

Digital Communication using GNU Radio

Prof. Kumar Appiah

Department of Electrical Engineering

Indian Institute of Technology Bombay

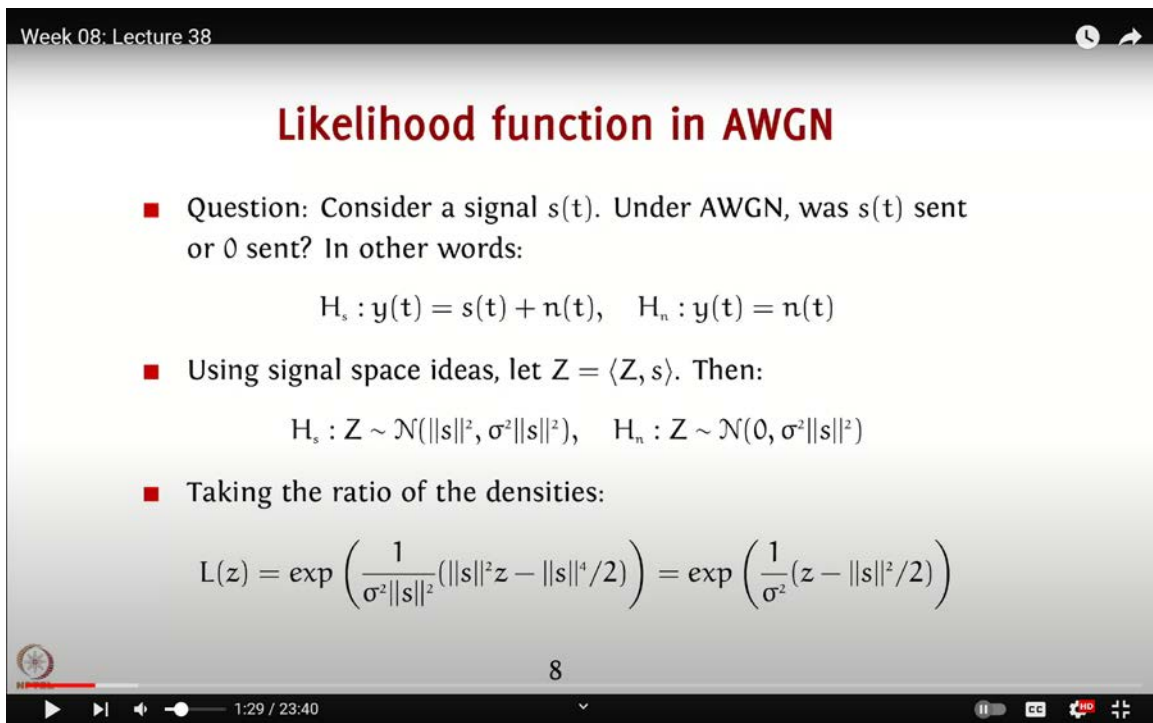
Week-08

Lecture-38

Phase Offset Estimation in GNU Radio

Hello and welcome to this lecture on Digital Communication using GNU Radio. My name is Kumar Appiah, and I am from the Department of Electrical Engineering at IIT Bombay. Today, we will use GNU Radio to delve into some of the concepts we have discussed in previous lectures. Specifically, we will focus on practically evaluating the likelihood function for the presence of a signal and examine how impairments such as noise affect the detection of phase offset. By understanding these impairments and their impact on parameter estimation, we can develop components crucial for practical communication systems. So, let's begin our exploration using GNU Radio.

(Refer Slide Time: 01:29)



Week 08: Lecture 38

Likelihood function in AWGN

- Question: Consider a signal $s(t)$. Under AWGN, was $s(t)$ sent or 0 sent? In other words:
$$H_s : y(t) = s(t) + n(t), \quad H_n : y(t) = n(t)$$
- Using signal space ideas, let $Z = \langle Z, s \rangle$. Then:
$$H_s : Z \sim \mathcal{N}(\|s\|^2, \sigma^2\|s\|^2), \quad H_n : Z \sim \mathcal{N}(0, \sigma^2\|s\|^2)$$
- Taking the ratio of the densities:
$$L(z) = \exp\left(\frac{1}{\sigma^2\|s\|^2}(\|s\|^2 z - \|s\|^4/2)\right) = \exp\left(\frac{1}{\sigma^2}(z - \|s\|^2/2)\right)$$

8

1:29 / 23:40

To start, let's consider the likelihood function for detecting the presence of a signal $s(t)$ under Additive White Gaussian Noise (AWGN). Here, we compare the hypotheses H_s and H_n based on the distribution of z . Similar to our binary signaling example, where we assessed the probability of making an error, the likelihood function in this case also follows a comparable form, with the decision region boundary given by the norm squared divided by 2. Since this situation closely mirrors the binary signaling example, though with the norm squared, you can try this on your own; we won't be repeating this specific example. The first practical example we will implement in GNU Radio is phase offset and maximum likelihood (ML) phase estimation.

