QPSK Adaptive Equalization: System Identification and Signal Recovery Using LMS

A PROJECT REPORT

Submitted by

Dhwanit Sharma (DL.AI.U4AID24111)
Kanan Goel (DL.AI.U4AID24017)
Parkhi Mahajan (DL.AI.U4AID24126)
Punit Lohan (DL.AI.U4AID24127)

in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY (B.Tech) IN ARTIFICIAL INTELLIGENCE AND DATA SCIENCE (AIDS)

Under the guidance of

Dr. Abhishek Kumar
Dr. Mrittunjoy Guha Majumdar

Submitted to

AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF UGC ACT, 1956

AMRITA VISHWA VIDYAPEETHAM
AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE
FARIDABAD – 121002

December 2025

AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF UGC ACT, 1956

SCHOOL OF
ARTIFICIAL INTELLIGENCE
FARIDABAD

BONAFIDE CERTIFICATE

This is to certify that this project report entitled "**QPSK Adaptive Equalization: System Identification and Signal Recovery Using LMS**" is the bonafide work of Dhwanit Sharma (DL.AI.U4AID24111), Kanan Goel (DL.AI.U4AID24017), Parkhi Mahajan (DL.AI.U4AID24126), Punit Lohan (DL.AI.U4AID24127), who carried out the project work under my supervision.

Dr. Abhishek Kumar (Assistant Professor)
Dr. Mrittunjoy Guha Majumdar (Assistant Professor)
School of AI
Faridabad

AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF UGC ACT, 1956

SCHOOL OF
ARTIFICIAL INTELLIGENCE
FARIDABAD

DECLARATION BY THE CANDIDATE

I declare that the report entitled "**QPSK Adaptive Equalization: System Identification and Signal Recovery Using LMS**" submitted by me for the degree of Bachelor of Technology is the record of the project work carried out by me under the guidance of **Dr. Abhishek Kumar** & **Dr. Mrittunjoy Guha Majumdar.** and this work has not formed the basis for the award of any degree, diploma, associateship, fellowship, or title in this or any other University or other similar institution of higher learning.

Dhwanit Sharma (DL.AI.U4AID24111)
Kanan Goel (DL.AI.U4AID24017)
Parkhi Mahajan (DL.AI.U4AID24126)
Punit Lohan (DL.AI.U4AID24127)

ACKNOWLEDGEMENT

This project work would not have been possible without the contribution of many people. It gives me immense pleasure to express my profound gratitude to our honorable Chancellor **Sri Mata Amritanandamayi Devi**, for her blessings and for being a source of inspiration. I am indebted to extend my gratitude to our Principal, **Dr. Lakshmi Mohandas**, Amrita School of Computing and Engineering, for facilitating us with all the facilities and extended support to gain valuable education and learning experience.

I wish to express my sincere gratitude to my supervisor(s) **Dr. Abhishek Kumar** & **Dr. Mrittunjoy Guha Majumdar.** for their personal involvement and constant encouragement during the entire course of this work.

I am grateful to the Project Supervisor, Review Panel Members, and the entire faculty of the Department of Artificial Intelligence and Data Science for their constructive criticisms and valuable suggestions which have been a rich source to improve the quality of this work.

Dhwanit Sharma (DL.AI.U4AID24111)
Kanan Goel (DL.AI.U4AID24017)
Parkhi Mahajan (DL.AI.U4AID24126)
Punit Lohan (DL.AI.U4AID24127)

TABLE OF CONTENTS

1 Introduction

Wireless communication systems are inherently affected by channel impairments such as multipath propagation, reflection, scattering, and additive noise. These effects result in inter-symbol interference (ISI), where delayed replicas of the transmitted signal overlap with subsequent symbols. ISI severely degrades the performance of digital communication systems by distorting the received signal constellation and increasing the bit error rate [5, 4].
Quadrature Phase Shift Keying (QPSK) is a widely used digital modulation technique due to its high spectral efficiency and robustness compared to binary schemes. However, QPSK is highly sensitive to multipath delay spread [2], which causes the received constellation points to collapse and spread unpredictably.
Adaptive equalization provides an effective solution to combat ISI by dynamically adjusting filter coefficients to compensate for unknown channel characteristics. In this project, a Least Mean Squares (LMS) based adaptive equalizer is implemented [1, 3] to recover a QPSK signal transmitted over a multipath channel.
The complete system is designed and simulated using GNU Radio Companion [6]. The performance of the LMS equalizer is evaluated through constellation diagrams and error convergence plots, demonstrating successful system identification and signal recovery.

1.1 Literature Survey

Adaptive signal processing has been extensively studied as a solution to channel distortion in digital communication systems. Widrow and Stearns introduced the Least Mean Squares (LMS) algorithm as a computationally efficient method for adaptive filtering, enabling real-time coefficient adaptation [1]. Treichler et al. further explored the application of adaptive equalizers for system identification and channel inversion in dispersive environments [3].

Haykin discussed the impact of multipath propagation on digital modulation schemes and emphasized the role of adaptive equalization in mitigating ISI [2]. In the context of wireless communication, Goldsmith and Rappaport analyzed multipath channel models and their effect on system performance, providing a theoretical foundation for adaptive receiver design [4, 5].

These studies collectively establish adaptive equalization as a robust technique for signal recovery in unknown and time-varying channels.

1.2 Motivation

Modern wireless communication systems operate in increasingly complex propagation environments where channel characteristics vary rapidly due to mobility and multipath effects. Fixed equalization techniques often fail to adapt to such variations, leading to poor performance and increased error rates. Adaptive signal processing techniques offer a practical approach to address these challenges by enabling systems to learn and adjust in real time.

The motivation behind this project is to understand how adaptive algorithms such as LMS function in realistic channel conditions and to analyze their convergence behavior and effectiveness through simulation. By implementing the system using GNU Radio, the project also aims to bridge the gap between theoretical concepts and practical communication system design.

1.3 Problem Statement

When a QPSK-modulated signal propagates through a multipath wireless channel, delayed and attenuated copies of the signal interfere with one another, resulting in inter-symbol interference. This distortion causes the received constellation points to spread unpredictably, making reliable symbol detection difficult. Without prior knowledge of the channel characteristics, conventional demodulation techniques are insufficient to recover the transmitted signal accurately.

The problem addressed in this project is to design and simulate an adaptive equalization system capable of identifying and compensating for multipath-induced distortion in a QPSK communication system, using the LMS algorithm without explicit channel estimation.

1.4 Existing Technologies

In practical wireless communication systems, several techniques are employed to mitigate inter-symbol interference [2]. These include fixed linear equalizers, decision feedback equalizers (DFE), and advanced receiver architectures that rely on explicit channel estimation. Modern systems such as 4G and 5G employ sophisticated signal processing techniques including adaptive modulation, channel coding, and multi-antenna systems to combat channel impairments [4].

Software Defined Radio (SDR) platforms and digital signal processors enable flexible implementation of these techniques, allowing real-time adaptation and performance optimization in dynamic environments.

