Introduction to Signal Processing with FFT

Jupyter Notebook Exercises in Python

Prepared by Mr. Heystek Grobler

1 Introduction

The Fast Fourier Transform (FFT) is a powerful tool in signal processing. It allows us to analyze signals not only in the *time domain* but also in the *frequency domain*.

In this document, we explore:

- How to generate signals
- How to apply the FFT
- How to interpret frequency spectra
- The effects of noise
- Simple frequency-domain filtering

Each section includes exercises and example Python code.

2 Generating a Simple Signal

We begin with a signal composed of two sine waves at 5 Hz and 20 Hz.

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

Fs = 200      # Sampling frequency
T = 1/Fs
t = np.arange(0, 1, T)

f1, f2 = 5, 20
signal = np.sin(2*np.pi*f1*t) + 0.5*np.sin(2*np.pi*f2*t)

plt.plot(t, signal)
plt.title("Time-domain Signal")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()
```

3 Applying the FFT

The FFT transforms the signal into the frequency domain.

```
n = len(signal)
fft_result = np.fft.fft(signal)
fft_freq = np.fft.fftfreq(n, T)

mask = fft_freq >= 0
fft_result = np.abs(fft_result[mask]) * (2/n)
fft_freq = fft_freq[mask]

plt.stem(fft_freq, fft_result, basefmt=" ")
plt.title("Frequency Spectrum")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")
plt.grid(True)
plt.show()
```

3.1 Observation

You should see peaks at 5 Hz and 20 Hz.

Exercise 1

Task: Add a sine wave at 50 Hz with amplitude 0.3. **Expected result:** The frequency spectrum should now show three peaks at 5 Hz, 20 Hz, and 50 Hz.

```
#
#
#
#
#
#
```

4 Noise and the FFT

Real-world signals often contain noise.

```
noisy_signal = signal + 0.5*np.random.randn(len(t))

fft_noisy = np.fft.fft(noisy_signal)
fft_noisy = np.abs(fft_noisy[mask]) * (2/n)

plt.stem(fft_freq, fft_noisy, basefmt=" ")
plt.title("Noisy Frequency Spectrum")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")
plt.grid(True)
plt.show()
```

Exercise 2

Task: Try different noise strengths (0.2, 0.5, 1.0). **Expected result:** Peaks at 5 Hz and 20 Hz become harder to distinguish under heavy noise, but they are still visible.

5 Filtering

We can filter out high-frequency noise by removing FFT components above 30 Hz.

```
fft_noisy = np.fft.fft(noisy_signal)
freqs = np.fft.fftfreq(n, T)

cutoff = 30
mask = np.abs(freqs) <= cutoff
fft_filtered = fft_noisy * mask

reconstructed = np.fft.ifft(fft_filtered).real

plt.plot(t, noisy_signal, label="Noisy")
plt.plot(t, reconstructed, label="Filtered")
plt.legend()
plt.title("Signal Filtering with FFT")
plt.show()
```

Exercise 3

Task: Reconstruct the signal using inverse FFT after filtering out frequencies above 30 Hz. **Expected result:** The reconstructed signal should closely resemble the original clean signal.

5.1 Observation

Filtering removes high-frequency noise but may introduce small artifacts (ringing). This is common with ideal FFT filters.

6 Summary

- The FFT converts signals from time domain to frequency domain.
- Peaks in the spectrum reveal which frequencies are present.
- Noise spreads across the frequency spectrum.
- Filtering in the frequency domain can recover useful signals.

7 Radio Astronomy Example

Radio telescopes detect extremely weak signals from space, often buried under thermal noise. One classic task is to identify narrow-band emission lines (e.g., the neutral hydrogen line at 1420 MHz) or periodic pulsar signals.

7.1 Simulated Observation

We simulate an observation where:

- The sampling frequency is 1000 Hz (simplified).
- The telescope is observing for 1 second.
- A weak sinusoidal signal at 60 Hz represents the astronomical source.
- Gaussian noise represents the receiver and sky noise.

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

Fs = 1000   # sampling frequency (Hz)
T = 1/Fs
t = np.arange(0, 1, T)
n = len(t)

# Simulated astronomical signal: weak sinusoid at 60 Hz
astro_freq = 60
astro_amp = 0.05  # very weak amplitude
astro_signal = astro_amp * np.sin(2*np.pi*astro_freq*t)

# Strong Gaussian noise
noise = np.random.randn(n)

# Total observed signal
observed = astro_signal + noise

plt.plot(t[:200], observed[:200])
plt.title("Simulated Radio Telescope Time Series (first 200
   samples)")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()
```

7.2 Frequency Analysis with FFT

In the time series the signal is invisible. The FFT can reveal the hidden tone.

```python
fft_result = np.fft.fft(observed)
freqs = np.fft.fftfreq(n, T)

mask = freqs >= 0
fft_mag = np.abs(fft_result[mask]) * (2/n)
fft_freqs = freqs[mask]

plt.stem(fft_freqs, fft_mag, basefmt=" ")
plt.xlim(0, 200)   # zoom into first 200 Hz
plt.title("Frequency Spectrum of Radio Observation")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")
plt.grid(True)
plt.show()
```

7.3 Observation

Although the signal amplitude is only 0.05 compared to unit-variance noise, a clear spectral peak emerges at 60 Hz. This is how astronomers detect hidden periodicities and spectral lines in noisy data.

Exercise 4

1. Try reducing the amplitude to 0.02 or 0.01. How low can the signal be before you can no longer see it?
2. Increase the integration time (observe for 2 seconds instead of 1). How does this affect your ability to detect the peak?
3. Replace the sinusoid with two nearby tones (e.g., 60 Hz and 65 Hz). Can you resolve them both?

7.4 Key Concept

Radio astronomy often relies on the fact that **integrating longer (more samples)** reduces noise variance while the signal peak remains. This is why telescopes observe for minutes or hours: the FFT spectrum gradually reveals hidden signals.