(Refer Slide Time: 02:40)

The screenshot shows a video player interface for a lecture titled "Week 08: Lecture 38". The main content is a slide with the following text:

ML Phase Estimation

- We consider $y(t) = s(t)e^{j\theta} + n(t)$, complex AWGN of variance N_0 , θ unknown. ML estimate:

$$L(y|\theta) = \exp\left(\frac{1}{\sigma^2}(\text{Re}(\langle y, se^{j\theta} \rangle) - \|se^{j\theta}\|^2)/2\right)$$

- Let $\langle y, s \rangle = |Z|e^{j\phi} = Z_c + jZ_s$. Then $\text{Re}(\langle y, se^{j\theta} \rangle) = \text{Re}(e^{-j\theta}Z) = |Z| \cos(\theta - \phi)$

$$L(y|\theta) = \exp\left(\frac{1}{\sigma^2}(|Z| \cos(\theta - \phi) - \|s\|^2)/2\right)$$

- Maximized for $\hat{\theta}_{ML} = \arg(\langle y, s \rangle) = \tan^{-1}(Z_s/Z_c)$

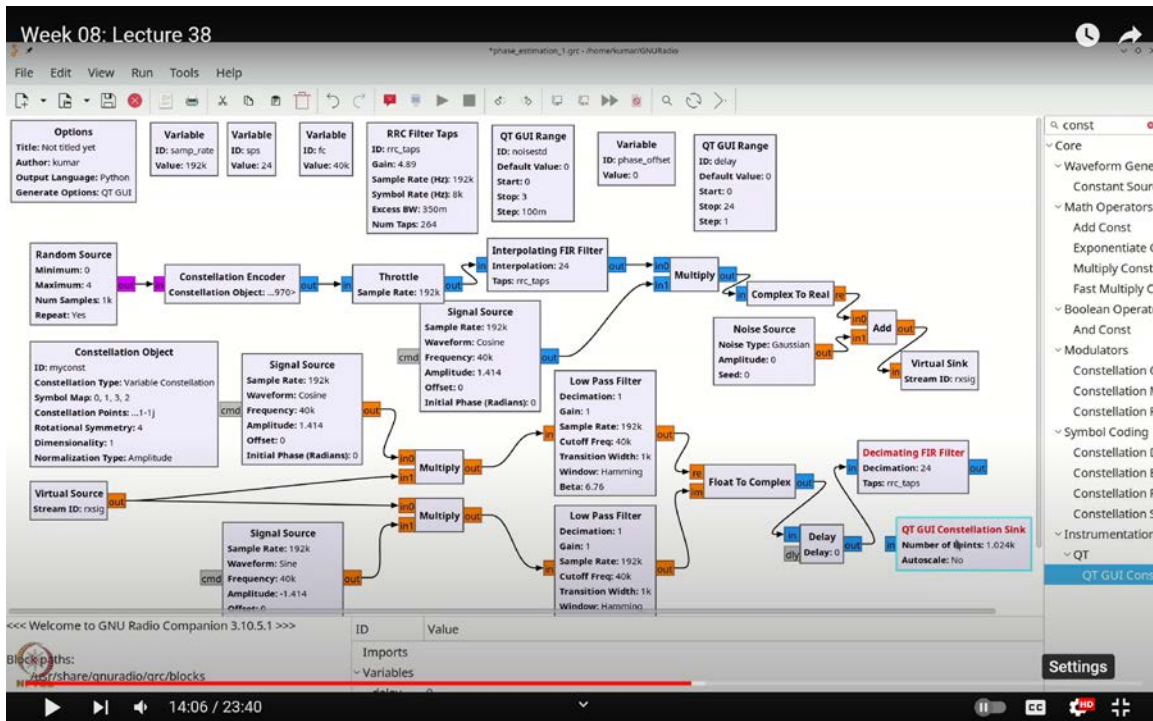
10

The video player controls at the bottom show a progress bar at 2:40 / 23:40 and various playback icons.

Remember that at the receiver, the carrier, or more specifically, the local oscillator, has a phase offset relative to the carrier at the transmitter. Our task here is to determine this phase offset. We can achieve this by comparing the received signal with a template function $s(t)$ and analyzing the phase of the resulting integral. Let's implement this using GNU Radio.

To implement our phase estimation approach, we need to assemble a complete basic digital communication setup, including baseband and passband components, which we've covered previously. So, let's quickly go through this process again.

