Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Моделювання складних систем

Лабораторна робота №1

“Дискретне перетворення Фур’є”

Варіант №18

Виконала студентка групи ІПС-31

Сенечко Дана Володимирівна

Київ - 2025

Постановка задачі та основна теорія

Нехай задано дискретну послідовність спостережень $\widehat{y}(t_j)$, $j = 0, 1, \ldots, N-1$, що отримана на інтервалі часу $[0, T]$, $T = 5$, де крок дискретизації дорівнює $\Delta t = t_{i+1} - t_i = 0.01$. Таким чином, ми маємо $N = T/\Delta t = 500$ значень, що представляють поведінку деякого сигналу у часі.

Мета роботи полягає у виявленні суттєвих частотних складових сигналу, тобто визначенні тих гармонік, які роблять найбільший внесок у його формування. Це завдання відоме як задача про приховану періодичність.

Для цього застосовується дискретне перетворення Фур'є (ДПФ), яке дозволяє перейти від часової області (сигнал як функція часу) до частотної області (розклад сигналу на гармонічні складові).

Дискретне перетворення Фур'є для дискретної послідовності $x(m)$, $m = 0, 1, \ldots, N-1$ визначається формулою:

$c_x(k) = \frac{1}{N} \sum_{m=0}^{N-1} x(m) e^{-i2\pi km/N}$, де $i^2 = -1$ – уявна (комплексна) одиниця.

Отримані коефіцієнти $c_x(k)$ є комплексними числами, що характеризують амплітуду та фазу відповідної гармонічної складової. На практиці для аналізу використовують модуль коефіцієнтів $\left|c_x(k)\right|$,

оскільки саме він показує величину внеску відповідної частоти у формування сигналу. Оскільки сигнал є дійсним, спектр $\left|c_x(k)\right|$ має властивість симетрії, тому достатньо розглядати лише половину значень:

$k = 0, 1, \ldots, N/2$.

Крок частоти визначається як $\Delta f = 1/T = 1/5 = 0.2$ Гц. Таким чином, кожному індексу k відповідає певна частота $f_k = k \cdot \Delta f$.

Для знаходження найбільш значущих частотних складових визначаються локальні максимуми у спектрі $\left|c_x(k)\right|$. Відповідні значення частот $f_k^* = k^* \cdot \Delta f$ і будуть тими гармоніками, які складають основу сигналу.

На завершальному етапі будується математична модель виду $y(t) = a_1 t^3 + a_2 t^2 + a_3 t + \sum_{i=4}^{k} a_i sin(2\pi f_{i-3} t) + a_{k+1}$, де a_1, a_2, a_3 – коефіцієнти поліноміальної частини, a_i – амплітуди гармонічних складових, а f_i – знайдені суттєві частоти.

Хід роботи

Програма реалізована мовою Python з використанням бібліотек NumPy, Matplotlib та SciPy. Умовно поділимо її на кілька етапів:

1) Реалізація власної функції ДПФ та перевірка її правильності.

Спочатку була написана функція `manualDFT`, яка обчислює ДПФ за означенням. Для її перевірки використовується функція `testDFT`, яка порівнює результати з вбудованою функцією `numpy.fft.fft`.

```
def manualDFT(samples: np.ndarray) -> np.ndarray:
    N = len(samples)
    m = np.arange(N)
    k = m.reshape((N, 1))
    e = np.exp(-2j * np.pi * k * m / N)
    return np.dot(e, samples) / N
```

```python
def testDFT(samples: np.ndarray) -> None:
    customDFTres = manualDFT(samples)
    npFFTres = np.fft.fft(samples) / len(samples)

    maxAbsoluteError = np.max(np.abs(customDFTres - npFFTres))

    if(maxAbsoluteError < 1e-6):
        print("Custom DFT is ok")
    else:
        print("Oops! Custom DFT is incorrect")
        print(f"Max absolute error: {maxAbsoluteError}")
```

```python
observations = np.loadtxt("f18.txt")
testDFT(observations)
```

```
Custom DFT is ok
```

2) Аналіз амплітудного спектра та пошук локальних максимумів.

