🔎 Introduction

In the world of **Digital Signal Processing (DSP)** and **Software Defined Radio (SDR)**, IQ data is the **fundamental building block** for representing real-world signals digitally. Whether you are plotting a spectrum, detecting a drone, decoding modulation, or feeding a machine learning model — **how you handle IQ data matters**.

One of the most critical operations in IQ processing is **normalization**.

This chapter will:

- Build your conceptual foundation of normalization
- Show why it's needed in SDR
- Demonstrate how to apply it in Python
- Clarify float vs integer decisions
- Include a robust code example
- Tie it all into real-world SDR signal flow

🔧 What is IQ Data?

IQ data is made up of:

- **I** (in-phase) and **Q** (quadrature) components
- Usually interleaved: `[I0, Q0, I1, Q1, ...]`
- Collected by SDR hardware as `uint8`, `int12`, or `int16`

> For basic SDRs like RTL-SDR, the samples are 8-bit unsigned integers (`uint8`), ranging from **0 to 255**.

1️⃣ Why Normalize?

Raw IQ data is **not centered around zero**, which is a **problem for DSP** operations such as:

- FFT
- Filtering

- Modulation/Demodulation
- Machine Learning
- Visualization (e.g., constellation diagrams)

❌ Without Normalization

- Values are always positive: `[0 to 255]`
- Math operations assume center at 0 → results are wrong
- Frequency and phase get distorted
- Filters and transforms behave incorrectly

2️⃣ The Two-Step Normalization Process

🎯 Step 1: Centering Around Zero

We shift the range `[0–255]` → `[-128 to +127]`:

```
centered = raw_value - 128
```

This makes:

- Midpoint = 0
- Symmetric swing on both sides

🎚 Step 2: Scaling to Float Range

We scale to float between approximately `[-1.0, +1.0]`:

```
normalized = (raw_value - 128) / 128.0
```

Resulting range:

- `-1.0` → `+0.9921875`

✅ This format is perfect for all DSP processing steps.

3️⃣ Why Use 128 and Not 127?

If you use 127, you get asymmetry:

```
(255 - 127) / 127 = +1.0078
```

That means:

- Signal is biased
- Not zero-centered
- Causes noise in modulation/demodulation and FFT

✅ Use `128.0` for perfect centering and symmetry.

4️⃣ Float vs Integer — Why It Matters

Format	Use Case	Pros	Cons
`uint8`	From SDR hardware	Compact, fast	Not DSP-ready
`int16`	High-res recording	Better range	Needs normalization too
`float32`	DSP, filtering, ML, FFT	Precise, widely supported	Takes more memory (4×)
`float16`	GPU optimization	Fast, compact float	Slight precision loss

📌 **Golden Rule**:

> Convert to float as soon as raw IQ enters your software.
> Stay in float for processing.
> Convert back to int only if saving/transmitting.

5️⃣ Python Code — Pairing and Normalizing IQ Data

Let's walk through code that:

- Takes raw IQ bytes from SDR
- Pairs them into I/Q tuples
- Normalizes them to float format

✅ Code

```python
# Raw IQ input (uint8 format)
raw_iq = [137, 250, 120, 10]  # [I0, Q0, I1, Q1]

# Step 1: Pair I and Q
def zip_raw_iq(iqdata_raw):
    i = iqdata_raw[0::2]  # Even-indexed → I
    q = iqdata_raw[1::2]  # Odd-indexed  → Q
    return i, q

# Step 2: Normalize each (I, Q) pair to float in [-1.0, +1.0]
def normalise(p_iqdata):
    n_p_iqdata = []
    for i_data, q_data in zip(p_iqdata[0], p_iqdata[1]):
        normalised_iqdata = (
            (i_data - 128) / 128.0,
            (q_data - 128) / 128.0
        )
        n_p_iqdata.append(normalised_iqdata)
        print("Normalized IQ:", normalised_iqdata)
    return n_p_iqdata

# Execution
paired_iqdata = zip_raw_iq(raw_iq)
print("Paired IQ Data:", list(zip(paired_iqdata[0], paired_iqdata[1])))

print("Now convert paired IQ Data into Normalized Data")
normalized_iqdata = normalise(paired_iqdata)
print("Final Normalized IQ Data Pairs:", normalized_iqdata)
```

◈ Output

```
Paired IQ Data: [(137, 250), (120, 10)]
Normalized IQ: (0.0703125, 0.953125)
Normalized IQ: (-0.0625, -0.921875)
Final Normalized IQ Data Pairs: [(0.0703125, 0.953125), (-0.0625, -0.921875)]
```

---

6 Use Cases Where Normalization is Critical

Application	Why Normalize?
FFT Analysis	Needs float values centered around zero
Modulation/Demodulation	Symbol mapping expects symmetry
Filtering	IIR/FIR filters fail on biased inputs
Machine Learning	Requires float input with known distribution
IQ Visualization	Constellation plots require true scaling

---

7 Reverse Process (When Saving or Transmitting)

If you want to convert normalized float IQ back to `uint8` for storage/transmission:

```python
def denormalize(normalized_iqdata):
    denorm = []
    for i_val, q_val in normalized_iqdata:
        i = int(round(i_val * 128 + 128))
        q = int(round(q_val * 128 + 128))
        denorm.extend([i, q])
    return denorm

# Example usage
restored_uint8 = denormalize(normalized_iqdata)
print("Restored uint8 data:", restored_uint8)
```

🔍 Advanced Concept: Preserving Precision

You asked a great question:

> “Why don’t we store the integer part and a fractional residue to preserve more info?”

This leads into:

- **Fixed-point representation**
- **Dual-word formats**
- **Quantization residuals**

These are valid and used in:

- High-precision radar
- Space applications
- Audio compression (FLAC, ALAC)

For SDR: typically not used unless your application **requires ultra-fidelity**.

✅ Summary: Key Takeaways

Concept	Action
Normalize IQ data	`(value - 128) / 128.0`

Concept	Action
Always convert to float	After receiving from SDR
Stay in float domain	For FFT, filters, demodulation
Reverse to int for saving	`(float * 128 + 128)`
Use `128`, not `127`	For symmetry
Paired I/Q → float complex	Required by most DSP libraries

🔷 Final Thought

> **Normalization is not optional.**
> It’s what makes digital signals DSP-ready, visualization-friendly, and machine-understandable.

It’s also one of the cleanest demonstrations of how raw physical data becomes structured digital intelligence.