(Refer Slide Time: 14:06)



First, we will set our sampling rate to 1920. We need to create a variable; to do this, press Ctrl+F (or Cmd+F), type "variable," and select the variable block. We will name this variable "sps" for samples per symbol. Since we aim for 8000 symbols per second, we'll set this variable to `samp_rate / 8000` using integer division.

Next, let's create another variable for our carrier frequency. Press Ctrl+F (or Cmd+F) and type "variable" again, but this time name the variable "fs" for carrier frequency.

Now we're ready to assemble our system. First, we need a random source. Press Ctrl+F (or Cmd+F) and type "random" to find the random source block. Change the type to "byte" and select "QPSK" modulation, with values ranging from 0 to 4.

The next component we'll add is a constellation encoder. Press Ctrl+F (or Cmd+F) and type "const" to find the constellation encoder block. Position this block below the random source.

I have my constellation encoder set up here. While I have this screen open, I will also add the constellation object needed to specify the constellation. I'll name this object ``myconst``. It's a standard QPSK constellation, so this will be our ``myconst``.

Next, we need to add a throttle. Press Ctrl+F (or Cmd+F) and type "thr" to find the throttle block and add it to our flowgraph. To ensure that we meet our bandwidth constraints, we need to include a pulse shaper.

Let's add an RRC (Root Raised Cosine) pulse shaper. Press Ctrl+F (or Cmd+F), type "RRC," and select the RRC filter taps block, which is a variable. Set the symbol rate to 8000 and leave the other parameters as default. The gain should be the square root of 24, approximately 4.89, so input that value.

Now, I need an interpolating FIR filter. Press Ctrl+F (or Cmd+F) and type "inter" to find the interpolating FIR filter block. Set the interpolation factor to ``SPS`` and use the RRC filter taps, which should be named ``RRC_taps``.

With the interpolating filter ready, we need to convert this to passband. This involves multiplying the signal by $e^{j2\pi f_c t}$ and taking the real part. To do this, we first add a multiplier block. Press Ctrl+F (or Cmd+F), type "mult," and add the multiply block to the flowgraph.

Next, add a signal source. Press Ctrl+F (or Cmd+F) and type "sig n" to find the signal source block. This signal source will generate a complex signal of the form $e^{j2\pi f_c t}$. For a cosine signal, this should be a float. Set the frequency to f_c and for scaling, use an amplitude of 1.414 (which is $\sqrt{2}$). Finally, we will take the real part of the resulting signal to obtain our passband signal.

Press Ctrl+F (or Cmd+F) and search for "complex to real" to find the block that will convert our complex signal to a real signal. We can now add noise to our signal. Press Ctrl+F (or

Cmd+F) and search for "noise source" to add a noise source block. Double-click the noise source block to set it to real by choosing "float" and name the amplitude ``noise_std``.

Next, add a QT GUI Range to control the noise level. Press Ctrl+F (or Cmd+F) and type "range" to add the QT GUI Range block. Set the name to ``noise_std``, with a default value of 0, a minimum of 0, a maximum of 3, and a step size of 0.1.

Now, add an "add" block. Press Ctrl+F (or Cmd+F), type "add," and double-click the block to make it float. Connect the noise source and the complex-to-real converter to this "add" block. To manage this setup efficiently, we need a virtual sink to monitor the receiver signal. Press Ctrl+F (or Cmd+F) and type "virtual sink" to add a virtual sink block. Name this block ``rx_sig`` to represent the receiver signal.

For the receiver side, create a virtual source to retrieve the ``rx_sig`` and get back the constellation. Add a virtual source block and set its stream ID to ``rx_sig`` to receive the same signal with added noise.

At the receiver, we will mix the signal with two sinusoids: one for the cosine and one for the sine component. To do this, add a signal source block. Press Ctrl+F (or Cmd+F) and type "signal source" to place a signal source block. Set its frequency to ``fc`` and its amplitude to 1.414. Additionally, we will add a phase offset, which we will define as a variable ``phase_offset``. Ensure that this block is set to float.

Copy this signal source block (Ctrl+C) and paste it (Ctrl+V) to create a second signal source block. In this second block, set the amplitude to be negative and the waveform to sine. We now have everything set up, but we need to define the ``phase_offset`` variable.

First, press Ctrl+F (or Cmd+F) and search for "var" to create a new variable. Name this variable ``phase_offset``, and set its initial value to 0 for now. We'll adjust this value later as needed.

Next, we need to multiply our signal sources with this variable. To do this, we will add a multiply block. Copy (Ctrl+C) and paste (Ctrl+V) this multiply block, then double-click it to change its type to float. Connect this block to the signal sources accordingly. Repeat the

process by duplicating the multiply block again (Ctrl+C, Ctrl+V), and make the necessary connections.

