Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Звіт

З лабораторної роботи №1

З курсу

«Моделювання складних систем»

Варіант 4

Виконав:

Студент групі ІПС-31

Павлюченко Василь Іванович

Київ

2024

**Постановка задачі**

На вході маємо значення деякої функції (сигналу) y з файлу «f4.txt»”. Значення є дискретними, крок аргументу функції

Завданням лабораторної є встановити аналітичний вигляд функції y(t), обрахувати її невідомі параметри.

**Хід роботи**

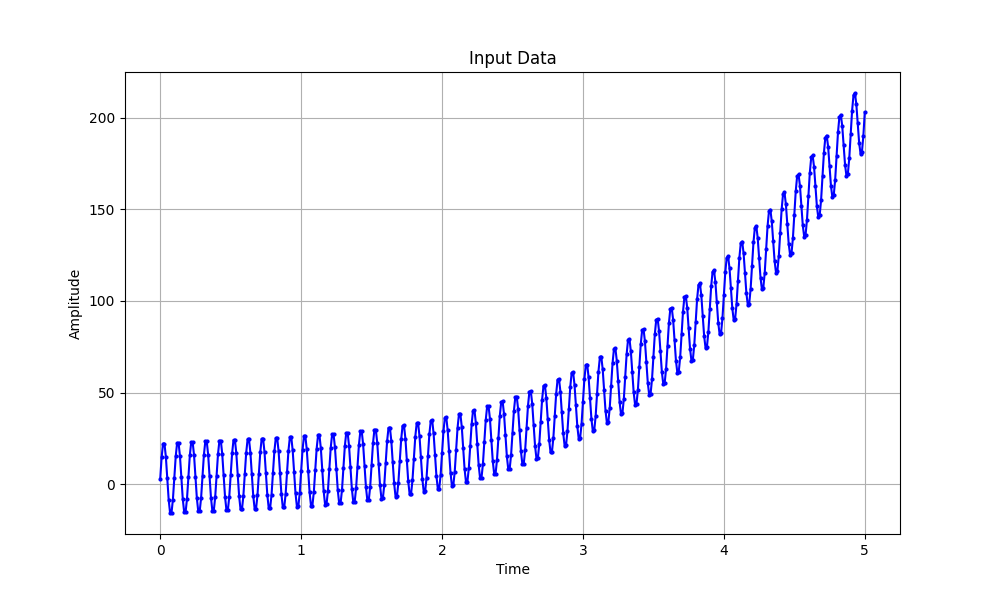
Завантажуємо дані з файлу:

# Load data  
y = load\_data('f4.txt')

Ініціалізовуємо необхідні змінні:

# Constants  
T = 5  
delta\_t = 0.01  
t = np.arange(0, T + delta\_t, delta\_t)  
N = t.size

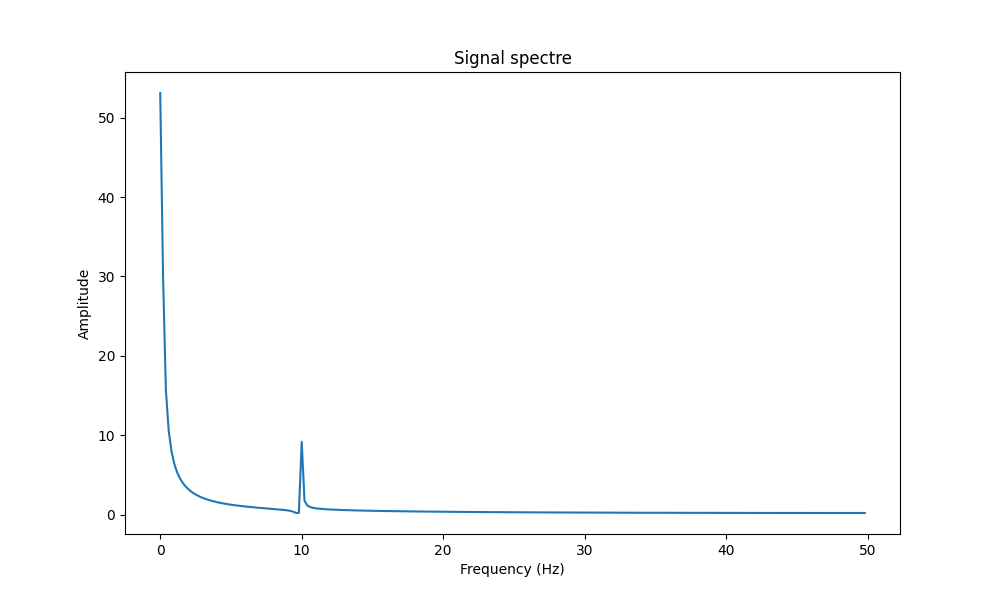
Проілюструємо вхідний сигнал на графіку:



Дізнаємось, які значні частоти наявні в сигналі. Для цього виконуємо дискретне перетворення Фур’є для отриманих значень:

# Discrete Fourier Transform  
def dft(x):  
 *"""Compute the Discrete Fourier Transform of the input signal."""* N = len(x)  
 X = np.zeros(N, dtype=complex)  
 for k in range(N):  
 X[k] = sum(x[n] \* np.exp(-2j \* np.pi \* k \* n / N) for n in range(N))  
 return X / N

Проілюструємо отримані результати на графіку. Графік є симетричним відносно середини, тому розглядаємо лише його першу половину:



Великий вклад частот, близьких до нуля, зумовлений поліноміальною частиною сигналу. Відповідно, нас цікавлять лише локальні максимуми, віддалені від нуля. На нашому графіку бачимо лише одну таку точку.

Знайдемо основні частоти за формулою , де *k* - локальні максимуми перетворення Фур’є, :

# Find significant frequencies  
def find\_significant\_frequencies(Y):  
 *"""Identify significant frequencies from the DFT result."""* delta\_f = 1 / T  
 magnitude = np.abs(Y)  
 k\_star, \_ = find\_peaks(magnitude[:N // 2])  
  
 peaks = []  
  
 for i in range(1, N - 1):  
 if (magnitude[i] > magnitude[i - 1] and magnitude[i] > magnitude[i + 1] and abs(magnitude[i] - magnitude[i - 1]) > 1):  
 peaks.append(i)  
  
 frequencies = k\_star \* delta\_f  
 return frequencies

Підставимо отримані значення у модель:

Отримуємо модель, яка є лінійною відносно невідомих параметрів . Використаємо метод найменших квадратів для оцінки цих параметрів.

Для цього знайдемо похідні функціонала S за кожним з параметрів та прирівняємо їх до 0 та розв’яжемо отриману СЛАР:

# Calculate square error  
def calculate\_square\_error(y, x, frequencies, a):  
 *"""Calculate the square error between the model and the observed data."""* n = len(a)  
 expression = 0  
 for i in range(len(y)):  
 t = x[i]  
 tmp = a[0] \* t \*\* 3 + a[1] \* t \*\* 2 + a[2] \* t + a[n - 1] - y[i]  
 for j in range(3, n - 1):  
 of = 2 \* np.pi \* frequencies[j - 3] \* t  
 tmp += a[j] \* sp.sin(of)  
 expression += tmp \*\* 2  
 return expression  
  
# Least squares method  
def custom\_lsq(y, x, frequencies):  
 *"""Fit a model to the data using the least squares method."""* n = 4 + len(frequencies)  
 a = generate\_a\_symbols(n)  
 expression = calculate\_square\_error(y, x, frequencies, a)  
 gradient = [sp.diff(expression, param) for param in a]  
 solution = sp.solve(gradient, a)  
 print("Found parameters:", solution)  
 return solution

Отримуємо сумісну систему з 5 рівнянь та 5 невідомих:

Розв’язуємо отриману систему та отримуємо наступну оцінку параметрів:



Підставляємо отримані параметри у модель:

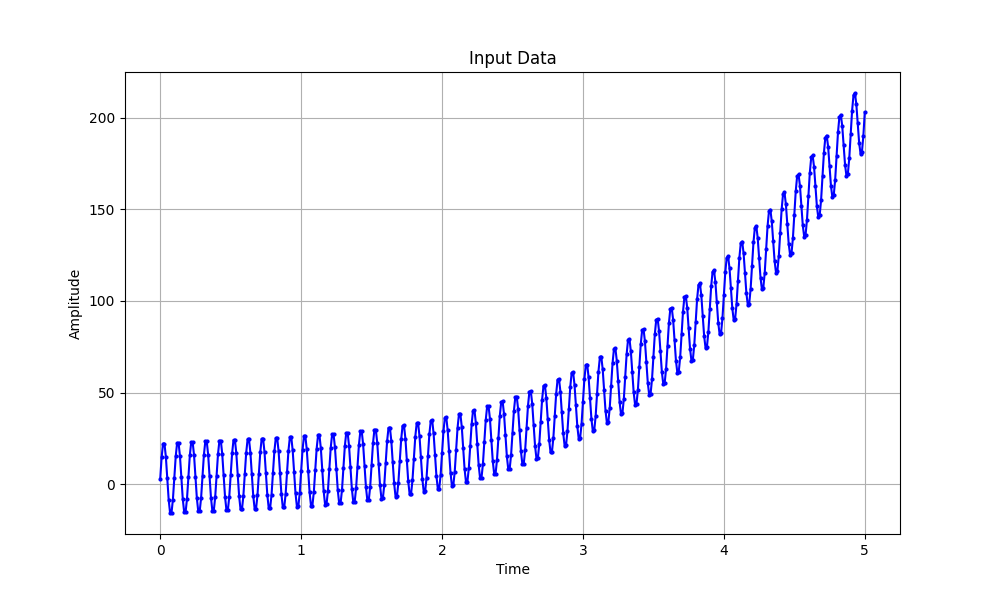
Обчислимо квадратичну похибку:

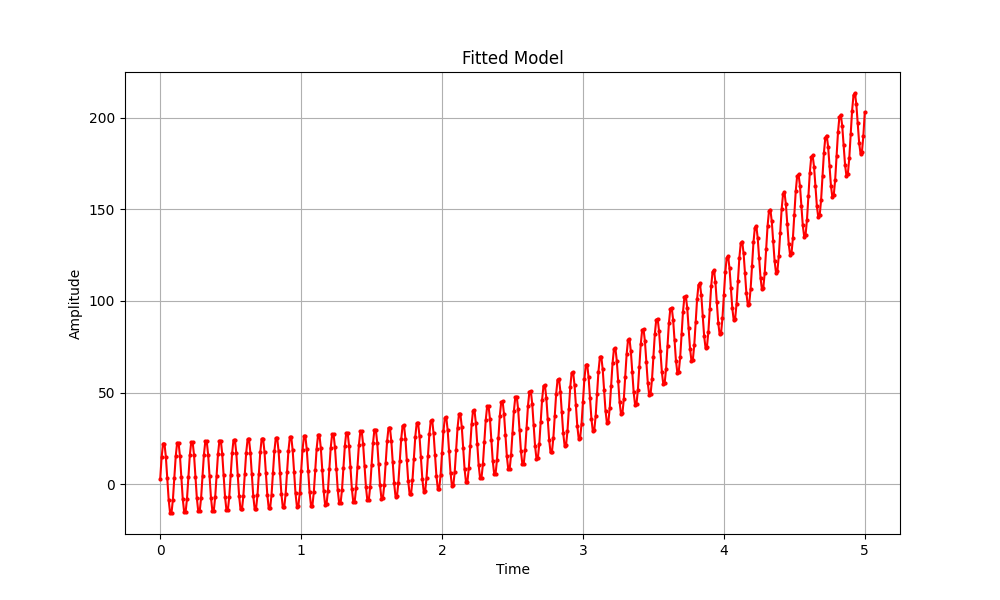
# Calculate square error  
def calculate\_square\_error(y, x, frequencies, a):  
 *"""Calculate the square error between the model and the observed data."""* n = len(a)  
 expression = 0  
 for i in range(len(y)):  
 t = x[i]  
 tmp = a[0] \* t \*\* 3 + a[1] \* t \*\* 2 + a[2] \* t + a[n - 1] - y[i]  
 for j in range(3, n - 1):  
 of = 2 \* np.pi \* frequencies[j - 3] \* t  
 tmp += a[j] \* sp.sin(of)  
 expression += tmp \*\* 2  
 return expression

Та отримаємо наступне значення:



Порівняємо отриманий графік з початковим:





Можемо побачити, що отриманий графік співпадає з початковим. Відповідно, можемо зробити висновок, що отримана модель є досить точною.

Отже, отримана відповідь:

**Повний код програми**

import numpy as np  
import matplotlib.pyplot as plt  
from scipy.signal import find\_peaks  
import sympy as sp  
  
# Constants  
T = 5  
delta\_t = 0.01  
t = np.arange(0, T + delta\_t, delta\_t)  
N = t.size  
  
# Load data  
def load\_data(file\_path):  
 *"""Load data from a text file."""* return np.loadtxt(file\_path)  
  
# Discrete Fourier Transform  
def dft(x):  
 *"""Compute the Discrete Fourier Transform of the input signal."""* N = len(x)  
 X = np.zeros(N, dtype=complex)  
 for k in range(N):  
 X[k] = sum(x[n] \* np.exp(-2j \* np.pi \* k \* n / N) for n in range(N))  
 return X / N  
  
