**Ministerul Educației și Cercetării al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**RAPORT**

Lucrare de laborator nr.7

la cursul „Programarea Declarativă”

**Tema 7:** „Calcule matematice cu ajutorul librăriilor Numpy, SymPy și SciPy”

A efectuat : **st. gr. TI-214** Buza Cătălin A verificat:  **M.Rusu**

**Chișinău 2023**

1. **Calculați √2 cu 100 de zecimale.**

Am utilizat librăria SymPy pentru a calcula rădăcina pătrată a lui 2 cu o precizie de 100 de zecimale. Rezultatul a fost afișat cu ajutorul funcției N și sqrt.

print(sympy.N(sympy.sqrt(2), 100))

1. **Calculați 1⁄2 + 1⁄3 în aritmetica rațională.**

Am utilizat SymPy pentru a efectua operații în aritmetica rațională, exemplificând adunarea a două fracții.

print(sympy.Rational(1, 2) + sympy.Rational(1, 3))

1. **Calculați forma extinsă a expresiei (x + y)6.**

Am folosit SymPy pentru a calcula forma extinsă a expresiei $(x + y)^6$.

print(sympy.expand((x + y)6))

1. **Simplificați expresia trigonometrică sin(x)⁄cos(x).**

Am utilizat SymPy pentru a simplifica expresia trigonometrică $\frac{\sin(x)}{\cos(x)}$.

print(sympy.simplify(sympy.sin(x) / sympy.cos(x)))

1. **Calculați lim sin(x)⁄x.**

x→0

Am folosit SymPy pentru a calcula limita $\lim\_{x \to 0} \frac{\sin(x)}{x}$.

print(sympy.limit(sympy.sin(x) / x, x, 0))

1. **Calculați derivata pentru log(x) pentru x**

Am utilizat SymPy pentru a calcula derivata funcției $\log(x)$.

print(sympy.log(sympy.log(x), x))

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

ec1 = sympy.Eq(2\*x + 3\*y, 5)

ec2 = sympy.Eq(4\*x + 3\*y, -4)

print(sympy.solve((ec1, ec2), (x, y)))

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

Am folosit SymPy pentru a determina dacă există valori booleane $x$ și $y$ care să satisfacă expresia $(x \lor \lnot y) \land (y \lor \lnot x)$.

print(sympy.satisfiable(sympy.And(sympy.Or(x, sympy.Not(y)), sympy.Or(y, sympy.Not(x)))))

1. **Rezolvați ecuația diferențială a lui Bernoulli .Rezolvati acceeasi ecuație folosind hint='Bernoulli'. Ce observati?**

Am rezolvat ecuația diferențială a lui Bernoulli $x \frac{df(x)}{dx} + f(x) - f(x)^2 = 0$ folosind SymPy, atât fără hint, cât și cu hint='Bernoulli'.

f = sympy.Function('f')

ec = sympy.Eq(x \* sympy.Derivative(f(x), x) + f(x) - f(x)2, 0)

print("Fara hint:", sympy.dsolve(ec))

print("Cu hint:", sympy.dsolve(ec, hint='Bernoulli'))

1. Folosind funcția quad() din librăria scipy, scrieți un program care rezolvă următoarea

integrală numerică: . De ce este important să avem o estimare a

0

preciziei (sau a erorii) integralei numerice?

Se efectuiază integrarea numerică a unei funcții folosind funcția `quad()` din librăria `scipy.integrate`.

def f(x):

return np.cos(2 \* np.pi \* x)

Se definește o funcție `f(x)` care reprezintă funcția ce va fi integrată numeric. În acest caz, funcția este $\cos(2 \pi x)$.

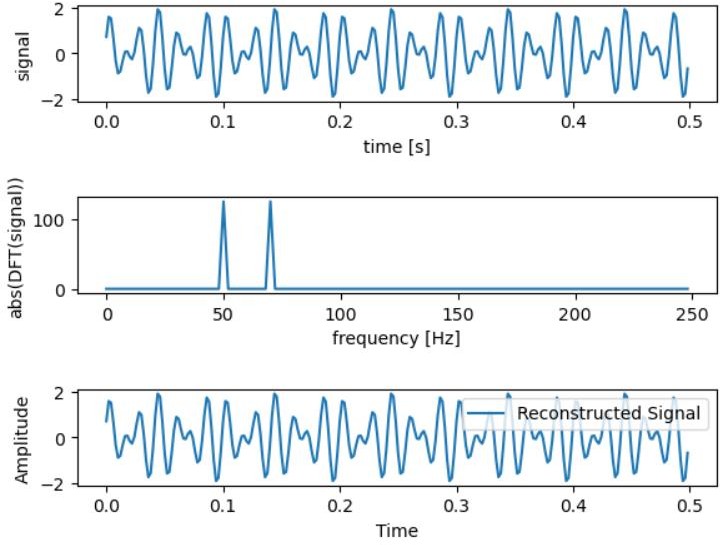
rez, err = scipy.integrate.quad(f, 0, 1)