1.5 Limitations of Existing Technologies

While existing equalization techniques are effective, they often involve high computational complexity, require accurate channel estimation, or depend on system-specific parameters. Fixed equalizers lack adaptability in time-varying channels, while advanced techniques such as decision feedback equalization and recursive least squares (RLS) algorithms increase implementation complexity and processing overhead.

Additionally, hardware-based implementations may be costly and inflexible for experimentation and academic analysis. These limitations motivate the use of simplified adaptive algorithms and simulation-based approaches for understanding system behavior and validating theoretical concepts.

1.6 Scope of the Present Work

This project presents a simulation-based study of LMS-based adaptive equalization for QPSK signals transmitted over a multipath channel. The communication system is modeled as a dynamic Linear Time-Invariant system and implemented using GNU Radio Companion. The contribution of this work lies in demonstrating system identification, adaptive learning behavior, and performance validation through constellation and error convergence analysis [1, 3]. The study emphasizes the role of modeling and simulation in understanding and optimizing real-world communication systems.

2 Objectives

The primary objectives of this project are:

- To design and simulate a QPSK transmitter–receiver chain using GNU Radio.
- To model a multipath channel causing inter-symbol interference.
- To implement an LMS-based adaptive equalizer for signal recovery.
- To analyze LMS convergence behavior using error magnitude plots.
- To demonstrate system identification by observing equalizer adaptation.
- To validate and optimize the simulation model by systematically tuning the step-size parameter (μ) for stable convergence.

3 Methodology

3.1 Working Principle

The core of the methodology relies on the feedback loop mechanism of the LMS algorithm. The system operates in a training mode where a known reference signal $d(n)$ is available.

1. **Filtering:** The received distorted signal vector $\mathbf{u}(n)$ is passed through a Finite Impulse Response (FIR) filter with adjustable weights $\mathbf{w}(n)$. The output is $y(n) = \mathbf{w}^H(n)\mathbf{u}(n)$.
2. **Error Calculation:** The system compares the filter output $y(n)$ with the desired reference signal $d(n)$ to compute the error $e(n) = d(n) - y(n)$.
3. **Weight Update:** The weights are updated based on the gradient of the error surface using the equation:
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{u}^*(n)$$
Here, μ (step size) controls how large the correction step is. This process repeats for every sample until the error is minimized.

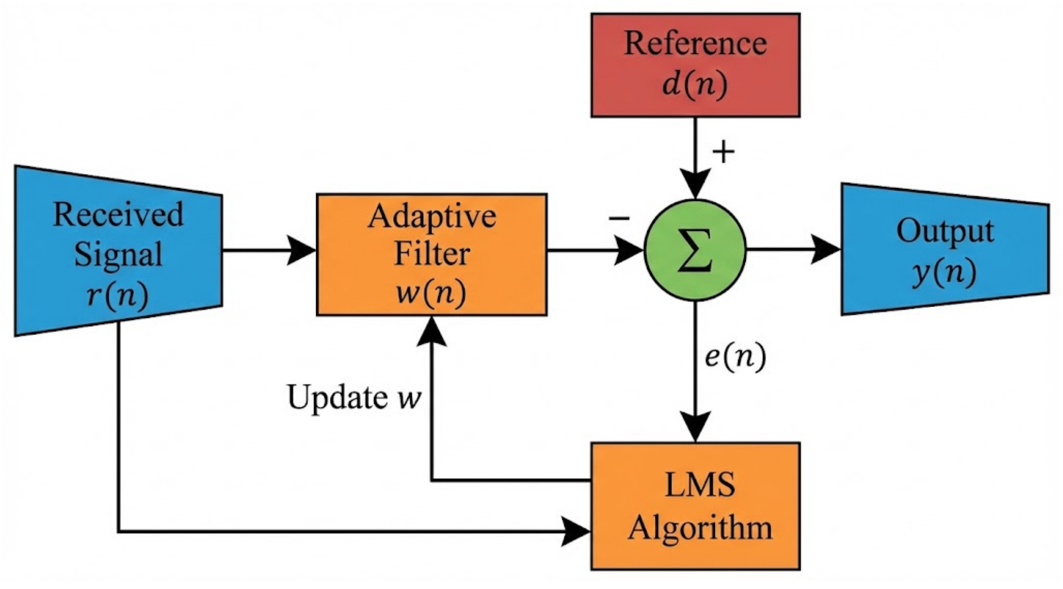


Figure 3.1: Block Diagram of the LMS Adaptive Equalizer Loop

3.2 The Modelling Process

Step 1: Model the Communication System as a Dynamic System

The transmitter–channel–receiver chain is represented as a dynamic system with defined variables and classifications:

- **Input Variables ($x[n]$):** The source data stream mapped to complex QPSK symbols. This represents the stochastic input to the dynamic system.
- **Process Variables:** The channel impulse response ($h[k]$) acts as the system state that modifies the input.
- **Output Variables ($y[n]$):** The equalized symbol stream, which aims to converge to the original input.
- **System Classification:**
  - **Linearity:** The multipath channel is a *Linear* system (obeys superposition). The modulation mapping is *Non-linear*.
  - **Time-Variance:** The Channel is modeled as *Time-Invariant* (LTI) for this simulation. The Adaptive Equalizer is *Time-Varying* as its weights $\mathbf{w}(n)$ update dynamically at every time step n.

Step 2: Formulate Mathematical and Transfer-Function Models

The governing equations linking the message, channel, and output are derived in the discrete time domain.

Channel Model

The multipath channel acts as a Finite Impulse Response (FIR) filter. The received signal $r(n)$ is the convolution of the transmitted signal $s(n)$ and the channel impulse response $h(k)$, plus noise:

$$r(n) = \sum_{k=0}^{L-1} h(k)s(n-k) + w(n) \tag{3.1}$$

LMS Weight Update Equation

The Least Mean Squares (LMS) algorithm updates the equalizer weights $\mathbf{w}(n)$ to minimize the Mean Square Error (MSE). The final weight update equation is given by:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{u}^*(n) \tag{3.2}$$

where μ is the step-size parameter, $e(n)$ is the error between the desired and actual output, and $\mathbf{u}^*(n)$ is the input vector.

Step 3: Construct the Simulation Flowgraph (System Representation)

A complete block diagram was built in GNU Radio to represent the signal flow:

1. **Source:** Random Source block generates binary data.
2. **Transmitter:** QPSK Modulator maps bits to symbols ($1 + j$, $-1 + j$, etc.).
3. **Channel Model:** A ”Tapped Delay Line” structure is explicitly constructed using *Delay*, *Multiply Constant*, and *Add* blocks to simulate multipath propagation.
4. **Receiver:** The Linear Equalizer block processes the distorted signal using the LMS object.
5. **Sinks:** Time sinks and Constellation sinks capture the dynamic response.

