Digital Communication using GNU Radio

Prof. Kumar Appaiah

Department of Electrical Engineering

Indian Institute of Technology Bombay

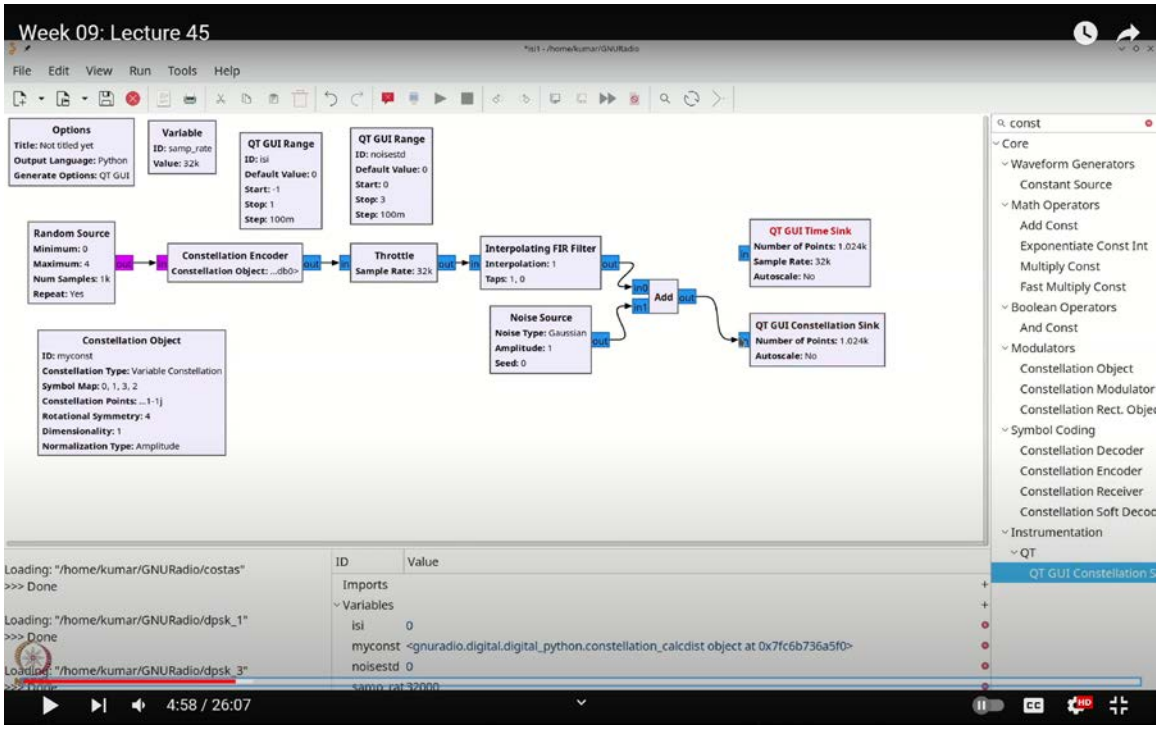
Week-09

Lecture-45

Zero forcing Receiver in GNU Radio

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. As you may recall, we have been discussing suboptimal equalization algorithms. In this context, our goal today is to implement the zero-forcing equalizer. The zero-forcing equalizer is notably simpler to implement and aims to completely eliminate interference whenever possible.

(Refer Slide Time: 04:58)



We will use GNU Radio to first implement a zero-forcing equalizer for a sample-sized

system, an equalizer that operates at the sample rate. We will then extend this to work with the equalizer we have been using as our running example, where the receiver samples at twice the rate of the symbols sent by the transmitter.

Let's dive into the GNU Radio implementation. We will start by exploring inter-symbol interference with a simple PSK example to observe how it affects our system. For simplicity, we will conduct a baseband simulation.

First, we need to add a random source. Press **`Ctrl + F`**, type "random," and select the random source. Double-click the random source and set the type to "byte" with values ranging from 0 to 4. Next, we will introduce our constellation object and constellation encoder to set up a QPSK constellation. Press **`Ctrl + F`**, type "CONSTANT," and you will find the constellation encoder and constellation object. Place these components onto the workspace.

Double-click the constellation object and name it "myconst." The constellation encoder will be configured as an instance of "myconst." Connect the random source to the encoder.

We need to add a throttle to control the simulation rate. Press **`Ctrl + F`** (or **`Cmd + F`** on macOS), type "throttle," and place it into the flowgraph.

Now, let’s add a channel to our setup. We’ll keep the channel simple by using a finite impulse response (FIR) filter to introduce inter-symbol interference. We are performing a one-symbol-per-sample simulation and will use an FIR filter to model the channel. Press **`Ctrl + F`** (or **`Cmd + F`**), type "interpolating FIR filter," and add it to the flowgraph. Connect this FIR filter to the rest of the components, leaving flexibility for adjusting the taps.

Double-click the interpolating FIR filter and set the taps to "1, ISI," where "ISI" represents inter-symbol interference. We will control the value of ISI using a QT GUI range. Press **`Ctrl + F`** (or **`Cmd + F`**), type "range," and add the range component to the flowgraph. Configure the ISI range to vary between -1 and 1.

