电子科技大学
格拉斯哥学院
Glasgow College, UESTC

Wireless & Optical Transmission Systems

Lab Manual

Student's Chinese Name	
Student's English Name	
Student's UESTC ID#	
Student's UoG ID#	

General Instructions

Here are some general instructions for Wireless & Optical Transmission Systems laboratory:

1. All students are advised to bring their lab manuals with them for each lab session. The lab manuals are supposed to be hard bound.
2. Students are required to complete the questionnaires at the end of each lab exercise.
3. A minimum of 8 out of the first 10 laboratory exercises must be completed to avoid CW.
4. The lab manuals must be submitted in the week following the next lab session.
5. All students must complete and sign the Declaration of Originality Form before submission of the lab manual.

Lab Rules and Health and Safety

Please read these rules carefully:

- Ask the teacher or a GTA in case of any problem.
- Familiarise yourself with the environment, locate the safety exits, the fire extinguisher, the first aid kit if available and the emergency contact numbers (UESTC teaching office).
- Leave your desk tidy as much as you can when you leave the lab.
- Do not drink or eat in the lab. Please throw away in the bin all your rubbish.
- Respect the lab equipment.
- Leave any item forgotten by other students in the teacher desk at the entrance of the lab.

University of Glasgow

Declaration of Originality Form

This form **must** be completed and signed and submitted with all assignments.

Please complete the information below (using BLOCK CAPITALS).

Name
Student Number
Course Name
Assignment Number/Name

An extract from the University's Statement on Plagiarism is provided overleaf. Please read carefully THEN read and sign the declaration below.

I confirm that this assignment is my own work and that I have:	
Read and understood the guidance on plagiarism in the Student Handbook, including the University of Glasgow Statement on Plagiarism	☐
Clearly referenced, in both the text and the bibliography or references, **all sources** used in the work	☐
Fully referenced (including page numbers) and used inverted commas for **all text quoted** from books, journals, web etc. (Please check with the Department which referencing style is to be used)	☐
Provided the sources for all tables, figures, data etc. that are not my own work	☐
Not made use of the work of any other student(s) past or present without acknowledgement. This includes any of my own work, that has been previously, or concurrently, submitted for assessment, either at this or any other educational institution, including school (see overleaf at 31.2)	☐
Not sought or used the services of any professional agencies to produce this work	☐
In addition, I understand that any false claim in respect of this work will result in disciplinary action in accordance with University regulations	☐

DECLARATION:

I am aware of and understand the University's policy on plagiarism and I certify that this assignment is my own work**,** except where indicated by referencing, and that I have followed the good academic practices noted above

Signed

The University of Glasgow Plagiarism Statement

The following is an extract from the University of Glasgow Plagiarism Statement. The full statement can be found in the University Calendar at http://www.gla.ac.uk/media/media_413985_en.pdf#page=53&view=fitH,225. This should be read in conjunction with the discipline specific guidance provided by the School.

31.1 The University's degrees and other academic awards are given in recognition of a student's **personal achievement**. All work submitted by students for assessment is accepted on the understanding that it is the student's own effort.

31.2 Plagiarism is defined as the submission or presentation of work, in any form, which is not one's own, without **acknowledgement of the sources**. Plagiarism includes inappropriate collaboration with others. Special cases of plagiarism can arise from a student using his or her own previous work (termed auto-plagiarism or self-plagiarism). Auto-plagiarism includes using work that has already been submitted for assessment at this University or for any other academic award.

31.3 The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism.

Work may be considered to be plagiarised if it consists of:

- a direct quotation;
- a close paraphrase;
- an unacknowledged summary of a source;
- direct copying or transcription.

With regard to essays, reports and dissertations, the rule is: if information **or ideas** are obtained from any source, that source must be acknowledged according to the appropriate convention in that discipline; and **any direct quotation must be placed in quotation marks** and the source cited immediately. Any failure to acknowledge adequately or to cite properly other sources in submitted work is plagiarism. Under examination conditions, material learnt by rote or close paraphrase will be expected to follow the usual rules of reference citation otherwise it will be considered as plagiarism. Departments should provide guidance on other appropriate use of references in examination conditions.

31.4 Plagiarism is considered to be an act of fraudulence and an offence against University discipline. Alleged plagiarism, at whatever stage of a student's studies, whether before or after graduation, will be investigated and dealt with appropriately by the University.

31.5 The University reserves the right to use plagiarism detection systems, which may be externally based, in the interests of improving academic standards when assessing student work.

If you are still unsure or unclear about what plagiarism is or need advice on how to avoid it,
SEEK HELP NOW! You can contact any one of the following for assistance:
Lecturer
Course Leader
Dissertation Supervisor
Adviser of Studies
Student Learning Service

OBJECTIVES

On completing these laboratories, you must be able to:

- declare and utilise MATLAB commands and codes
- generate wireless channel models
- analyse performance of wireless channels in presence of noise
- plot various performance metrics in logarithmic scale
- understand various blocks in Simulink (Communications Toolbox)
- design basic wireless system in Simulink (Communications Toolbox)

These exercises / tasks will be assessed by milestones.

Introduction to MATLAB

Matlab is a computing environment specially designed for matrix computations. It is widely utilised for the study of a variety of applications, including circuits, signal processing, control systems, communications, image processing, symbolic mathematics, statistics, neural networks, wavelets, and system identification. Its large library of built-in functions and toolboxes, as well as its graphical capabilities, makes it a valuable tool for electrical engineering education and research.

Matlab has an interactive mode in which user commands are interpreted immediately as they are typed. Alternatively, a program (called a script) can be written in advance using a text editor, saved as a file, and then executed in Matlab.

Introduction to Simulink

Simulink is a program for simulating signals and dynamic systems. As an extension of Matlab, Simulink adds many features specific to the simulation of dynamic systems while retaining all of Matlab's general purpose functionality.

Simulink has two phases of use: *model definition* and *model analysis*. A typical session starts by either defining a new model or by recalling a previously defined model, and then proceeds to analyse that model. In order to facilitate the model definition, Simulink has a large class of windows called block diagram windows. In these windows, models are created and edited principally by mouse-driven commands. An important part of mastering Simulink is to become familiar with manipulations of various model components in these windows.

After you define a model, you can analyse it either by choosing options from the Simulink menus or by entering commands in the Matlab command window. The progress of an ongoing simulation can be viewed while it is running, and the results can be made available in the Matlab workspace when the simulation is complete.

Introduction to Communications Toolbox

Communications Toolbox software implements a variety of communications-related tasks. Many of the functions in the toolbox perform computations associated with a component of a communication system, such as a demodulator or equaliser. Other functions are designed for visualisation or analysis.

EXERCISE 1: Random Signal

Process a binary data stream using a communication system that consists of a baseband modulator, channel, and demodulator. Compute the system's bit error rate (BER). Also, display the transmitted and received signals in a scatter plot.

The following table indicates the key tasks insolving the problem, along with relevant Communications Toolbox functions. The exercise arbitrarily chooses baseband 16-QAM (quadrature amplitude modulation) as the modulation scheme and AWGN (additive white Gaussian noise) as the channel model.

Task	Function or Method
Generate a random binary data stream	`randint`
Modulate using 16-QAM	`modulate` method on `modem.qammod` object
Add white Gaussian noise	`awgn`
Create a scatter plot	`scatterplot`
Demodulate using 16-QAM	`modulate` method on `modem.qamdemod` object
Compute the system's BER	`biterr`

The discussion below describes each step in more detail, introducing M-code along the way. To view all the code in one editor window, enter the following in the MATLAB Command Window.