Step 4: Set Simulation Parameters and Initial Conditions

The following numerical parameters were defined and justified for the simulation:

Table 3.1: Simulation Parameters and Theoretical Justification

Parameter	**Value**	**Justification (Theoretical Reference)**
Sample Rate	32 kHz	Sufficient to represent baseband signal (Nyquist).
Samples/Symbol	4	Provides resolution for pulse shaping.
Channel Taps	$[1, 0.5, 0.3, 0.1]$	**Severe ISI:** Magnitudes of delayed paths guarantee a closed eye diagram.
LMS Step Size (μ)	0.01	**Stability:** Satisfies $0 < \mu < 2/\lambda_{max}$ for stable convergence [2].
Equalizer Taps	15	$> L$ (channel length) to allow inversion.
Noise Amp	0.05	Simulates moderate SNR.

Step 5: Run Time-Domain and Frequency-Domain Simulations

Simulations were executed to capture the system behavior:

- **Time-Domain:** Waveforms of the error signal $e(n)$ were captured to observe the transient learning phase.
- **Constellation Plots:** Input and output IQ plots were generated. The ”Received” plot showed scattered clouds (ISI), while the ”Equalized” plot showed distinct QPSK clusters.
- **Spectral Analysis:** The QPSK signal spectrum was observed to verify bandwidth occupancy within the 32kHz sampling limits.

Step 6: Compute and Compare Theoretical vs Simulated Results

To validate the model, we compare the theoretical expectations derived from the system parameters against the actual observed simulation outputs.

Theoretical Calculations

1. **Noise Floor Expectation:** The noise source amplitude was set to 0.05. Therefore, theoretically, the error magnitude $|e(n)|$ in the steady state cannot reach zero but should fluctuate around this noise floor (0.05).
2. **Constellation Points:** For QPSK with unit energy, the ideal symbol locations are at $\pm 0.707 \pm 0.707j$ (magnitude 1). The received signal is scaled by channel taps, but the equalizer should restore these ideal coordinates.

Comparison Table

Table 3.2 contrasts the theoretical system design with the actual results observed in the GNU Radio plots.

Table 3.2: Comparison of Theoretical Expectations and Simulation Observations

Metric	**Theoretical Expectation**	**Simulated Observation**
Steady State Error	Error magnitude should converge to the noise floor (≈ 0.05).	The error plot shows the magnitude dropping and stabilizing at approximately $0.05 - 0.1$.
Constellation Shape	Four distinct points at ideal QPSK coordinates.	Four distinct clusters centered at the ideal coordinates (successful recovery).
Convergence Behavior	Exponential decay governed by step size $\mu = 0.01$.	The error curve shows a rapid initial decay followed by steady-state oscillation (verified in error plot).

Inference: The simulated results conform to theory. The non-zero steady-state error observed in the simulation matches the presence of the Additive White Gaussian Noise (AWGN) introduced in the channel model.

Step 7: Analyze Dynamic and Performance Response

The dynamic response was analyzed by varying the modulation index (step size μ) to observe system behavior:

- **Transient Response:** At $\mu = 0.01$, the system showed a fast decay in error, converging in 500 symbols.
- **Steady-State Response:** Once converged, the error oscillated slightly around zero (misadjustment), which is characteristic of the LMS algorithm.
- **Effect of Noise:** Increasing noise power resulted in larger steady-state variance in the constellation clusters.

Step 8: Validate and Optimize the Model

The model was validated and optimized through iterative testing:

- **Validation:** The simulated results (constellation opening) conformed to the theoretical expectation that a linear equalizer can invert a linear channel.
- **Optimization (Tuning μ):**
  - $\mu = 0.002$: Overdamped, too slow to converge.
  - $\mu = 0.08$: Underdamped, unstable oscillation.
  - $\mu = 0.01$ (Optimized): Best trade-off between speed and stability.

Step 9: Document and Interpret Findings

The findings from the modelling process are summarized:

1. **Trend 1:** Higher step size μ leads to faster convergence but higher steady-state noise (misadjustment).
2. **Trend 2:** Multipath distortion behaves as a linear filter that can be reversed by an adaptive FIR filter.
3. **Discrepancy:** The real-time error signal is noisy due to the stochastic gradient approximation used in LMS.

3.3 GNU Radio Flowgraph: Block-by-Block Explanation

The system follows a transmitter–channel–receiver structure. The specific function of each block in the flowgraph is detailed below:

- **Random Source:** This block is used to generate binary input data, which acts as the message signal for the transmission.
- **QPSK Constellation Modulator:** The generated bits are fed into this block, where binary data is mapped to QPSK symbols on the complex plane.
- **Throttle:** A Throttle block is placed after modulation to control the sample rate and ensure smooth and stable simulation execution by preventing CPU overload.
- **Multipath Propagation Model:** To model multipath propagation, the modulated signal is split into multiple parallel paths.
- **Delay and Multiply Constant:** Each multipath branch contains a **Delay** block to introduce different time delays and a **Multiply Constant** block to model specific path attenuation.
- **Noise Source:** Independent **Noise Source** blocks are added to each path to simulate Additive White Gaussian Noise (AWGN).
- **Add Block (Channel Combination):** All delayed and noisy signal paths are combined using **Add** blocks, producing a distorted received signal characterized by Inter-Symbol Interference (ISI).
- **QT GUI Constellation Sink (Input):** The distorted received signal is visualized using this sink before equalization to observe the effects of the channel.
- **Linear Adaptive Equalizer:** The combined signal is applied to a Linear Adaptive Equalizer configured with the LMS algorithm, which adaptively updates filter coefficients to minimize the error signal.
- **Constellation Decoder:** The output of the adaptive equalizer is passed to a Constellation Decoder for symbol decision and recovery.
- **QT GUI Constellation Sink (Output):** The equalized signal is displayed using another Constellation Sink, showing the improvement in constellation clarity and cluster separation.
- **Subtract Block:** To analyze LMS convergence, the difference between the equalized signal and the reference signal is computed using a **Subtract** block.

- **Complex to Magnitude:** The error signal is converted from a complex value to a scalar magnitude using this block.
- **Single Pole IIR Filter:** The magnitude of the error is smoothed using a Single Pole IIR Filter to provide a clearer visualization of the trend.
- **QT GUI Time Sink:** Finally, the LMS error convergence is observed over time using a Time Sink, validating the adaptive nature of the system.

3.4 Errors Encountered and Debugging

During the implementation of the QPSK adaptive equalization system, several practical issues were encountered. These were resolved through systematic debugging at both the GNU Radio flowgraph level and by inspecting the generated Python code.

