📡 Chapter - Frequency-Domain Analysis of Captured IQ Data

🔎 What We're Going to Do

When we capture I/Q samples from the RTL-SDR, we're essentially freezing the **raw RF baseband signal** (complex-valued time series). Time-domain plots are useful to check if data is valid, but most of the real insights come from transforming into the **frequency domain** using the **Fourier Transform (FFT)**.

Our goals in this step:

1. **Convert IQ samples → Spectrum**
   - Use FFT to transform from time domain → frequency domain.
   - This tells us *which frequencies are present* and *their relative power*.
2. **Plot the Power Spectrum (FFT magnitude)**
   - x-axis: frequency bins (Hz)
   - y-axis: power (dB)
3. **Plot the Spectrogram (Waterfall)**
   - Spectrum over time (sliding FFTs).
   - Useful for seeing bursts, frequency hopping, or temporal activity.
4. **Extract Key Insights**
   - Where is the signal centered?
   - What's the occupied bandwidth?
   - Are there bursts or hopping activity?

🛠 The Method

1. **Read IQ Data**
   - We'll start from your `.bin` capture (e.g., `Prac_1.bin`).
   - Reshape into I/Q pairs (complex signal).
2. **Compute FFT**
   - Choose FFT size (`NFFT`), typically power of 2 like 1024, 2048, 4096.

- Apply a window (Hann) to reduce spectral leakage.
- Use `fftshift` so that 0 Hz is in the center of the plot.

3. **Convert FFT Magnitude to dB**
   - `20 * log10(|FFT|)`
   - This gives us the familiar spectrum view.
4. **Spectrogram**
   - Split samples into overlapping windows.
   - Compute FFT per window.
   - Stack results → gives time vs frequency power view.

First the Idea:

Imagine your IQ samples are like a song recorded in a tape.
You hear the sound in time — but you don't know which notes (frequencies) are being played strongly or softly.

☞ ### That's where the FFT comes in.

The FFT is like a magic prism: it takes your time-domain IQ samples (the messy wave) and splits them into frequency pieces.
Time domain → tells you what the wave looks like over time.
Frequency domain → tells you which frequencies are present and how strong they are.
That's the whole point:
Convert IQ samples → FFT → see the spectrum.
Capture samples (IQ = I + jQ, complex).
Run FFT (Fast Fourier Transform).
FFT is your spectrum analyzer in Python.
So, as Step 1: What are we trying to do?
We have IQ samples captured from the RTL-SDR.
In time-domain, IQ samples just look like noisy squiggles.
To know what frequencies are present, we need to look in the frequency domain.
That's where the FFT (Fast Fourier Transform) comes in → it's like turning a messy sound waveform into a "spectrum analyzer."

☞ ### So the job is simple:

IQ samples (time-domain) → FFT → Spectrum (frequency vs power).
Note:-
IQ data is stored as complex numbers:
I (In-phase) → real part
Q (Quadrature) → imaginary part
Together they form a complex number: I + jQ.
That's why we don't use float here → floats can't carry both I & Q at once.
complex64 = 32-bit real + 32-bit imaginary → compact and standard for SDR data.
Plot the magnitude → boom, you see the spectrum.

☞ ## np.fft.fft(raw)

Takes the IQ samples (time series) and transforms them to frequency domain.
The output array shows how much of each frequency is present.
Think of it as:
☞ "If I play all possible pure tones, how much does each tone match my captured signal?"

☞ ### np.fft.fftshift(spectrum)

By default, FFT puts 0 Hz (DC) at the left edge of the plot.
That looks odd, because we usually expect negative frequencies on the left and positive frequencies on the right.
fftshift just re-centers the spectrum:
Middle = 0 Hz
Left = -Fs/2
Right = +Fs/2
☞ This makes the spectrum look like what a spectrum analyzer shows.

☞ ### np.abs(spectrum_shifted) ** 2

FFT gives complex numbers (amplitude + phase).
But what we usually want is power (energy at each frequency).
np.abs(x) gives magnitude.
**2 squares it to get power.
☞ So this is basically: "How strong is each frequency component?"