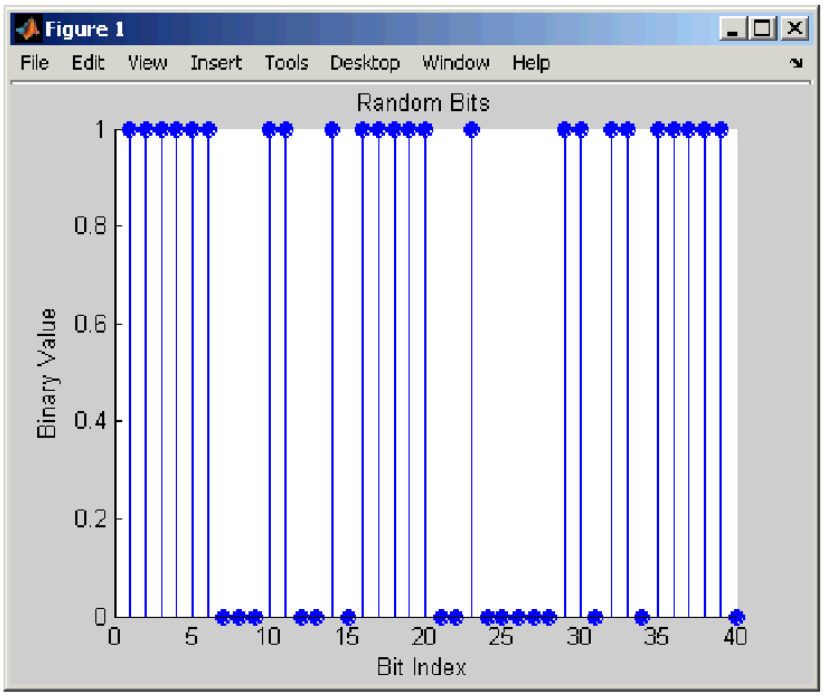
```
edit commdoc_mod
```

1. Generate a Random Binary Data Stream. The conventional format for representing a signal in MATLAB is a vector or matrix. This exercise uses the `randint` function to create a column vector that lists the successive values of a binary data stream. The length of the binary data stream (that is, the number of rows in the column vector) is arbitrarily set to 30,000.

Note: The sampling times associated with the bits do not appear explicitly, and MATLAB has no inherent notion of time. For the purpose of this exercise, knowing only the values in the data stream is enough to solve this.

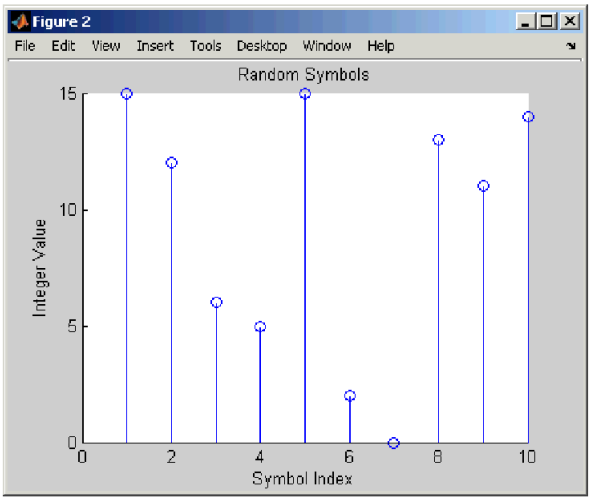
The code below also creates a stem plot of a portion of the data stream, showing the binary values. Your plot might look different because the exercise uses random numbers. Notice the use of the colon (:) operator in MATLAB to select a portion of the vector. For more information about this syntax, see "The Colon Operator" in the MATLAB documentation set.

```
%% Setup
% Define parameters.
M = 16; % Size of signal constellation
k = log2(M); % Number of bits per symbol
n = 3e4; % Number of bits to process
nsamp = 1; % Oversampling rate

%% Signal Source
% Create a binary data stream as a column vector.
x = randint(n,1); % Random binary data stream

% Plot first 40 bits in a stem plot.
stem(x(1:40),'filled');
title('Random Bits');
xlabel('Bit Index'); ylabel('Binary Value');
```



2. Prepare to Modulate. The `modem.qammod` object implements an M-ary QAM modulator, M being 16 in this exercise. It is configured to receive integers between 0 and 15 rather than 4-tuples of bits. Therefore, you must pre-process the binary data stream `x` before using the `modulate` method of the object. In particular, you arrange each 4-tuple of values from `x` across a row of a matrix, using the reshape function in MATLAB, and then apply the `bi2de` function to convert each 4-tuple to a corresponding integer. (The `.'` characters after the reshape command form the unconjugated array transpose operator in MATLAB. For more information about this and the similar `'` operator, see Reshaping a Matrix" in the MATLAB documentation set.)

```
%% Bit-to-Symbol Mapping
% Convert the bits in x into k-bit symbols.
xsym = bi2de(reshape(x,k,length(x)/k).','left-msb');

%% Stem Plot of Symbols
% Plot first 10 symbols in a stem plot.
figure; % Create new figure window.
stem(xsym(1:10));
title('Random Symbols');
xlabel('Symbol Index'); ylabel('Integer Value');
```



3. Modulate Using 16-QAM. Having defined `xsym` as a column vector containing integers between 0 and 15, you can use the `modulate` method of the `modem.qammod` object to modulate `xsym` using the baseband representation. Recall that `M` is 16, the alphabet size.

```
%% Modulation
y = modulate(modem.qammod(M), xsym); % Modulate using 16-QAM.
```

The result is a complex column vector whose values are in the 16-point QAM signal constellation. A later step in this exercise will show what the constellation looks like. Also, note that the `modulate` method of the `modem.qammod` object does not apply any pulse shaping. For an example that uses rectangular pulse shaping with PSK modulation, see `basicsimdemo`.

4. Add White Gaussian Noise. Applying the `awgn` function to the modulated signal adds white Gaussian noise to it. The ratio of bit energy to noise power spectral density, Eb/N0, is arbitrarily set at 10dB.

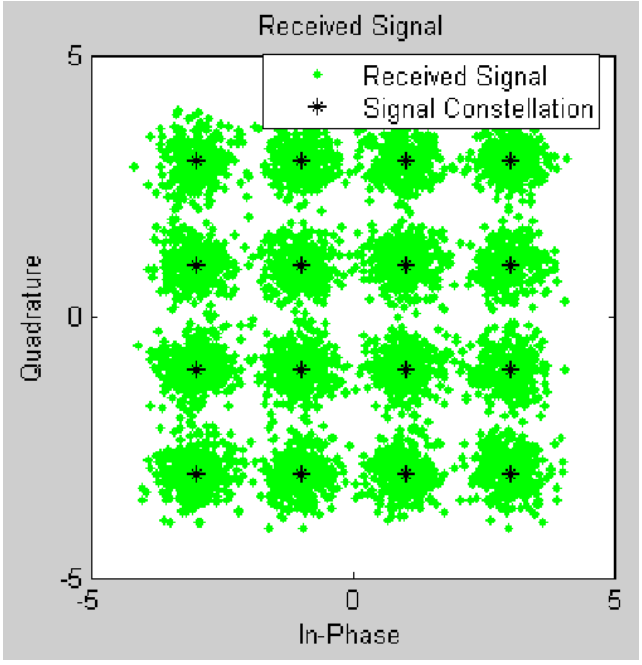
The expression to convert this value to the corresponding signal-to-noise ratio (SNR) involves `k`, the number of bits per symbol (which is 4 for 16-QAM), and `nsamp`, the oversampling factor (which is 1 in this exercise). The factor `k` is used to convert Eb/N0 to an equivalent Es/N0, which is the ratio of symbol energy to noise power spectral density. The

factor `nsamp` is used to convert Es/N0 in the symbol rate bandwidth to an SNR in the sampling bandwidth.

```
%% Transmitted Signal
ytx = y;

%% Channel
% Send signal over an AWGN channel.
EbNo = 10; % In dB
snr = EbNo + 10*log10(k) - 10*log10(nsamp);
ynoisy = awgn(ytx,snr,'measured');

%% Received Signal
yrx = ynoisy;
```

5. Create a Scatter Plot. Applying the `scatterplot` function to the transmitted and received signals shows what the signal constellation looks like and how the noise distorts the signal. In the plot, the horizontal axis is the in-phase component of the signal and the vertical axis is the quadrature component. The code below also uses the `title`, `legend`, and `axis` functions in MATLAB to customise the plot.

```
%% Scatter Plot
% Create scatter plot of noisy signal and transmitted
% signal on the same axes.
h = scatterplot(yrx(1:nsamp*5e3),nsamp,0,'g.');
hold on;
scatterplot(ytx(1:5e3),1,0,'k*',h);
title('Received Signal');
legend('Received Signal','Signal Constellation');
axis([-5 5 -5 5]); % Set axis ranges.
hold off;
```



6. Demodulate Using 16-QAM. Applying the `demodulate` method of the `modem.qamdemod` object to the received signal demodulates it. The result is a column vector containing integers between 0 and 15.

```
%% Demodulation
% Demodulate signal using 16-QAM.
zsym = demodulate(modem.qamdemod(M), yrx);
```

7. Convert the Integer-Valued Signal to a Binary Signal. The previous step produced `zsym`, a vector of integers. To obtain an equivalent binary signal, use the `de2bi` function to convert each integer to a corresponding binary 4-tuple along a row of a matrix. Then use the `reshape` function to arrange all the bits in a single column vector rather than a four-column matrix.

```
%% Symbol-to-Bit Mapping
% Undo the bit-to-symbol mapping performed earlier.
z = de2bi(zsym, 'left-msb'); % Convert integers to bits.
% Convert z from a matrix to a vector.
z = reshape(z.', prod(size(z)),1);
```

8. Compute the System's BER. Applying the `biterr` function to the original binary vector and to the binary vector from the demodulation step above yields the number of bit errors and the bit error rate.

```
%% BER Computation
% Compare x and z to obtain the number of errors and the bit
% error rate.
[number_of_errors, bit_error_rate] = biterr(x, z)
```

The statistics appear in the MATLAB Command Window. Your results might vary because the exercise uses random numbers.

```
number_of_errors =

    71


bit_error_rate =

    0.0024
```

Milestone: Demonstrate your code and the output to a demonstrator.

Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1. **2.**
Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.

Code: Please write your code here. Do not forget to include comments.
Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

EXERCISE 2: Scatter Plots

A scatter plot of a signal shows the signal's value at a given decision point. In the best case, the decision point should be at the time when the eye of the signal's eye diagram is the most widely open.

To produce a scatter plot from a signal, use the `scatterplot` object.

Scatter plots are often used to visualise the signal constellation associated with digital modulation. A scatter plot can be useful when comparing system performance to a published standard, such as 3GPP or DVB standards.

The scatter plot feature is part of the `commscope` package. Users can create the scatter plot object in two ways: using a default object or by defining parameter-value pairs. For more information, see the `commscope.ScatterPlot` help page.

Scatter Plots
In this exercise, you will observe the received signals for a QPSK modulated system. The output symbols are pulse shaped, using a raised cosine filter.

1 Create a QPSK modulator object. Type the following at the MATLAB command line:

```
hMod = modem.pskmod('M', 4, 'PhaseOffset', pi/4);
```

2 Create an upsampling filter, with an upsample rate of 16. Type the following at the MATLAB command line:

```
Rup = 16; % up sampling rate
hFilDesign = fdesign.pulseshaping(Rup, 'Raised Cosine', ...
'Nsym, Beta', Rup,0.50);
hFil = design(hFilDesign);
```

3 Create the transmit signal. TypethefollowingattheMATLAB command line:

```
d = randi([0 hMod.M-1], 100, 1); % Generate data symbols
sym = modulate(hMod, d); % Generate modulated symbols
xmt = filter(hFil, upsample(sym, Rup));
```

4 Create a scatter plot and set the samples per symbol to the upsampling rate of the signal. Type the following at the MATLAB command line:

```
hScope = commscope.ScatterPlot
hScope.SamplesPerSymbol = Rup;
```

In this simulation, the absolute sampling rate or symbol rate is not specified. Use the default value for Sampling Frequency, which is 8000. This results in 2000 symbols per second symbol rate.

5 Set the constellation value of the scatter plot to the expected constellation. Type the following at the MATLAB command line:

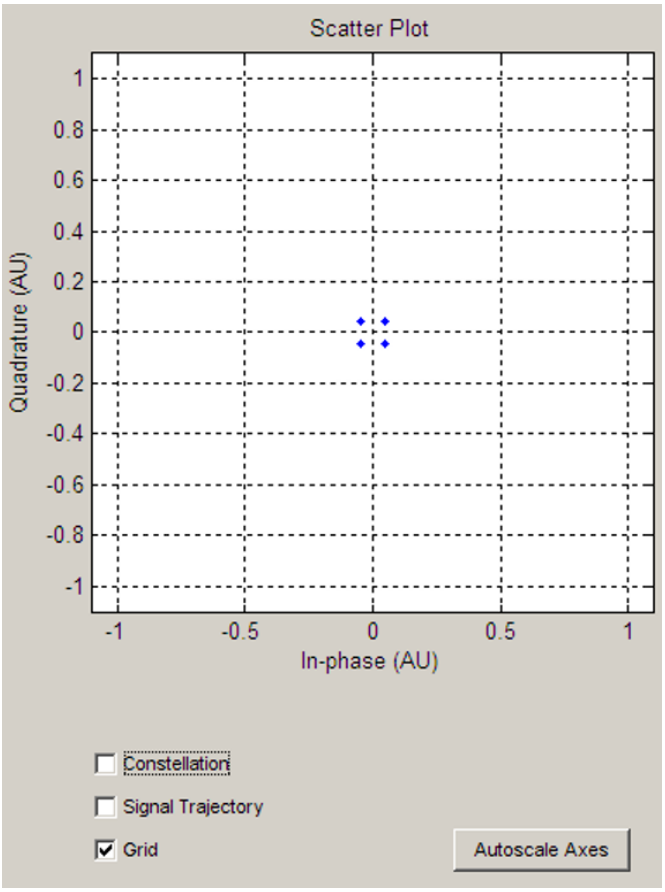
```
hScope.Constellation = hMod.Constellation;
```

6 Since the pulse shaping filter introduces a delay, discard these transient values by setting `MeasurementDelay` to the group delay of the filter, which is four symbol durations or 4/Rs seconds. Type the following at the MATLAB command line:

```
groupDelay = (hFilDesign.NumberOfSymbols/2);
hScope.MeasurementDelay = groupDelay /hScope.SymbolRate;
```

7 Update the scatter plot with transmitted signal by typing the following at the MATLAB command line:

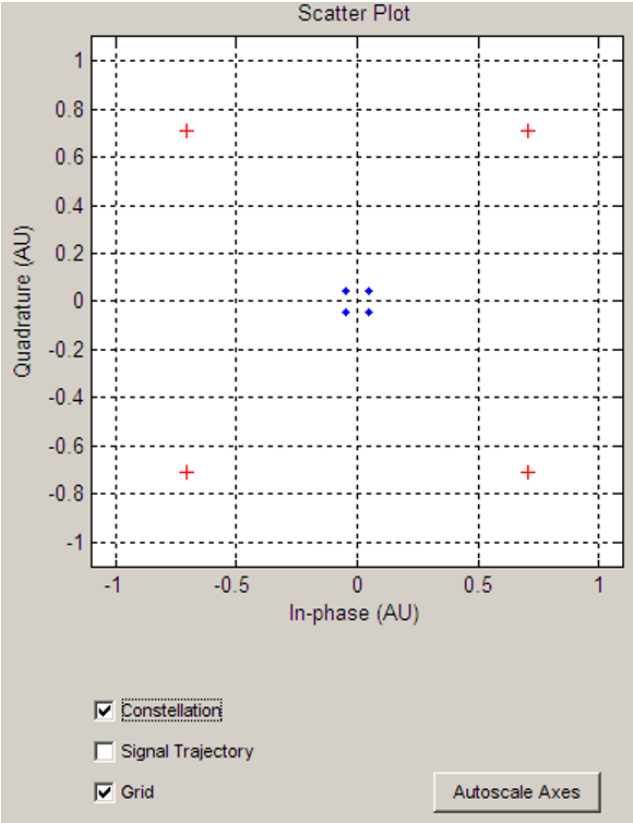
```
update(hScope, xmt)
```



The Figure window updates, displaying the transmitted signal.

8 Display the ideal constellation and evaluate how closely it matches the transmitted signal. To display the ideal constellation, type the following at the MATLAB command line:

```
hScope.PlotSettings.Constellation = 'on';
```



The Figure window updates, displaying the ideal constellation and the transmitted signal.

