Explanation of FFT and DFT Comparison Code

Henri Setyo Pambudi

Overview

This Python code compares the Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) using simulated oscilloscope data of electrical waveforms. The analysis includes signal generation, spectral analysis, and visualization. The objective is to illustrate the computational efficiency and equivalency of FFT and DFT in transforming time-domain signals to the frequency domain.

Code Explanation

1. Importing Libraries

The code starts by importing essential libraries:

- **NumPy**: For numerical computations and generating signals.
- **Matplotlib**: For visualizing signals and frequency spectra.
- **SciPy**: For optimized FFT computation.

Listing 1: Import Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import time
```

2. Signal Generation

A synthetic signal is generated using a combination of sine waves:

- Frequencies: 50 Hz and 120 Hz.
- Sampling Rate: 3000 Hz.
- Duration: 1 second.
- Noise: Random noise is added to simulate a real-world signal.

Listing 2: Signal Generation

```
sampling_rate = 3000
T = 1 / sampling_rate
duration = 1


t = np.linspace(0, duration, int(sampling_rate * duration),
    endpoint=False)


freq1 = 50
freq2 = 120


signal = 5 * np.sin(2 * np.pi * freq1 * t) + 3 * np.sin(2 *
    np.pi * freq2 * t) + np.random.normal(0, 0.5, t.shape)
```

3. DFT Implementation

The Discrete Fourier Transform (DFT) is implemented manually using the mathematical formula:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\frac{2\pi}{N}kn}$$

This function takes a time-domain signal and returns the DFT result.

Listing 3: DFT Implementation

```
def dft(x):
    N = len(x)
    X = np.zeros(N, dtype=complex)
    for k in range(N):
        for n in range(N):
            X[k] += x[n] * np.exp(-2j * np.pi * k * n / N)
    return X
```

4. FFT Implementation

The Fast Fourier Transform (FFT) is calculated using the optimized function from the `scipy.fft` library. The FFT is expected to provide a faster computation compared to the DFT, especially for large datasets.

Listing 4: FFT Implementation

```
fft_result = fft(signal)
```

5. Frequency Domain Visualization

The signal's frequency-domain representation is plotted using both DFT and FFT results. This allows for comparison of the amplitude spectrum for both transforms.

Listing 5: Plotting Frequency Spectra

```
frequencies = np.fft.fftfreq(len(signal), T)


fft_magnitude = np.abs(fft_result)
dft_result = dft(signal)
dft_magnitude = np.abs(dft_result)


plt.figure(figsize=(12, 8))


plt.subplot(3, 1, 1)
plt.plot(t, signal, label='Time-Domain Signal', color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Oscilloscope Signal')
plt.legend()
plt.grid()


plt.subplot(3, 1, 2)
plt.stem(frequencies[:len(frequencies) // 2], fft_magnitude[:
    len(frequencies) // 2], label='FFT', linefmt='r-',
    markerfmt='ro', basefmt='r-')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title('FFT of the Signal')
plt.legend()
plt.grid()


plt.subplot(3, 1, 3)
plt.stem(frequencies[:len(frequencies) // 2], dft_magnitude[:
   len(frequencies) // 2], label='DFT', linefmt='g-',
   markerfmt='go', basefmt='g-')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title('DFT of the Signal')
plt.legend()
plt.grid()


plt.tight_layout()
plt.show()
```

6. Full Code

Below is the complete code that generates a signal, computes both the FFT and DFT, and visualizes the results in both time and frequency domains.

Listing 6: Full Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft


sampling_rate = 3000
T = 1 / sampling_rate
duration = 1
t = np.linspace(0, duration, int(sampling_rate * duration),
   endpoint=False)  # Time vector
freq1 = 50
freq2 = 120
signal = 5 * np.sin(2 * np.pi * freq1 * t) + 3 * np.sin(2 *
   np.pi * freq2 * t) + np.random.normal(0, 0.5, t.shape)

def dft(x):
    N = len(x)
    X = np.zeros(N, dtype=complex)
    for k in range(N):
        for n in range(N):
            X[k] += x[n] * np.exp(-2j * np.pi * k * n / N)
    return X
```

```python
fft_result = fft(signal)
dft_result = dft(signal)


frequencies = np.fft.fftfreq(len(signal), T)


fft_magnitude = np.abs(fft_result)
dft_magnitude = np.abs(dft_result)


plt.figure(figsize=(12, 8))
plt.subplot(3, 1, 1)
plt.plot(t, signal, label='Time-Domain Signal', color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Oscilloscope Signal')
plt.legend()
plt.grid()


plt.subplot(3, 1, 2)
plt.stem(frequencies[:len(frequencies) // 2], fft_magnitude[:
    len(frequencies) // 2], label='FFT', linefmt='r-',
    markerfmt='ro', basefmt='r-')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title('FFT of the Signal')
plt.legend()
plt.grid()


plt.subplot(3, 1, 3)
plt.stem(frequencies[:len(frequencies) // 2], dft_magnitude[:
    len(frequencies) // 2], label='DFT', linefmt='g-',
    markerfmt='go', basefmt='g-')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title('DFT of the Signal')
plt.legend()
plt.grid()

plt.tight_layout()
plt.show()
```

Key Insights

- **Equivalence of Results**: Both the FFT and DFT give identical frequency-domain representations of the signal.
- **Computational Efficiency**: FFT is significantly faster than DFT, especially for large datasets.
- **Applications**: These methods are foundational in signal processing tasks, such as filtering, spectral analysis, and feature extraction.