☞ ### Plotting

X-axis: Frequency bins (not yet converted to Hz, but we can later with np.fft.fftfreq)
Y-axis: Power
✅ That's it. You just built your first spectrum analyzer in ~10 lines of Python.
Input: messy IQ squiggles

Output: clear spectrum of frequencies present

When you do:

spectrum = np.fft.fft(raw)

raw has N samples (say, N = 4096).

The FFT output is also length N.

Each element spectrum[k] corresponds to a frequency bin, i.e., a tiny slice of the total frequency range.

How bins map to actual frequencies

Bin index k goes from 0 to N-1.

Frequency per bin (Δf) = fs / N, where fs = sample rate.

Bin 0 = 0 Hz (DC)

Bin 1 = fs/N Hz

…

Bin N-1 = (N-1)*(fs/N) Hz

If you use fftshift, the bins get rearranged:

Left half → negative frequencies (-fs/2 … 0)

Right half → positive frequencies (0 … fs/2)

Quick example :

fs = 2.4 MHz

N = 4096

Then:

Δf = fs / N = 2_400_000 / 4096 ≈ 585.94 Hz per bin

Bin 0 → 0 Hz

Bin 1 → 585.94 Hz

Bin 2 → 2 * 585.94 ≈ 1171.88 Hz

…

Bin 2048 (middle after fftshift) → 0 Hz

Bin 0 after fftshift → -fs/2 ≈ -1.2 MHz

Why we talk in bins vs frequency

“Bin” is the index in the FFT array.

Frequency is what the bin represents in Hz: f = k * fs / N (or after shift, f = (k-N/2)*fs/N).

📜 Python Practical

import numpy as np
import matplotlib.pyplot as plt

1. Load the raw IQ samples

raw = np.fromfile("Prac_1.bin", dtype=np.complex64)

2. Apply FFT to move from time → frequency domain

spectrum = np.fft.fft(raw)

3. Shift zero frequency (DC) to the center for better plotting

spectrum_shifted = np.fft.fftshift(spectrum)

4. Convert to power (magnitude squared)

power = np.abs(spectrum_shifted) ** 2

5. Plot the result

plt.plot(power)
plt.title("Frequency Spectrum")
plt.xlabel("Frequency Bin")
plt.ylabel("Power")
plt.show()

1️⃣ Read the IQ samples

```
raw = np.fromfile(filename, dtype=np.uint8)
I = raw[0::2].astype(np.float32) - 127.5
Q = raw[1::2].astype(np.float32) - 127.5
IQ = I + 1j*Q
```

- `raw[0::2]` → picks all **I samples**
- `raw[1::2]` → picks all **Q samples**
- `astype(np.float32)` → converts 8-bit integers to floats for calculations
- `-127.5` → centers the data around **0** (from `[0,255]` → `[-127.5,127.5]`)
- `IQ = I + 1j*Q` → builds a **complex64 array** that FFT can understand

2️⃣ FFT: Time → Frequency domain

```
spectrum = np.fft.fft(IQ)
```

- Converts your **time-domain complex signal** into **frequency bins**.
- Each element `spectrum[k]` corresponds to **the strength of a specific frequency component**.

3️⃣ FFT Shift: Center 0 Hz

```
spectrum_shifted = np.fft.fftshift(spectrum)
```

- Moves **DC (0 Hz)** to the center of the array.
- Makes plotting intuitive: negative frequencies on the left, positive on the right.

4️⃣ Power Spectrum

```
power = np.abs(spectrum_shifted) ** 2
```

- FFT output is **complex** → has magnitude and phase.
- `np.abs()` → magnitude
- `**2` → power (energy at each frequency bin)

5️⃣ Plotting

- **First plot** → raw FFT magnitude (not shifted)
- **Second plot** → shifted FFT for clear frequency visualization
- **Third plot** → power spectrum vs frequency bins

> ✅ Right now the X-axis is in **bins**, not Hz. Each bin corresponds to `fs/N` Hz.

📊 What You'll See & Analyze

- **FFT Plot (Spectrum)**
  - Peaks → carriers or strong tones.
  - Width of main lobe → occupied bandwidth.
  - Noise floor visible at bottom.
- **Spectrogram (Waterfall)**
  - Horizontal bright lines → continuous signals (carriers).
  - Bursts → packet transmissions.
  - Slanted lines → frequency drift or Doppler.

- ◦ Multiple bands → possible hopping or multi-signal environment.

✅ This builds directly on your **basic signal properties**