9 One way to create a better match between the two signals is to normalise the filter. Normalise the filter by typing the following at the MATLAB command line:

```
hFil.Numerator = hFil.Numerator / max(hFil.Numerator);
```

10 Refilter the signal using a normalised filter.

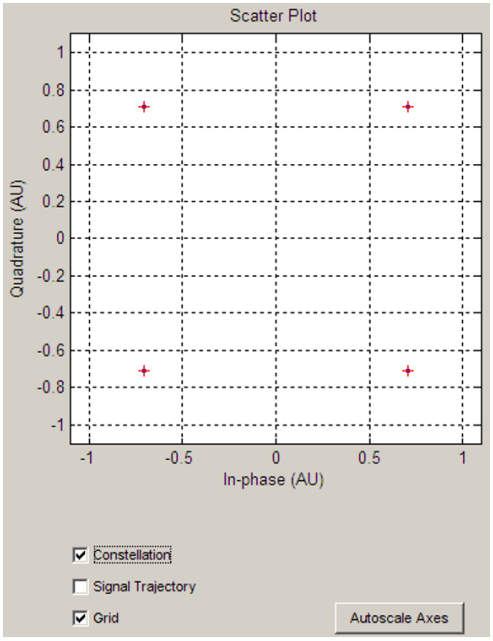
```
xmt = filter(hFil, upsample(sym, Rup));
```

11 Reset the scope before displaying the transmitted signal. Resetting the scope also resets the counter for measurement delay, discarding the transient filter values. To reset the scope, type the following at the MATLAB command line:

```
reset(hScope)
```

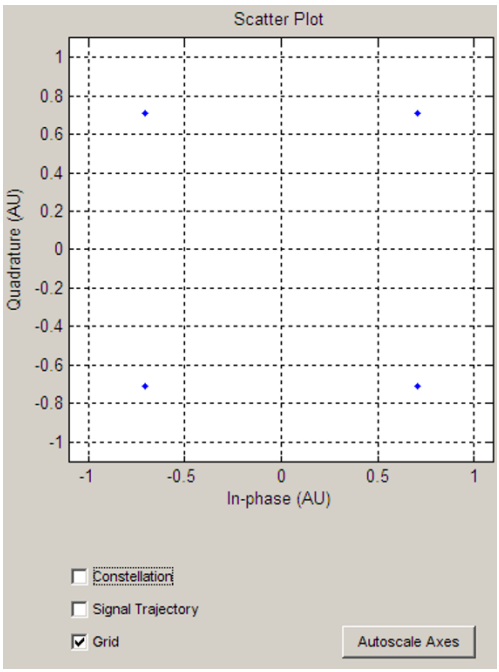
12 Update the scatter plot so it displays the signal.

```
Update(hScope, xmt)
```



The match between the ideal constellation points and the transmitted signal is nearly identical.

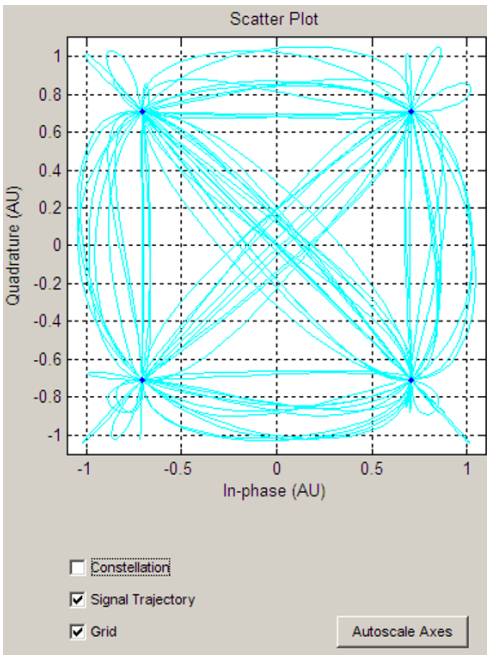
13 To view the transmitted signal more clearly, turn off the ideal constellation by clicking **Constellation** in the Figure window.



The Figure window updates, displaying only the transmitted signal.

14 View the signal trajectory. Type the following at the MATLAB command line:

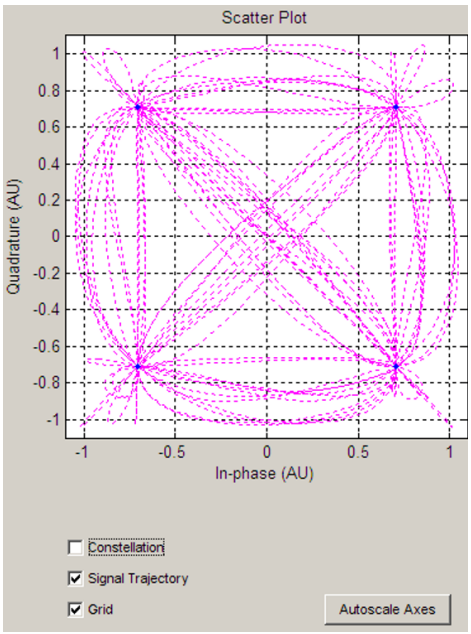
```
hScope.PlotSettings.SignalTrajectory = 'on';
```



The Figure window updates, displaying the trajectory. An alternate way to display the signal trajectory is to click the **Signal Trajectory**.

15 Change the linestyle. Type the following at the MATLAB command line:

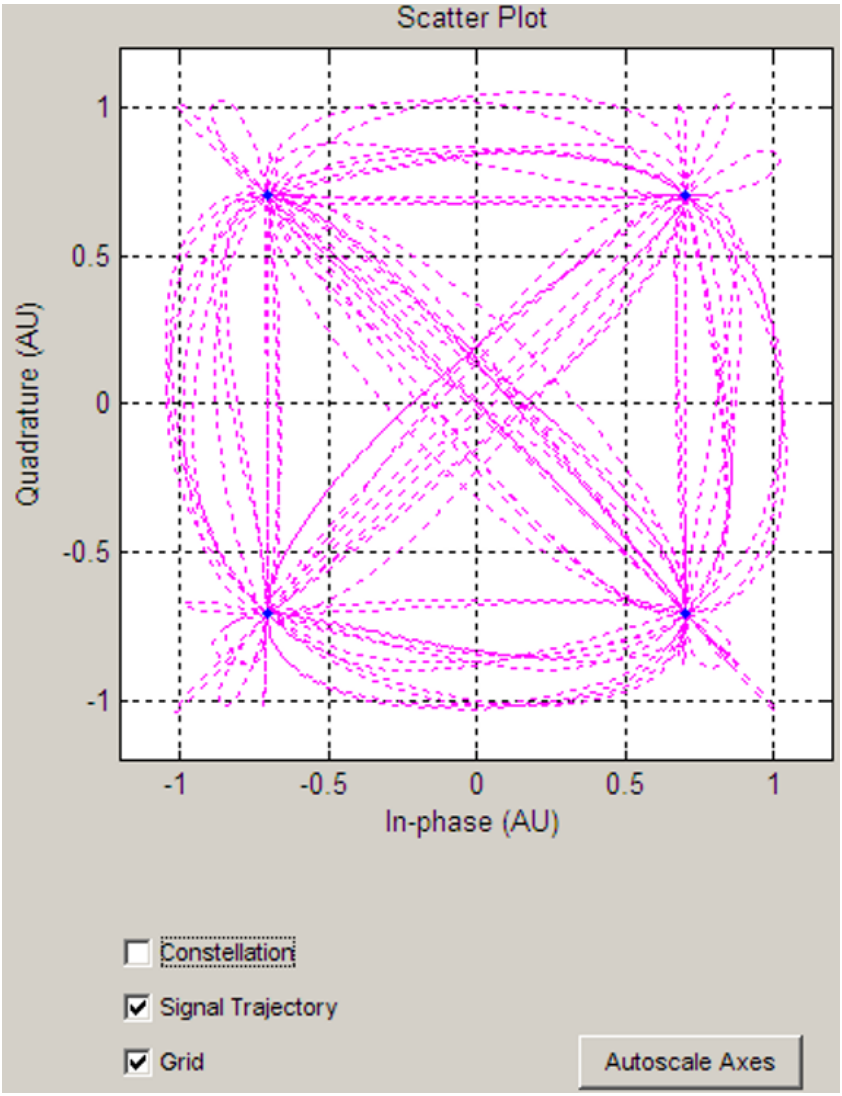
```
hScope.PlotSettings.SignalTrajectoryStyle = ':m';
```



The Figure window updates, changing the linestyle making up the signal trajectory.