Now, we need to account for the delay introduced by the low pass filters. To handle this, add a delay range and a delay block. Press Ctrl+F (or Cmd+F) and type "range" to add a QT GUI Range block. Name this block ``delay``, set the default value to 0, and adjust the range to go from 0 up to ``sps - 1``. Set the type to integer. Also, add a delay block by searching for "delay" and place it in the workflow. Ensure that the delay block is linked to the ``delay`` range by typing "delay."

We now need a pair of low pass filters. Press Ctrl+F (or Cmd+F) and search for "low pass filter" to add the first filter. Set it to operate in real mode, with a decimation factor of 1 (no decimation), a cutoff frequency of ``fc``, and a transition width of 1000 Hz. Duplicate this low pass filter (Ctrl+C, Ctrl+V) to create the second filter. Adjust the layout if necessary to make space.

Next, add multipliers to connect the output of the low pass filters. Once these connections are in place, convert the signal to baseband by using a "float to complex" block. Press Ctrl+F (or Cmd+F) and search for "float to complex" to add this block to the workflow.

Finally, add a decimating FIR filter to complete the setup. Press Ctrl+F (or Cmd+F), type "decimating FIR filter," and place it in the workflow. Set the decimation factor to ``sps`` and use the same RRC taps as before. Since the taps are symmetric, there is no need to reverse them.

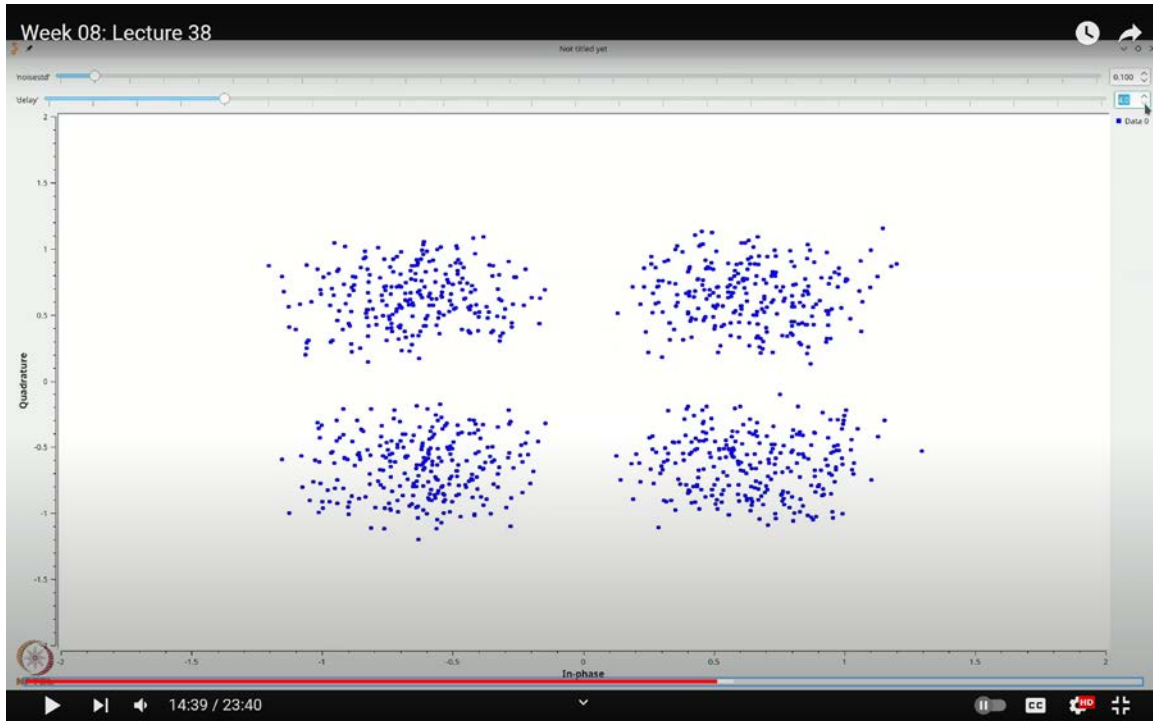
This completes the setup for your phase offset estimation and signal processing.

To ensure that our constellation is received accurately, we will add a QT GUI Constellation Sync block. To do this, press Ctrl+F (or Cmd+F) and search for "const" to locate the QT GUI Constellation Sync block. You can rotate this block using the arrow keys if needed; I have rotated it and connected it accordingly.