Для знаходження найбільш суттєвих частотних внесків було реалізовано функцію `findFreqContribution`. Вона обчислює модулі спектра та визначає локальні максимуми.

```python
def findFreqContribution(samples: np.ndarray, observInterval: int = 5):
    numSamples = len(samples)
    deltaF = 1 / observInterval
    modulesDFT = np.abs(manualDFT(samples)) # or np.fft.fft(samples)

    half = numSamples // 2
    localMaxIndices = []
    for k in range(1, half - 1):
        if modulesDFT[k] > modulesDFT[k-1] and modulesDFT[k] > modulesDFT[k+1]:
            localMaxIndices.append(k)

    plotDFTModules(
        modulesDFT,
        deltaF,
        'dft-modules.png',
        peakIndices = localMaxIndices
    )

    frequencies = []
    for k in localMaxIndices:
        freq = k * deltaF
        frequencies.append(freq)

    return frequencies
```

```python
peak_frequencies = findFreqContribution(observations)
print(peak_frequencies)
```

Отримуємо такий результат: `[25.0, 38.800000000000004]`

3) Побудова графіка спектра амплітуд ДПФ.

Спектр було зображено за допомогою функції `plotDFTModules`. На графіку відображаються значення амплітуд у частотній області, а локальні максимуми виділяються червоними точками, оскільки їх іноді не видно.

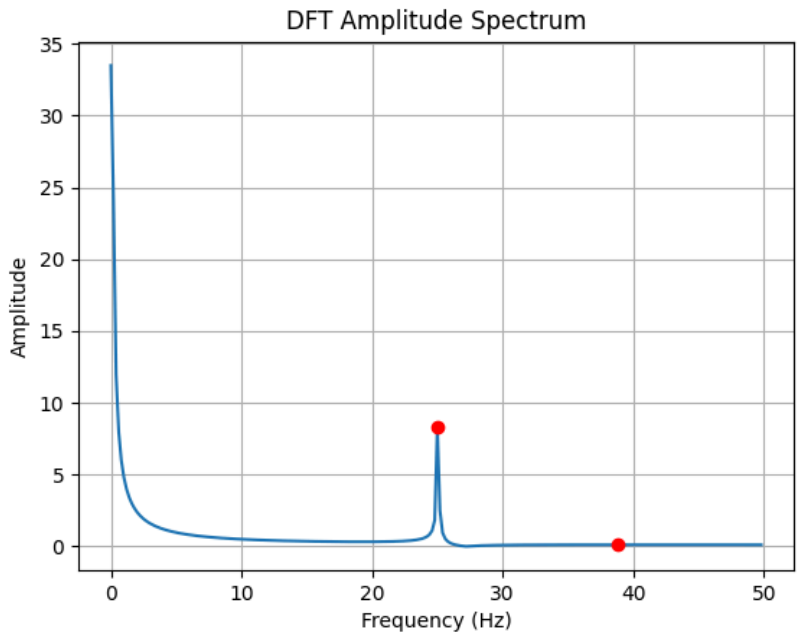
```python
def plotDFTModules(
        modulesDFT: np.ndarray,
        deltaF: float,
        saveImgPath: str,
        peakIndices = None
) -> None:
    numSamples = len(modulesDFT)
    halfModules = modulesDFT[:numSamples//2]
    frequencies = np.arange(len(halfModules)) * deltaF

    plt.plot(frequencies, halfModules)

    if peakIndices is not None:
        peakFrequencies = np.array(peakIndices) * deltaF
        peakAmplitudes = halfModules[peakIndices]
        plt.scatter(peakFrequencies, peakAmplitudes, color='red', zorder=5)

    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Amplitude")
    plt.grid(True)
    plt.title("DFT Amplitude Spectrum")
    plt.savefig(saveImgPath)
    plt.show()
```

Отримуємо такий графік:



4) Побудова математичної моделі сигналу.

```
def fit_model(t, observations, peak_frequencies):
    def model(t, a1, a2, a3, *params):
        k = len(params) // 2
        y = a1 * t**3 + a2 * t**2 + a3 * t
        for i in range(k):
            fi = params[i]
            ai = params[k + i]
            y += ai * np.sin(2 * np.pi * fi * t)
        return y

    # starting guess for parameters
    initial_guess = [0, 0, 0] + peak_frequencies + [1]*len(peak_frequencies)

    params, covariance = curve_fit(model, t, observations, p0=initial_guess)
    fitted_values = model(t, *params)
    return params, fitted_values
```

```
# time vector
t = np.arange(len(observations)) * 0.01  # delta_t = 0.01

# model fitting
params, fitted_values = fit_model(t, observations, peak_frequencies)
print("Found parameters:", params)
```

Отримуємо:

```
Found parameters: [ 1.00000060e+00  1.99999534e+00 -4.99998985e+00 -2.00000493e+00
  2.50000000e+01  3.86762147e+01 -1.80000049e+01 -4.33100353e-07]
```

Можемо підставити отриманні значення в формулу моделі:

$$y(t) = t^3 + 2t^2 - 5t - 2 - 18sin(2\pi * 25t) - 4.3 * 10^{-7}sin(2\pi * 38.8t)$$

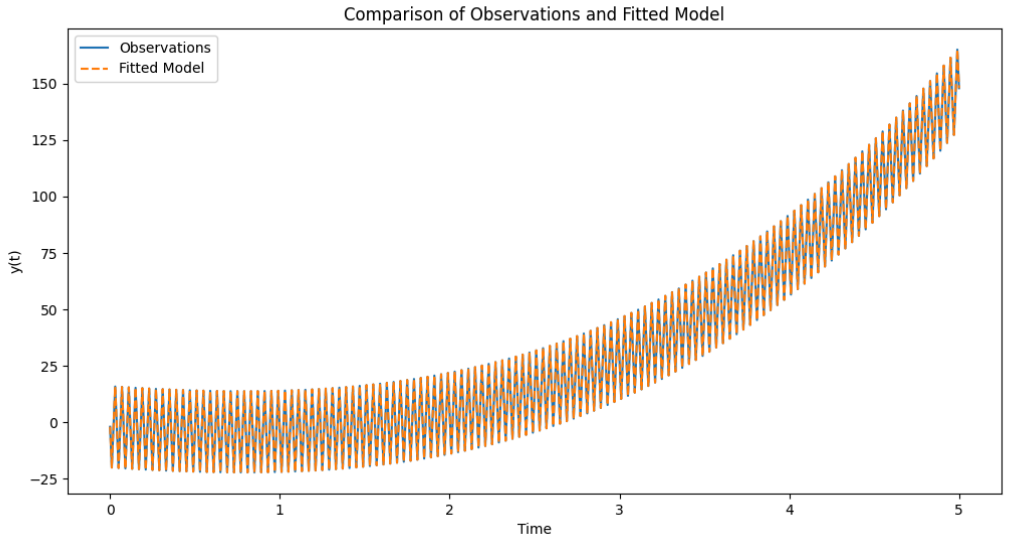
Другий синус практично нульовий, тому його вплив на модель можна вважати незначним.

5) Порівняння моделі з експериментальними даними.

Для перевірки адекватності моделі побудовано графік, на якому відображаються вихідні дані та апроксимуюча крива.

```
plt.figure(figsize=(12, 6))
plt.plot(t, observations, label='Observations')
plt.plot(t, fitted_values, label='Fitted Model', linestyle='--')
plt.legend()
plt.xlabel('Time')
plt.ylabel('y(t)')
plt.title('Comparison of Observations and Fitted Model')
plt.savefig("fitted_model_comparison.png")
plt.show()
```

Отриманий графік:



Графік показав, що відновлена модель добре описує спостереження: підібрані синусоїди й поліном відтворюють структуру сигналу з невеликою похибкою.

Додаткова перевірка:

```
#additional analysis
error_value = np.sum((fitted_values - observations) ** 2)
print(f"Sum of Squared Errors (SSE): {error_value}")
```

```
Sum of Squared Errors (SSE): 3.509159129894941e-07
```

Маємо досить малу похибку: $3.5 * 10^{-7}$, тому апроксимуюча функція добре відтворює спостереження.

Код та згенеровані графіки можна переглянути на моєму [GitHub](#).