16 Autoscale the scatterplot display to fit the entire plot. Type the following at the MATLAB command line:

```
autoscale(hScope)
```



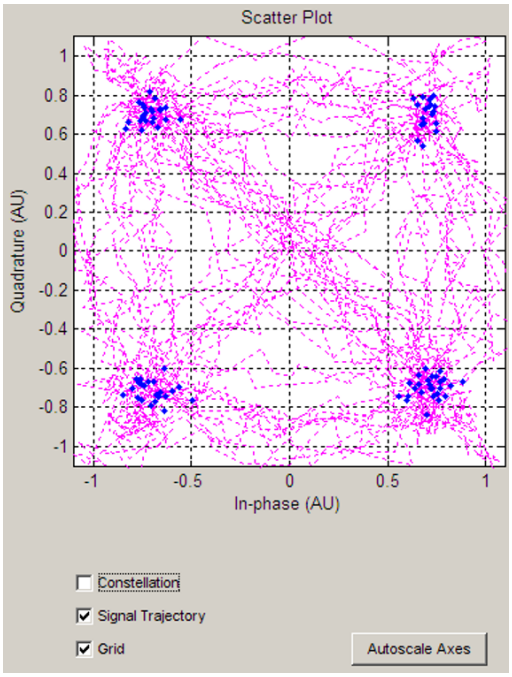
The Figure window updates. An alternate way to autoscale the fit is to click the **Autoscale Axes** button.

17 Create a noisy signal by Passing `xmt` through an AWGN channel. Type the following at the MATLAB command line:

```
rcv = awgn(xmt, 20, 'measured'); % Add AWGN
```

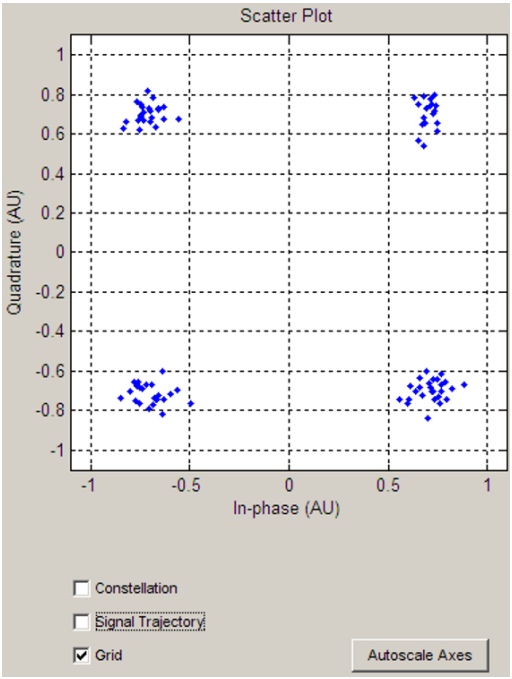
18 Send the received signal to the scatter plot. Before sending the signal, reset the scatter plot to remove the old data. Type the following at the MATLAB command line:

```
reset(hScope) update(hScope, rcv)
```



The Figure window updates, displaying the noisy signal.

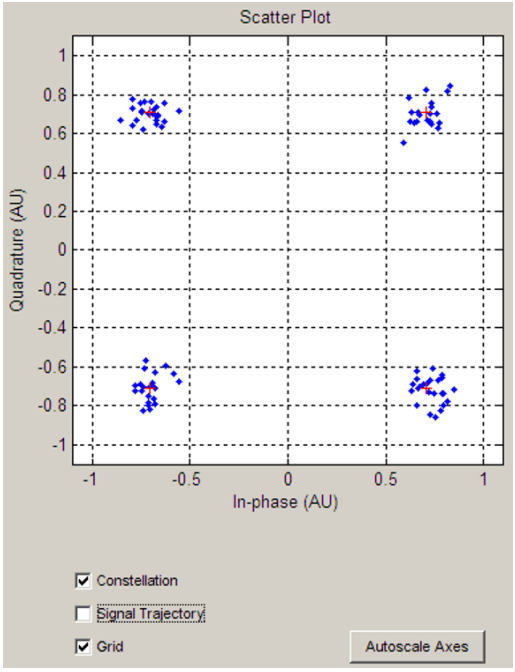
19 Turn off the signal trajectory by clicking **Signal Trajectory** in the Figure window.



The Figure window updates, displaying the signal plot without the signal trajectory. An alternate way to turn off the signal trajectory is typing the following at the MATLAB command line:

```
hScope.PlotSettings.SignalTrajectory = 'off';
```

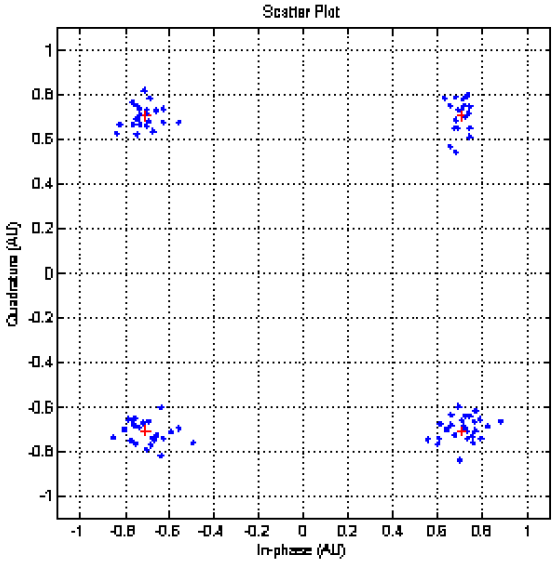
20 View the constellation by clicking **Constellation** in the Figure window.



The Figure window updates, displaying both the ideal constellation and the transmitted signal. An alternate way to view the constellation is by typing the following at the MATLAB command line:

```
hScope.PlotSettings.Constellation = 'on';
```

21 Print the scatter plot by making the following selection in the Figure window: **File > Print**



When you print the scatter plot, you print the axes, not the entire GUI.

Milestone: Demonstrate your code and the output to a demonstrator.

Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1. **2.**
Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.

Code: Please write your code here. Do not forget to include comments.
Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

EXERCISE 3: Signal Constellations

To plot the signal constellation associated with a modulation process, follow these steps:

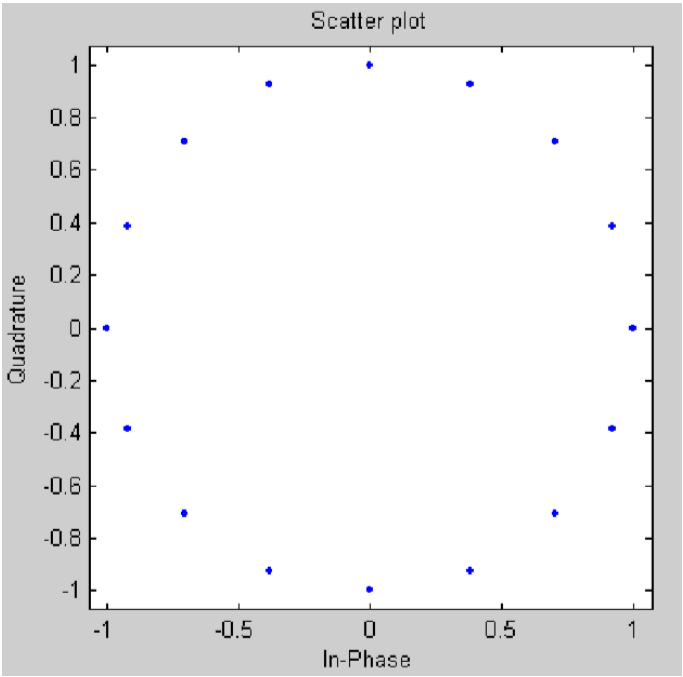
1 If the alphabet size for the modulation process is M, then create the signal `[0:M-1]`. This signal represents all possible inputs to the modulator.

2 Use the appropriate modulation function to modulate this signal. If desired, scale the output. The result is the set of all points of the signal constellation.

3 Apply the scatterplot function to the modulated output to create a plot.

Constellation for 16-PSK. The code below plots a PSK constellation having 16 points.