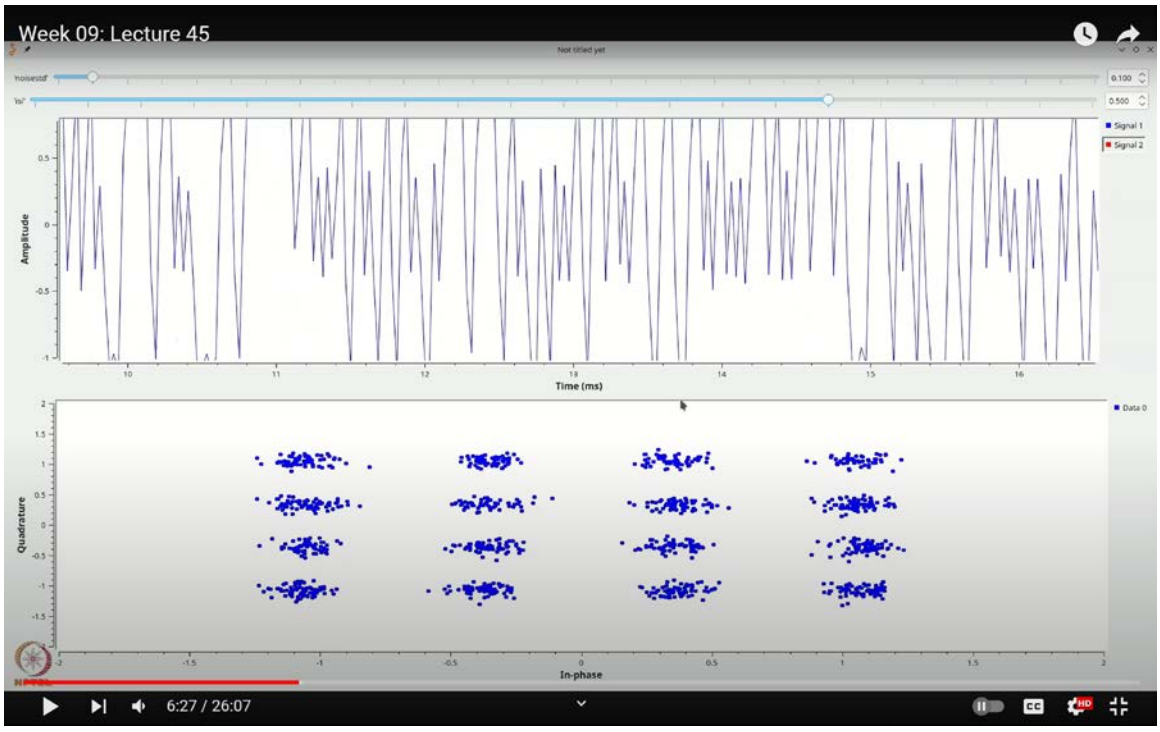
This setup will allow us to explore the impact of inter-symbol interference and implement

the zero-forcing equalizer effectively.

Let's start by setting the default value to 0. We will configure it to range from -1 to 1 with a step size of 0.1. This parameter represents a certain amount of the signal being received. Additionally, we will introduce some noise into the system. Press **`Ctrl + F`** (or **`Cmd + F`**), search for "range," and add another range component. We will name this range "noise std," standing for noise standard deviation. Set its standard value to 0, with a range from 0 to 3 and a step size of 0.1.

Next, add a noise source by pressing **`Ctrl + F`** (or **`Cmd + F`**), typing "noise," and selecting the noise source. Connect this noise source to the flowgraph by pressing **`Ctrl + F`** (or **`Cmd + F`**) and selecting "add."

(Refer Slide Time: 06:27)



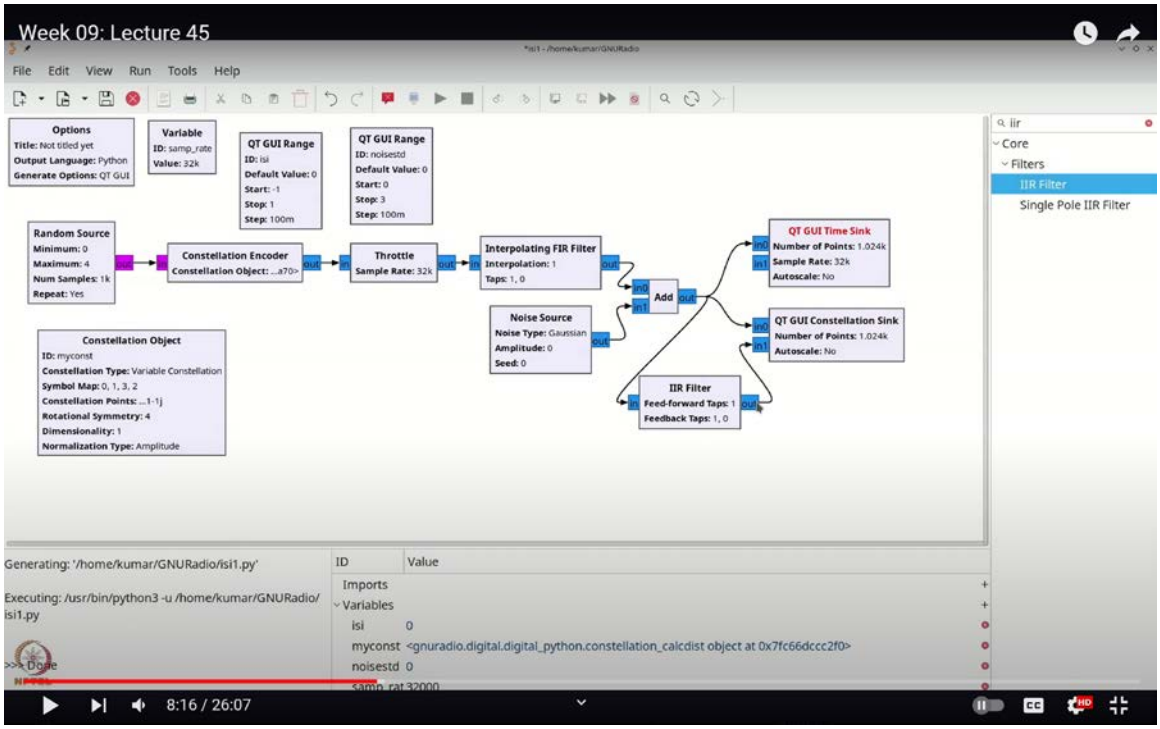
Once the components are added, we can inspect both the time domain view and the QT GUI constellation sink. Press **`Ctrl + F`** (or **`Cmd + F`**), select "time sink" for the time domain view, and similarly, add the QT GUI constellation sink. Connect the QT GUI constellation sink to the range component we previously set for the noise standard

deviation. Ensure that the amplitude of the noise is controlled by this "noise std" parameter.

