



Objective

This laboratory will introduce several fundamental concepts for digital communication system design and analysis. Specifically, methods used to represent binary information in terms of amplitude, phase, and frequency quantities will be discussed and studied via experimentation. Moreover, MATLAB® and Simulink® will also be introduced as development tools for digital communication systems, especially the constructure of prototype software-defined radio (SDR).

Contents

1	Theoretical Preparation	3
1.1	Random Variables	3
1.2	Central Limit Theorem	3
1.3	Gaussian Processes	4
1.4	The Q Function	4
1.5	Power Spectral Density	5
1.6	Complex Baseband & Signal Representation	6
1.7	Suggested Readings	7
1.8	Problems	8
2	Simulation Experiments	10
2.1	Random Number Generators	10
2.1.1	Generation Process	10
2.1.2	Random Variable Transformation	10
2.1.3	New Random Variables	10
2.1.4	Noise Generation	11
2.2	AM Transmission in MATLAB	11
2.3	Using Simulink in Communications	12
2.3.1	Data Sequence Generators	13
2.3.2	AM Transmission in Simulink	14
2.3.3	Additional Information on Simulink	15
3	USRP2 Hardware Implementation	16
3.1	Basic Setup	16
3.2	Frequency Offset	16
3.3	Digital Communication	18
3.4	IEEE 802.11 Wireless Local Area Networks	19

3.5	Accelerate Your Simulink Model that Uses USRP Blocks	20
4	Open-ended Design Problem: Automatic Frequency Offset Compensator	22
4.1	Introduction	22
4.2	Objective	22
4.3	Theoretical Background	22
5	Lab Report Preparation & Submission Instructions	24

1 Theoretical Preparation

The fundamental concepts of digital communication systems and related theoretical background material covered in this section will serve as a basis for the implementation and design of prototype systems throughout the rest of this course.

1.1 Random Variables

A random variable (RV) is a number assigned to every outcome of an experiment. This number could be the gain in a game of chance, the voltage of a random source, the cost of a random component, or any other numerical quantity that is of interest in the performance of the experiment.

A random variable is a mapping function whose domain is a sample space and whose range is some set of real numbers:

$$X = X(s) \tag{1}$$

where X is the RV and $X(s)$ is the outcome of an experiment. The cumulative distribution function (CDF) describes how the RV behaves probabilistically, and is defined as [5]:

$$F_x(x) = P(X \leq x). \tag{2}$$

Eq. (2) describes the probability that the outcome of an experiment described by the RV X is less than or equal to the dummy variable x .

Several fundamental characteristics of the CDF include the following:

- $F_x(x)$ is bounded between zero and one.
- $F_x(x)$ is a non-decreasing function, i.e., $F_x(x_1) \leq F_x(x_2)$ if $x_1 \leq x_2$.

The probability density function (PDF) is the derivative of the CDF in terms of the dummy variable x , which we can define as [5]:

$$f_x(x) = \frac{d}{dx} F_x(x). \tag{3}$$

For more information about random variables, please refer to Section 4.1 of the course textbook [6].

1.2 Central Limit Theorem

In probability theory, the central limit theorem (CLT) states conditions under which the mean of a sufficiently large number of independent random variables, each with finite mean and variance, can be approximated by a Normal (i.e., Gaussian) distribution. The CLT also requires the random variables to be identically distributed. Since real-world quantities are often the balanced sum of many unobserved random events, this theorem provides a partial explanation for the prevalence of the normal probability distribution.

The CLT states that if you have a set of random variables $\{X_i\}_{i=1}^{i=N}$ such that X_i is independently and identically distributed (i.i.d.), i.e.:

- X_i are statistically independent.

- X_i have the same pdf with mean μ_x and variance σ_x^2 .

Then as $N \rightarrow \infty$, the distribution of the sample expectation will be Gaussian regardless of the original distribution.

1.3 Gaussian Processes

In probability theory and statistics, a Gaussian process is a stochastic process whose realizations consist of random values associated with every point in a range of times (or of space) such that each such random variable has a normal distribution. Moreover, every finite collection of those random variables has a multivariate normal distribution.

Gaussian processes are important in statistical modeling because of properties inherited from the normal distribution. For example, if a random process is modeled as a Gaussian process, the distributions of various derived quantities can be obtained explicitly. Such quantities include the average value of the process over a range of times, and the error in estimating the average using sample values at a small set of times.

Given the following expression:

$$y = \int_0^T g(t)X(t) dt \quad (4)$$

we can say that $X(t)$ is a Gaussian process if:

- $E(y^2)$ is finite, i.e., does not blow up.
- Y is Gaussian-distributed for every $g(t)$.

Note that the RV Y has a Gaussian distribution, where its PDF is defined as:

$$f_y(y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(y-\mu_y)^2}{2\sigma_y^2}}, \quad (5)$$

where μ_y is the mean and σ_y^2 is the variance. Such processes are important because they closely match the behavior of numerous physical phenomena, such as additive white Gaussian noise (AWGN).

1.4 The Q Function

The Q function is a convenient way to express right-tail probabilities for normal (Gaussian) random variables. When dealing with Gaussian distributions, it is sometimes necessary to use the Q function, such as for computing $P(x) \geq Y(x)$, or the tail probability of an event, as shown in Figure 1.