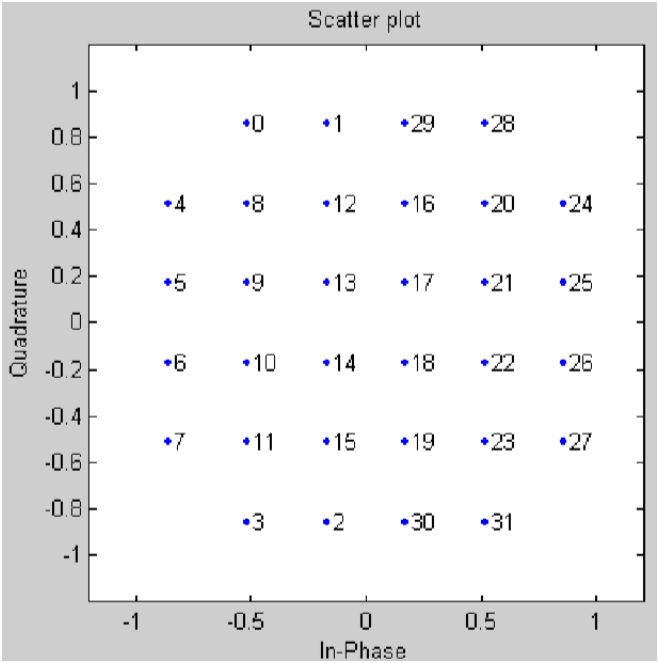
```
M = 16;
x = [0:M-1];
scatterplot(modulate(modem.pskmod(M), x));
```



Constellation for 32-QAM. The code below plots a QAM constellation having 32 points and a peak power of 1 watt. The exercise also illustrates how to label the plot with the numbers that form the input to the modulator.

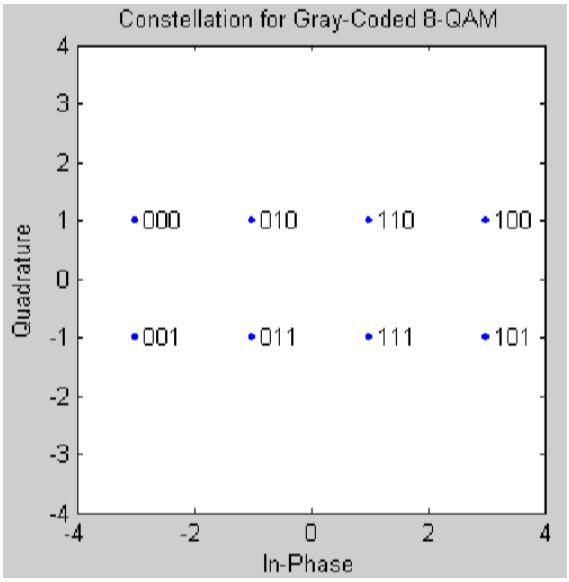
```
M = 32;
x = [0:M-1];
y = modulate(modem.qammod(M), x);
scale = modnorm(y, 'peakpow', 1);
y = scale*y; % Scale the constellation.
```

```
scatterplot(y); % Plot the scaled constellation.

% Include text annotations that number the points.
hold on; % Make sure the annotations go in the same figure.
for jj=1:length(y)
     text(real(y(jj)), imag(y(jj)), [' ' num2str(jj-1)]);
end
hold off;
```



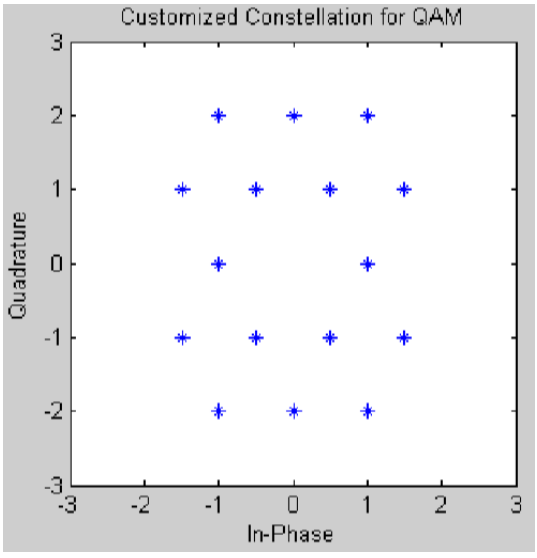
Gray-Coded Signal Constellation. The exercise below plots an 8-QAM signal Gray-coded constellation, labelling the points using binary numbers so you can verify visually that the constellation uses Graycoding.

```
M = 8;
x = [0:M-1];
y = modulate(modem.qammod('M', M, 'SymbolOrder', 'Gray'), x);
% Plot the Gray-coded constellation.
scatterplot(y, 1, 0, 'b.'); % Dots for points.
% Include text annotations that number the points in binary.
hold on; % Make sure the annotations go in the same figure.
annot = dec2bin([0:length(y)-1], log2(M));
text(real(y)+0.15, imag(y), annot);
axis([-4 4 -4 4]);
title('Constellation for Gray-Coded 8-QAM');
hold off;
```



Customised Constellation for QAM. The code below describes and plots a constellation with a customised structure.

```
% Describe constellation.
inphase = [1/2 -1/2 1 0 3/2 -3/2 1 -1];
quadr = [1 1 0 2 1 1 2 2];
inphase = [inphase; -inphase]; inphase = inphase(:);
quadr = [quadr; -quadr]; quadr = quadr(:);
const = inphase + j*quadr;

% Plot constellation.
scatterplot(const, 1, 0, '*');
hold on;
axis([-3 3 -3 3]);
title('Customised Constellation for QAM');
hold off;
```



Milestone: Demonstrate your code and the output to a demonstrator.

Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1. **2.**
Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.

Code: Please write your code here. Do not forget to include comments.
Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

EXERCISE 4: Using BERTool

The exercise above has performed tasks associated with various components of a communication system. In some cases, you might need to create a more sophisticated simulation that uses one or more of these techniques:

- Looping over a set of values of a specific parameter, such as Eb/N0, the alphabet size, or the oversampling rate, so you can see the parameter's effect on the system
- Processing data in multiple smaller sets rather than in one large set, to reduce the memory requirement
- Dynamically determining how much data to process to get reliable results, instead of trying to guess at the beginning

This exercise discusses these issues and provides exercises of constructs that you can use in your simulations of communication systems.

Using BERTool to Run Simulations

Communications Toolbox software includes a graphical user interface called BERTool. Using the BERTool GUI, you can solve problems like the following:

Modify the previous Exercise 1 so that it computes the BER for integer values of EbNo between 0 and 7. Plot the BER as a function of EbNo using a logarithmic scale for the vertical axis.

BERTool solves the problem by managing a series of simulations with different values of Eb/N0, collecting the results, and creating a plot. You provide the core of the simulation, which in this case is a minor modification of the previous exercise. This exercise introduces BERTool as well as some simulation-related issues, in these topics:

- "Comparing with Theoretical Results"
- "More About the Simulation Structure".

This exercise uses code from `commdoc_gray.m` as well as code from a template file that is tailored for use with BERTool. To view the original code in an editor window, enter these commands in the MATLAB Command Window.

```
edit commdoc_gray
edit bertooltemplate
```

To view a completed M-file for this exercise, enter `edit commdoc_bertool` in the MATLAB Command Window.

1. Save Template in Your Own Directory. Navigate to a directory where you want to save your own files. Save the BERTool template (`bertooltemplate`) under the file name `my_commdoc_bertool` to avoid overwriting the original template.

Also, change the first line of `my_commdoc_bertool`, which is the function declaration, to use the new filename.

```
function [ber, numBits] = my_commdoc_bertool(EbNo, maxNumErrs,
maxNumBits)
```

2. Copy Setup Code Into Template. In the `my_commdoc_bertool` file, replace

```
% --- Set up parameters. –
% --- INSERT YOUR CODE HERE.
```

with the following setup code adapted from the example in `commdoc_gray.m`.

```
% Setup
% Define parameters.
M = 16; % Size of signal constellation
k = log2(M); % Number of bits per symbol
n = 1000; % Number of bits to process
nsamp = 1; % Oversampling rate
```

To save time in the simulation, the code above changes the value of `n` from its original value. At small values of EbNo, it is not necessary to process tens of thousands of symbols to compute an accurate BER; at large values of EbNo, the loop structure in the template file (described later) causes the simulation to include at least 100 errors even if it must iterate several times through the loop to accumulate that many errors.

3. Copy Simulation Code Into Template. In the `my_commdoc_bertool` file, replace

```
% --- Proceed with simulation.
% --- Be sure to update totErr and numBits.
% --- INSERT YOUR CODE HERE.
```

with the rest of the code (that is, the code following the Setup section) from the example in `commdoc_gray.m`. Also, type a semicolon at the end of the last line of the pasted code (the `biterr` command) to suppress screen output when BERTool runs the simulation.

4. Update numBits and totErr. After the pasted code from the last step and before the end statement from the template, insert the following code.

```
%% Update totErr and numBits.
totErr = totErr + number_of_errors;
numBits = numBits + n;
```

These commands enable the function to keep track of the number of bits processed and the number of errors detected.