Se folosește funcția `quad()` pentru a efectua integrarea numerică a funcției `f(x)` în intervalul [0, 1]. Rezultatul integrării este returnat în variabila `rez`, iar eroarea estimată este returnată în variabila `err`.

print("rezultat:", rez)

print("err:", err)

Se afișează rezultatul integrării sub formă de mesaj, unde rez reprezintă valoarea aproximativă a integralei, iar err este o estimare a erorii asociate acestei aproximări.

1. 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).

Am generat un semnal ca o suprapunere de două unde sinusoidale și am calculat transformata Fourier, apoi am afișat spectrul semnalului.

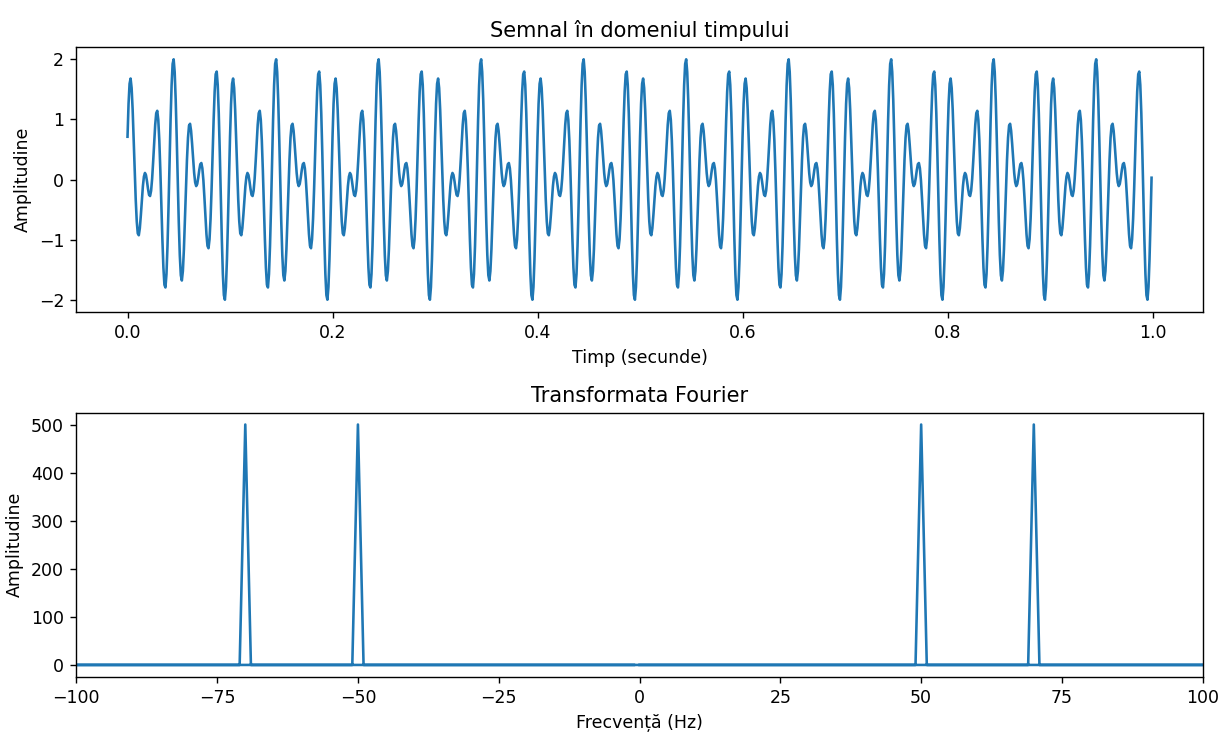
plt.plot(np.abs(frecvente), np.abs(transformata))

