

Київський національний університет імені Тараса Шевченка  
Факультет комп'ютерних наук і кібернетики

Звіт  
з лабораторної роботи №1  
з моделювання складних систем  
Варіант 4

Виконав:  
Студент групи ІПС-31  
Міцкевич Костянтин Олександрович

Київ  
2024

# Мета і завдання роботи

## Мета:

- Ознайомитися з дискретним перетворенням Фур'є та методом найменших квадратів.
- Визначити суттєві частоти коливань у дискретних спостереженнях.
- Побудувати математичну модель спостережуваної функції та знайти її параметри.

## Завдання:

- Проаналізувати дискретне перетворення Фур'є та його властивості.
- Визначити суттєві внески частот за спостереженнями.
- Побудувати програму для обчислення модуля перетворення Фур'є та його локальних максимумів.
- Визначити параметри моделі через метод найменших квадратів.
- Побудувати графіки спостережень та моделі, порівняти їх.

## Модель функції:

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

# Теоретична частина

**Дискретне перетворення Фур'є** - це математична процедура, що використовується для визначення гармонічного, або частотного, складу дискретних сигналів.

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

## Властивості ДПФ:

1) Симетрія:  $x(N-m) = \sum_{n=0}^{N-1} x(n) * e^{-i2\pi nm/N}$ , тобто амплітуди гармонік у другій половині спектра повторюють першу половину, але можуть мати протилежну фазу.

2) Лінійність, тобто: вхідна послідовність  $x_1(n)$  має ДПФ  $X_1(m)$ , а інша вхідна послідовність  $x_2(n)$  має ДПФ  $X_2(m)$ , то ДПФ суми цих послідовностей  $x_{\text{sum}}(n) = x_1(n) + x_2(n)$  рівна:  $X_{\text{sum}}(m) = X_1(m) + X_2(m)$

3) Зсув у часі, тобто, якщо сигнал зсунутий по часу на певну кількість відліків, його спектр змінюється лише фазово, а амплітуди залишаються ті ж.

## Модуль та фазовий спектр:

Модуль спектра визначає амплітуду гармоніки частоти  $f_k$ :

$$|c_x(k)| = \sqrt{\Re(c_x(k))^2 + \Im(c_x(k))^2}$$

Фазовий спектр показує фазовий зсув гармоніки частоти  $f_k$ :

$$\phi_x(k) = \arctan \frac{\Im(c_x(k))}{\Re(c_x(k))}$$

### **Постановка задачі прихованої періодичності:**

Для аналізу сигналу задається інтервал спостереження  $[0, T]$ , де  $T = 5$ . Спостереження здійснюються у дискретні моменти часу з кроком  $\Delta t = 0.01$ . Метою є визначення суттєвих частот сигналу за допомогою дискретного перетворення Фур'є. Для цього спочатку обчислюється модуль спектра  $|C_y(k)|$ , а потім знаходяться його локальні максимуми. Частоти, що відповідають цим максимумам, визначають основні гармонійні складові сигналу, тобто суттєві внески у його періодичність.

### **Метод найменших квадратів**

Метод найменших квадратів використовується для визначення параметрів математичної моделі сигналу шляхом мінімізації різниці між спостережуваними даними та значеннями моделі. Функціонал похибки визначається як:

$$F(a_1, a_2, \dots, a_{k+1}) = \frac{1}{2} \sum_{j=0}^{N-1} \left( a_1 t_j^3 + a_2 t_j^2 + a_3 t_j + \sum_{i=4}^k a_i \sin(2\pi f_{i-3} t_j) + a_{k+1} - \hat{y}(t_j) \right)^2$$

Для знаходження параметрів  $a_i$ ,  $i = 1, 2, \dots, k+1$  необхідно розв'язати систему рівнянь:

$$\frac{\partial F(a_1, a_2, \dots, a_{k+1})}{\partial a_j} = 0$$

Таким чином, метод мінімізує суму квадратів відхилень моделі від спостережуваних даних і дозволяє побудувати найкращу апроксимацію сигналу.

Тобто це дозволяє знайти параметри математичної моделі так, щоб модель максимально точно відтворювала спостережувані дані. Для цього обчислюється різниця між значеннями моделі та реальними спостереженнями в усіх точках часу, підноситься до квадрата і сумується. Потім ці параметри вибираються так, щоб ця сума була мінімальною.

# Хід роботи

## 1) Завантаження спостережень з файлу f4.txt та ініціалізація всіх параметрів:

```
def read_from_file(filename: str = 'f4.txt') -> np.ndarray | None:
    # This function reads data from a file and returns it as a NumPy array.
    try:
        data = np.loadtxt(filename)
        return data
    except FileNotFoundError:
        print(f"File with name '{filename}' not found.")
        return None
```

```
def main():
    data = read_from_file()
    dt = 0.01

    if data is not None:
        N = len(data)
        t = np.arange(0, N*dt, dt)
```