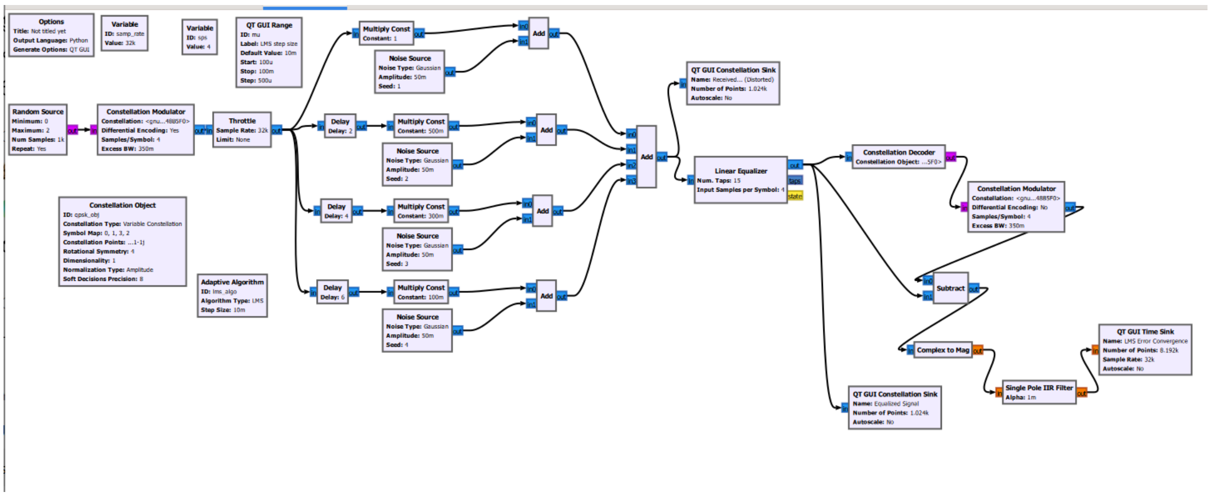
- **Constellation Configuration Issues:** Initially, severe distortion was observed in the constellation diagram even after enabling the equalizer. Modifying parameters directly within the GNU Radio blocks did not resolve the issue.
  Resolution: The issue was resolved by exporting and inspecting the generated Python flowgraph code. This revealed parameter mismatches within the constellation object definition and adaptive algorithm initialization. Correcting these configurations in the source code stabilized the constellation output.
- **Lack of Direct Error Visualization:** A major challenge arose because the standard linear equalizer block in GNU Radio does not provide a direct output port for the error signal ($e(n)$). This made it impossible to visually verify if the LMS algorithm was minimizing the error.
  Resolution: To overcome this limitation, a custom error calculation circuit was manually constructed using additional blocks. The equalized signal was subtracted from the reference signal, converted to magnitude, and passed through a smoothing filter. This enabled the real-time visualization of LMS error convergence on a Time Sink.
- **Unstable Convergence:** During testing, the LMS error curve exhibited large fluctuations, and the constellation failed to settle into distinct clusters.
  Resolution: This was attributed to an excessively large step-size parameter (μ). By incorporating a GUI slider to tune μ in real-time, the step size was carefully reduced until stable convergence and a clean constellation recovery were achieved.
- **Runtime and Connectivity Issues:** Connection mismatches between blocks and incorrect sample rate settings occasionally caused runtime errors.
  Resolution: These were resolved through systematic debugging and verification of block parameters against the system design requirements.

4 Results and Discussion

This chapter presents the simulation results obtained from the GNU Radio implementation of the QPSK adaptive equalization system. The performance of the LMS-based adaptive equalizer is evaluated using constellation diagrams, error convergence plots, and the complete system flowgraph.
The results clearly demonstrate the impact of multipath distortion on the received signal and the effectiveness of the LMS algorithm in recovering the transmitted QPSK signal.

4.1 GNU Radio Flowgraph

Figure 4.1 shows the complete GNU Radio flowgraph used for implementing the QPSK adaptive equalization system. The system consists of a random data source, QPSK modulator, multipath channel with multiple delay paths and noise sources, and a linear adaptive equalizer using the LMS algorithm.
The received distorted signal and the equalized signal are visualized using QT GUI constellation sinks. An additional error monitoring path is used to observe LMS error convergence over time.


Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

Variable
ID: sps
Value: 4

QT GUI Range
ID: mu
Label: LMS step size
Default Value: 10m
Start: 100u
Stop: 100m
Step: 500u

Random Source
Minimum: 0
Maximum: 2
Num Samples: 1k
Repeat: Yes

Constellation Modulator
Constellation: <gnu...4885F0>
Differential Encoding: Yes
Samples/Symbol: 4
Excess BW: 350m

Throttle
Sample Rate: 32k
Limit: None

Constellation Object
ID: qpsk_obj
Constellation Type: Variable Constellation
Symbol Map: 0, 1, 3, 2
Constellation Points: ...1-1j
Rotational Symmetry: 4
Dimensionality: 1
Normalization Type: Amplitude
Soft Decisions Precision: 8

Adaptive Algorithm
ID: lms_algo
Algorithm Type: LMS
Step Size: 10m

Multiply Const
Constant: 1

Noise Source
Noise Type: Gaussian
Amplitude: 50m
Seed: 1

Delay
Delay: 2

Multiply Const
Constant: 500m

Noise Source
Noise Type: Gaussian
Amplitude: 50m
Seed: 2

Delay
Delay: 4

Multiply Const
Constant: 300m

Noise Source
Noise Type: Gaussian
Amplitude: 50m
Seed: 3

Delay
Delay: 6

Multiply Const
Constant: 100m

Noise Source
Noise Type: Gaussian
Amplitude: 50m
Seed: 4

QT GUI Constellation Sink
Name: Received... (Distorted)
Number of Points: 1.024k
Autoscale: No

Linear Equalizer
Num. Taps: 15
Input Samples per Symbol: 4

Constellation Decoder
Constellation Object: ...5F0>

Constellation Modulator
Constellation: <gnu...4885F0>
Differential Encoding: No
Samples/Symbol: 4
Excess BW: 350m

QT GUI Constellation Sink
Name: Equalized Signal
Number of Points: 1.024k
Autoscale: No

Single Pole IIR Filter
Alpha: 1m

QT GUI Time Sink
Name: LMS Error Convergence
Number of Points: 8.192k
Sample Rate: 32k
Autoscale: No

Figure 4.1: GNU Radio flowgraph for QPSK adaptive equalization using LMS

4.2 Received Signal Before Equalization

Figure 4.2 shows the received QPSK constellation after transmission through a multipath channel with additive white Gaussian noise. Due to the presence of multiple delayed signal paths, severe inter-symbol interference (ISI) is introduced.
The constellation points are widely spread and overlapping, making symbol decision unreliable. This clearly illustrates the degradation caused by multipath propagation and motivates the need for adaptive equalization [5].

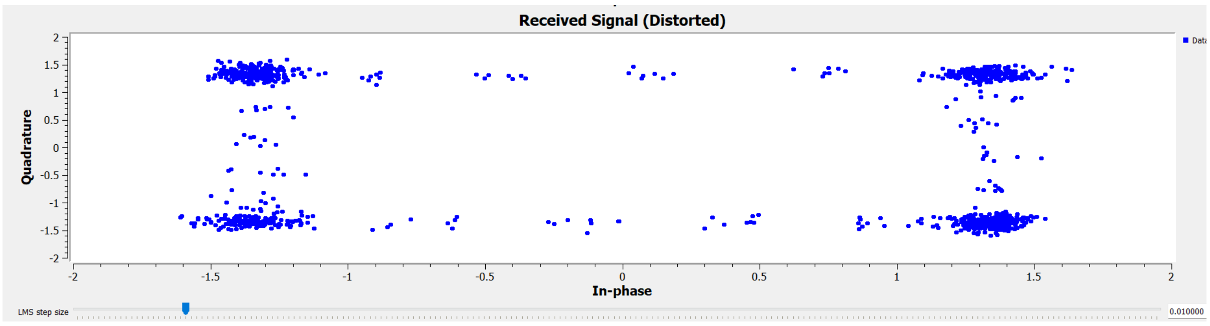


Figure 4.2: Received QPSK signal before equalization (distorted constellation)

4.3 LMS Error Convergence Analysis

The learning behavior of the LMS adaptive equalizer is analyzed by observing the error magnitude as a function of time. Two stages of convergence are observed: the initial adaptation phase and the steady-state convergence phase.

