**MINISTERUL EDUCAŢIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Ingineria Software și Automatică**

**Programul de studii: Tehnologia informației**

RAPORT

# LUCRARE DE LABORATOR NR. 7

# la Programarea Declarativă

**Tema: Utilizarea librării NumPy, SciPy și SymPy pentru efectuarea diferitor calcule în limbajul Python**

A efectuat:

st. gr. TI-211 Popa Cătălin

A verificat: lect.dr. Mariana Rusu

UTM, Chișinău 2023

**Tema**

Utilizarea librăriilor NumPy, SciPy și SymPy pentru efectuarea diferitor calcule în limbajul Python.

1.Calculați √2 cu 100 de zecimale.

from mpmath import mp  
mp.dps = 100  
raspuns = mp.sqrt(2)  
print(raspuns)



2. Calculați 1/2 + 1/ 3 în aritmetica rațională.

from fractions import Fraction  
fractie1 = Fraction(1, 2)  
fractie2 = Fraction(1, 3)  
rezultat = fractie1 + fractie2  
print("Rezultatul adunării:", rezultat)



3. Calculați forma extinsă a expresiei (𝑥 + 𝑦)^6.

from sympy import symbols, expand  
x, y = symbols('x y')  
expresie = (x + y)\*\*6  
forma\_extinsa = expand(expresie)  
print(forma\_extinsa)



4. Simplificați expresia trigonometrică sin(𝑥)/cos(𝑥).

from sympy import symbols, simplify, sin, cos  
x = symbols('x')  
expresie = sin(x) / cos(x)  
expresie\_simplificata = simplify(expresie)  
print("Expresia simplificată:", expresie\_simplificata)



5. Calculați lim 𝑥→0 sin (𝑥)/𝑥.

from sympy import symbols, limit, sin  
x = symbols('x')  
expresie = sin(x) / x  
rezultat\_limita = limit(expresie, x, 0)  
print("Limita când x se apropie de 0 pentru sin(x)/x:", rezultat\_limita)



6. Calculați derivata pentru log(𝑥) pentru 𝑥.

from sympy import symbols, diff, log  
x = symbols('x')  
expresie = log(x)  
derivata = diff(expresie, x)  
print("Derivata pentru log(x):", derivata)



7. Rezolvați sistemul de ecuații 2𝑥 + 3𝑦 = 5, 4𝑥 − 3𝑦 = −4.

from sympy import symbols, Eq, solve  
x, y = symbols('x y')  
ec1 = Eq(2\*x + 3\*y, 5)  
ec2 = Eq(4\*x - 3\*y, -4)  
rezultat = solve((ec1, ec2), (x, y))  
print(f"x = {rezultat[x]}, y = {rezultat[y]}")



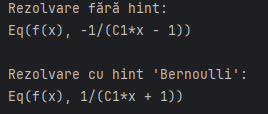
8. Există valori booleene x, y care fac expresia (𝑥∨¬𝑦)∧(𝑦∨¬𝑥) adevărată? Argumentați răspunsul, folosiți sym.satisfiable.

from sympy import symbols, Not, Or, And, satisfiable  
x, y = symbols('x y')  
expresie = And(Or(x, Not(y)), Or(y, Not(x)))  
# Verificăm dacă există valori care satisfac expresia  
asignare = satisfiable(expresie)  
  
if asignare:  
 print("Există valori pentru x și y care fac expresia adevărată:")  
 print(asignare)  
else:  
 print("Nu există valori pentru x și y care să facă expresia adevărată.")