Initially, we will start with no noise. When observing the constellation, it should display a standard QPSK constellation with values like $\frac{1+j}{\sqrt{2}}$ and so on, which is why the values range between $-\frac{1}{\sqrt{2}}$ and $\frac{1}{\sqrt{2}}$. Everything should look as expected. Now, introduce a small amount of noise to see its effect.

With noise introduced, the constellation will start to show distortions. Next, let's focus on introducing inter-symbol interference (ISI). Temporarily, we will only view the real part of the signal. Adding ISI will cause noticeable splitting in the constellation. This splitting occurs because the symbols are not reproduced faithfully at the receiver. The received signal is convolved with a filter of the form $\delta[n] + \text{ISI} \cdot \delta[n-1]$. For example, if ISI is 0.2, the received signal Y[n] is actually $X[n] + 0.2 \cdot X[n-1]$, which results in a QPSK constellation superimposed with a smaller QPSK constellation.

(Refer Slide Time: 08:16)



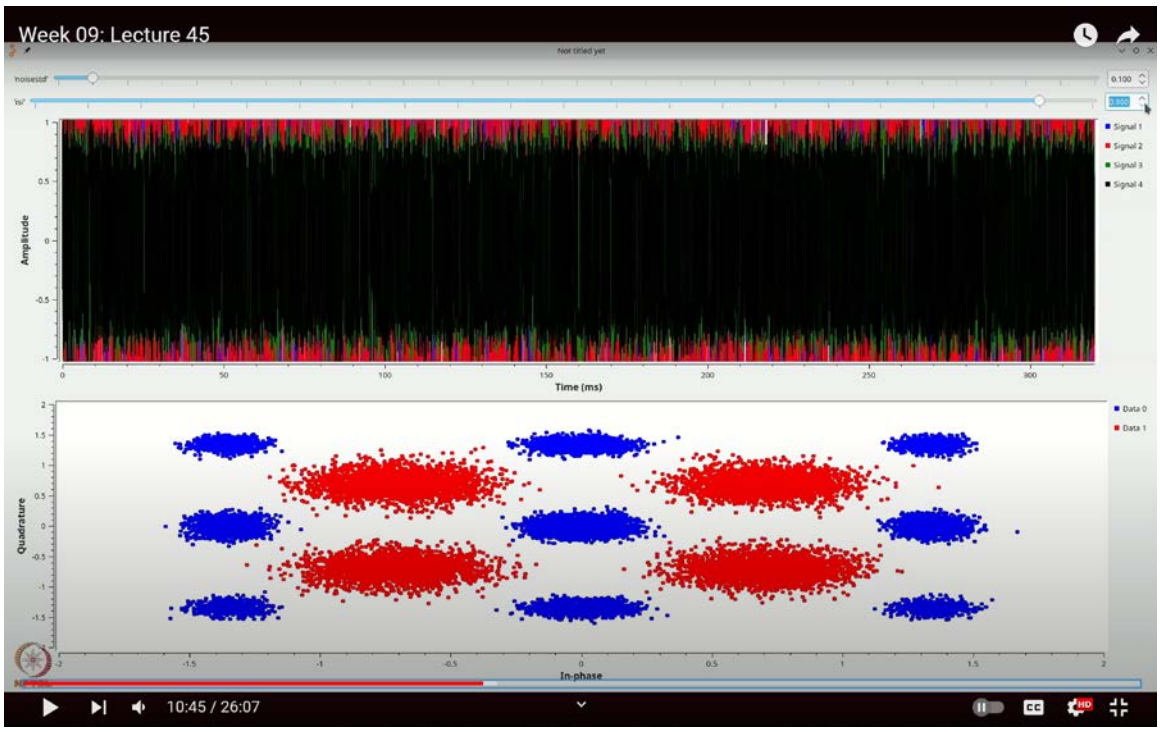
In fact, if you examine the constellation closely, the distance between the center of this

mini QPSK constellation and any of the standard QPSK constellation points will be approximately 0.3 times the original distance. This indicates that the system is less robust to noise due to this effect.

Our first step to address this is to perform zero-forcing filtering. To implement zero-forcing filtering, we will construct the inverse filter. Given that the filter's impulse response is $\delta[n] + \text{ISI} \cdot \delta[n-1]$, its Z-transform is $1 + Z^{-1} \cdot \text{ISI}$. To apply this, press **`Ctrl + F`** (or **`Cmd + F`**), search for "interpolating FIR filter," and add the filter to the flowgraph.

Let's start by double-clicking on the filter. We can't use a standard FIR filter for this purpose; instead, we need to use an IIR filter. Press **`Ctrl + F`** (or **`Cmd + F`**), search for "IIR filter," and select it. Set it to operate in complex-to-complex mode. The "flow taps" setting is acceptable for our case.

(Refer Slide Time: 10:45)