Now, let's execute the flow graph and observe the results. As you can see, there is noticeable distortion. From previous experiments, you might remember that setting the

delay to 9 corrects the constellation even in the presence of noise. When the delay is set correctly, the constellation appears quite accurate. If the delay is incorrect, the quality degrades.

(Refer Slide Time: 14:39)



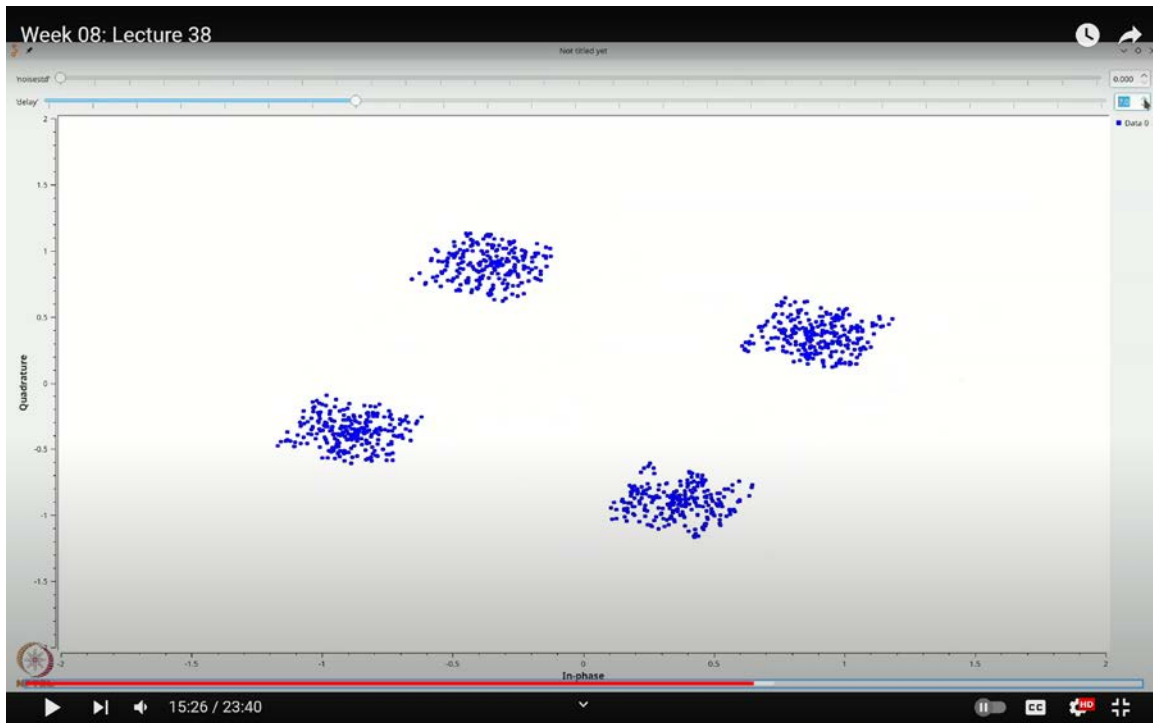
Next, we will examine the effect of the phase offset. Double-click on the signal source and change the label from "phase offset" to "initial phase." We will apply the same change to the corresponding block. Set the initial phase to $\pi/8$ by typing `3.1415 / 8`. Execute the flow graph again, and you'll observe that the constellation becomes skewed.

The skew in the constellation arises because the I and Q components, which should have been correctly aligned if the phase at the receiver were accurate, are missing. This additional rotation occurs due to the phase offset. As we derived in the lecture, multiplying $\cos(2\pi f_c t)$ with $\cos(2\pi f_c t + \phi)$ and $\sin(2\pi f_c t)$ with $\sin(2\pi f_c t + \phi)$ results in a rotation of the constellation.

If you check other phase offsets, for instance, $\pi/4$, you will see that it rotates the constellation accordingly. Specifically, a $\pi/4$ rotation changes a point at $1 + j$ (divided

by $\sqrt{2}$) to $\hat{1} + \hat{j}$. If you rotate by $\pi/2$, you essentially recover the original constellation, though you must account for the fact that the point is essentially rotated $\pi/2$.