9. Rezolvați ecuația diferențială a lui Bernoulli 𝑥 𝑑𝑓(𝑥) 𝑥 + 𝑓(𝑥) − 𝑓(𝑥)2 = 0. Rezolvați aceeași ecuație folosind hint='Bernoulli'. Ce observati?

from sympy import symbols, Function,dsolve  
x = symbols('x')  
# Definim funcția necunoscută f(x)  
f = Function('f')  
# Definim ecuația diferențială Bernoulli  
ecuatie = x \* f(x).diff(x) + f(x) - f(x)\*\*2  
# Rezolvare fără hint  
sol\_fara\_hint = dsolve(ecuatie)  
# Rezolvare cu hint 'Bernoulli'  
sol\_cu\_hint = dsolve(ecuatie, hint='Bernoulli')  
  
print("Rezolvare fără hint:")  
print(sol\_fara\_hint)  
print("\nRezolvare cu hint 'Bernoulli':")  
print(sol\_cu\_hint)



10. Folosind funcția quad() din librăria scipy, scrieți un program care rezolvă următoarea integrală numerică: 𝐼 = ∫ cos(2𝜋𝑥) 𝑑𝑥 1 0 . De ce este important să avem o estimare a preciziei (sau a erorii) integralei numerice?

from scipy.integrate import quad  
import numpy as np  
def f(x):  
 return np.cos(2 \* np.pi \* x)  
integrala, eroare = quad(f, 0, 1)  
print("Valoarea integrală:", integrala)  
print("Estimarea erorii:", eroare)



11. Creați un semnal ca o suprapunere a unei unde sinusoidale de 50 Hz și 70 Hz (cu o ușoară schimbare de fază între ele). Apoi transformați semnalul Fourier și trasați valoarea absolută a coeficienților (complexi) discreți de transformare Fourier în funcție de frecvență (observați vârfuri la 50Hz și 70Hz).