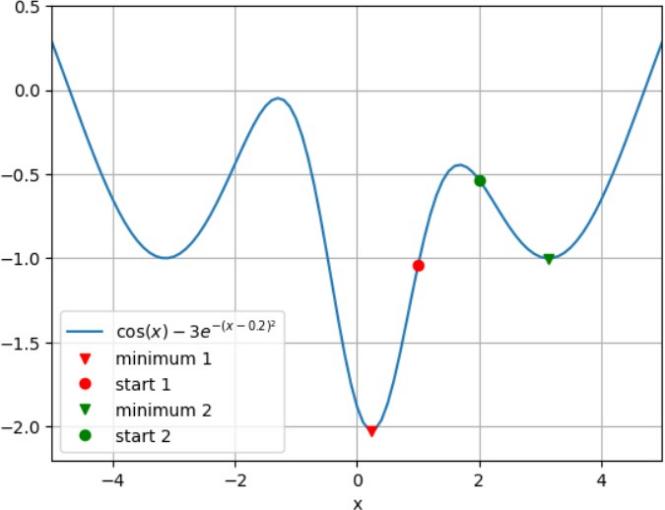
plt.xlabel('Frecvență (Hz)')

plt.ylabel('Amplitudine')

plt.title('Transformata Fourier a semnalului')

plt.show()



1. 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*.

Am folosit funcția fmin din SciPy pentru a găsi valoarea minimă a funcției $\cos(x) - 3e^{-(x-0,2)^2}$, pornind de la două valori inițiale diferite.

def functie\_optimizare(x):

return np.cos(x) - 3 \* np.exp(-(x-0.2)2)

valoare\_minima\_x1 = fmin(functie\_optimizare, 1.0)

valoare\_minima\_x2 = fmin(functie\_optimizare, 2.0)

1. 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.

Codul oferă exemple practice privind manipularea imaginilor folosind funcționalitățile din modulele `scipy.ndimage` și `scipy.signal`. Vom explica fiecare secțiune a codului:

**1. Generarea imaginii de test:**

original\_image = np.random.random((256, 256))

Se generează o imagine alb-negru de dimensiune 256x256 pixeli, unde intensitățile pixelilor sunt aleatoare, distribuite uniform între 0 și 1.

**2. Schimbarea orientării imaginii:**

rotated\_image = ndimage.rotate(original\_image, angle=45)

Se rotește imaginea originală cu un unghi de 45 de grade în sensul acelor de ceasornic folosind funcția `rotate` din `scipy.ndimage`.

**3. Schimbarea rezoluției imaginii:**

resized\_image = ndimage.zoom(original\_image, zoom=(0.5, 0.5))

Se realizează redimensionarea imaginii originali la jumătate din dimensiunile inițiale folosind funcția `zoom` din `scipy.ndimage`.

**4. Generarea de zgomot asupra imaginii:**

noisy\_image = original\_image + 0.5 \* np.random.normal(size=original\_image.shape)

Se adaugă zgomot gaussiano-normalizat la imaginea originală. Zgomotul este generat folosind funcția `normal` din NumPy.

**5. Aplicarea unui filtru Gaussian:**

gaussian\_filtered = ndimage.gaussian\_filter(noisy\_image, sigma=2)

Se aplică un filtru Gaussian asupra imaginii zgomotoase folosind funcția `gaussian\_filter` din `scipy.ndimage`. Parametrul `sigma` controlează dispersia filtrului Gaussian.

**6. Aplicarea unui filtru Median:**

median\_filtered = ndimage.median\_filter(noisy\_image, size=3)

Se aplică un filtru median asupra imaginii zgomotoase folosind funcția `median\_filter` din `scipy.ndimage`. Parametrul `size` specifică dimensiunea ferestrei mediane.

**7. Aplicarea filtrului Wiener:**

wiener\_filtered = signal.wiener(noisy\_image)

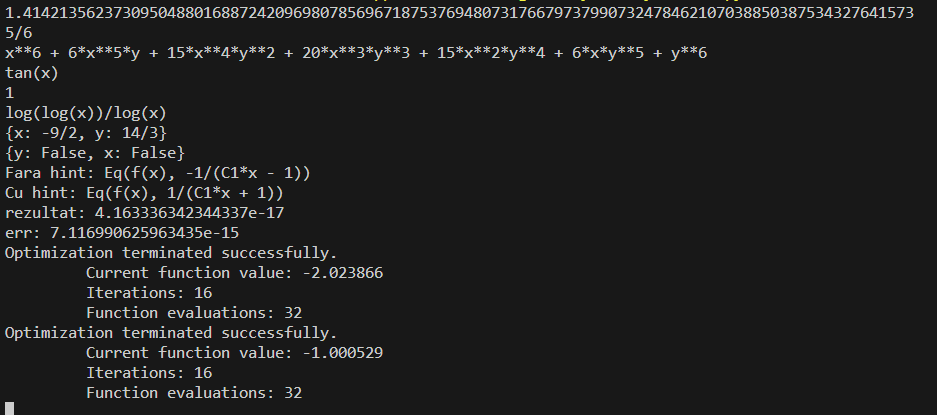
Se aplică filtrul Wiener asupra imaginii zgomotoase folosind funcția `wiener` din `scipy.signal`.