We can define the Q function as the following:

$$Q(x) = \frac{1}{2} \left(1 - \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right), \quad (6)$$

where $\operatorname{erf}(x)$ is the error function. Thus, instead of working out the mathematics, a communications engineer would instead refer to a look-up table for the value of $Q(x)$. Several key values of the Q function include:

$$Q(-\infty) = 1 \quad Q(0) = 0.5 \quad Q(\infty) = 0$$

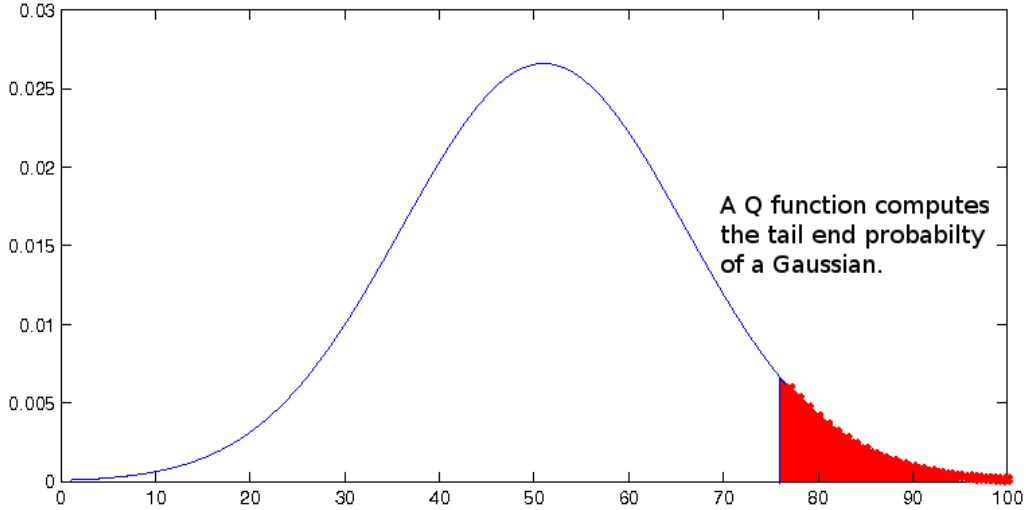


Figure 1: Tail of a Gaussian distribution.

Tables of Q function values can be found from the book by Abramowitz and Stegun [2]. For more information about Q function, please refer to Section 4.2 of the course textbook [6].

1.5 Power Spectral Density

To analyze a signal in the frequency domain, the power spectral density (PSD), $S_x(f)$, is often used to characterize the signal, which is obtained by taking the Fourier Transform of the autocorrelation $R_x(\tau)$ of the signal $X(t)$. The PSD and the autocorrelation of a function, $R_x(\tau)$, are mathematically related by the Einstein-Wiener-Khinchin (EWK) relations, namely:

$$S_x(f) = \int_{-\infty}^{\infty} R_x(\tau) e^{-j2\pi f\tau} d\tau \quad (7)$$

$$R_x(f) = \int_{-\infty}^{\infty} S_x(\tau) e^{+j2\pi f\tau} df \quad (8)$$

Using the EWK relations, we can derive some general properties of the power spectral density of a stationary process:

- $S_x(0) = \int_{-\infty}^{\infty} R_x(\tau) d\tau$
- $E\{X^2(t)\} = \int_{-\infty}^{\infty} S_x(f) df$
- $S_x(f) \geq 0$ for all f
- $S_x(-f) = S_x(f)$
- The power spectral density, appropriately normalized, has the properties usually associated with a probability density function:

$$p_x(f) = \frac{S_x(f)}{\int_{-\infty}^{\infty} S_x(f) df} \quad (9)$$

Using $H(f)$ to denote the frequency response of the system, we can relate the power spectral density of input and output random processes by the following equation:

$$Y(f) = |H(f)|^2 X(f), \quad (10)$$

where $X(f)$ is the PSD of input random process and $Y(f)$ is the PSD of output random process.

For more information about power spectral density, please refer to Section 4.4 of the course textbook [6].

1.6 Complex Baseband & Signal Representation

Understanding how a signal is represented can greatly enhance one's ability to analyze and design baseband digital communication systems. We need a convenient mathematical framework to represent signal and noise. We usually have two: Envelope/Phase and In-phase/Quadrature.

A bandpass signal can be represented by the sum of its in-phase (I) and quadrature (Q) components:

$$x(t) = R_I(t) \cos(2\pi f_c t) - R_Q(t) \sin(2\pi f_c t). \quad (11)$$

where $R_I(t)$ is in-phase amplitude, $R_Q(t)$ is quadrature amplitude and f_c is carrier frequency. It can also be represented as a sinusoid by its envelope and phase:

$$x(t) = R(t) \cos(\omega t + \phi(t)), \quad (12)$$

where $R(t)$ is the amplitude and $\phi(t)$ is phase offset.

Consider a 4-QAM signal constellation and how it would be represented by I and Q or its envelope and phase. If you needed to compute the distance between two points, which representation would be easier to work with? What about with other signal constellations? Table 1 shows these relationships graphically.

Given the importance of representing signals in an efficient format, consider how you would compute the distance between two constellation points in QPSK whose four points given in I/Q format are:

$$S_1(t) = +A \cos(2\pi f_c t) + A \sin(2\pi f_c t),$$

$$S_2(t) = -A \cos(2\pi f_c t) + A \sin(2\pi f_c t),$$

$$S_3(t) = -A \cos(2\pi f_c t) - A \sin(2\pi f_c t),$$

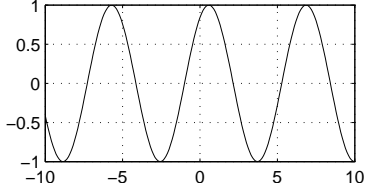
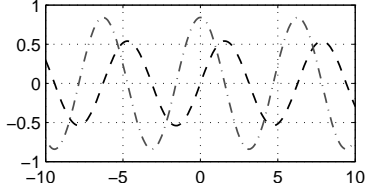
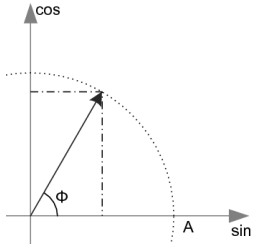
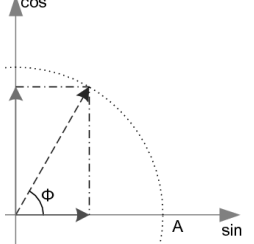
$$S_4(t) = +A \cos(2\pi f_c t) - A \sin(2\pi f_c t).$$

Using basic trigonometry, the distance between any two points can be quickly computed using the law of cosines, namely:

$$C^2 = A^2 + B^2 - 2AB \cos(\theta). \quad (13)$$

This solution yields the distance between the two points. The same principle holds true for other signal constellations including M-QAM, M-PSK, M-PAM, 7-around-1, BOX, etc. Although the trigonometry can be different for various constellations, it is almost always simpler to evaluate the math if you represent the signals in complex baseband.