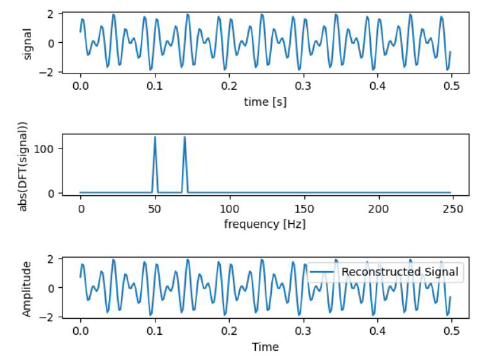
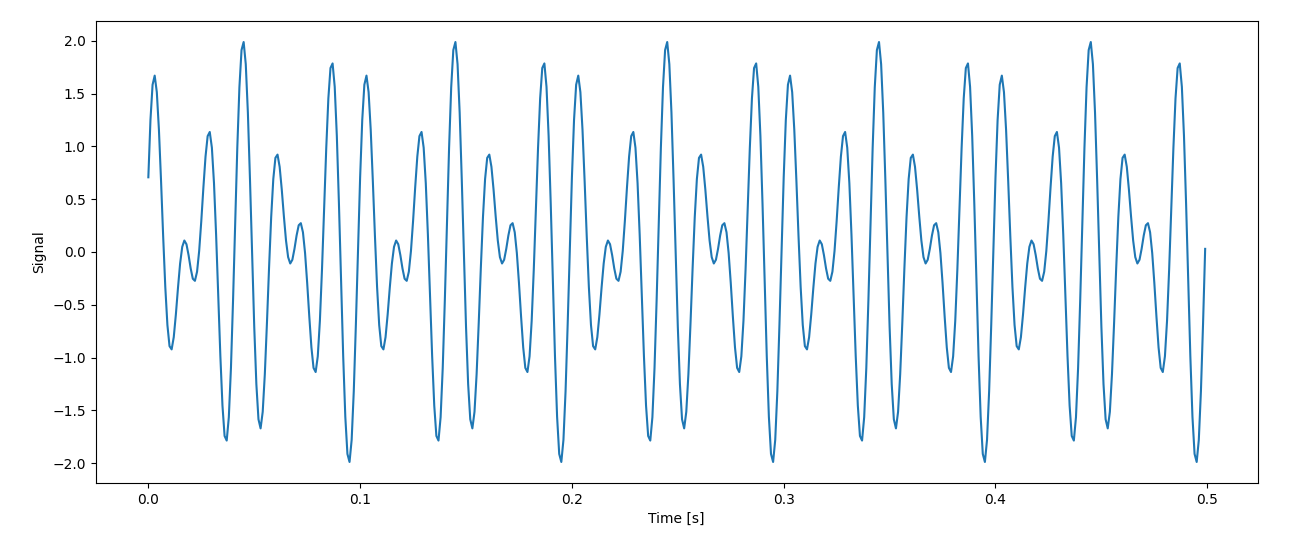
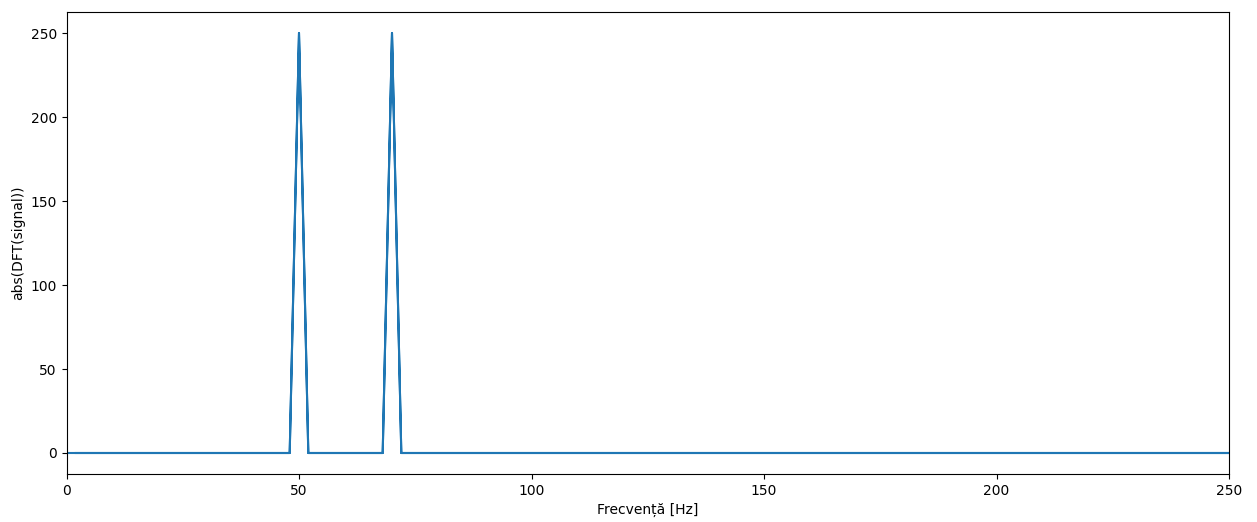
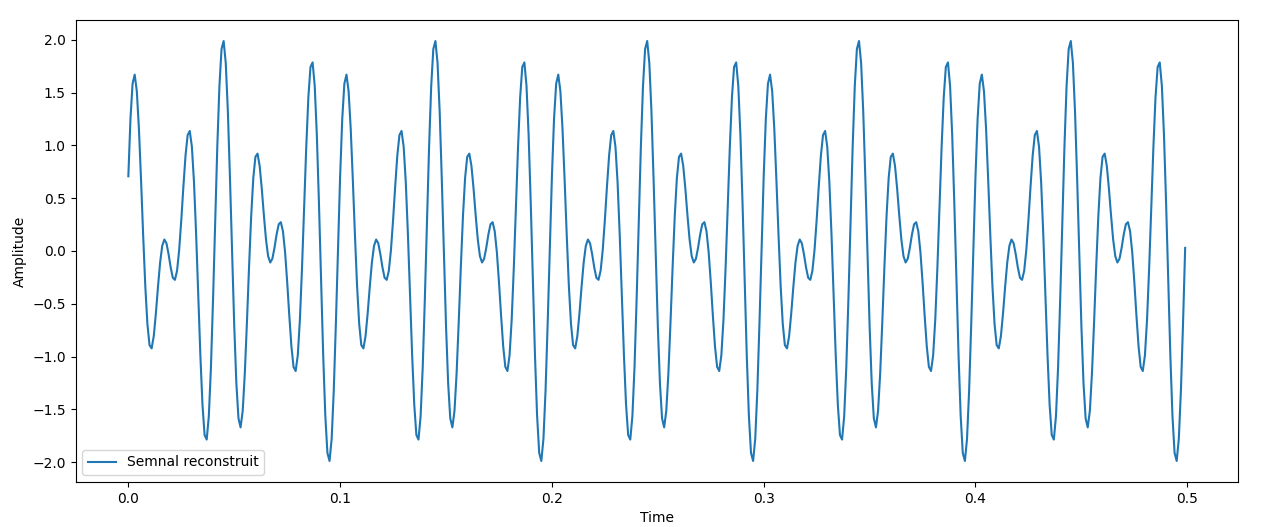


