferențială cu hint-ul 'Bernoulli'  
sol\_bernoulli = dsolve(ecuatie, hint='Bernoulli')  
  
# Afișăm rezultatele  
print("Soluție directă:", sol\_direct)  
print("Soluție cu hint='Bernoulli':", sol\_bernoulli)

Figura . Codul exercitiului 9

Figura . Rezultatul exercitiului 9

**Exercitiul 10.**

from scipy.integrate import quad  
import numpy as np  
  
# Definim funcția integrată  
def f(x):  
 return np.cos(2 \* np.pi \* x)  
  
# Aplicăm funcția quad pentru a rezolva integrala  
rezultat, eroare = quad(f, 0, 1)  
  
# Afișăm rezultatele  
print("Rezultatul integrală:", rezultat)  
print("Eroarea estimată:", eroare)

Figura . Codul exercitiului 10



Figura . Rezultatul exercitiului 10

**Exercitiul 11.**

import numpy as np  
import matplotlib.pyplot as plt  
  
# Parametrii semnalului  
frecventa\_50hz = 50  
frecventa\_70hz = 70  
faza\_50hz = 0  
faza\_70hz = np.pi / 4  
amplitudinea = 1  
durata\_semnalului = 1  
frecventa\_esantionarii = 1000  
  
# Generarea timpului  
timp = np.arange(0, durata\_semnalului, 1 / frecventa\_esantionarii)  
  
# Generarea semnalului ca o suprapunere de sinusoide  
semnal = amplitudinea \* np.sin(2 \* np.pi \* frecventa\_50hz \* timp + faza\_50hz) + \  
 amplitudinea \* np.sin(2 \* np.pi \* frecventa\_70hz \* timp + faza\_70hz)  
  
# Calculul Transformatei Fourier  
transformata\_fourier = np.fft.fft(semnal)  
frecvente = np.fft.fftfreq(len(transformata\_fourier), 1 / frecventa\_esantionarii)  
  
# Reconstruirea semnalului din spectrul de amplitudini  
semnal\_reconstruit = np.fft.ifft(transformata\_fourier)  
  
# Generarea al doilea semnal identic cu semnalul reconstruit  
semnal\_identical = np.real(semnal\_reconstruit)  
  
# Afișarea ambelor semnale și a transformatei Fourier  
plt.figure(figsize=(12, 8))  
  
plt.subplot(3, 1, 1)  
plt.plot(timp, semnal)  
plt.title('Semnal ca suprapunere de 50 Hz și 70 Hz')  
plt.xlabel('Timp (s)')  
plt.ylabel('Amplitudine')  
plt.grid(True)  
  
plt.subplot(3, 1, 2)  
plt.plot(frecvente, np.abs(transformata\_fourier))  
plt.title('Transformata Fourier')  
plt.xlabel('Frecvență (Hz)')  
plt.ylabel('abs(DFT(semnal))')  
plt.xlim(0, 250)  
plt.grid(True)  
  
plt.subplot(3, 1, 3)  
plt.plot(timp, semnal\_identical)  
plt.title('Semnal identic cu semnalul reconstruit')  
plt.xlabel('Timp (s)')  
plt.ylabel('Amplitudine')  
plt.grid(True)  
  
plt.tight\_layout()  
plt.show()