## 2) Обчислення дискретного перетворення Фур'є для всіх точок $k = 0, \dots, N-1$ за допомогою власної DFT функції та перевірка коректності

```
def DFT(signal: np.ndarray) -> np.ndarray:
    # Computes the Discrete Fourier Transform (DFT) of a given signal.
    signal = np.asarray(signal, dtype=float)
    N = signal.shape[0]
    c_x = np.zeros(N, dtype=complex)

    for k in range(N):
        c_x[k] = np.sum(signal * np.exp(-2j * np.pi * k * np.arange(N) / N))

    return c_x

def test_dft_implementation(samples: np.ndarray) -> None:
    # Tests whether the custom DFT implementation matches NumPy's FFT.
    my_dft_result = DFT(samples)
    numpy_fft_result = np.fft.fft(samples)
    max_abs_error = np.max(np.abs(my_dft_result - numpy_fft_result))

    if max_abs_error < 1e-6:
        print("DFT implementation matches NumPy's FFT.")
    else:
        print("DFT implementation does not match NumPy's FFT.")
```

Результат:

```
Checking DFT against np.fft.fft:  
DFT implementation matches NumPy's FFT.
```

### 3) Аналіз спектра та визначення суттєвих частот. Побудова графіка.

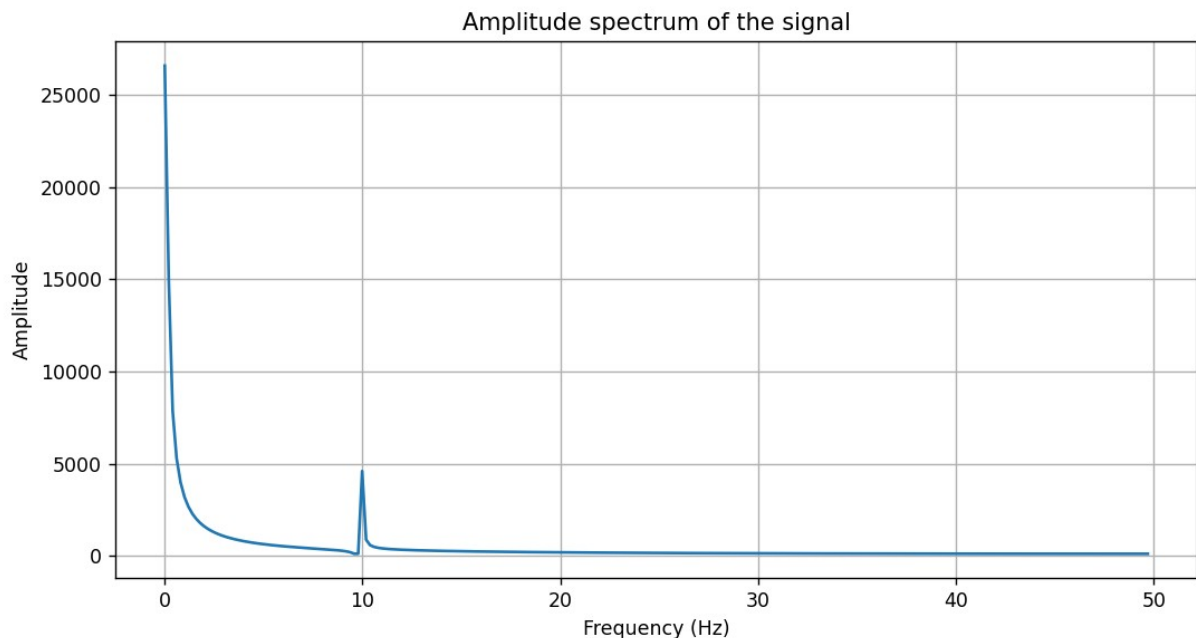
```
amplitude, significant_freqs = analyze_DFS(c_x, dt)  
print("\nAmplitude spectrum (first 10 values):", amplitude[:10])  
print("\nDetected significant frequencies:", significant_freqs)  
  
plot_amplitude_spectrum(amplitude, dt)  
  
def analyze_DFS(dft: np.ndarray, dt: float) -> tuple[np.ndarray, np.ndarray]:  
    # Analyzes the spectrum: amplitude and frequency.  
    amplitude = np.abs(dft)  
    frequencies = np.fft.fftfreq(len(dft), dt)  
    peaks = find_peaks(amplitude)  
    significant_frequencies = frequencies[peaks]  
    return amplitude, significant_frequencies  
  
def find_peaks(amplitude: np.ndarray, threshold_ratio: float = 0.1) -> np.ndarray:  
    # Finding local amplitude maxima with a threshold.  
    N = len(amplitude)  
    half_N = N // 2  
    amp = amplitude[:half_N]  
    threshold = threshold_ratio * np.max(amp)  
  
    peaks = [i for i in range(1, half_N - 1)  
             if amp[i] > amp[i - 1] and amp[i] > amp[i + 1] and amp[i] >= threshold]  
  
    return np.array(peaks)
```

```
def plot_amplitude_spectrum(amplitude, dt):  
    # A 'Amplitude spectrum of the signal' plotting function  
    N = len(amplitude)  
    frequencies = np.fft.fftfreq(N, dt)  
    half_N = N // 2  
    plt.figure(figsize=(10,5))  
    plt.plot(frequencies[:half_N], amplitude[:half_N])  
    plt.xlabel('Frequency (Hz)')  
    plt.ylabel('Amplitude')  
    plt.title('Amplitude spectrum of the signal')  
    plt.grid(True)  
    plt.show()
```

Результат виконання цього блоку:

```
Amplitude spectrum (first 10 values): [26603.10053 15039.03431652 7854.12997539 5287.99966126
3979.82565079 3188.72617765 2659.19270358 2280.03415783
1995.19358789 1773.37279744]

Detected significant frequencies: [9.98003992]
```



Маємо єдиний значущий вклад частоти 9.98 Гц (близьке до 0 значення ігноруємо, так як це поліноміальний вклад).