Table 1: Signal representations of $A \sin(\omega t + \phi)$.

	Envelope/Phase \mathbf{A}, ϕ	In-phase/Quadrature $(\mathbf{A} \sin \phi), (\mathbf{A} \cos \phi)$
Time	$\mathbf{A} \sin(\omega t + \phi)$	$(\mathbf{A} \cos \phi) \sin(\omega t) + (\mathbf{A} \sin \phi) \cos(\omega t)$
Waveform		
Vector		

1.7 Suggested Readings

Although this laboratory handout provides some information about the fundamentals of digital communications, the reader is encouraged to review the material from the following references in order to gain further insight on these topics.

- Overview of Signals and Systems:
 - Chapter 2 in Reference [6].
- Overview of Probability Theory:
 - Chapter 4 in Reference [6].
- Introduction to Communication Simulation Techniques:
 - Chapter 1 in Reference [7].

1.8 Problems

1. The power spectral density of a narrowband noise signal, $n(t)$, is shown in Figure 2. The carrier frequency is 10 Hz.
 - (a) Find the power spectral densities of the in-phase and quadrature components of $n(t)$.
 - (b) Find their cross-spectral densities.

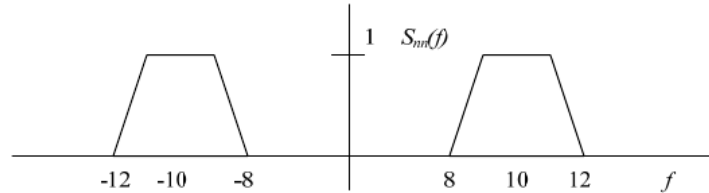


Figure 2: Power spectral density of a narrowband noise signal, $n(t)$.

2. The input to a time-invariant linear system with impulse response $h(t)$ is a white Gaussian noise process $X(t)$ with two-sided spectral density $\frac{N_0}{2}$. The output is the random process $Y(t)$. The filter's response is given in terms of $p_T(t)$, the rectangular pulse of duration T , by $h(t) = \frac{\sin(2\pi t)}{T} * p_T(t)$.
 - (a) Find the cross-correlation function $R_{X,Y}(\tau)$, $-\infty < \tau < \infty$.
 - (b) Find $R_Y(0)$.
3. Prove the following two properties of the autocorrelation function $R_X(\tau)$ of a random process $X(t)$:
 - (a) If $X(t)$ contains a DC component equal to A , then $R_X(\tau)$ will contain a constant component equal to A^2 .
 - (b) If $X(t)$ contains a sinusoidal component, then $R_X(\tau)$ will also contain a sinusoidal component of the same frequency.
4. Consider a pair of stationary processes $X(t)$ and $Y(t)$. Show that the cross-correlations $R_{XY}(\tau)$ and $R_{YX}(\tau)$ of these processes have the following properties:
 - (a) $R_{XY}(\tau) = R_{YX}(-\tau)$
 - (b) $|R_{XY}(\tau)| \leq \frac{1}{2}[R_X(0) + R_Y(0)]$
5. Exercise 2.21 from the course textbook [6].
6. Exercise 2.53 from the course textbook [6].
7. Exercise 4.1 from the course textbook [6].
8. Exercise 4.3 from the course textbook [6].
9. Exercise 4.6 from the course textbook [6].
10. Exercise 4.7 from the course textbook [6].

11. Exercise 4.26 from the course textbook [6].
12. Exercise 4.27 from the course textbook [6].

2 Simulation Experiments

The MATLAB software uses a matrix language, which means it is designed for vector and matrix operations. You can often speed up your code by using vectorizing algorithms that take advantage of this design. *Vectorization* means converting **for** and **while** loops to equivalent vector or matrix operations. In this laboratory, MATLAB is used as a digital communication system design and evaluation tool. Due to the nature of many the mathematical operations used in communications, it is also important to vectorize operations in MATLAB as much as possible. Avoiding loops will save you many computational cycles and ultimately result in much shorter simulation times.

Read the online documentation [Techniques for Improving Performance](#) for more information about vectorization, as well as some other techniques for improving performance.

2.1 Random Number Generators

We will now focus on the generation of additive Ricean noise via the manipulation of uniform and Gaussian random number generators. The goal of this laboratory exercise is to refresh your knowledge about probability and prepare you for conducting communication simulations in MATLAB.

2.1.1 Generation Process

Generate two uniform random variables, $x = \{x_1, x_2, \dots, x_n\}$ and $y = \{y_1, y_2, \dots, y_n\}$, on the interval $[0,1]$. Use `rand()` function of MATLAB. Let $n = 20,000$.

2.1.2 Random Variable Transformation

Apply the following transformations to obtain two new random variables, $x_{std-normal}$ and $y_{std-normal}$:

$$x_{std-normal} = \mu_1 + \sqrt{-2\log(x)}\cos(2\pi y) \quad (14)$$

$$y_{std-normal} = \mu_2 + \sqrt{-2\log(x)}\sin(2\pi y) \quad (15)$$

where μ_1 and μ_2 are equal to zero. Plot the histograms of $x_{std-normal}$ and $y_{std-normal}$. Describe and explain your observations.

2.1.3 New Random Variables

Apply the following transformation to obtain a new random variable, $z_{rayleigh}$:

$$z_{rayleigh} = \sqrt{x_{std-normal}^2 + y_{std-normal}^2} \quad (16)$$

Make $\mu_1=1$ and $\mu_2=2$ and apply the same transformation to obtain another random variable, z_{rician} :

$$z_{rician} = \sqrt{x_{std-normal}^2 + y_{std-normal}^2} \quad (17)$$

Plot the histograms of $z_{rayleigh}$ and z_{rician} . Describe and explain your observations.