5. Suppress Earlier Plots. Running multiple iterations would result in many plots, which this exercise suppresses for simplicity. In the `my_commdoc_bertool` file, remove the lines of code that use these functions: `stem`, `title`, `xlabel`, `ylabel`, `figure`, `scatterplot`, `hold`, `legend`, and `axis`.

6. Omit Direct Assignment of EbNo. When BERTool invokes a simulation function, it specifies a value of EbNo. The `my_commdoc_bertool` function must not directly assign EbNo. Therefore, remove or comment out the line that you pasted into `my_commdoc_bertool` (within the Channel section) that assigns EbNo directly.

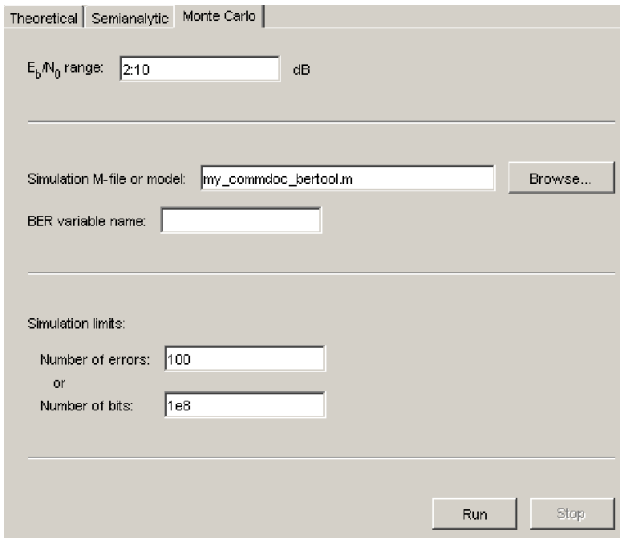
```
% EbNo = 10; % In dB % COMMENT OUT FOR BERTOOL
```

7. Save Simulation Function. The simulation function, `my_commdoc_bertool`, is complete. Save the file so that BERTool can use it.

8. Open BERTool and Enter Parameters. To open BERTool, enter

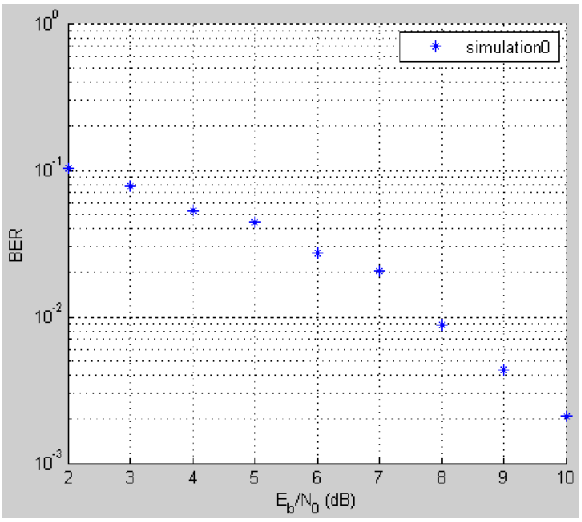
```
Bertool
```

in the MATLAB Command Window. Then click the **Monte Carlo** tab and enter parameters as shown below.



These parameters tell BERTool to run your simulation function, `my_commdoc_bertool`, for each value of EbNo in the vector `2:10` (that is, the vector `[2 3 4 5 6 7 8 9 10]`). Each time the simulation runs, it continues processing data until it detects 100 bit errors or processes a total of 1e8bits, whichever occurs first.

9. Use BERTool to Simulate and Plot. Click the **Run** button on BERTool. BERTool begins the series of simulations and eventually reports the results to you in a plot like the one below.

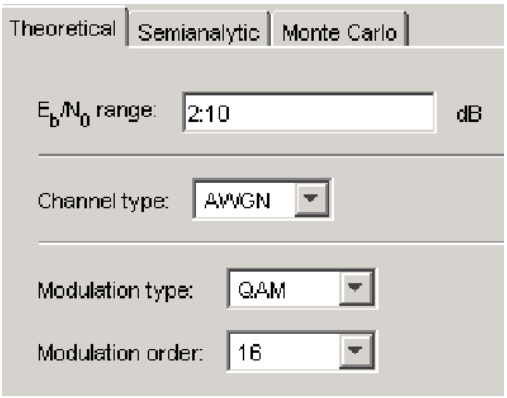


To compare these BER results with theoretical results, leave BERTool open and use the procedure below.

Comparing with Theoretical Results

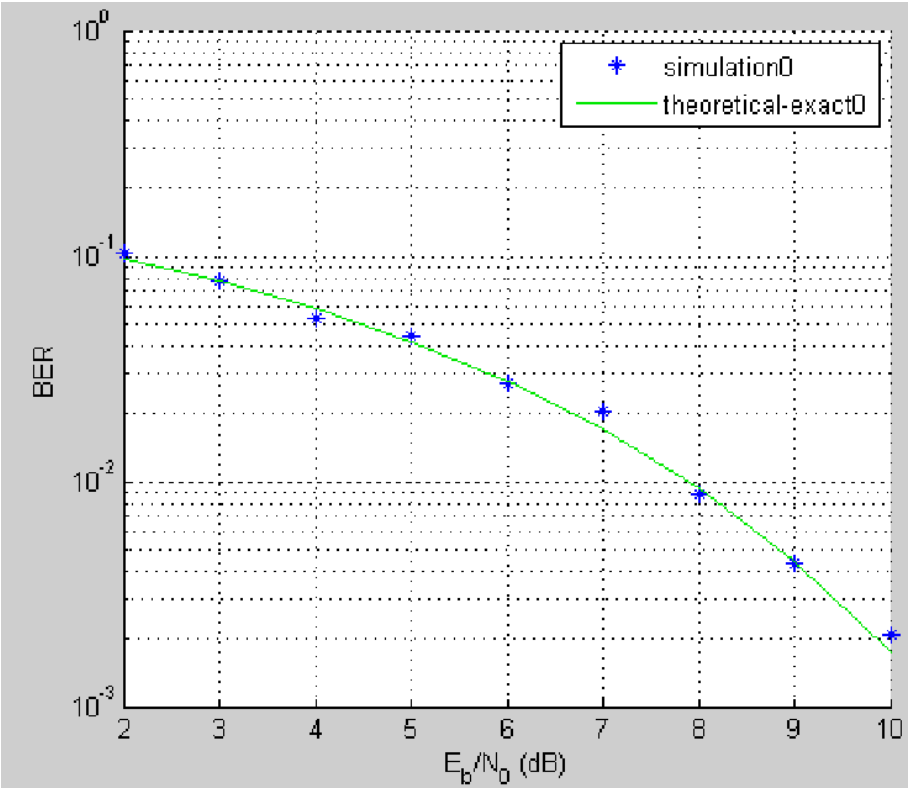
To check whether the results from the exercise above are correct, use BERTool again. This time, use its **Theoretical** panel to plot theoretical BER results in the same window as the simulation results from before. Follow this procedure:

1 In the BERTool GUI, click the **Theoretical** tab and enter parameters as shown below.



The parameters tell BERTool to compute theoretical BER results for 16-QAM over an AWGN channel, for Eb/N0 values in the vector `2:10`.

2 Click the **Plot** button. The resulting plot shows a solid curve for the theoretical BER results and plotting markers for the earlier simulation results.



Notice that the plotting markers are close to the theoretical curve. It is relevant that the simulation code used a Gray-coded signal constellation, unlike the previous exercise. The theoretical performance results assume a Gray-coded signal constellation. To continue exploring BERTool, you can select the **Fit** check box to fit a curve to the simulation data, or set Confidence Level to a numerical value to include confidence intervals in the plot.

Milestone: Demonstrate your code and the output to a demonstrator.

Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1. **2.**

Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.

Code: Please write your code here. Do not forget to include comments.
Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

EXERCISE 5: Preparing Simulink Models for Use with BERTool

A Simulink model must satisfy these requirements before you can use it with BERTool, where the case-sensitive variable names must be exactly as shown below:

- The channel block must use the variable EbNo rather than a hard-coded value for Eb/N0.
- The simulation must stop when the error count reaches the value of the variable `maxNumErrs` or when the number of processed bits reaches the value of the variable `maxNumBits`, whichever occurs first.
  You can configure the Error Rate Calculation block in Communications Blockset software to stop the simulation based on such criteria.
- The simulation must send the final error rate data to the MATLAB workspace as a variable whose name you enter in the **BER variable name** field in BERTool. The variable must be a three-element vector that lists the BER, the number of bit errors, and the number of processed bits.
  This three-element vector format is supported by the Error Rate Calculation block.