Figura 2 – Grafic.

import numpy as np  
import matplotlib.pyplot as plt  
  
fr1 = 50   
fr2 = 70   
  
timp\_total = 0.5   
frecventa\_esantionare = 1000   
  
timp = np.linspace(0, timp\_total, int(timp\_total \* frecventa\_esantionare), endpoint=False)  
semnal = np.sin(2 \* np.pi \* fr1 \* timp) + np.sin(2 \* np.pi \* fr2 \* timp + np.pi/4)  
  
plt.figure(figsize=(15, 6))  
plt.plot(timp, semnal)  
plt.xlabel('Time [s]')  
plt.ylabel('Signal')  
plt.show()  
  
transformata = np.fft.fft(semnal)  
frecvente = np.fft.fftfreq(len(semnal), 1 / frecventa\_esantionare)  
  
plt.figure(figsize=(15, 6))  
plt.plot(np.abs(frecvente), np.abs(transformata))  
plt.xlabel('Frecvență [Hz]')  
plt.ylabel('abs(DFT(signal))')  
plt.xlim(0, 250)  
plt.show()  
  
componenta\_50Hz = np.where((np.abs(frecvente) >= fr1 - 1) & (np.abs(frecvente) <= fr1 + 1))[0][0]  
componenta\_70Hz = np.where((np.abs(frecvente) >= fr2 - 1) & (np.abs(frecvente) <= fr2 + 1))[0][0]  
  
semnal\_reconstruit = np.fft.ifft(transformata)  
semnal\_reconstruit = semnal\_reconstruit.real  
  
plt.figure(figsize=(15, 6))  
plt.plot(timp, semnal\_reconstruit, label='Semnal reconstruit')  
plt.xlabel('Time')  
plt.ylabel('Amplitude')  
plt.legend()  
plt.show()







12. Găsiți valoarea minimă a lui 𝑥 pentru optimizarea expresiei cos(𝑥) − 3𝑒−(𝑥−0,2)2 .

Apelați funcția scipy.optimize.fmin care ia ca argument o funcție *f* pentru a minimiza și o valoare inițială *x0* de la care să pornească căutarea pentru minim și care returnează valoarea lui *x* pentru care *f(x)* este (local) minimizat. Repetați căutarea minimului pentru două valori (x0 = 1.0 și, respectiv, x0 = 2.0) pentru a demonstra că în funcție de valoarea de pornire putem găsi minime diferite ale funcției *f*.

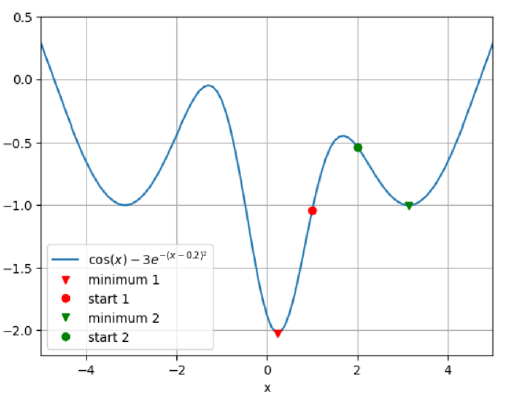
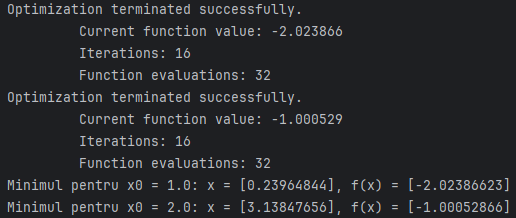
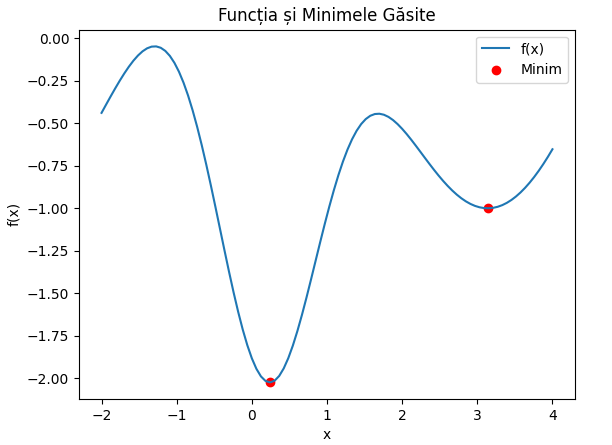