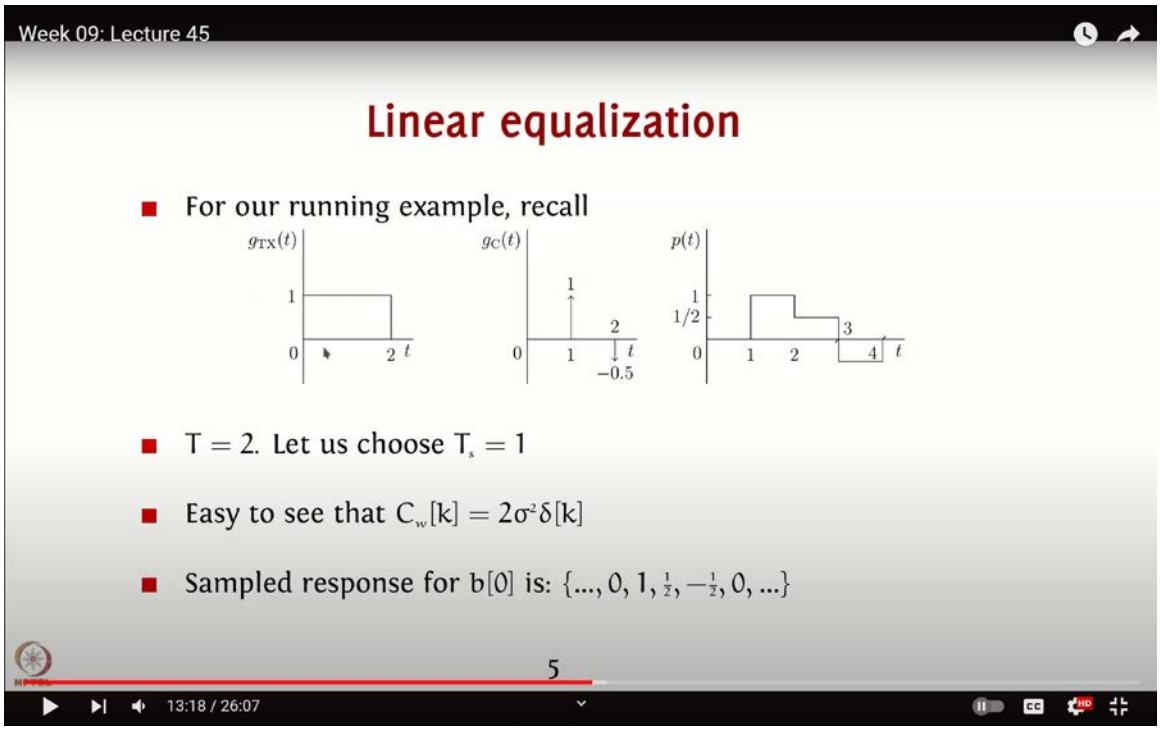
For an IIR filter, you need to specify both the feedforward taps and the feedback taps. In this instance, the feedforward tap is set to 1, and the feedback taps are 1 and ISI. Additionally, ensure that the "old style of taps" setting is turned off. This setting was

originally used for compatibility with MATLAB or other software-based filter designs but is no longer necessary.

So, we set it to false. This filter now represents $\frac{1}{1+Z^{-1}}$. Next, connect the filter output to both of the viewing components simultaneously. Double-click on the QT GUI Time Sink and set the number of inputs to 2. Do the same for the QT GUI Constellation Sink. Connect the outputs accordingly and then execute the flowgraph.

Initially, if you observe the constellation, it should appear correct. As you increase the noise level, you will notice that the constellation remains largely unaffected because the ISI is set to 0. In this case, the equalizer is essentially behaving as a regular all-pass filter with no delay.

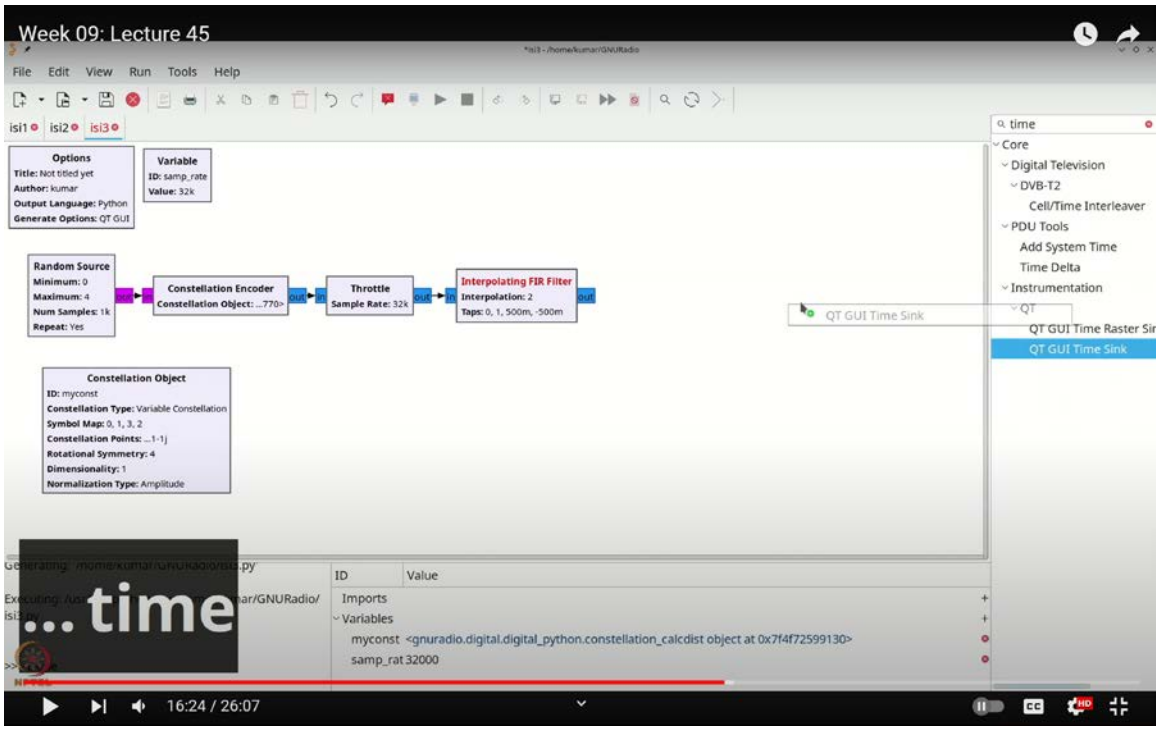
(Refer Slide Time: 13:18)



However, as you introduce ISI, you will observe that while the unequalized constellation starts to spread, the equalized constellation should return to its correct position. Yet, one issue remains: the equalized values are highly susceptible to noise.

Let’s adjust the setup for a better observation. Increase the number of samples to 10,000 and adjust the number of points in both the constellation and time sinks to match. As you raise the noise level, you will see the effect on the constellation. With higher ISI, the constellation spread becomes evident. Increasing both ISI and noise significantly exacerbates the spread.