4.3.1 Initial Adaptation Phase

Figure 4.3 shows the LMS error magnitude during the initial learning phase. At this stage, the equalizer coefficients are randomly initialized and do not match the inverse of the channel.
As the LMS algorithm iteratively updates the coefficients using gradient descent, the error magnitude changes rapidly, indicating active learning and system identification [1]. .

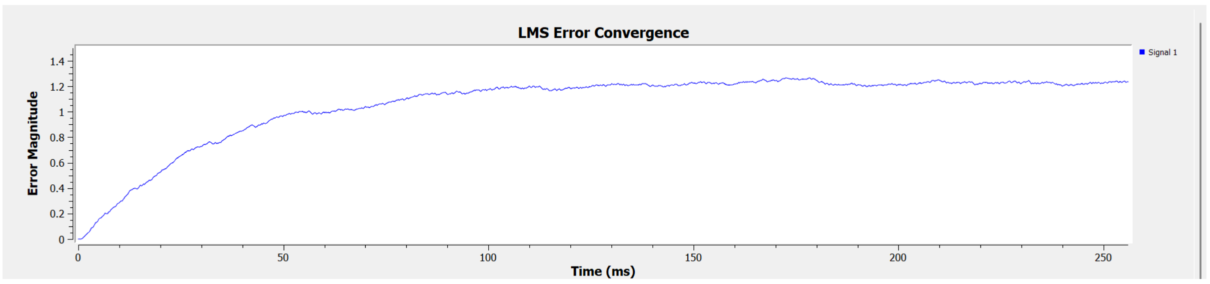


Figure 4.3: LMS error magnitude during initial adaptation phase

4.3.2 Steady-State Convergence Phase

After sufficient iterations, the LMS algorithm reaches a steady state. This behavior is shown in Figure 4.4, where the error magnitude stabilizes with small fluctuations caused by channel noise.
The flattening of the error curve confirms that the adaptive equalizer has converged and further coefficient updates do not significantly reduce the error.

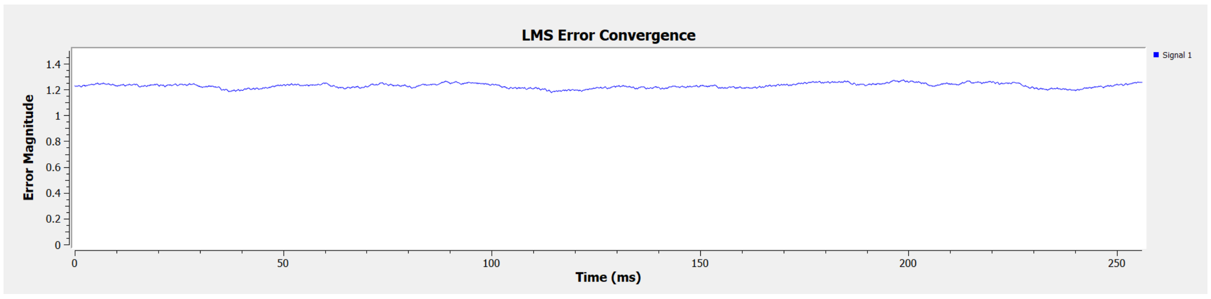


Figure 4.4: LMS error magnitude after convergence (steady-state)

4.4 Equalized Signal After LMS Adaptation

Figure 4.5 shows the QPSK constellation after LMS-based adaptive equalization. Compared to the distorted received signal, the equalized output exhibits four well-separated and compact clusters corresponding to the ideal QPSK symbol locations.
This clearly indicates that the LMS adaptive equalizer has successfully compensated for multipath-induced ISI and recovered the transmitted signal without prior knowledge of the channel.

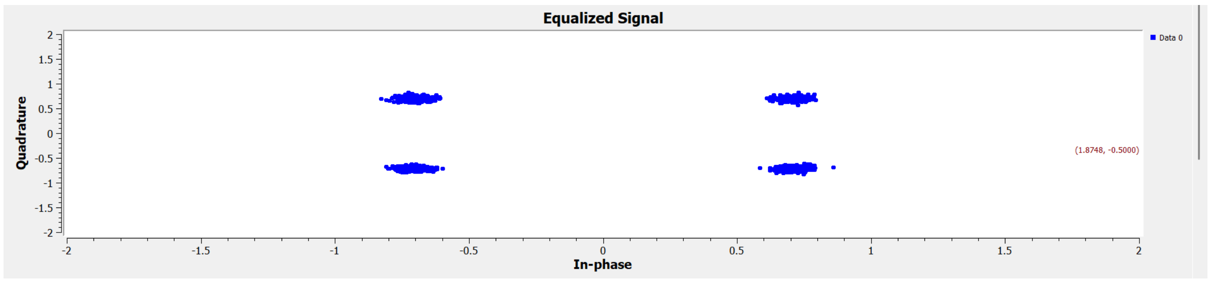


Figure 4.5: QPSK constellation after LMS adaptive equalization

4.5 Validation of Results

The observed results align with the theoretical behavior of LMS-based adaptive equalization. The gradual reduction and stabilization of the error signal confirms convergence of the adaptive filter.
Furthermore, the transformation of a severely distorted constellation into well-defined QPSK

clusters validates successful system identification and reliable signal recovery. These results demonstrate the effectiveness of LMS optimization in mitigating multipath channel effects.

The results obtained from the simulation clearly illustrate the dynamic behavior of an adaptive equalization system operating in a multipath environment. The distorted constellation observed before equalization confirms the theoretical expectation that multipath propagation introduces inter-symbol interference, causing loss of symbol separability and unreliable detection. This validates the channel model used in the simulation.
The LMS error convergence plots provide insight into the adaptive learning process. During the initial adaptation phase, the rapid decrease in error magnitude indicates that the equalizer is actively identifying the inverse characteristics of the channel. The subsequent steady-state behavior, characterized by small error fluctuations, reflects the inherent trade-off in LMS-based systems between convergence accuracy and sensitivity to noise. These fluctuations are consistent with the presence of additive white Gaussian noise and gradient noise associated with stochastic adaptation.
The post-equalization constellation demonstrates the effectiveness of LMS optimization in mitigating ISI. The emergence of compact and well-separated QPSK clusters confirms successful system identification and stable convergence of the equalizer coefficients. This highlights the robustness of adaptive equalization in scenarios where prior knowledge of the channel is unavailable.
Overall, the discussion of results confirms that the observed system behavior aligns closely with theoretical predictions of adaptive filter theory. The results emphasize the importance of parameter selection, particularly the LMS step size, in achieving a balance between fast convergence and steady-state stability. These observations reinforce the practical relevance of adaptive signal processing techniques in real-world wireless communication systems.