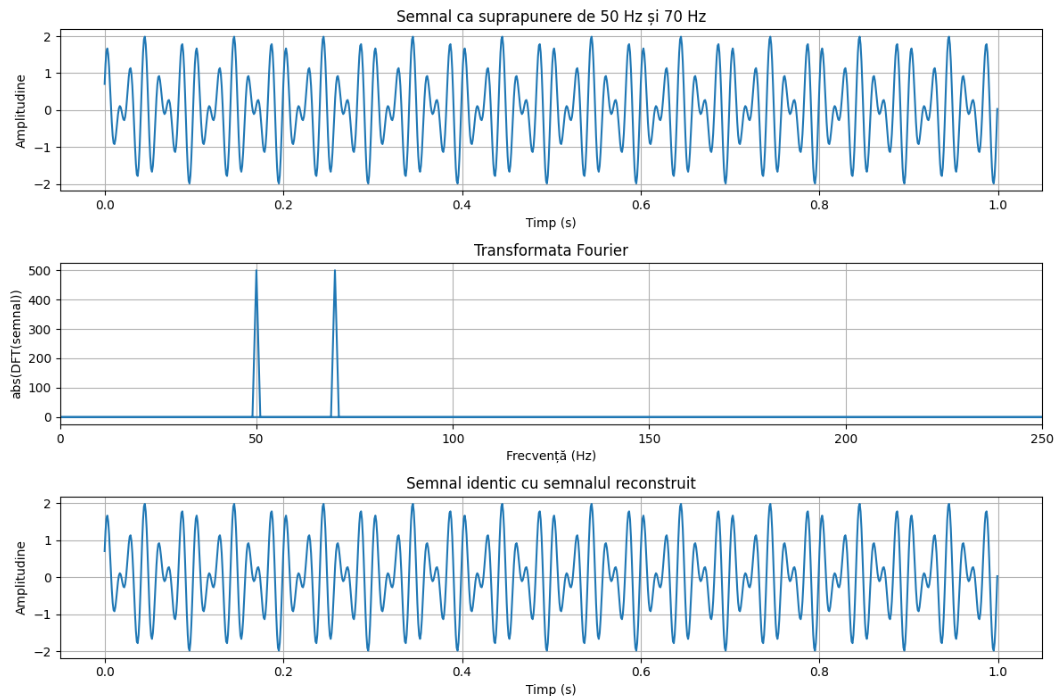
Figura . Codul exercitiului 11

Figura . Rezultatul exercitiului 11

**Exercitiul 12.**

import numpy as np  
from scipy.optimize import fmin  
import matplotlib.pyplot as plt  
  
# Definirea funcției de minimizat  
def f(x):  
 return np.cos(x) - 3 \* np.exp(-(x - 0.2)\*\*2)  
  
# Apelarea funcției de optimizare pentru x0 = 1.0  
x\_min\_1 = fmin(f, 1.0, disp=False)  
y\_min\_1 = f(x\_min\_1)  
  
# Apelarea funcției de optimizare pentru x0 = 2.0  
x\_min\_2 = fmin(f, 2.0, disp=False)  
y\_min\_2 = f(x\_min\_2)  
  
# Valoarea funcției pentru x = 1 și x = 2  
y\_at\_x1 = f(1)  
y\_at\_x2 = f(2)  
  
# Afișarea rezultatelor  
print("Valoarea minimă a lui x pentru x0 = 1.0:", x\_min\_1)  
print("Valoarea funcției pentru x = 1:", y\_at\_x1)  
  
print("Valoarea minimă a lui x pentru x0 = 2.0:", x\_min\_2)  
print("Valoarea funcției pentru x = 2:", y\_at\_x2)  
  
# Crearea unui grafic pentru funcția dată  
x\_values = np.linspace(-2, 4, 100)  
y\_values = f(x\_values)  
  
plt.plot(x\_values, y\_values, label='f(x)')  
  
plt.scatter(x\_min\_1, y\_min\_1, color='red', marker='^', label='Minimizare pentru x0=1.0')  
plt.scatter(x\_min\_2, y\_min\_2, color='blue', marker='^', label='Minimizare pentru x0=2.0')  
plt.scatter(1, y\_at\_x1, color='red', marker='o', label='f(1)')  
plt.scatter(2, y\_at\_x2, color='blue', marker='o', label='f(2)')  
plt.title('Graficul funcției și punctele de minimizare')  
plt.xlabel('x')  
plt.ylabel('f(x)')  
plt.legend()  
  
# Adăugarea unui grid  
plt.grid(True)  
  
plt.show()