(Refer Slide Time: 15:26)

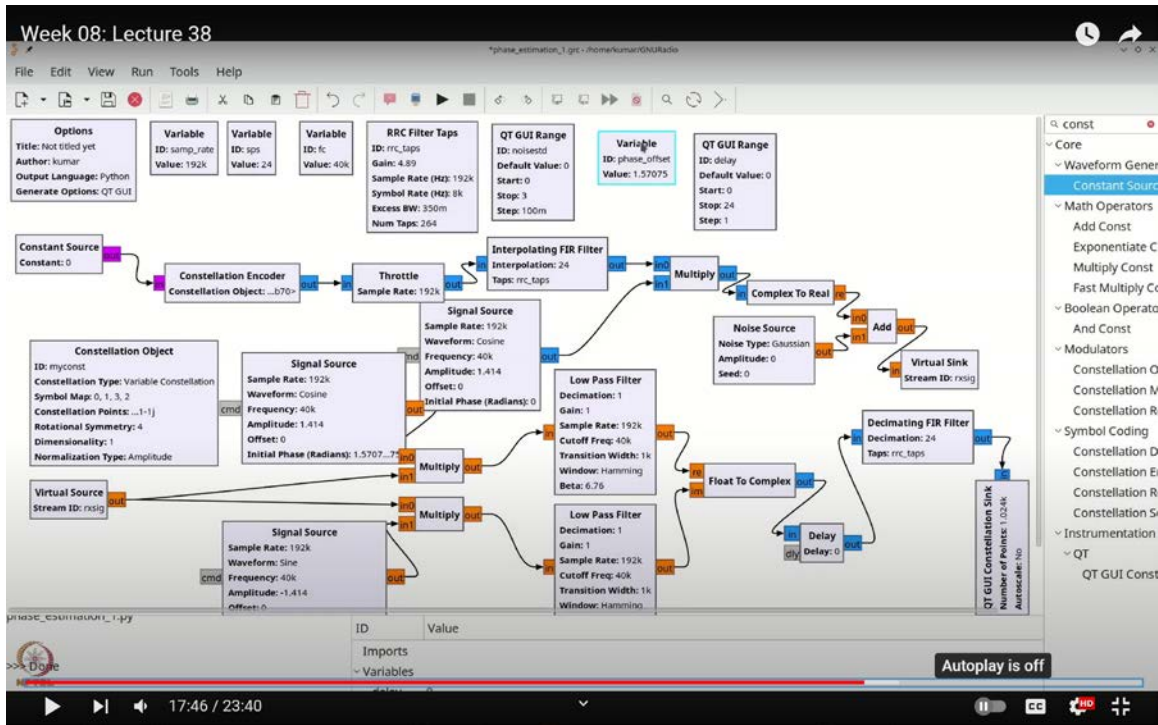


These calibrations are crucial. If the phase offset is unknown, it will lead to errors in detection. To determine the phase offset, one effective strategy is to use a known symbol approach. You can send a known signal for a period to identify and correct the phase offset. Let's explore a simple example to understand how this phase detection works.

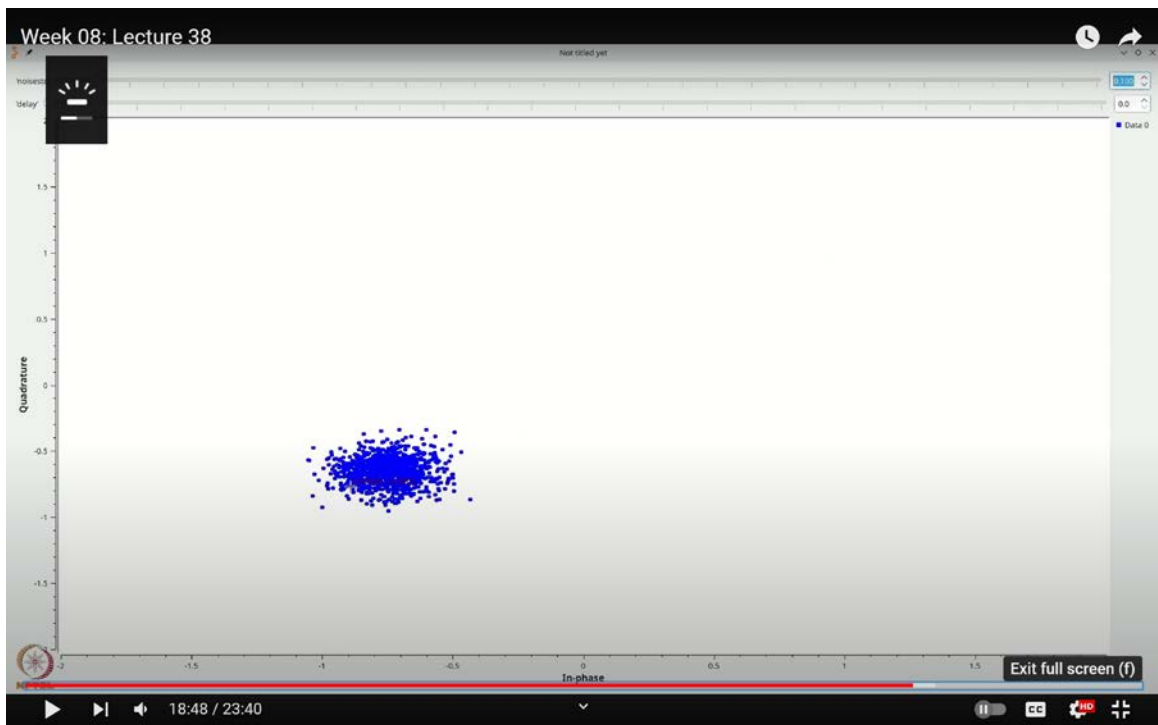
Let's temporarily remove the random source by selecting it and pressing delete. Instead, we'll add a constant source. Press Ctrl+F (or Cmd+F) and type `CONST` to find the Constant Source block. Double-click on this block and set it to output a constant byte value of 0. This byte 0 corresponds to the first element of our constellation, which is $\hat{-1} - \hat{j}$.