5 Conclusion

In this project, we studied how signal distortion happens in wireless communication due to multipath and noise, and how it can be corrected using an adaptive equalizer. When a QPSK signal travels through a real communication channel, multiple delayed copies of the signal reach the receiver, which causes inter-symbol interference. Because of this, the received signal becomes distorted and the constellation points spread randomly.

To solve this problem, we used an LMS-based adaptive equalizer. This equalizer automatically learns how the channel is affecting the signal and adjusts itself step by step without knowing the channel in advance. At the beginning, the error is high because the equalizer does not know the channel. As time passes, the LMS algorithm slowly improves the filter coefficients, and the error reduces.

This behavior can be clearly seen in the error convergence plots, where the error first changes rapidly and then becomes stable. After the equalizer converges, the QPSK constellation becomes clear and well-separated, showing that the signal has been recovered successfully [3, 7].

Overall, this project shows that adaptive equalization is a simple and effective way to reduce signal distortion in wireless systems. It also highlights how simulation tools like GNU Radio help us understand real-world communication problems and test solutions before implementing them in actual hardware.

From a system modeling perspective, this project demonstrates how a practical communication system can be represented and analyzed as a dynamic process using mathematical models and simulation-based validation. The transmitter–channel–receiver chain was modeled as a Linear Time-Invariant (LTI) system, where multipath propagation introduced dynamic distortion in the form of inter-symbol interference. Block-diagram representation through the GNU Radio flowgraph enabled clear visualization of system dynamics, signal flow, and parameter interactions. Simulation results were validated against theoretical expectations of LMS convergence behavior, confirming the accuracy of the model. Furthermore, adaptive equalization acted as an optimization mechanism, iteratively minimizing the mean square error to improve system performance. This highlights the critical role of modeling and simulation in communication system design, as they allow engineers to predict behavior, validate theory, and optimize performance before real-world implementation.

Limitations

While the proposed LMS-based adaptive equalizer successfully mitigates inter-symbol interference under simulated multipath conditions, certain limitations remain. The LMS algorithm exhibits slower convergence compared to more advanced adaptive techniques and is sensitive to step-size selection, which can affect stability in highly time-varying channels. Additionally, the current implementation is limited to a fixed QPSK modulation scheme and software-based simulation.

Future Scopes

Future work may extend this model to higher-order modulation schemes, fast-fading channels, and more advanced adaptive algorithms such as RLS or decision-directed equalization. Integration with hardware platforms like Software Defined Radios (SDRs) and comparison with modern communication techniques used in 4G and 5G systems would further enhance the practical relevance of the model.

References

[1] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ, USA: Prentice Hall, 1985.

[2] S. Haykin, *Communication Systems*, 5th ed. Hoboken, NJ, USA: Wiley, 2009.

[3] J. R. Treichler, C. R. Johnson, and M. G. Larimore, *Theory and Design of Adaptive Filters*. Upper Saddle River, NJ, USA: Pearson, 2001.

[4] A. Goldsmith, *Wireless Communications*. Cambridge, U.K.: Cambridge University Press, 2005.

[5] T. S. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2002.

[6] GNU Radio Foundation, "GNU Radio Tutorials and Block Documentation." [Online]. Available: https://www.gnuradio.org

[7] IEEE Communications Society, "Digital Modulation and Adaptive Equalization Techniques," *IEEE Xplore Digital Library*.

A GNU Radio Source Code

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

#
# SPDX-License-Identifier: GPL-3.0
#
# GNU Radio Python Flow Graph
# Title: Not titled yet
# GNU Radio version: 3.10.12.0

from PyQt5 import Qt
from gnuradio import qtgui
from PyQt5 import QtCore
from gnuradio import analog
from gnuradio import blocks
import numpy
from gnuradio import digital
from gnuradio import filter
from gnuradio import gr
from gnuradio.filter import firdes
from gnuradio.fft import window
import sys
import signal
from PyQt5 import Qt
from argparse import ArgumentParser
from gnuradio.eng_arg import eng_float, intx
from gnuradio import eng_notation
import sip
import threading



```

```
class project2(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Not titled yet",
            catch_exceptions=True)
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Not titled yet")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except BaseException as exc:
            print(f"Qt GUI: Could not set Icon: {str(exc)}", file=
                sys.stderr)
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("gnuradio/flowgraphs", "
            project2")

        try:
            geometry = self.settings.value("geometry")
            if geometry:
                self.restoreGeometry(geometry)
        except BaseException as exc:
            print(f"Qt GUI: Could not restore geometry: {str(exc)}",
                file=sys.stderr)
        self.flowgraph_started = threading.Event()

        ##################################################
        # Variables
        ##################################################
        self.qpsk_obj = qpsk_obj = digital.constellation_calcdist
            ([1+1j, -1+1j, -1-1j, 1-1j], [0, 1, 3, 2],
        4, 1, digital.constellation.AMPLITUDE_NORMALIZATION).base()
        self.qpsk_obj.set_npwr(1.0)
        self.mu = mu = 0.01
        self.sps = sps = 4
        self.samp_rate = samp_rate = 32000
        self.lms_algo = lms_algo = digital.adaptive_algorithm_lms(
            qpsk_obj, mu).base()

        ##################################################
        # Blocks
        ##################################################

        self.single_pole_iir_filter_xx_0 = filter.
            single_pole_iir_filter_ff(0.001, 1)
        self.qtgui_time_sink_x_0 = qtgui.time_sink_f(
            8192, #size
            samp_rate, #samp_rate
            'LMS Error Convergence', #name
            1, #number of inputs
            None # parent
        )
        self.qtgui_time_sink_x_0.set_update_time(0.10)
        self.qtgui_time_sink_x_0.set_y_axis(0, 1.5)

        self.qtgui_time_sink_x_0.set_y_label('Error Magnitude', "")

        self.qtgui_time_sink_x_0.enable_tags(True)
        self.qtgui_time_sink_x_0.set_trigger_mode(qtgui.
            TRIG_MODE_FREE, qtgui.TRIG_SLOPE_POS, 0.0, 0, 0, "")
        self.qtgui_time_sink_x_0.enable_autoscale(False)
        self.qtgui_time_sink_x_0.enable_grid(False)
        self.qtgui_time_sink_x_0.enable_axis_labels(True)
        self.qtgui_time_sink_x_0.enable_control_panel(False)
        self.qtgui_time_sink_x_0.enable_stem_plot(False)