Figura . Codul exercitiului 12

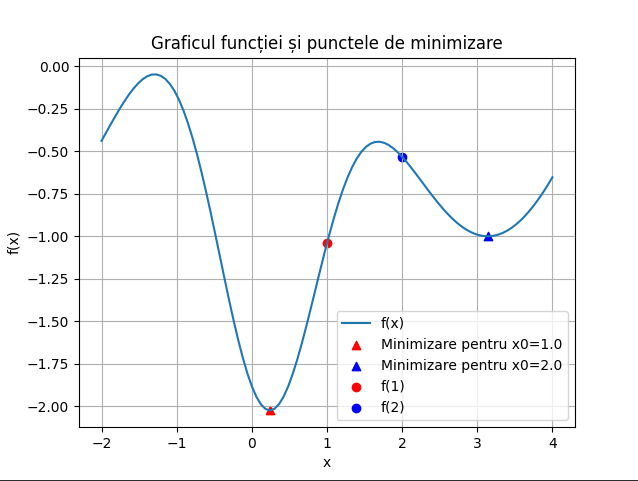


Figura . Rezultatul exercitiului 12

**Exercitiul 13.**

import numpy as np  
import matplotlib.pyplot as plt  
from scipy import ndimage  
from scipy.ndimage import gaussian\_filter, median\_filter  
from scipy.signal import wiener  
from skimage.transform import rotate  
  
# Încărcați o imagine de exemplu (puteți utiliza o imagine proprie)  
# Iată un exemplu cu o imagine simplă generată în mod aleator:  
image = plt.imread("download.jpg") # Încărcați imaginea folosind plt.imread  
  
# Convertiți imaginea la scală de gri dacă are mai multe canale de culoare  
if len(image.shape) == 3:  
 image = np.mean(image, axis=-1)  
  
# Schimbați orientarea imaginii (rotire)  
rotated\_image = rotate(image, angle=45, resize=True)  
  
# Schimbați rezoluția imaginii (zoom)  
zoom\_factor = 2  
zoomed\_image = ndimage.zoom(image.astype(float), zoom=(zoom\_factor, zoom\_factor))  
  
# Generați zgomot asupra imaginii  
noise = 0.1 \* np.random.random(zoomed\_image.shape)  
noisy\_image = zoomed\_image + noise  
  
# Aplicați filtre Gaussian, Median și Wiener  
gaussian\_filtered = gaussian\_filter(noisy\_image, sigma=1)  
median\_filtered = median\_filter(noisy\_image, size=3)  
wiener\_filtered = wiener(noisy\_image, mysize=3)  
  
# Afișați imaginile  
plt.figure(figsize=(15, 10))  
  
plt.subplot(231)  
plt.imshow(image, cmap='gray')  
plt.title('Imagine originală')  
  
plt.subplot(232)  
plt.imshow(rotated\_image, cmap='gray')  
plt.title('Imagine rotită')  
  
plt.subplot(233)  
plt.imshow(zoomed\_image, cmap='gray')  
plt.title('Imagine mărită')  
  
plt.subplot(234)  
plt.imshow(noisy\_image, cmap='gray')  
plt.title('Imagine cu zgomot')  
  
plt.subplot(235)  
plt.imshow(gaussian\_filtered, cmap='gray')  
plt.title('Filtru Gaussian')  
  
plt.subplot(236)  
plt.imshow(median\_filtered, cmap='gray')  
plt.title('Filtru Median')  
  
plt.show()

****

Filtru Wiener

Filtru Median

Filtru Gaussian

Imaginea cu zgomot

Imaginea rotita

Imaginea marita

Imaginea originala

**Conzluzie**

Laboratorul nr. 7 a fost dedicat explorării utilizării bibliotecilor NumPy, SciPy și SymPy în Python pentru efectuarea diverselor calcule matematice și manipulări de date. În cadrul acestui laborator, s-au abordat aspecte precum calculul rădăcinilor pătrate cu precizie extinsă, operațiuni cu fracții, extinderea expresiilor, simplificarea și evaluarea acestora. Problemele matematice rezolvate au inclus ecuații, calculul limitelor, derivarea funcțiilor și realizarea de integrări numerice.

Pe lângă aceste aspecte, cunoștințele acumulate au fost aplicate în procesul de manipulare a imaginilor, incluzând adăugarea de zgomot și aplicarea de filtre Gaussian, median și Wiener pentru îmbunătățirea calității imaginilor. Astfel, s-a obținut o experiență practică semnificativă în utilizarea eficientă a acestor biblioteci în contextul diverselor scopuri matematice și științifice.