🛠 BPSK Modulation Workshop: Digital Tx-Side Simulation Using NumPy

🎯 Objective:

To understand what **BPSK (Binary Phase Shift Keying)** is, why it is used, and how we simulate its signal chain **digitally using NumPy** as part of a **Software-Defined Radio (SDR) transmitter**.

◈ 1. What is BPSK?

BPSK = Binary Phase Shift Keying
It is the **simplest form of phase modulation** where:

- A bit `1` is transmitted as a sine wave with **0° phase**
- A bit `0` is transmitted as a sine wave with **180° phase (inverted)**
  Instead of changing the amplitude or frequency, **BPSK shifts the phase** of the carrier wave to represent binary data.
  🌀 Think of it like:
- `1` → cos(2πft)
- `0` → −cos(2πft)

✅ 2. Why Use BPSK?

Benefit	Description
◈ Simple	Only two symbols: +1, −1 (0°, 180°) — easiest to implement
📶 Robust	High noise immunity in low-bandwidth environments
🛰 Used In	GPS, satellite comms, military links, narrowband radios
◈ Ideal For	Starting with digital modulation fundamentals

The word “symbol” is fundamental to digital communication — yet often misunderstood.

❓ What is a Symbol in Digital Communication?

A symbol is a distinct waveform or signal state that represents data. It is not always the same as a single bit.

🔁 Analogy:

Think of **letters and words**:

- A **bit** is like a letter (smallest unit: 0 or 1)
- A **symbol** is like a whole word (can represent 1, 2, or more bits)
- A **modulated waveform** (like a sine wave at 0° or 180°) is the **physical form** of a symbol

---

🎯 Why Use Symbols?

Because transmitting raw bits is inefficient.
We group bits into symbols so we can represent multiple bits using a single waveform unit (e.g., amplitude, phase, frequency).

🔢 Examples of Symbol Mapping

Modulation	Bits per Symbol	Example Mapping
BPSK	1	1 → +1, 0 → −1 (phase: 0°/180°)
QPSK	2	00, 01, 10, 11 → 4 different phases
16-QAM	4	Each symbol = 4 bits (amplitude + phase)

🚀 How Symbols Relate to Data Rate and Bandwidth

⏱️ Symbol Rate (Baud Rate):

- **Number of symbols sent per second**
- Measured in **baud**
- E.g., 1,000 baud = 1,000 symbols/second

💾 Bit Rate:

- **Bits per second = Symbols per second × Bits per symbol**
- So if:
    - BPSK: 1 bit/symbol → Bit rate = Symbol rate

    - QPSK: 2 bits/symbol → Bit rate = 2 × Symbol rate

🌐 Bandwidth Usage:

- **Bandwidth depends on symbol rate**, not bit rate
- That's why **higher-order modulation** (QPSK, 16-QAM) sends **more bits per Hz of bandwidth**

⯑ Why Mapping to Symbols Is Necessary

Because:

- Physical systems (antennas, DACs) send **waveforms**, not 0s and 1s
- Mapping bits to symbols defines **how we turn digital data into analog signals**
- Modulation schemes **map bits** → **symbols** → **waveforms**

🔄 In NumPy Terms (What You're Doing)

When you do:

```
symbols = 2 * bits - 1
```

You're:

- Turning bits into symbols
- Each symbol becomes a **modulated signal unit**
- Then you represent it as a complex number (I + jQ) for SDR transmission

---

📊 Summary Table

Concept	Meaning
Bit	Smallest digital unit (0 or 1)
Symbol	Signal unit representing 1+ bits
Modulation	Method to turn symbols into waveforms
Symbol Rate	Number of symbols per second
Bit Rate	Symbol rate × bits per symbol
Bandwidth	Related to **symbol rate**, not bit rate

🛠 3. Digital Representation of BPSK

What happens in real hardware:

Step	Action
1	Bits are mapped to symbols
2	Symbols modulate a carrier signal
3	Signal is passed to DAC and transmitted

What we do in digital SDR simulation:

Step	Action
1	Create binary bitstream using NumPy
2	Map bits to symbols: `1 → +1`, `0 → -1`
3	Create complex IQ samples (`I = ±1`, `Q = 0`)

Step	Action
4	(Optional) Plot constellation
5	Interleave I and Q for DAC or file output

◈ 4. NumPy-Based BPSK Simulation (Tx Side)

✓ Step-by-step

🔹 a) Create Bitstream

```
import numpy as np
bits = np.array([1, 0, 1, 1, 0])
```

🔹 b) Map Bits to BPSK Symbols

```
symbols = 2 * bits - 1  # 1 → +1, 0 → -1
```

🔹 c) Form Complex I/Q Samples (BPSK: Q = 0)

```
iq_samples = symbols + 0j  # complex array with Q=0
```

🔹 d) Visualize Constellation (Optional)

```
import matplotlib.pyplot as plt
plt.scatter(iq_samples.real, iq_samples.imag)
plt.title("BPSK Constellation Diagram")
plt.xlabel("In-phase (I)")
plt.ylabel("Quadrature (Q)")
plt.grid(True)
plt.axis('equal')
plt.show()
```

This gives you **two points on the I-axis**: `+1` and `-1`, confirming the BPSK mapping.

