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

$$y(t) = a_1 t^3 + a_2 t^2 + a_3 t + \sum_{i=4}^{k} a_i \, sin(\, 2\pi f_{i-3} t) + a_{k+1}$$

На вході маємо значення деякої функції (сигналу) y з файлу «f4.txt»”.

Значення є дискретними, крок аргументу функції $\Delta t = 0.01$

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

Хід роботи

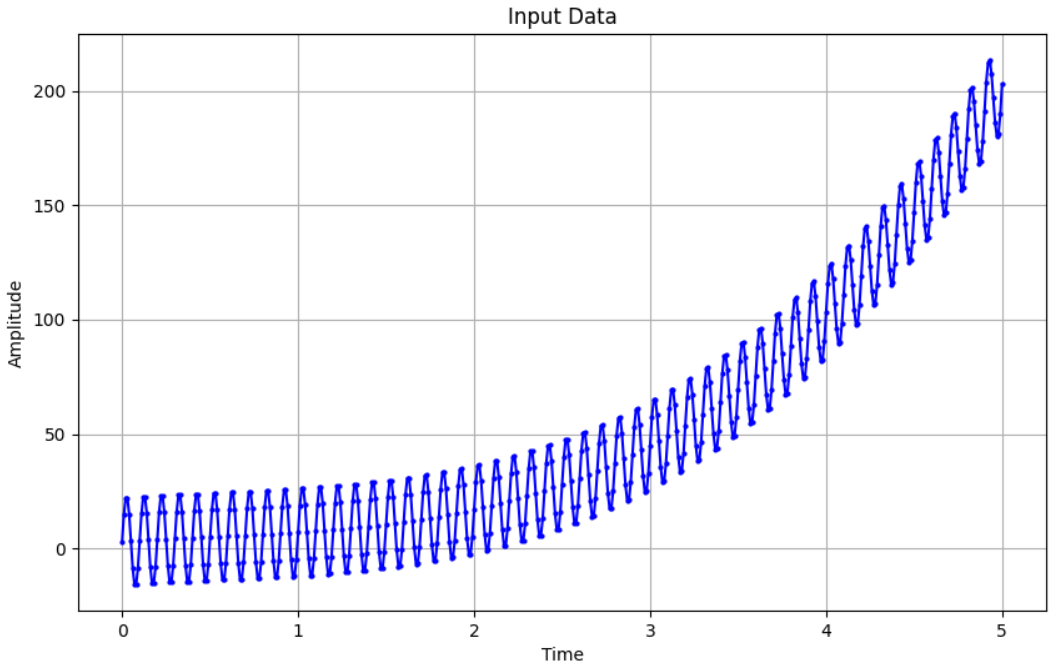
Завантажуємо дані з файлу:

```
# 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:

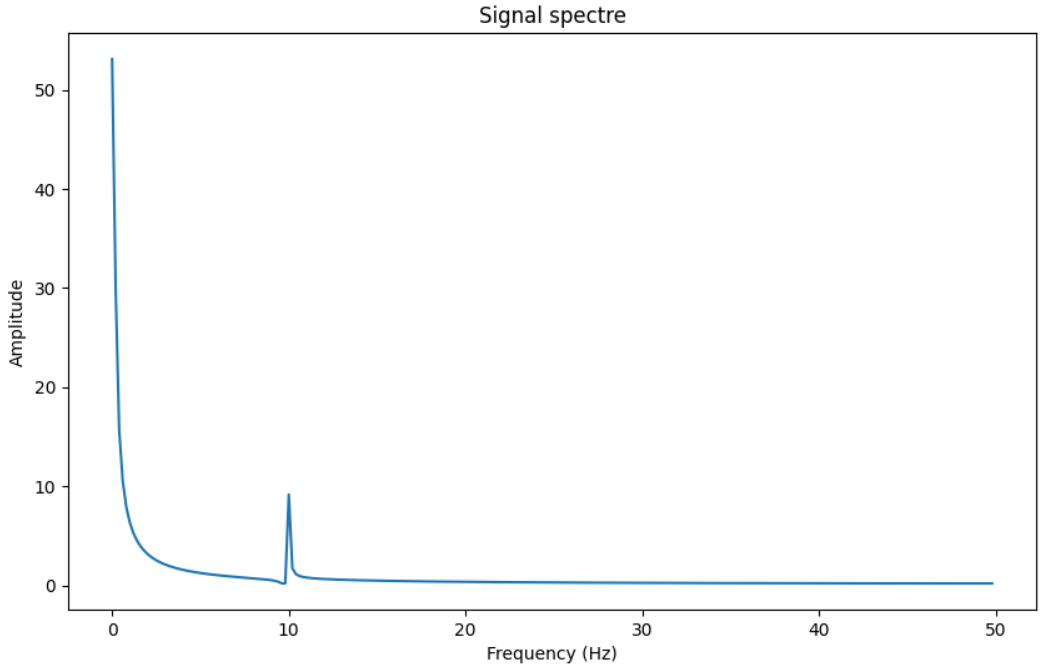


Дізнаємось, які значні частоти наявні в сигналі. Для цього виконуємо дискретне перетворення Фур'є для отриманих значень:

$$c_x(k) = \frac{1}{N}\sum_{m=0}^{N-1} x(m)\, e^{-i2\pi km/N}$$