2.1.4 Noise Generation

We will now use the random variables $x_{std-normal}$ (or $y_{std-normal}$) and $z_{rayleigh}$ to simulate the performance of an idealized binary phase shift keying (BPSK) transceiver system generate a symbol vector consisting of 20,000 random “1” and “-1” values.

Additive Gaussian Noise: Add the elements of $x_{std-normal}$ to the symbol vector and round the obtained noisy symbols to the nearest constellation point (i.e., “1” or “-1”). Repeat the procedure by generating zero mean Gaussian random variables of different variances. Use the following variance values:

$$\sigma^2 = \{0.1, 0.1259, 0.1585, 0.1995, 0.2512, 0.3162, 0.3981, 0.5012, 0.6310, 0.7943, 1\} \quad (18)$$

for generating different Gaussian random variables. Note that a standard normal random variable, i.e., zero mean and unit variance, can be converted to a random variable of desired mean $\mu_{desired}$ and variance $\sigma_{desired}^2$, namely $x \sim N(\mu_{desired}, \sigma_{desired}^2)$, by the transformation:

$$x = \mu_{desired} + \sigma_{desired} \times x_{std-normal}. \quad (19)$$

Count the number of errors resulting for each value of the variance and generate a plot, where the y-axis is the ratio of the number of errors to the total number of symbols transmitted, while the x-axis consists of the variance values chosen. Use `semilogy()` function from MATLAB. For the x-axis, use the vector obtained by converting each element of the above variance vector into a vector $N = 10 \cdot \log_{10}(1/\sigma^2)$. What do you observe?

Additive Rayleigh Noise: Now add the elements of $z_{rayleigh}$ to the symbol vector and round the obtained noisy symbols to the nearest constellation point (i.e., “1” or “-1”). Repeat the procedure by generating Rayleigh random variables of different variances (use the Gaussian random variables obtained previously and apply the procedure described in Section 2.1.3).

Count the number of errors resulting for each value of the variance and generate a plot, where the y-axis consists of number of errors divided by the total number of symbols, and the x-axis is the variance values chosen. Use `semilogy()` function from MATLAB. What do you observe?

2.2 AM Transmission in MATLAB

In this section, we will implement a basic amplitude modulation (AM) transmitter [6]. AM transmission defines a relationship between the amplitude of the received signal and the data being sent. To simulate the AM transmitter, you will need to construct four vectors in MATLAB:

- A time vector:

```
t=linspace(0,1,10000)
```

- A cosine at the carrier frequency:

```
fc=500;c=cos(2*pi*fc*t)
```

- A message vector to modulate against the carrier:

```
m=[cos(2*pi*20*t(1:length(t)/2))cos(2*pi*50*t(length(t)/2+1:end))]
```

- A demodulating cosine accounting for frequency and phase offset:

```
c2 = cos(2*pi*(fc+fo)*t+phi)
```

Consider your message as an impulse train being multiplied by a sinusoid. After modulation, the input to the sinusoid has properties of both the pulse shape (the sinusoid) and the original impulse.

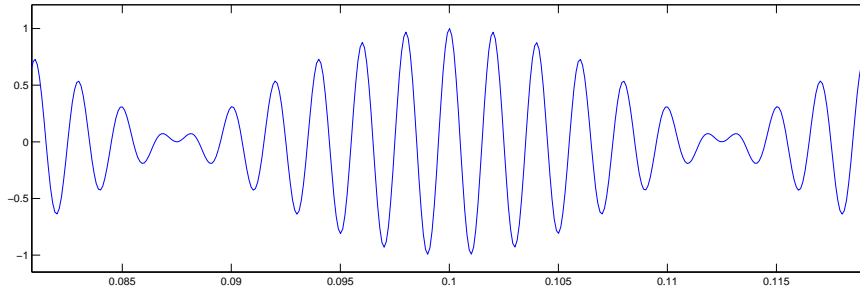


Figure 3: Amplitude Modulated Message, where x-axis is time, and y-axis is amplitude modulated message.

Perform the following tasks:

- Construct a basic AM modulator / demodulator in MATLAB. Assume that the frequency and phase offsets are zero.
- Plot the following information signals:
 1. The message vector.
 2. The modulated message in the time domain.
 3. The demodulated message in the time domain.

Why doesn't your demodulated message look like your message signal? What might you need to do to make it look like your original message? Increase the phase and frequency offsets. How does this affect your demodulated message vector?

2.3 Using Simulink in Communications

Model-based design is a useful approach that provides a more visual realization to an engineering solution. A communications system is made up of many different sub-systems. A very basic transmitter can be modeled as a data stream with some redundancy encoded into a signal constellation and passed through a pulse filter. Transmitted over an ideal channel, the receiver in this scenario just needs to perform the inverse of the coding operations performed on the original signal.

2.3.1 Data Sequence Generators

Referring to the extensive documentation of Simulink available online [4], make sure you know how to create a model and add blocks, which has been introduced previously during the laboratory tutorial. To gain additional insight into the capabilities of Simulink, you might want to examine some of the pre-built demos that accompany Simulink, e.g., ADSL and HIPERLAN.

The next step of the lab demonstrates the concepts you should be familiar with at this point. Two important optimization considerations when working in Simulink are:

1. Which **solver** you use can dramatically affect your simulation time. For communications applications, it is almost always ideal to use a Fixed Step Discrete solver. To change your solver in your model, go to **Simulation** → **Configuration Parameters**. Under the **Solver** tab change **Variable-Step** to **Discrete-Step**.
2. Some Simulink blocks may seem redundant. For example, if you search for a **sin** block you will find that sinusoid generation in several blocksets. For optimization purposes, it is ideal to use appropriate blockset for your application. Whenever possible, you want to use blocks available in the Communications Blockset.

We will now implement a simple system that observes the power of band-limited white noise.