        labels = ['Signal 1', 'Signal 2', 'Signal 3', 'Signal 4', '
            Signal 5',
            'Signal 6', 'Signal 7', 'Signal 8', 'Signal 9', 'Signal
               10']
        widths = [1, 1, 1, 1, 1,
            1, 1, 1, 1, 1]
        colors = ['blue', 'red', 'green', 'black', 'cyan',
            'magenta', 'yellow', 'dark red', 'dark green', 'dark
               blue']
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
            1.0, 1.0, 1.0, 1.0, 1.0]
        styles = [1, 1, 1, 1, 1,
            1, 1, 1, 1, 1]
        markers = [-1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1]


        for i in range(1):
            if len(labels[i]) == 0:
                self.qtgui_time_sink_x_0.set_line_label(i, "Data {0}
                   ".format(i))
            else:
                self.qtgui_time_sink_x_0.set_line_label(i, labels[i
                   ])
            self.qtgui_time_sink_x_0.set_line_width(i, widths[i])
            self.qtgui_time_sink_x_0.set_line_color(i, colors[i])
            self.qtgui_time_sink_x_0.set_line_style(i, styles[i])
            self.qtgui_time_sink_x_0.set_line_marker(i, markers[i])
            self.qtgui_time_sink_x_0.set_line_alpha(i, alphas[i])

        self._qtgui_time_sink_x_0_win = sip.wrapinstance(self.
           qtgui_time_sink_x_0.qwidget(), Qt.QWidget)
        self.top_layout.addWidget(self._qtgui_time_sink_x_0_win)
        self.qtgui_const_sink_x_1 = qtgui.const_sink_c(
            1024, #size
            'Equalized Signal', #name
            1, #number of inputs
            None # parent
        )
        self.qtgui_const_sink_x_1.set_update_time(0.10)
        self.qtgui_const_sink_x_1.set_y_axis((-2), 2)
        self.qtgui_const_sink_x_1.set_x_axis((-2), 2)
        self.qtgui_const_sink_x_1.set_trigger_mode(qtgui.
           TRIG_MODE_FREE, qtgui.TRIG_SLOPE_POS, 0.0, 0, "")
        self.qtgui_const_sink_x_1.enable_autoscale(False)
        self.qtgui_const_sink_x_1.enable_grid(False)
        self.qtgui_const_sink_x_1.enable_axis_labels(True)


        labels = ['', '', '', '', '',
            '', '', '', '', '']
        widths = [1, 1, 1, 1, 1,
            1, 1, 1, 1, 1]
        colors = ["blue", "red", "green", "black", "cyan",
            "magenta", "yellow", "dark red", "dark green", "dark
               blue"]
        styles = [0, 0, 0, 0, 0,
            0, 0, 0, 0, 0]
        markers = [0, 0, 0, 0, 0,
            0, 0, 0, 0, 0]
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
            1.0, 1.0, 1.0, 1.0, 1.0]

        for i in range(1):
            if len(labels[i]) == 0:
                self.qtgui_const_sink_x_1.set_line_label(i, "Data
                   {0}".format(i))
            else:
                self.qtgui_const_sink_x_1.set_line_label(i, labels[i
                   ])
            self.qtgui_const_sink_x_1.set_line_width(i, widths[i])
            self.qtgui_const_sink_x_1.set_line_color(i, colors[i])
            self.qtgui_const_sink_x_1.set_line_style(i, styles[i])
            self.qtgui_const_sink_x_1.set_line_marker(i, markers[i])
            self.qtgui_const_sink_x_1.set_line_alpha(i, alphas[i])

        self._qtgui_const_sink_x_1_win = sip.wrapinstance(self.
           qtgui_const_sink_x_1.qwidget(), Qt.QWidget)
        self.top_layout.addWidget(self._qtgui_const_sink_x_1_win)
        self.qtgui_const_sink_x_0 = qtgui.const_sink_c(
            1024, #size
            'Received Signal (Distorted)', #name
            1, #number of inputs
```

```python
            None # parent
        )
        self.qtgui_const_sink_x_0.set_update_time(0.10)
        self.qtgui_const_sink_x_0.set_y_axis((-2), 2)
        self.qtgui_const_sink_x_0.set_x_axis((-2), 2)
        self.qtgui_const_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, qtgui.TRIG_SLOPE_POS, 0.0, 0, "")
        self.qtgui_const_sink_x_0.enable_autoscale(False)
        self.qtgui_const_sink_x_0.enable_grid(False)
        self.qtgui_const_sink_x_0.enable_axis_labels(True)


        labels = ['', '', '', '', '',
            '', '', '', '', '']
        widths = [1, 1, 1, 1, 1,
            1, 1, 1, 1, 1]
        colors = ["blue", "red", "green", "black", "cyan",
            "magenta", "yellow", "dark red", "dark green", "dark blue"]
        styles = [0, 0, 0, 0, 0,
            0, 0, 0, 0, 0]
        markers = [0, 0, 0, 0, 0,
            0, 0, 0, 0, 0]
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
            1.0, 1.0, 1.0, 1.0, 1.0]

        for i in range(1):
            if len(labels[i]) == 0:
                self.qtgui_const_sink_x_0.set_line_label(i, "Data {0}".format(i))
            else:
                self.qtgui_const_sink_x_0.set_line_label(i, labels[i])
            self.qtgui_const_sink_x_0.set_line_width(i, widths[i])
            self.qtgui_const_sink_x_0.set_line_color(i, colors[i])
            self.qtgui_const_sink_x_0.set_line_style(i, styles[i])
            self.qtgui_const_sink_x_0.set_line_marker(i, markers[i])
            self.qtgui_const_sink_x_0.set_line_alpha(i, alphas[i])