🔷 e) Prepare for Transmission or File Output

Interleave I and Q components (as float32 values):

```
iq_interleaved = np.empty(2 * len(iq_samples), dtype=np.float32)
iq_interleaved[0::2] = iq_samples.real  # I
iq_interleaved[1::2] = iq_samples.imag  # Q
```

This is now ready for:

- DAC
- SDR transmission
- File writing (e.g., IQ.bin or .wav)

📈 5. Summary Flow: Tx Side Signal Chain

```
[Bitstream]
   ↓
[Map: 0 → -1, 1 → +1]
   ↓
[Create I/Q: I = Symbol, Q = 0 (BPSK)]
   ↓
[Form complex NumPy array]
   ↓
[Plot or interleave for SDR output]
```

🧭 6. Real-World Relevance

- This same I/Q logic powers **RTL-SDR transmitters**, **HackRF One**, and **GNU Radio flowgraphs**
- **Before hitting the antenna**, this is exactly what SDR software does to digital data

- This can be extended to **QPSK, 16-QAM, FSK**, etc., using same NumPy flow

---

✅ Your Workshop Checklist

Task	Done?
Understood what BPSK is	☐
Wrote code to map bits to symbols	☐
Created complex I/Q signal	☐
Plotted constellation	☐
Interleaved I/Q for Tx	☐

Excellent. You're very close to grasping the **core process** of digital modulation — especially **BPSK using I/Q representation**. Let's now **amplify** what you've understood into a fully intuitive and technically clear explanation.

Your Understanding So Far (Summarized)

- You have **digital data** (a stream of 0s and 1s).
- You decide to use **BPSK** (1 bit per symbol).
- In BPSK:
  - `1` → waveform = `cos(2πft)` (0° phase, or +1 amplitude)
  - `0` → waveform = `-cos(2πft)` (180° phase, or -1 amplitude)
- You represent these as `+1` and `-1` **symbols**.
- You then want to map this into **I/Q waveform samples** (baseband signal).

So far, ✔.

Now, let's go step-by-step on how this mapping to **I/Q waveforms** and eventually **samples (data)** works.

◈ Step-by-Step: How BPSK Maps to IQ Data

🔹 Step 1: Digital Bits → Modulation Symbols

We take:

```
bits = [1, 0, 1, 1, 0]
```

We map them using:

```
symbols = 2 * bits - 1  # → [1, -1, 1, 1, -1]
```

These are **abstract modulation symbols**: each one tells us whether to send `cos(2πft)` or `-cos(2πft)`.

🔹 Step 2: Why Do We Need IQ?

Real radio signals are sinusoidal (cosine/sine).
To make them using DSP, we **create them in baseband (I/Q form)** first, and then shift them to RF (via mixers).

BPSK baseband signal can be seen as:

```
s(t) = I(t) * cos(2πft) + Q(t) * sin(2πft)
```

For **BPSK**:

- We only vary the phase: 0° (**I**=+1) or 180° (**I**=−1)
- **Q is always 0** → no quadrature component

So:

```
s(t) = I(t) * cos(2πft)
```

🔷 Step 3: Represent This in DSP (IQ Data)

Now, we **don’t yet make the actual cosine waveform** — that happens at the DAC or in the RF stage.

Instead, we **store the I and Q components** in an array.

So:

```
iq_samples = symbols + 0j
```

This creates a **complex array** of:

- I values on the **real axis**: `+1` or `-1`
- Q values on the **imaginary axis**: all `0`

→ These are **baseband complex samples** representing each symbol.

⯑ Visual: Constellation of BPSK

```
    Q (Imaginary)
        ↑
        |
        |
        |
        |
        ●         ●
    (-1+0j)     (+1+0j)
        |
        |
        |
        +-------------------→ I (Real)
                  (0, 0)
```

Only two points: `+1` and `-1` on I-axis (Q = 0)

🔷 Step 4: Why Complex Numbers?

Because:

- Complex numbers = `I + jQ` = baseband signal
- Allows **simple mixing, filtering, and visualization**
- Efficient for **FFT, modulation, decoding**

Even though **BPSK has no Q part**, we still use complex format for compatibility with SDR processing chains.

🔷 Step 5: Turn IQ Samples into Transmittable Data

Now, the array `iq_samples = [1+0j, -1+0j, 1+0j, ...]` is ready.

To **send it to DAC** or **write to a file**, you interleave it:

```
I1, Q1, I2, Q2, I3, Q3, ...
```

In NumPy:

```
iq_interleaved = np.empty(2 * len(iq_samples), dtype=np.float32)
iq_interleaved[0::2] = iq_samples.real
iq_interleaved[1::2] = iq_samples.imag
```

This gives you a **flat array of float32s** → ready for:

- DAC input
- File saving (like `.bin`, `.wav`)
- SDR software (like GNU Radio)

◈ Summary Flow

```
[Bits: 1, 0, 1]
   ↓
[Symbols: +1, -1, +1]
   ↓
[IQ Samples: +1+0j, -1+0j, +1+0j]
   ↓
[Interleaved: 1, 0, -1, 0, 1, 0]
   ↓
[DAC/Transmit]
```

✓ Final Clarification

- `cos(2πft)` and `-cos(2πft)` are the **physical RF signals**
- We **don't generate them directly** in NumPy
- We generate their **baseband digital equivalents**: the I values

- At the SDR transmitter, the hardware or software **mixes them up** with cosine carrier to generate actual RF