1. Create a new .mdl (model) file. **File** → **New** → **Model**
This creates a new ‘workspace’ for you to build your model in.
2. To add a data source, you can find several under the Sources category in the Simulink library. Choose a **Band-Limited White Noise** source and drag it to your workspace.
3. The next step is to observe the waveform you just introduced to your model. Simulink calls such observation devices **sinks** and categorizes them as such. Drag a **Scope** to the workspace.
4. You are now observing band-limited white noise on the scope. What if you wanted to observe the power of the signal? Search the Simulink library for **math** and double click on the sub-category of math functions that appear.

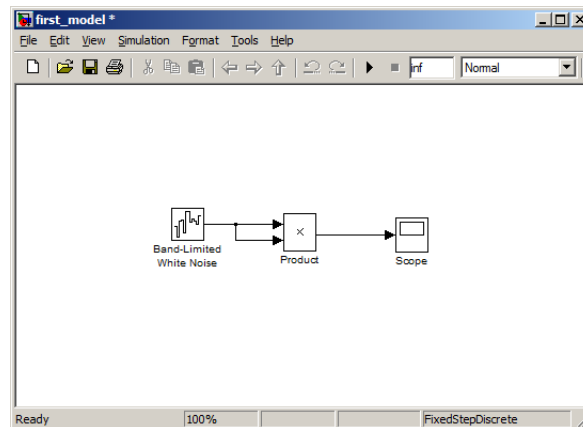


Figure 4: Final Model.

5. Now connect the source to the sink. To observe your data, press the play button at the top of your model. Note the integer next to it that represents the simulation length. Instead of 10, type `inf`. Double click on the scope while the simulation is running. Note that to make additional changes to your model, you must stop your simulation.
6. Delete the direct link between the noise and the scope blocks. Place the appropriate math block between the source and scope such that the power of the signal is observed in the scope. Record your scope output and model design.

Simulink also allows you to implement sub-systems with *Embedded MATLAB code*. Embedded MATLAB code is used within the Simulink product family to include embedded-ready MATLAB code in Simulink models. For example, you can call Embedded MATLAB functions contained in one or more M-files on the MATLAB path from **Embedded MATLAB Function** blocks in Simulink block diagram models. This allows you to create libraries of Embedded MATLAB code that can be reused in multiple Simulink models. As a result, you can combine graphical models and MATLAB code in Simulink to develop, simulate, and verify algorithms in a complete system-level context.

On the model you have just constructed, perform the following tasks:

- Delete the math operation between the source and sink.
- Search the library for **Embedded MATLAB Function**. Drag this block between the sink and source and hook them up to their respective ports.
- Double click on the function block and modify the code such that the output of this block is the power of the input signal.
- Compare your results to that of the pre-made block.

2.3.2 AM Transmission in Simulink

We will now manipulate the parameters of a double sideband amplitude modulator (DSB-AM) in Simulink and observe the system on several scopes. When the message signal is modulated, a sideband appears on either side of the carrier frequency. These double sidebands are a source of interference. In some more sophisticated AM schemes, these double sidebands are suppressed (DSB-Suppressed Carrier).

Open `DSBAMbasic.mdl` from the course website, as shown in Figure 5, and perform the following tasks. You will manipulate the parameters of this model and observe the effects on the scopes.

- Observe the outputs of scopes 1 and 2. They should be identical. Shift the carrier estimate of the DSB-AM demodulator. What happens to the output of scope 2?
- What happens if you replace the DSB-AM modulator blocks with DSBSC-AM modulator blocks. How does the PSD change?
- Make the channel non-ideal by including an AWGN block in between the modulator and demodulator. How does this affect the output on scope 2?

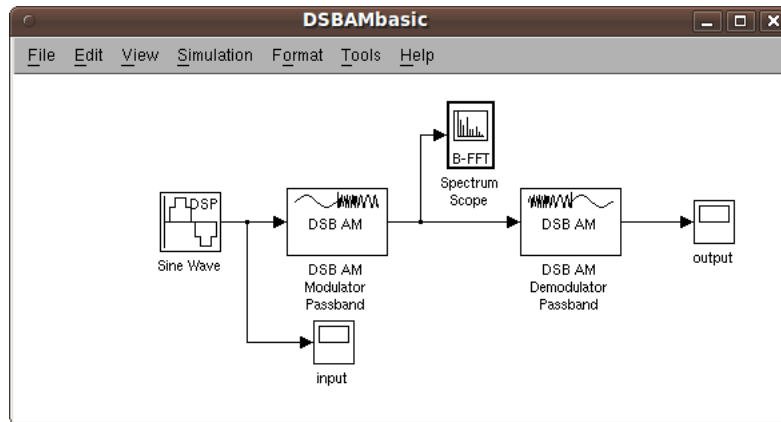


Figure 5: A double sideband amplitude modulator model.

2.3.3 Additional Information on Simulink

Several other features in Simulink that can assist you throughout the laboratory components of this course include:

- Pre-built blocks are often sub-systems driven by many smaller blocks. Right click on a block and click **Look under mask** to see the blocks comprising a system.
- For communications model, it's always helpful to learn the *sample time* and *data type* of the model. In order to display sample time, go to the **Format** menu, and click **Sample Time Display**. In order to display data type, go to the **Format** menu, and click **Port/Signal Displays**.
- The best way to get comfortable with Simulink is to experiment with basic models like the ones in Section 2.3.1 and Section 2.3.2.

3 USRP2 Hardware Implementation

This section of the laboratory will familiarize you with some of the useful Simulink tools for digital communication system design via SDR using the USRP2 platforms. Specifically, you will implement several simple digital communication scripts that wirelessly transmit packets of information.

3.1 Basic Setup

If you have not gone through the host computer configuration steps described in the laboratory tutorial, please do so at this time.

3.2 Frequency Offset

Oscillator crystals are not perfect, since they do not oscillate at exactly the specified frequency. Due to imperfections and tolerances in the manufacturing process, oscillator crystals typically have inaccuracies of about 20 or 50 parts-per-million (ppm), which are guaranteed by the manufacturer. We will ignore other sources of inaccuracy, such as load capacitance, that change the RLC characteristics of the oscillator circuit.