Now, if we set the phase offset to 0 and execute this flow graph in the absence of noise, we should obtain $\hat{-1} - \hat{j}$ divided by $\sqrt{2}$, which is the expected constellation point. Even with noise, you should still see this expected result.

(Refer Slide Time: 17:46)



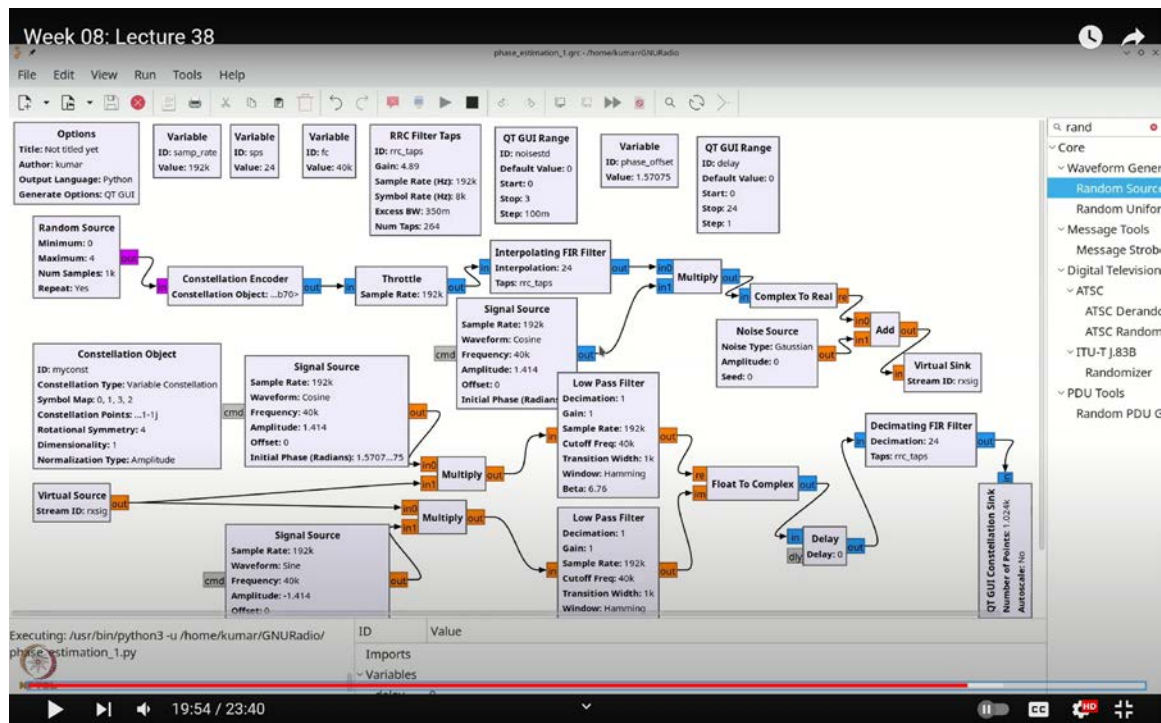
(Refer Slide Time: 18:48)



If there is a phase offset, the point $-1 - j$ will rotate by that amount. For example, with a phase offset of $\pi/4$ ($3.1415 / 4$), executing the flow graph will show that this constellation point has undergone a clockwise rotation of $\pi/4$. This rotation occurs because the phase offset is applied at the receiver.