Figura 1 – Grafic.

import numpy as np  
import matplotlib.pyplot as plt  
from scipy.optimize import fmin  
  
# Definirea funcției de minimizat  
def f(x):  
 return np.cos(x) - 3 \* np.exp(-(x - 0.2)\*\*2)  
  
# Căutarea minimului pentru x0 = 1.0  
x\_min\_1 = fmin(f, x0=1.0)  
min\_value\_1 = f(x\_min\_1)  
  
# Căutarea minimului pentru x0 = 2.0  
x\_min\_2 = fmin(f, x0=2.0)  
min\_value\_2 = f(x\_min\_2)  
  
# Afișarea rezultatelor  
print(f"Minimul pentru x0 = 1.0: x = {x\_min\_1}, f(x) = {min\_value\_1}")  
print(f"Minimul pentru x0 = 2.0: x = {x\_min\_2}, f(x) = {min\_value\_2}")  
  
# Generarea unui set de puncte pentru a afișa funcția  
x\_values = np.linspace(-2, 4, 100)  
y\_values = f(x\_values)  
  
# Afișarea funcției și punctelor de minim pe un plot  
plt.plot(x\_values, y\_values, label='f(x)')  
plt.scatter([x\_min\_1, x\_min\_2], [min\_value\_1, min\_value\_2], color='red', label='Minim')  
  