# Find significant frequencies  
def find\_significant\_frequencies(Y):  
 *"""Identify significant frequencies from the DFT result."""* delta\_f = 1 / T  
 magnitude = np.abs(Y)  
 k\_star, \_ = find\_peaks(magnitude[:N // 2])  
  
 peaks = []  
  
 for i in range(1, N - 1):  
 if (magnitude[i] > magnitude[i - 1] and magnitude[i] > magnitude[i + 1] and abs(magnitude[i] - magnitude[i - 1]) > 1):  
 peaks.append(i)  
  
 frequencies = k\_star \* delta\_f  
 return frequencies  
  
# Generate symbols for parameters  
def generate\_a\_symbols(n):  
 *"""Generate symbolic variables for parameters in SymPy."""* return sp.symbols(f'a0:{n}')  
  
# Calculate square error  
def calculate\_square\_error(y, x, frequencies, a):  
 *"""Calculate the square error between the model and the observed data."""* n = len(a)  
 expression = 0  
 for i in range(len(y)):  
 t = x[i]  
 tmp = a[0] \* t \*\* 3 + a[1] \* t \*\* 2 + a[2] \* t + a[n - 1] - y[i]  
 for j in range(3, n - 1):  
 of = 2 \* np.pi \* frequencies[j - 3] \* t  
 tmp += a[j] \* sp.sin(of)  
 expression += tmp \*\* 2  
 return expression  
  
# Least squares method  
def custom\_lsq(y, x, frequencies):  
 *"""Fit a model to the data using the least squares method."""* n = 4 + len(frequencies)  
 a = generate\_a\_symbols(n)  
 expression = calculate\_square\_error(y, x, frequencies, a)  
 gradient = [sp.diff(expression, param) for param in a]  
 solution = sp.solve(gradient, a)  
 print("Found parameters:", solution)  
 return solution  
  
  
# Fitted model  
def fitted\_model(t, a, fund\_frequencies):  
 *"""Construct the fitted model using the found parameters."""* n = len(a)  
 model = a[0] \* t\*\*3 + a[1] \* t\*\*2 + a[2] \* t + a[n - 1]  
 for j in range(3, n - 1):  
 model += a[j] \* np.sin(2 \* np.pi \* fund\_frequencies[j - 3] \* t)  
 return model  
  
# Signal spectre visualization  
def plot\_signal\_spectre(Y):  
 *"""Plot the signal spectre for given transformation."""* plt.figure(figsize=(10, 6))  
 plt.plot(np.arange(N // 2) / T, np.abs(Y[:N // 2]), label='Signal spectre')  
 plt.title('Signal spectre')  
 plt.xlabel('Frequency (Hz)')  
 plt.ylabel('Amplitude')  
 plt.show()  
  
# Visualization functions  
def plot\_signal(t, signal, title, xlabel, ylabel, color='blue', label=None):  
 *"""Plot a signal with specified properties."""* plt.figure(figsize=(10, 6))  
 plt.plot(t, signal, label=label, color=color, linestyle='-', marker='o', markersize=2)  
 plt.title(title)  
 plt.xlabel(xlabel)  
 plt.ylabel(ylabel)  
 if label:  
 plt.legend()  
 plt.grid()  
 plt.show()  
  
def plot\_comparison(t, signal1, signal2, label1, label2):  
 *"""Plot a comparison between two signals."""* plt.figure(figsize=(10, 6))  
 plt.plot(t, signal1, label=label1, color='blue', linestyle='-', marker='o', markersize=2)  
 plt.plot(t, signal2, label=label2, color='red', linestyle='-')  
 plt.title('Comparison of Model and Input Data')  
 plt.xlabel('Time')  
 plt.ylabel('Amplitude')  
 plt.legend()  
 plt.grid()  
 plt.show()  
  
# Main execution flow  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Load data  
 y = load\_data('f4.txt')  
  
 # Compute DFT  
 Y = dft(y)  
  
 # Find significant frequencies  
 significant\_freq = find\_significant\_frequencies(Y)  
 print("Significant frequencies:", significant\_freq)  
  
 # Plot signal spectre  
 plot\_signal\_spectre(Y)  
  
 # Call least squares method to find model parameters  
 solution = custom\_lsq(y, t, significant\_freq)  
  
 # Extract parameter values from the solution  
 a\_values = [solution[param] for param in solution]  
  
 # Calculate square error  
 square\_error = calculate\_square\_error(y, t, significant\_freq, a\_values)  
 print("Square error:", square\_error)  
  
 # Generate fitted values  
 y\_fitted = fitted\_model(t, a\_values, significant\_freq)  
  
 # Plot the input data and fitted model  
 plot\_signal(t, y, 'Input Data', 'Time', 'Amplitude')  
 plot\_signal(t, y\_fitted, 'Fitted Model', 'Time', 'Amplitude', 'red')  
 plot\_comparison(t, y, y\_fitted, 'Input Data', 'Fitted Model')