        self._qtgui_const_sink_x_0_win = sip.wrapinstance(self.qtgui_const_sink_x_0.qwidget(), Qt.QWidget)
        self.top_layout.addWidget(self._qtgui_const_sink_x_0_win)
        self._mu_range = qtgui.Range(0.0001, 0.1, 0.0005, 0.01, 200)
        self._mu_win = qtgui.RangeWidget(self._mu_range, self.set_mu
            , "LMS step size", "counter_slider", float, QtCore.Qt.
            Horizontal)
        self.top_layout.addWidget(self._mu_win)
        self.digital_linear_equalizer_0 = digital.linear_equalizer
            (15, 4, lms_algo, True, [ ], 'corr_est')
        self.digital_constellation_modulator_1 = digital.generic_mod
            (
             constellation=qpsk_obj,
             differential=False,
             samples_per_symbol=4,
             pre_diff_code=True,
             excess_bw=0.35,
             verbose=False,
             log=False,
             truncate=False)
        self.digital_constellation_modulator_0 = digital.generic_mod
            (
             constellation=qpsk_obj,
             differential=True,
             samples_per_symbol=sps,
             pre_diff_code=True,
             excess_bw=0.35,
             verbose=False,
             log=False,
             truncate=False)
        self.digital_constellation_decoder_cb_1 = digital.
            constellation_decoder_cb(qpsk_obj)
        self.blocks_throttle2_0 = blocks.throttle( gr.
            sizeof_gr_complex*1, samp_rate, True, 0 if "auto" == "
            auto" else max( int(float(0.1) * samp_rate) if "auto" ==
            "time" else int(0.1), 1) )
        self.blocks_sub_xx_0 = blocks.sub_cc(1)
        self.blocks_multiply_const_vxx_3 = blocks.multiply_const_cc
            (0.1)
        self.blocks_multiply_const_vxx_2 = blocks.multiply_const_cc
            (0.3)
        self.blocks_multiply_const_vxx_1 = blocks.multiply_const_cc
            (1.0)
        self.blocks_multiply_const_vxx_0 = blocks.multiply_const_cc(0.5)
        self.blocks_delay_2 = blocks.delay(gr.sizeof_gr_complex*1, 6)
        self.blocks_delay_1 = blocks.delay(gr.sizeof_gr_complex*1, 4)
        self.blocks_delay_0 = blocks.delay(gr.sizeof_gr_complex*1, 2)
        self.blocks_complex_to_mag_0 = blocks.complex_to_mag(1)
        self.blocks_add_xx_4 = blocks.add_vcc(1)
        self.blocks_add_xx_3 = blocks.add_vcc(1)
        self.blocks_add_xx_2 = blocks.add_vcc(1)
        self.blocks_add_xx_1 = blocks.add_vcc(1)
        self.blocks_add_xx_0 = blocks.add_vcc(1)
        self.analog_random_source_x_0 = blocks.vector_source_b(list(map(int, numpy.random.randint(0, 2, 1000))), True)
        self.analog_noise_source_x_3 = analog.noise_source_c(analog.GR_GAUSSIAN, 0.05, 4)
        self.analog_noise_source_x_2 = analog.noise_source_c(analog.GR_GAUSSIAN, 0.05, 3)
        self.analog_noise_source_x_1 = analog.noise_source_c(analog.GR_GAUSSIAN, 0.05, 2)
        self.analog_noise_source_x_0 = analog.noise_source_c(analog.GR_GAUSSIAN, 0.05, 1)


        ##################################################
        # Connections
        ##################################################
        self.connect((self.analog_noise_source_x_0, 0), (self.blocks_add_xx_3, 1))
        self.connect((self.analog_noise_source_x_1, 0), (self.blocks_add_xx_0, 1))
        self.connect((self.analog_noise_source_x_2, 0), (self.blocks_add_xx_2, 1))
        self.connect((self.analog_noise_source_x_3, 0), (self.blocks_add_xx_1, 1))
        self.connect((self.analog_random_source_x_0, 0), (self.digital_constellation_modulator_0, 0))
        self.connect((self.blocks_add_xx_0, 0), (self.blocks_add_xx_4, 1))
        self.connect((self.blocks_add_xx_1, 0), (self.
           blocks_add_xx_4, 3))
        self.connect((self.blocks_add_xx_2, 0), (self.
           blocks_add_xx_4, 2))
        self.connect((self.blocks_add_xx_3, 0), (self.
           blocks_add_xx_4, 0))
        self.connect((self.blocks_add_xx_4, 0), (self.
           digital_linear_equalizer_0, 0))
        self.connect((self.blocks_add_xx_4, 0), (self.
           qtgui_const_sink_x_0, 0))
        self.connect((self.blocks_complex_to_mag_0, 0), (self.
           single_pole_iir_filter_xx_0, 0))
        self.connect((self.blocks_delay_0, 0), (self.
           blocks_multiply_const_vxx_0, 0))
        self.connect((self.blocks_delay_1, 0), (self.
           blocks_multiply_const_vxx_2, 0))
        self.connect((self.blocks_delay_2, 0), (self.
           blocks_multiply_const_vxx_3, 0))
        self.connect((self.blocks_multiply_const_vxx_0, 0), (self.
           blocks_add_xx_0, 0))
        self.connect((self.blocks_multiply_const_vxx_1, 0), (self.
           blocks_add_xx_3, 0))
        self.connect((self.blocks_multiply_const_vxx_2, 0), (self.
           blocks_add_xx_2, 0))
        self.connect((self.blocks_multiply_const_vxx_3, 0), (self.
           blocks_add_xx_1, 0))
        self.connect((self.blocks_sub_xx_0, 0), (self.
           blocks_complex_to_mag_0, 0))
        self.connect((self.blocks_throttle2_0, 0), (self.
           blocks_delay_0, 0))
        self.connect((self.blocks_throttle2_0, 0), (self.
           blocks_delay_1, 0))
        self.connect((self.blocks_throttle2_0, 0), (self.
           blocks_delay_2, 0))
        self.connect((self.blocks_throttle2_0, 0), (self.
           blocks_multiply_const_vxx_1, 0))
        self.connect((self.digital_constellation_decoder_cb_1, 0), (
           self.digital_constellation_modulator_1, 0))
        self.connect((self.digital_constellation_modulator_0, 0), (
           self.blocks_throttle2_0, 0))
```

```python
        self.connect((self.digital_constellation_modulator_1, 0), (self.blocks_sub_xx_0, 0))
        self.connect((self.digital_linear_equalizer_0, 0), (self.blocks_sub_xx_0, 1))
        self.connect((self.digital_linear_equalizer_0, 0), (self.digital_constellation_decoder_cb_1, 0))
        self.connect((self.digital_linear_equalizer_0, 0), (self.qtgui_const_sink_x_1, 0))
        self.connect((self.single_pole_iir_filter_xx_0, 0), (self.qtgui_time_sink_x_0, 0))


    def closeEvent(self, event):
        self.settings = Qt.QSettings("gnuradio/flowgraphs", "project2")
        self.settings.setValue("geometry", self.saveGeometry())
        self.stop()
        self.wait()

        event.accept()

    def get_qpsk_obj(self):
        return self.qpsk_obj

    def set_qpsk_obj(self, qpsk_obj):
        self.qpsk_obj = qpsk_obj
        self.digital_constellation_decoder_cb_1.set_constellation(self.qpsk_obj)

    def get_mu(self):
        return self.mu

    def set_mu(self, mu):
        self.mu = mu

    def get_sps(self):
        return self.sps

    def set_sps(self, sps):
        self.sps = sps

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.blocks_throttle2_0.set_sample_rate(self.samp_rate)
        self.qtgui_time_sink_x_0.set_samp_rate(self.samp_rate)

    def get_lms_algo(self):
        return self.lms_algo

    def set_lms_algo(self, lms_algo):
        self.lms_algo = lms_algo




def main(top_block_cls=project2, options=None):

    qapp = Qt.QApplication(sys.argv)

    tb = top_block_cls()

    tb.start()
    tb.flowgraph_started.set()

    tb.show()

    def sig_handler(sig=None, frame=None):
        tb.stop()
        tb.wait()

        Qt.QApplication.quit()

    signal.signal(signal.SIGINT, sig_handler)
    signal.signal(signal.SIGTERM, sig_handler)

    timer = Qt.QTimer()
    timer.start(500)
    timer.timeout.connect(lambda: None)

```

```python
    qapp.exec_()

if __name__ == '__main__':
    main()


# ==================== END OF CODE ========================
```