Oscillator circuits are used in the radio frequency (RF) front-end electronics to modulate and demodulate signals to and from RF. Inaccuracies in the oscillator crystal frequency cause errors in the RF carrier frequency. As you may recall from ECE 3311, even relatively small differences in frequency between the transmitter and the receiver can prevent the receiver from correctly decoding the transmission.

It is also worth noting that oscillator circuits provide the clock source for the digital electronics. In particular, the clocks for the analog-to-digital and digital-to-analog converters affect their sampling rates. Therefore, accurate oscillators are essential for both the RF and digital systems.

The GNU Radio FAQ [1] states that the “USRP2 reference clock stability” is “about 20 ppm unless you lock to an external reference.” Note that the USRP2 has an input connector for an external frequency reference, but we will not be using it in these labs. Suppose that we select a carrier frequency of 2.45 GHz for our USRP2. An offset of 20 ppm may sound small, but it is quite significant at high frequencies; note that 20 ppm is equal to 20,000 parts-per-billion. Doing out the math, $2.45\text{E}+9$ times $20,000\text{E}-9 = 49,000$. In other words, when using a carrier frequency of 2.45 GHz, the worst case frequency offset could be as much as about 50 kHz! Note that if you have two USRP2s trying to communicate with each other, where one has an offset of -50kHz and the other has an offset of +50kHz, the overall difference in frequency will be 100kHz!

It is necessary to compensate for this frequency offset in order to achieve successful digital communication between the USRP2s. For example, we have measured some of the USRP2s to have frequency offsets of as much as 45kHz ($45\text{kHz}/2.45\text{GHz} = 18\text{ppm}$). With compensation, the USRP2 works fine, but without it, digital communication does not work at all.

Please do the following steps to measure and compensate for the frequency offset.

1. Arbitrarily select a carrier somewhere within either of the supported bands of the XCVR2450 daughtercard (2.4 to 2.5 GHz and 4.9 to 5.9 GHz). For example, suppose 2.45 GHz. Note: You probably do not want to choose this exact frequency since everyone in the class will be transmitting at once; there will be interference if everyone chooses this frequency.
2. Run `siggen.mdl` on one USRP2 and `observeFFT.mdl` on another USRP2, as shown in Figure 6 and Figure 7. By setting the “Center Frequency (Hz)” parameter in both USRP2 Transmitter and USRP2 Receiver blocks, you can use the carrier frequency you have selected in the previous step.

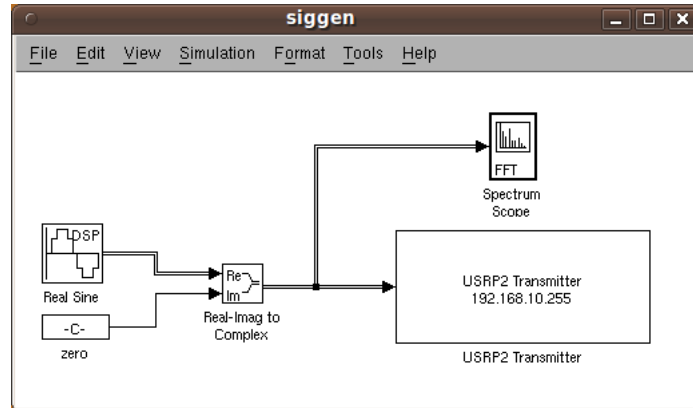


Figure 6: The structure of `siggen.mdl`. In this model, since the USRP2 Transmitter block requires the complex input, the Real-Imag to Complex block converts real and imaginary inputs to a complex-valued output signal.

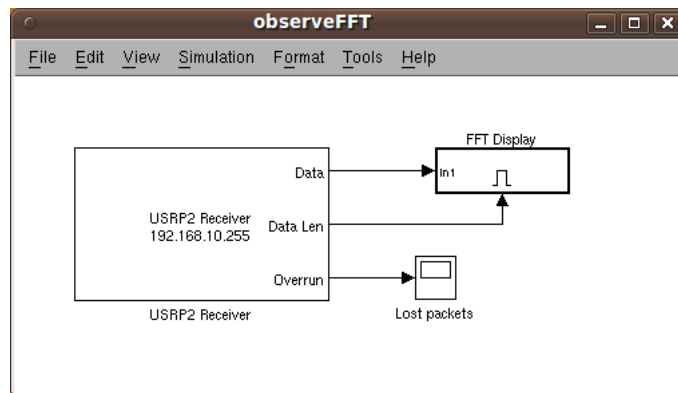


Figure 7: The structure of `observeFFT.mdl`. In this model, FFT Display is an *enabled subsystem*. We use the Data Len parameter to qualify the execution of this part. Specifically, when Data Len contains a zero value, there is no data, so FFT Display cannot be enabled.

3. Using your FFT plot from Spectrum Scope, measure the frequency offset between the received signal and the desired carrier. In order to have a better observation from the scope, you can right click on the plot from your scope, and choose “Autoscale”.
4. Now adjust the carrier frequency of the transmitter by the amount of this offset.

- By iteratively adjusting the carrier frequency and observing the result, you should be able to determine the carrier frequency that you must use on the transmitter in order to observe the correct carrier frequency on the receiver.

Keep in mind that both the transmitter and the receiver have frequency offsets. For example, “2.45 GHz” on the receiver is somewhere within 2.45 GHz +/- 20ppm, not exactly “2.45 GHz.” When you tune the transmitter to eliminate the frequency offset on the receiver, you are essentially compensating for the offset between them. It is the relative offset between the transmitter and receiver that matters. Note that you can adjust either the transmitter or the receiver (or both) to compensate for the offset. The basic idea is to get the two devices synchronized in carrier frequency with each other. If you have more than two USRP2s communicating with each other, you have to make sure that each pair of transmitters and receivers are all tuned to each other.