#### 4) Визначення невідомих параметрів методом найменших квадратів:

```
params, model_values = fit_model(t, data, significant_freqs)
print("\nFitted model parameters:", params)
```

Результат виконання цього блоку:

```
( Fitted model parameters: [ 2.21213162 -4.8129402  9.51750308 19.67630199  6.59672866]
□ # Fits a signal model (cubic polynomial + harmonics) to the observed data using least squares
def model(t, a1, a2, a3, *params):
    # Inner function to build the signal model with a cubic polynomial and harmonics
    y = a1*t**3 + a2*t**2 + a3*t
    k = len(peak_frequencies)
    for i in range(k):
        ai = params[i]
        phi_i = params[k + i]
        fi = peak_frequencies[i]
        y += ai * np.sin(2*np.pi*fi*t + phi_i)
    return y

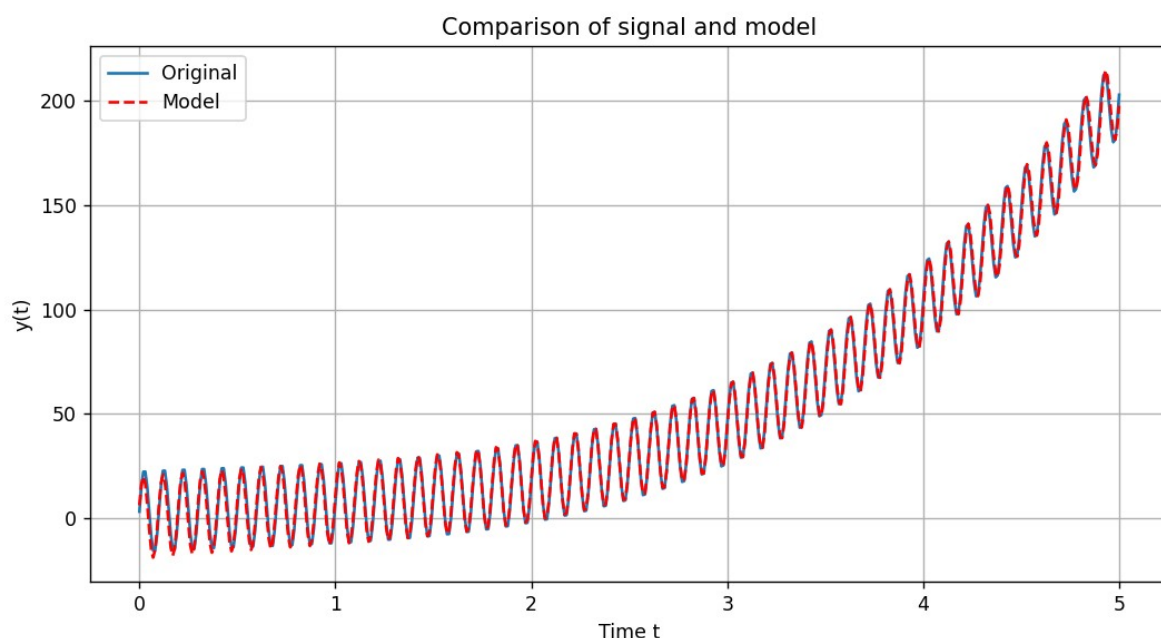
k = len(peak_frequencies)
initial_guess = [1, 1, 1] + [1]*k + [0]*k
params, _ = curve_fit(lambda t, *p: model(t, *p), t, observations, p0=initial_guess)
return params, model(t, *params)
```

**5) Побудова оригінального графіку та графіку моделі:**

```
plot_signal_vs_model(t, data, model_values)
```

```
def plot_signal_vs_model(t, observations, model_values):
    # A 'signal vs model' plotting function
    plt.figure(figsize=(10,5))
    plt.plot(t, observations, label='Original')
    plt.plot(t, model_values, '--', color='red', label='Model')
    plt.xlabel('Time t')
    plt.ylabel('y(t)')
    plt.title('Comparison of signal and model')
    plt.legend()
    plt.grid(True)
    plt.show()
```

Результат виконання цього блоку:



## Висновок

Було реалізовано дискретне перетворення Фур'є (DFT) та перевірено його точність порівняно з NumPy; результати повністю збігаються. Аналіз амплітудного спектра показав, що суттєва частота сигналу становить приблизно 9.98 Гц. Підгонка моделі сигналу за допомогою методу найменших

квадратів дозволила знайти параметри кубічного полінома та гармоніки, які добре відтворюють спостережувані дані.