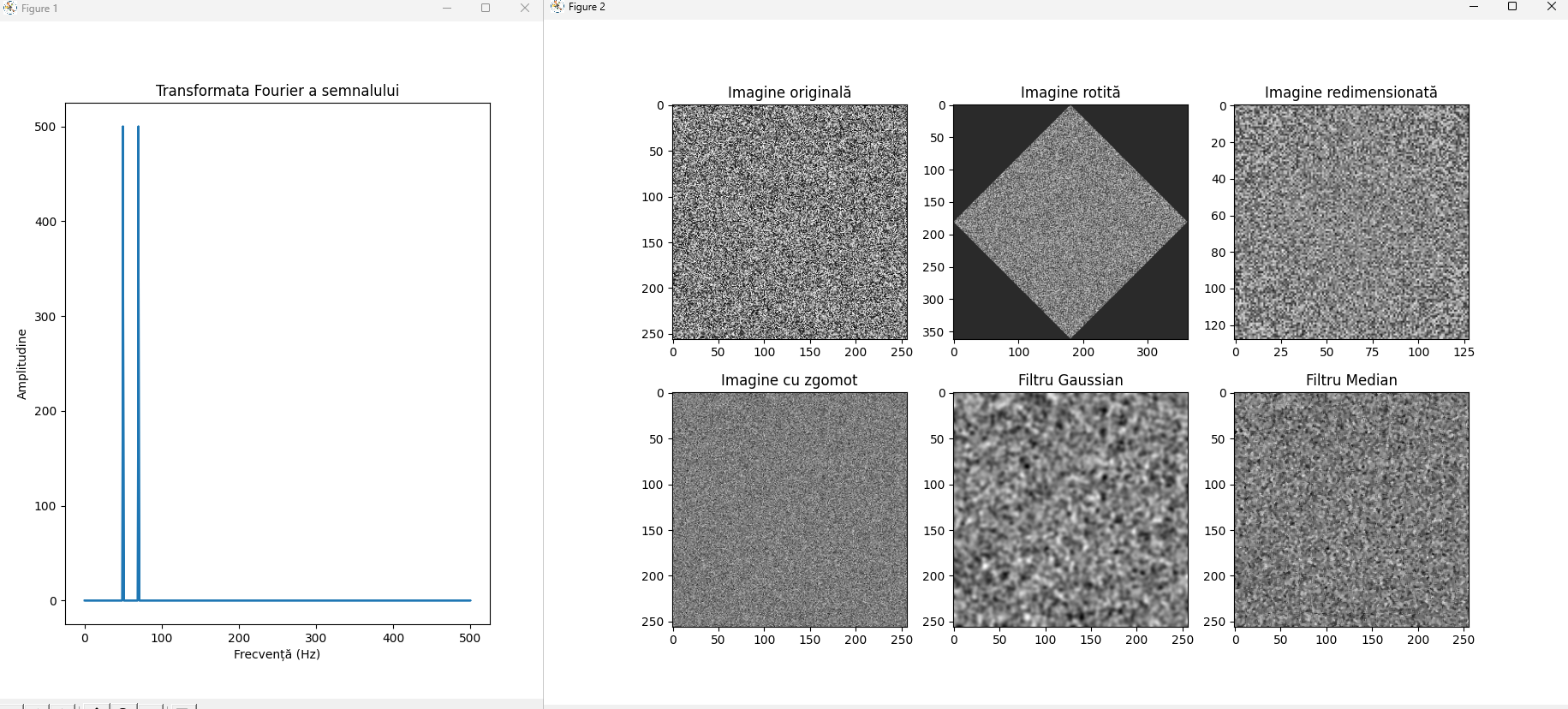
**8. Afișarea imaginilor:**

plt.show()

Se afișează imaginile generate și procesate într-un set de sub-figuri folosind funcționalitatea `subplot` și `imshow` din Matplotlib.

**Rezultatul executatiei**





**Concluzie**

Lucrarea de laborator a oferit o perspectivă detaliată asupra capacităților extinse ale librăriilor NumPy, SymPy și SciPy în manipularea și rezolvarea problemelor matematice variate în limbajul Python. Utilizarea acestor librării a simplificat semnificativ procesele de calcul și analiză matematică, oferind soluții eficiente pentru o gamă largă de probleme.