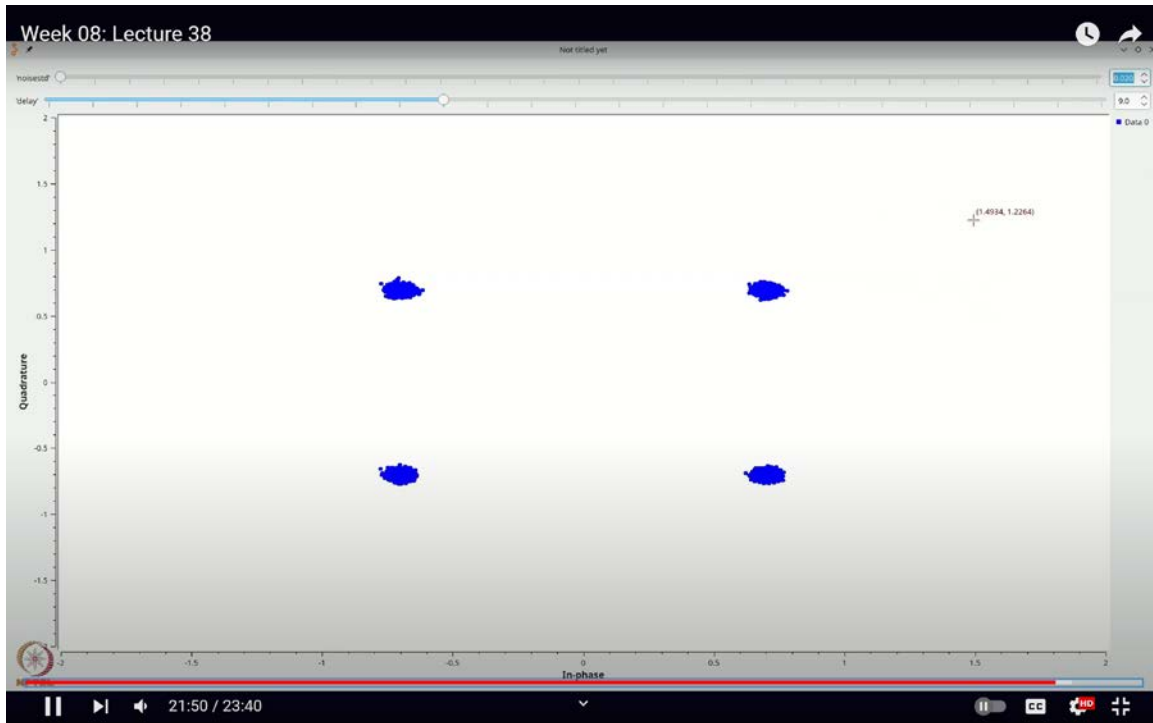
If we set the phase offset to $\pi/2$ ($\pi / 2$), in the presence of noise, you will notice that the point $-1 - j$ shifts to $-1 + j$. This demonstrates that sending a known training sequence from the transmitter allows us to identify and correct for the phase offset. In training modes such as PSK or similar constellations, this technique can be used to correct phase offsets effectively.

(Refer Slide Time: 19:54)



To complete the setup, let's revert to our random source. Remove the constant source, press Ctrl+F (or Cmd+F), and search for "random" to add a Random Source block. Double-click on this block and configure it to output values from 0 to 3 as bytes. Connecting this back will restore the original performance of the system.

(Refer Slide Time: 21:50)



Set the delay to 9 by default to avoid repetitive adjustments. Additionally, let's introduce a bit of noise by setting the default noise level to 0.02. With noise, it's crucial to be cautious because excessive noise can cause the signal to fall into neighboring bins.

To handle increased noise, consider increasing the number of constellation points. Double-click on the QT GUI Constellation Sync block and set it to 10,240 points. Similarly, set the random source to 10,000 values. In noisy conditions, averaging multiple measurements can help improve the signal-to-noise ratio. By averaging out the noise over a prolonged training period, you can mitigate its impact and enhance calibration accuracy.

Of course, even blind strategies are not immune to noise; in fact, they are often significantly impacted by it and may require even more extensive training. To summarize, whether you are dealing with noise with training or without, or using a blind approach, you must estimate the phase using some method based on the signal's properties. Only then can you reliably detect the transmitted symbols.

Next, we will focus on phase calculation. One important point to note is that if you have a constellation like QAM-4 (which is essentially QPSK) or something like QAM-16, once you determine the phase offset and frequency offset, you can use the distances of these symbols from the boundary to calculate the amplitude A . Remember, finding the amplitude A was also one of our challenges, and this approach can help with that.

There are both blind and training-based strategies for this, but the core principle is to measure the distance of the constellation points from the origin or to fit the constellation properly. It is worth noting that the maximum likelihood estimate of the amplitude is implicitly addressed through this method.

In this lecture, you've gained insight into some receiver impairments that can be modeled using GNU Radio, particularly the impact of phase offsets and how they manifest in the presence of noise. Additionally, you've explored aspects like parameter estimation of amplitude. In subsequent lectures, we will examine other receiver impairments, such as frequency offsets, and explore methods to address them. Thank you.