(Refer Slide Time: 16:24)



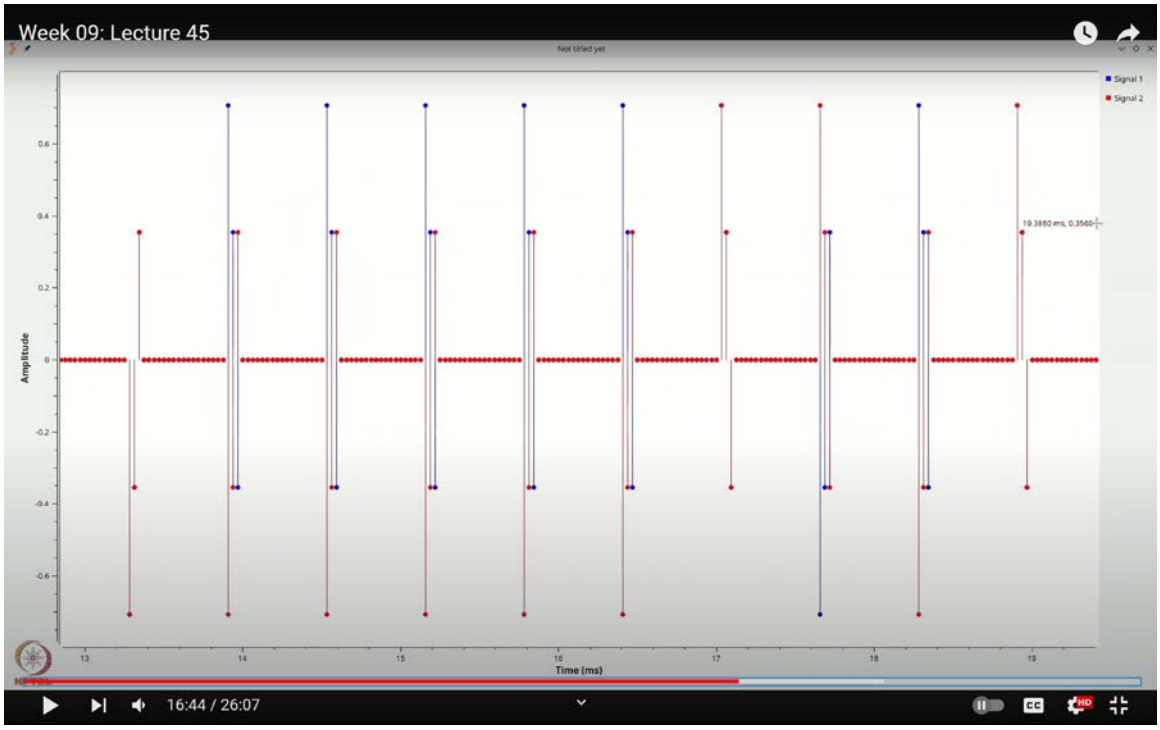
In the absence of ISI (i.e., ISI = 0.5), reducing ISI to 0 results in minimal noise enhancement, and the constellation remains quite stable. However, as ISI increases, the noise enhancement also increases, causing the red constellation to spread.

This issue arises because the zero-forcing equalizer amplifies the noise in its effort to cancel out the inter-symbol interference. To effectively neutralize ISI, the equalizer needs to extract the required component from the vector space, which, as we observed, results in increased noise enhancement.

Indeed, you'll observe that if we keep the noise level very low, say at 0.1, and then start increasing the inter-symbol interference (ISI), you’ll notice that as ISI approaches around

1, the data essentially becomes corrupted and spreads out significantly. This happens because when ISI equals 1, the filter essentially becomes $\frac{1}{1+Z^{-1}}$, which has a pole at $Z = e^{j\pi}$. This configuration will drastically amplify the noise, particularly around π, leading to significant noise enhancement.

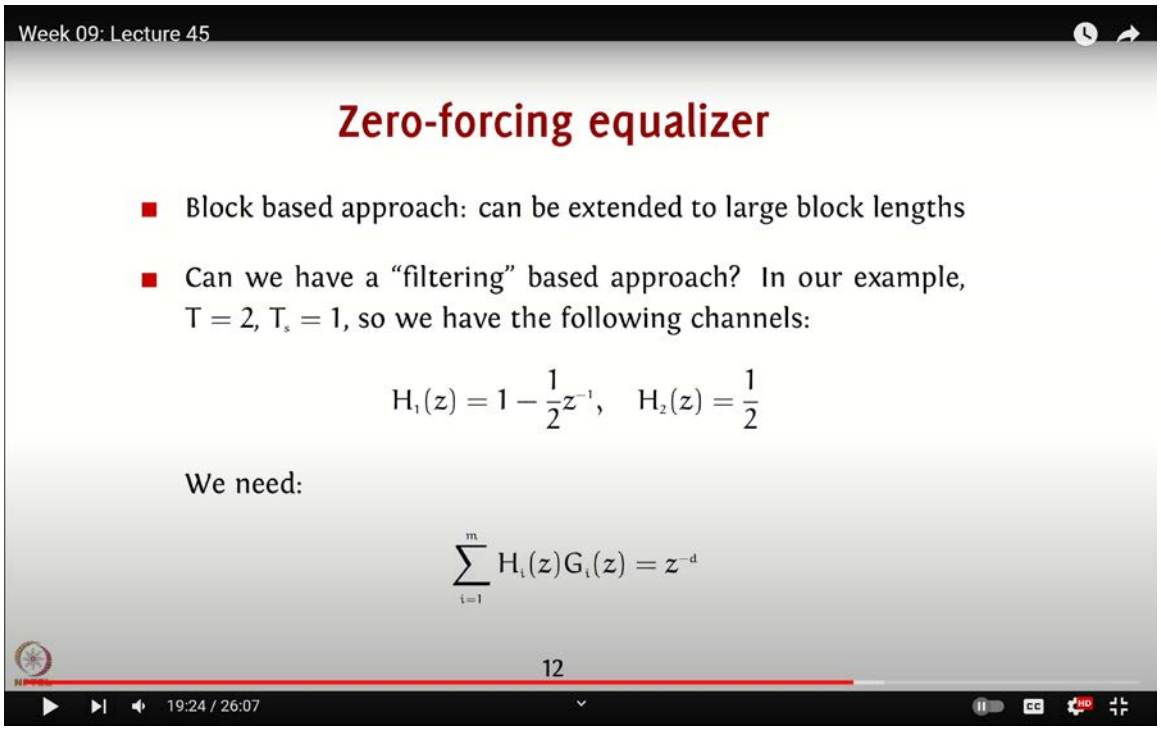
(Refer Slide Time: 16:44)