3.3 Digital Communication

Now that you have learned how to compensate for the frequency offset, you are ready to experiment with some of the digital communication examples. Do the following tasks concerning a BPSK example:

- Try running `BPSKTx_lab1.mdl` on one USRP2 and `BPSKRx_lab1.mdl` on another USRP2, as shown in Figure 8 and Figure 9.

Please note that for the carrier frequency, you should use the values that you found in Section 3.2 that compensate for the frequency offset.

On the receiving computer workstation, you will see that `Display` inside `BPSK Receiver` subsystem shows the received data. What do you observe?

You might need to stop and restart the receiver before it starts receiving data in order to flush the USRP2’s Ethernet buffers.

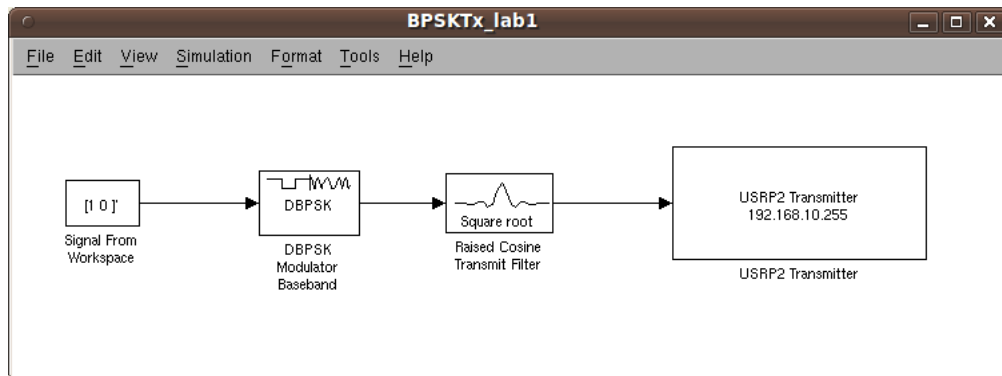


Figure 8: The structure of `BPSKTx_lab1.mdl`. In this model, the `Signal From Workspace` block specifies a data source of repeated 10.

- Now try running these models using the carrier frequencies **without compensation**. What do you observe from the `Display`?

Some of the USRP2s have more severe frequency offsets than others, depending on the severity of imperfections in the oscillator hardware. On some of the USRP2s, you will see an increase in the number of incorrect packets. Some of the USRP2s will not receive any packets at all because the frequency offset is so severe.

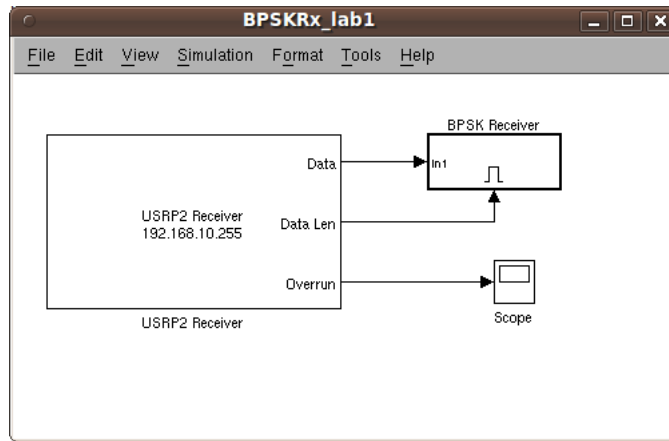


Figure 9: The structure of `BPSKRx_lab1.mdl`. In this model, `BPSK Receiver` is an *enabled subsystem*, which includes the DBPSK demodulator.

3. Now, use `To Workspace` block instead of `Display` block and repeat the previous two steps. By using `To Workspace` block, you can record more data than by just observing the `Display` block. You need to change the following three parameters in `To Workspace` block:
 - (a) **Variable name:** You can specify the name of the array that holds the data.
 - (b) **Limit data points to last:** You can specify the maximum number of input samples to save. Choose “1000”.
 - (c) **Save format:** You can specify the format in which to save simulation output to the workspace. Choose “Array”.

Plot the data you have collected. What are your observations regarding this communication system?

4. As you may recall from ECE 3311, frequency synchronization between the transmitter and receiver is crucial. To get a feel for this, try experimenting with a few other carrier frequency offsets, stepping in increments of about 10 kHz or 15 kHz. Which offsets result in many incorrect packets?
5. Try changing the distance between the transmitter and receiver. Try moving the transmitter and receiver to disrupt the line of sight communication between them to increase channel impairments such as multi-path interference. Observe how this affects the packet error rate.

3.4 IEEE 802.11 Wireless Local Area Networks

Another interesting experiment is using the USRP2 to observe the spectrum of wireless local area networks within the vicinity (WLANs). Most high population density areas employ numerous wireless communication networks for a variety of applications, such as the WPI wireless network. Consequently, you can use your USRP2 experimentation platform to plot their magnitude spectrum. IEEE 802.11 [3] is one type of WLAN standard that possesses a list of carrier frequencies for a collection of Wi-Fi channels. For example, The IEEE 802.11 standard defines Channel 1 of the 2.4 GHz band to be centered at 2.412 GHz.

- Specify the “Center frequency” parameter of **USRP2 Receiver** with this carrier frequency, use the **Spectrum Scope** to plot their magnitude spectrums.
- Since your XCVR2450 daughtercard supports the 5 GHz band as well, also use the **Spectrum Scope** to plot spectrum in the 5 GHz band.

You might want to turn on “Autoscale” (right click the scope display) and adjust the amplitude, since the peaks of these wireless signals could be rather low. It might also be useful to change the “Decimation” parameter of **USRP2 Receiver** to adjust the frequency resolution for a better graph of the spectrum.