# Adăugarea etichetelor și a titlului  
plt.xlabel('x')  
plt.ylabel('f(x)')  
plt.title('Funcția și Minimele Găsite')  
  
# Adăugarea unei legende  
plt.legend()  
  
# Afișarea plotului  
plt.show()

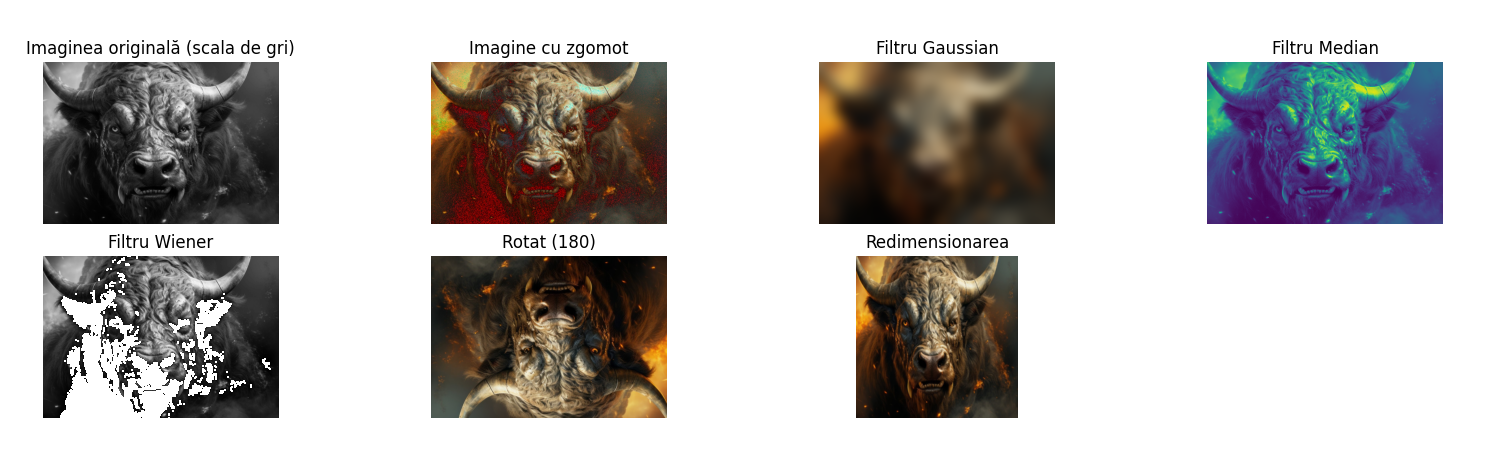




13.

Efectuați diverse manipulări asupra unei imagini : schimbați orientarea, rezoluția (scipy.ndimage oferă manipularea tablourilor n-dimensionale ca imagini). Generați zgomot asupra imaginii, apoi folosiți pe rând filtrele Gaussian, median, Wiener. Observați eficacitatea fiecărui filtru.

import numpy as np  
import matplotlib.pyplot as plt  
from scipy import signal, ndimage  
from PIL import Image, ImageFilter  
  
  
image\_path = 'Screenshot\_2.png'  
original\_image = Image.open(image\_path)  
  
# Convertiți imaginea la scala de gri  
gray\_image = original\_image.convert('L')  
gray\_array = np.array(gray\_image)  
  
# Afișați imaginea originală  
plt.figure(figsize=(8, 8))  
plt.subplot(4, 4, 1)  
plt.imshow(gray\_array, cmap='gray')  
plt.title('Imaginea originală (scala de gri)')  
plt.axis('off')  
   
# Generați zgomot aleator  
noisy\_array = np.random.normal(loc=0, scale=25, size=(original\_image.height, original\_image.width)).astype(np.uint8)  
noisy\_image = original\_image.copy()  
noisy\_image\_array = np.array(noisy\_image)  
noisy\_image\_array[:, :, 0] += noisy\_array  
  
# afisare zgomot  
plt.subplot(4, 4, 2)  
plt.imshow(noisy\_image\_array)  
plt.title('Imagine cu zgomot')  
plt.axis('off')  
  
# Aplicați filtrele Gaussian  
gaussian\_filtered = noisy\_image.filter(ImageFilter.GaussianBlur(radius=30))  
  
# Afișați rezultatele filtrării  
plt.subplot(4, 4, 3)  
plt.imshow(gaussian\_filtered)  
plt.title('Filtru Gaussian')  
plt.axis('off')  
  
#median  
median\_filtered = gray\_image.filter(ImageFilter.MedianFilter(size=1))  
plt.subplot(4, 4, 4)  
plt.imshow(median\_filtered)  
plt.title('Filtru Median')  
plt.axis('off')  
  
# Aplicați filtrul Wiener  
wiener\_filtered = signal.wiener(gray\_array, mysize=(1, 1))  
plt.subplot(4, 4, 5)  
plt.imshow(wiener\_filtered, cmap='gray')  
plt.title('Filtru Wiener')  
plt.axis('off')  
  
# rotarea  
rotat = original\_image.rotate(180)  
plt.subplot(4, 4, 6)  
plt.imshow(rotat, cmap='gray')  
plt.title('Rotat (180)')  
plt.axis('off')  
  
  
im\_resized = original\_image.resize((250, 250))  
plt.subplot(4, 4, 7)  
plt.imshow(im\_resized)  
plt.title('Redimensionarea')  
plt.axis('off')  
  
plt.show()



**Concluzie**

În urma laboratorului dat, am calculat √2 cu 100 de zecimale, rezolvat probleme de aritmetică rațională și trigonometrie, derivate și integrale, sisteme de ecuații și ecuații diferențiale. Am demonstrat importanța preciziei în integrarea numerică și am aplicat transformata Fourier în analiza semnalului. În plus, am optimizat funcții și efectuat manipulări asupra imaginilor, utilizând diverse filtre și evaluând eficacitatea acestora.