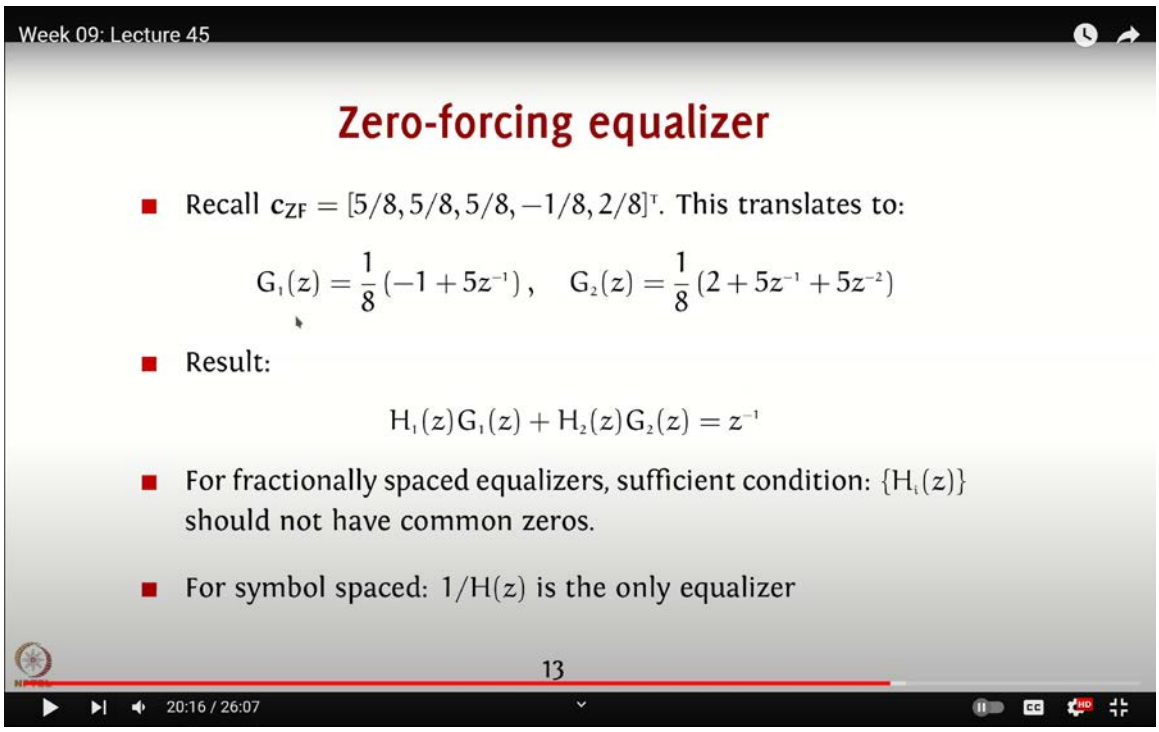
Let’s fine-tune the resolution of the ISI range to 0.01. By running the simulation with a small amount of noise and setting the ISI close to 0.95, you can clearly see the dramatic noise enhancement. This enhancement occurs because noise at higher frequencies gets significantly amplified. Conversely, if you explore negative ISI values, you will see that noise at lower frequencies is similarly enhanced. Thus, regardless of the sign of ISI, you will experience some degree of noise enhancement.

The issue with the zero-forcing equalizer in low Signal-to-Noise Ratio (SNR) scenarios becomes apparent from this observation. The signal spreads out, and the blue blobs on the constellation diagram represent the received signal.

(Refer Slide Time: 19:24)



(Refer Slide Time: 20:16)