```
# 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 $f_* = k_* \Delta f$, де k - локальні максимуми перетворення Фур'є, $\Delta f = \frac{1}{T}$:

```
# 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:

$$y(t) = a_1 t^3 + a_2 t^2 + a_3 t + a_4 \sin(20\pi t) + a_5$$

Отримуємо модель, яка є лінійною відносно невідомих параметрів a_i.

Використаємо метод найменших квадратів для оцінки цих параметрів.

$$S(a_1, a_2, a_3, a_4, a_5) = \sum_{i=1}^{501} (y_i - y(t_i)^2$$

$$S \rightarrow min$$

Для цього знайдемо похідні функціонала S за кожним з параметрів a_i 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:

$$A = \begin{pmatrix} 2247799.1 & 523963.5 & 125625.8 & -384.7 & 31375.1 \\ 523963.5 & 125625.8 & 31375.1 & -76.9 & 8358.4 \\ 125625.8 & 31375.1 & 8358.4 & -15.4 & 2505.0 \\ -384.7 & -76.9 & -15.4 & 500.0 & -5.1e-14 \\ 31375.1 & 8358.4 & 2505.0 & -5.1 & 1002.0 \end{pmatrix}$$

$$b = \begin{pmatrix} 3638268.5 \\ 851461.4 \\ 206125.3 \\ 9384.5 \\ 53206.2 \end{pmatrix}$$

$$Aa = b$$

Розв'язуємо отриману систему та отримуємо наступну оцінку параметрів:

```
Found parameters: {a0: 1.99999946277886, a1: -2.99999603336482, a2: 4.99999262793685, a3: 19.9999961336770, a4: 3.00000322143062}
```

Підставляємо отримані параметри у модель:

$$y(t) = 2t^3 - 3t^2 + 5t + 20\sin(20\pi t) + 3$$

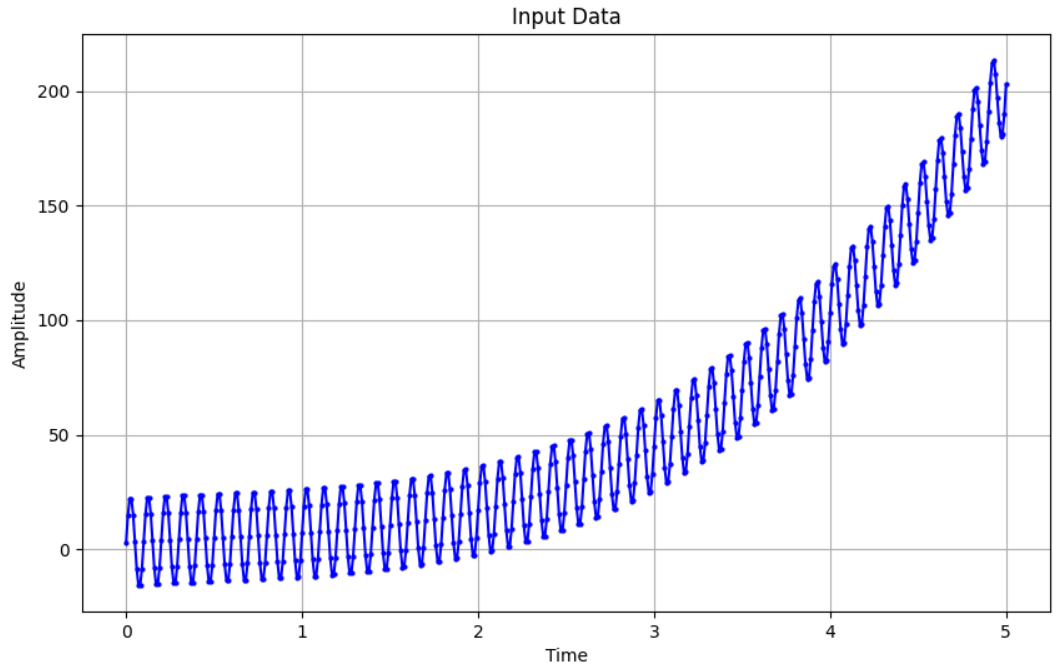
Обчислимо квадратичну похибку:

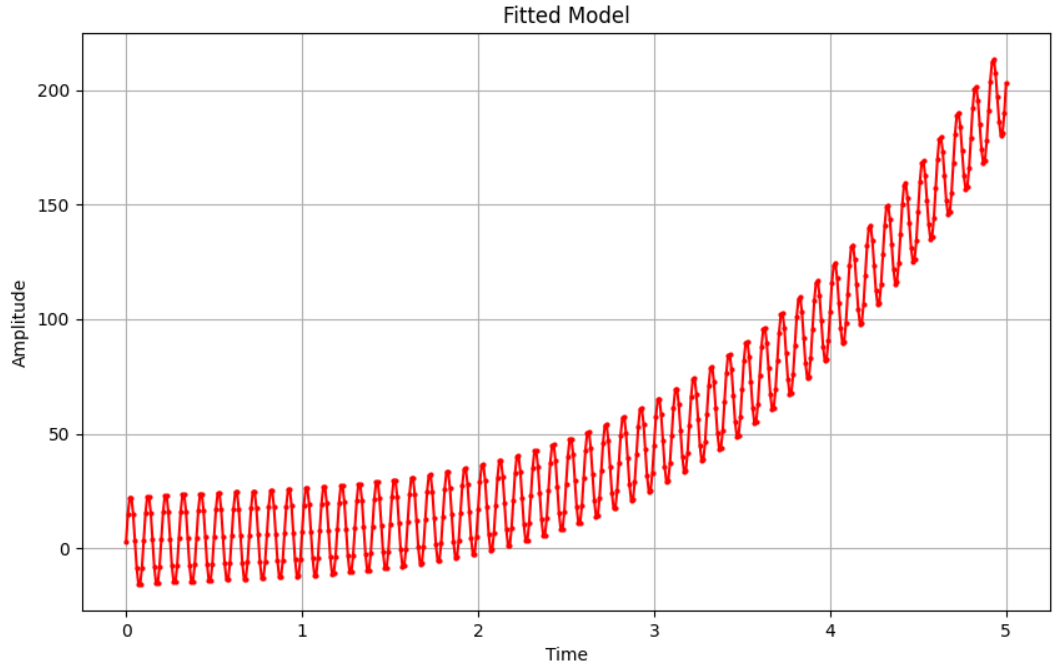
```python
# Calculate square error
def calculate_square_error(y, x, frequencies, a):
    """Calculate the square error between the model and the observed data."""
    n = len(a)
    expression = 0
    for i in range(len(y)):
        t = x[i]
        tmp = a[0] * t ** 3 + a[1] * t ** 2 + a[2] * t + a[n - 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:

```
Square error: 4.502554073260 66e-7
```

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:

$$(t) = 2t^3 - 3t^2 + 5t + 20\sin(20\pi t) + 3$$

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')
```