3.5 Accelerate Your Simulink Model that Uses USRP Blocks

In Section 3.3, when you plot your received data from DBPSK, you may find some constant zeros even if you have compensated the frequency offset. At the same time, by observing **Overflow** of **USRP2 Receiver** block, we can find that there are packets being dropped during the USRP2 transmission to the host. This problem is mainly due to your Simulink model being not fast enough to keep up with the processing speed of the USRP2 board, such that the model cannot be executed in real-time. Here is a collection of performance improvements you can make in your Simulink models to approach, if not achieve, real-time:

1. In the **Simulation** → **Configuration Parameters** → **Data Import/Export** dialog, turn off all logging.
2. Make sure that the model is single-rate. If the model requires resampling, then choose rational coefficients that will keep the model single-rate. For instance, the FM demos in the Communications Blockset that employ the USRP2 blocks use a native frame size of 358, with an initial sample rate of roughly 195 kHz. The signal is eventually inputted into an output audio device, which runs at 48 kHz. The model then uses a resampling factor of 44/179 to get close to 48 kHz, and results in decent audio quality. To obtain those demos, type `demo toolbox commblks` at the MATLAB command prompt, then navigate in the help browser to Simulink Demos → SDR Hardware.
3. Do not add any **Buffer** blocks to the model. Although it is tempting to create frame lengths other than 358, doing so by using a **Buffer** will severely degrade performance.
4. You should try to run with *Rapid Accelerator* instead of *Normal* mode. Be aware that some scopes do not plot data when run in *Rapid Accelerator* mode, but scopes inevitably slow down a model in any case. (See Point 6.)
5. Try to avoid feedback loops. Typically, such loops imply scalar processing, which will slow down the model considerably.
6. Perhaps the most obvious trick of all is not to use scopes unless absolutely necessary. To visualize your data, send it to a workspace variable and post-process it.
7. If you are using *Accelerator* or *Rapid Accelerator*, set the **Simulation** → **Configuration Parameters** → **Optimization** → **Compiler optimization level** to ‘Optimizations on (faster runs)’.

8. If the model has lots of **Constant** blocks, it could help slightly if **Simulation** → **Configuration Parameters** → **Optimization** → **Inline parameters** is checked on. This will cause the sample time of those **Constant** blocks to truly become inf such that Simulink will get the values once and only once during a run.
9. If the model generates code, the Solver setting should be *Fixed-step/discrete*. The tasking mode should be *Single Tasking*.

4 Open-ended Design Problem: Automatic Frequency Offset Compensator

4.1 Introduction

Beginning from this laboratory, you will be given an open-ended design problem at the end of each experiment. These problems are related to what you have learned and done in each laboratory, but requires additional thinking and investigation. There is no single solution for these problems, and each student group is expected to come up with their own innovative design.

4.2 Objective

The objective of this problem is to design and implement a software-defined radio (SDR) communication system capable of automatically calculating the frequency offset between two USRP platforms.

In Section 3.2, you have already found the frequency offset of two USRPs manually, namely, by comparing the FFT of the transmitted and received signals. In this problem, you are expected to discover this offset in an automatic way. In other words, in the receiver model you have designed, the only thing you need to do is to hit the “start simulation” button, which will result in this model providing you with the value of the frequency offset.

4.3 Theoretical Background

1. Before you apply your method to USRP2 boards, it is highly recommended that you test your method with Simulink-only model without USRP and see whether it works.
2. In the Simulink-only model, you can introduce the frequency offset by **Phase/Frequency Offset** block. Specify a frequency offset in this block, and see whether your Simulink model can give you the number you have specified.
3. An important hint: If you take the square of a signal, the FFT of the received signal will be shifted double of the frequency offset.
4. In your model, you might need the following blocks:
 - (a) **Random Integer Generator**
 - (b) **Baseband Modulator**
 - (c) **Raised Cosine Transmit Filter**
 - (d) **Magnitude FFT**
 - (e) **Probe**
 - (f) **Maximum**

The first three blocks are used on the transmitter side, and the last three blocks are used on the receiver side. Using **Probe** and **Maximum**, you will be able to find the location of the peak of the FFT. But of course, you are not constrained to these blocks. You can use all the blocks available in Simulink. If you want, you can even use MATLAB to implement your method.

5. Although you are required to find the frequency offset of two USRP2 boards, you are actually trying to find the frequency offset of the received signal.
6. Compare your result here from what you have obtained in Section 3.2.

5 Lab Report Preparation & Submission Instructions

Include all your answers, results, and source code in a laboratory report formatted as follows:

- Cover page: includes course number, laboratory title, names and student numbers of team, submission date.
- Table of contents, list of tables, list of figures.
- Pre-laboratory experiment (as an appendix).
- Responses to laboratory questions and explanation of observations.
- Responses to open-ended design problem.
- Source code (as an appendix).

Remember to write your laboratory report in a narrative approach, explaining your experience and observations in such a way that it provides the reader with some insight as to what you have accomplished. Furthermore, please include images and outputs wherever possible in your laboratory report document.

Each group is required to submit a single report electronically (in PDF format not exceeding 2MB) to `alexw@ece.wpi.edu` by the scheduled due date and time. Reports that do not meet these specifications will be returned without evaluation and will receive a grade of “0” for the report segment of the laboratory experiment.

References

- [1] USRP2 FAQ. [Online]: <http://gnuradio.org/redmine/wiki/gnuradio/USRP2GenFAQ>.
- [2] M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*. Dover Publications, June 1965.
- [3] IEEE 802.11 Working Group. IEEE 802.11: The working group setting the standards for wireless LANs. [Online]: <http://www.ieee802.org/11/>.
- [4] The MathWorks. Simulink documentation. <http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/index.html?/access/helpdesk/help/toolbox/simulink/>.
- [5] Athanasios Papoulis and S. Unnikrishna Pillai. *Probability, Random Variables and Stochastic Processes*. McGraw Hill Higher Education, 4th edition, 2002.
- [6] Michael Rice. *Digital Communications: A Discrete-Time Approach*. Pearson/Prentice Hall, Upper Saddle River, New Jersey, 2009.
- [7] Dennis Silage. *Digital Communication Systems using MATLAB and Simulink*. Bookstand Publishing, Gilroy, CA, 2009.