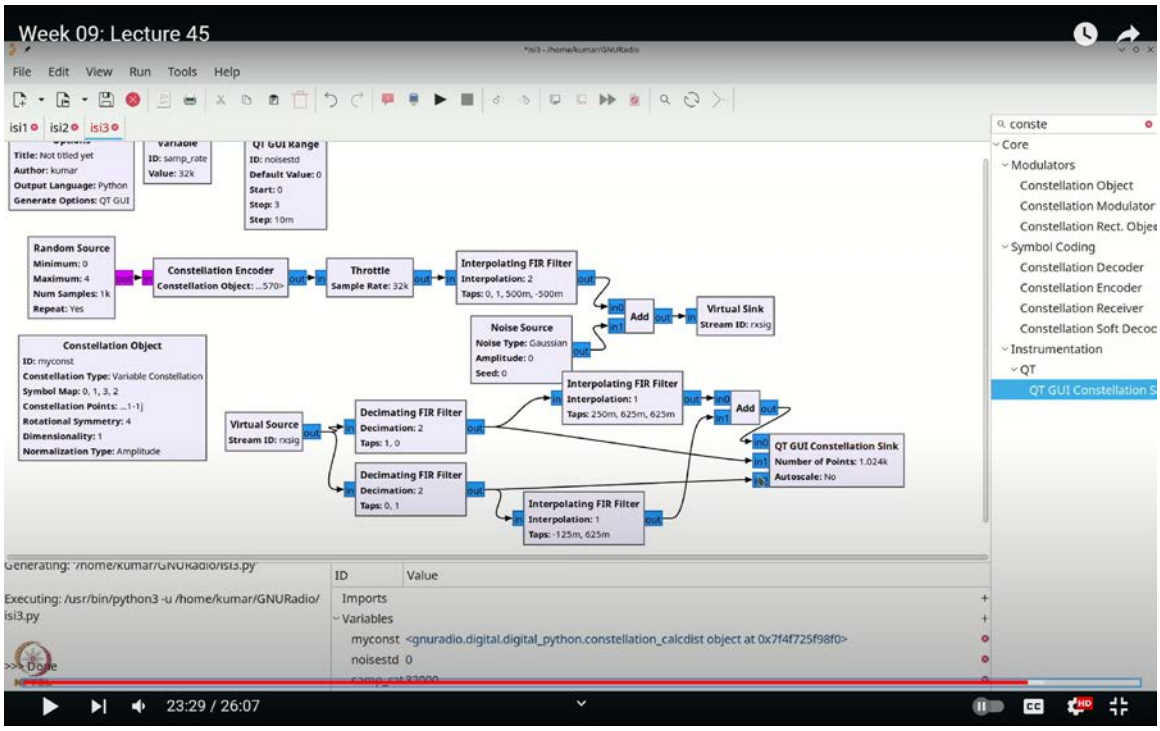


While the signal appears relatively clean, using advanced equalization methods like the Viterbi algorithm-based maximum likelihood sequence estimation can provide much better results, especially when noise levels are low. The trade-off is the increased computational complexity. Blindly using a zero-forcing equalizer in such scenarios can lead to severe noise enhancement, which is evident from our current visualization.

Now, let's evaluate the running example we discussed in class using GNU Radio to see how it performs. Recall that our transmit pulse was a rectangular pulse with an amplitude of 1 from 0 to 2 seconds. We modeled the effective channel with two impulses: one with an amplitude of 1 at one sample and another with an amplitude of -0.5 at the second sample.

When you convolve GTX and GC, the resulting P(T) appears as follows: between the first and second samples, it is 1; between the second and third samples, it is 0.5; and between the third and fourth samples, it is -0.5. Given that the symbol rate is two samples per symbol, the length of P(T) exceeds one symbol's length, which inevitably introduces inter-symbol interference.

(Refer Slide Time: 23:29)



Let's now model this scenario using GNU Radio. We'll start by adding a random source. To do this, press Control + F (or Command + F) and search for "random" to add the random source block. Since we are using a QPSK constellation, double-click on the random source block and set its maximum value to 4, and choose the data type as byte.

Next, we need to add a constellation object and an encoder that utilizes this object. To do this, press Control + F (or Command + F) and search for "CONST" to find the constellation object. Select the default QPSK constellation, rename it to "myconst," and connect it to the constellation encoder. Double-click on the constellation encoder block and set its constellation object to "myconst."

Before proceeding, let's add a throttle block to ensure that the simulation doesn't overload our computer. Press Control + F (or Command + F), type "throttle," and add the throttle block to your flow graph.

Now, let's incorporate the channel effects. We'll account for the impact of P(T), which has a response of 0, 1, 0.5, -0.5, with symbols sent at a rate of two samples per symbol. Press Control + F (or Command + F), and search for "interpolating FIR filter" to add this block. Set the interpolation factor to 2 to match the samples per symbol, and configure the taps to 0, 1, 0.5, -0.5.

To verify that we're achieving the desired shape, let's add a time sink. Press Control + F (or Command + F), type "time," and add the time sink block. Temporarily set the interpolation to 20 to better visualize the response. Run the simulation and use a stem plot to observe the output. Focus on the blue trace; you should see a pattern that closely resembles 1, 0.5, -0.5. Note that the actual value is slightly adjusted to 0.7 because the constellation points are in the form $\frac{1}{\sqrt{2}} + j\frac{1}{\sqrt{2}}$. The amplitude of half and negative half, with a total of 20 samples in between, confirms the response is correct. Reset the interpolation to 2 and remove the time sink once verification is complete.

To finalize the transmission-to-reception phase, we need to add noise. We'll use a QTGUI range to control the noise level. Press Control + F (or Command + F), search for "range," and add the QTGUI range block. Set the default value to 0, the maximum to 3, and the step

to 0.01.

Next, add a noise source block by pressing Control + F (or Command + F) and searching for "noise." Double-click the noise block and set the noise standard deviation (noise std) to be controlled by the QTGUI range. Name this range block "noise std."

Finally, add an "add" block by pressing Control + F (or Command + F) and searching for "add." Connect this block to our signal path. To visualize the output, add a virtual sink by pressing Control + F (or Command + F), searching for "virtual sink," and connecting it to the output. Name the stream "RxSig."

With these steps, we have set up the complete signal processing flow in GNU Radio.

Now we need to apply zero-forcing equalization to the signal **`RxSig`**. Let's recall the optimal zero-forcing equalizer for sampling at twice the symbol rate. According to our lecture, the effective channel P(T), when sampled at twice the symbol rate, can be represented as two channels. One channel has the form $1 - \frac{1}{2}Z^{-1}$, corresponding to the part of P(T) where the response is 1 followed by $-\frac{1}{2}$ two samples later. The second channel has the form $\frac{1}{2}$. Therefore, we have $H_1(Z) = 1 - \frac{1}{2}Z^{-1}$ and $H_2(Z) = \frac{1}{2}$.

To design the equalizer, we need it to satisfy $\sum H_i(Z)G_i(Z) = Z^{-D}$. This means we want to combine them to get either one or just a causal delay, a perfect delay.

From our matrix operations, we determined that the zero-forcing equalizer coefficients are $\frac{5}{8}, \frac{5}{8}, \frac{5}{8}, -\frac{1}{8}, \frac{2}{8}$. When you reverse and carefully select the coefficients, you get $-\frac{1}{8}$ and $\frac{5}{8}$ for $G_1(Z)$, and $\frac{2}{8} + \frac{5}{8}Z^{-1} + \frac{5}{8}Z^{-2}$ for $G_2(Z)$. You can verify that $H_1(Z)G_1(Z) + H_2(Z)G_2(Z)$ yields Z^{-1}, confirming our equalizer design.

We'll now implement G_1 and G_2 in GNU Radio to verify their effectiveness in equalizing the signal. First, add a virtual source to provide the signal we want to equalize. Double-click the virtual source and name it **`RxSig`**.

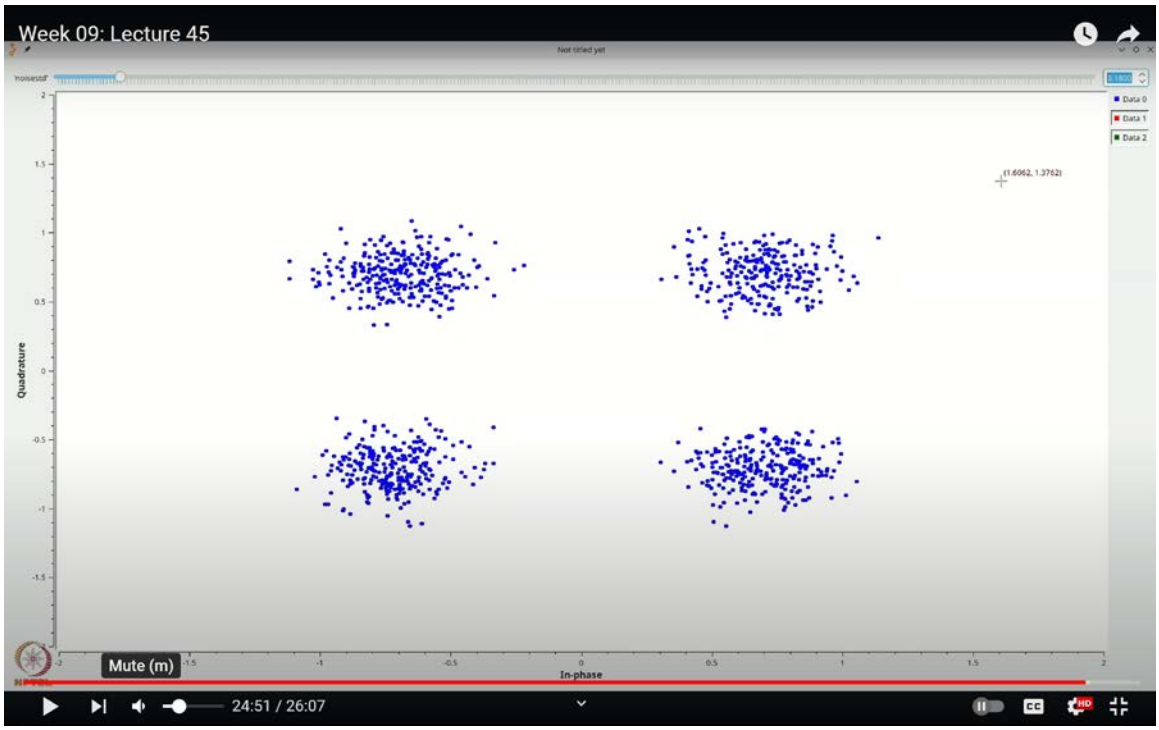
Since we are sampling at twice the symbol rate, we need to split **`RxSig`** into two parts.

We'll add two decimating filters to take alternate samples. Press Control + F (or Command + F), search for "decimating FIR filter," and add it to the flow graph. Set the decimation factor to 2. For the first filter, configure it to take the first set of samples. Copy and paste this filter, and configure the second filter to take the second set of samples, effectively processing the alternate samples.

Now connect these filters. You will have the first set of samples corresponding to 0 and $\frac{1}{2}$ and the second set corresponding to 1 and $-\frac{1}{2}$.

Next, add the filters corresponding to $H_1(Z)$ and $H_2(Z)$. For $H_1(Z)$, use a filter with coefficients $-\frac{1}{8}$ and $\frac{5}{8}$. For $H_2(Z)$, use coefficients $\frac{2}{8}, \frac{5}{8}, \frac{5}{8}$. Add an interpolating FIR filter for this purpose. Press Control + F (or Command + F), search for "interpolating FIR filter," and configure the interpolation to 1 with the specified coefficients. Duplicate and configure another interpolating FIR filter with the coefficients for $H_2(Z)$.

(Refer Slide Time: 24:51)



Finally, use an add block to combine the outputs of these filters. Press Control + F (or

Command + F), search for "add," and add the block to your flow graph. Connect the filters to the add block to obtain the desired response.

To confirm the effectiveness of the equalization, add a constellation sink. Press Control + F (or Command + F), search for "QTGUI constellation sink," and add it to the flow graph. Run the simulation to observe the constellation and verify that it is well-formed, indicating successful equalization.

Even with added noise, the constellation remains equalized effectively. To verify this, let's take an additional step. We'll add three inputs to our constellation sink. Set the number of inputs to 3 and label them as follows: the first input as blue, the second as red, and the third as magenta. Now, we'll connect the outputs of our two filters to these inputs. I'll make this setup visible for you.

Observe the constellation as we add some noise. You'll notice that the red trace looks quite good, while the magenta trace appears less clear. Why is this? The red trace represents the output of $H_2(Z)$, which is essentially $\frac{1}{2}$. Recall that P(T) had components $\frac{1}{2}$ and $-\frac{1}{2}$. With proper sampling, one part of the signal will not experience any inter-symbol interference (ISI), as the first and last parts are where ISI will be encountered. Thus, you see this trace with an amplitude of $\frac{1}{2} \times \frac{1}{\sqrt{2}}$.

However, for proper systematic zero-forcing equalization, you need to achieve this exact result: it eliminates the ISI and restores the signal to its correct amplitude. This is the essence of zero-forcing equalization.

Keep in mind that adding more noise can degrade performance. As seen previously with ISI, zero-forcing equalization is not very robust to noise. This technique can amplify noise, particularly in low SNR scenarios, leading to significant performance degradation.

In this lecture, we demonstrated a practical approach to implementing zero-forcing equalization using GNU Radio. While it effectively cancels interference under limited noise conditions, its performance declines in lower SNR environments due to increased noise amplification.

In the next lecture, we'll explore an alternative suboptimal equalization method that performs somewhat better than zero-forcing equalization under certain conditions and assess its effectiveness in GNU Radio. Thank you.