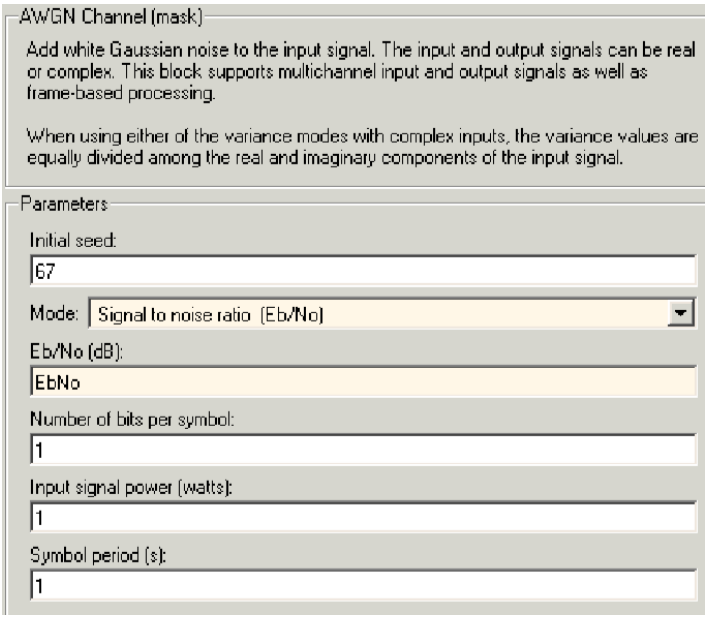
Tips for Preparing Models

Here are some tips for preparing a Simulink model for use with BERTool:

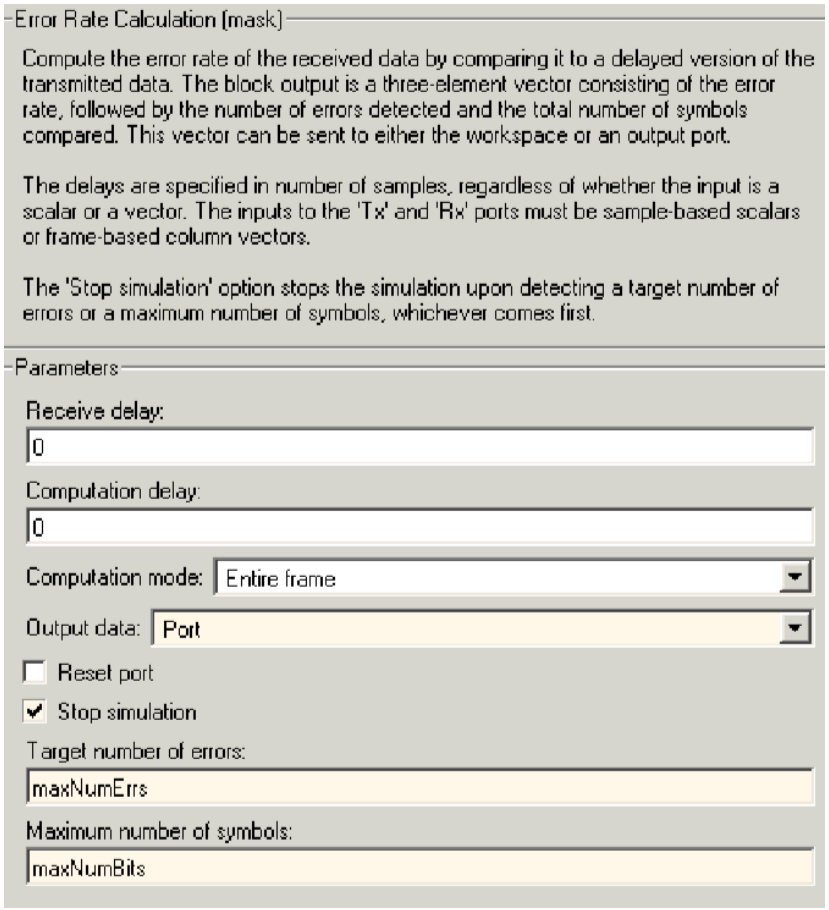
- To avoid using an undefined variable name in the dialog box for a Simulink block in the steps that follow, setup variables in the MATLAB workspace using a command such as the one below.

  ```
  EbNo = 0; maxNumErrs = 100; maxNumBits = 1e8;
  ```

  You might also want to put the same command in the model's preload function callback, to initialise the variables if you reopen the model in a future MATLAB session.
  When you use BERTool, it provides the actual values based on what you enter in the GUI, so the initial values above are somewhat arbitrary.
- To model the channel, use the AWGN Channel block in Communications Blockset software with these parameters:
  - **Mode** = Signal to noise ratio (Eb/No)
  - **Eb/No** = EbNo



- To compute the error rate, use the Error Rate Calculation block in Communications Blockset software with these parameters:
  - Check **Stop simulation**.
  - **Target number of errors** = `maxNumErrs`
  - **Maximum number of symbols** = `maxNumBits`



- To send data from the Error Rate Calculation block to the MATLAB workspace, set **Output data** to `Port`, attach a Signal to Workspace block from Signal Processing Blockset™ software, and set the latter block's **Limit data points to last** parameter to `1`. The **Variable name** parameter in the Signal to Workspace block must match the value you enter in the **BER variable name** field of BERTool.
- If your model computes a symbol error rate instead of a bit error rate, use the Integer to Bit Converter block in Communications Blockset software to convert symbols to bits.
- Frame-based simulations often run faster than sample-based simulations for the same number of bits processed. The number of errors or number of processed bits might exceed the values you enter in BERTool, because the simulation always processes a fixed amount of data in each frame.
- If you have an existing model that uses the AWGN Channel block using a **Mode** parameter other than `Signal to noise ratio (Eb/No)`, you can adapt the block to use the Eb/No mode instead. To learn about how the block's different modes are related to each other, press the AWGN Channel block's **Help** button to view the online reference page.
- If your model uses a preload function or other callback to initialise variables in the MATLAB workspace upon loading, make sure before you use the **Run** button in BERTool that one of these conditions is met:
    - The model is not currently in memory. In this case, BERTool loads the model into memory and runs the callback functions.
    - The model is in memory (whether in a window or not), and the variables are intact.

  If you clear or overwrite the model's variables and want to restore their values before using the **Run** button in BERTool, you can use the `bdclose` function in the MATLAB Command Window to clear the model from memory. This causes BERTool to reload the model after you click **Run**. Similarly, if you refresh your workspace by issuing a `clear all` or `clear variables` command, you should also clear the model from memory by using `bdclose all`.

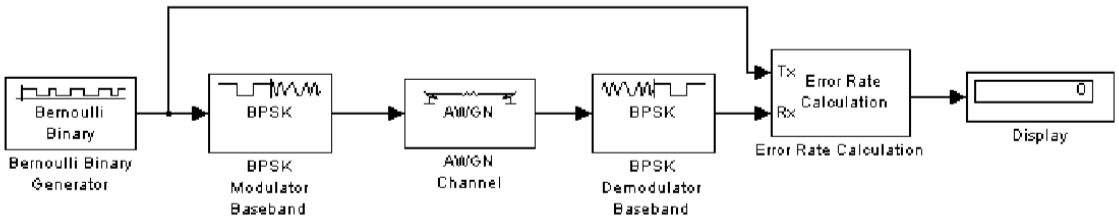
Preparing a Model for Use with BERTool

This exercise starts from a Simulink model originally created as an example in the Communications Blockset Getting Started documentation and shows how to tailor the model for use with BERTool. The exercise also illustrates how to compare the BER performance of a Simulink simulation with theoretical BER results. The exercise assumes that you have Communications Blockset software installed.

To prepare the model for use with BERTool, follow these steps, using the exact case-sensitive variable names as shown:

1 Open the model by entering the following command in the MATLAB Command Window.

```
bpskdoc
```



2 To initialise parameters in the MATLAB workspace and avoid using undefined variables as block parameters, enter the following command in the MATLAB Command Window.

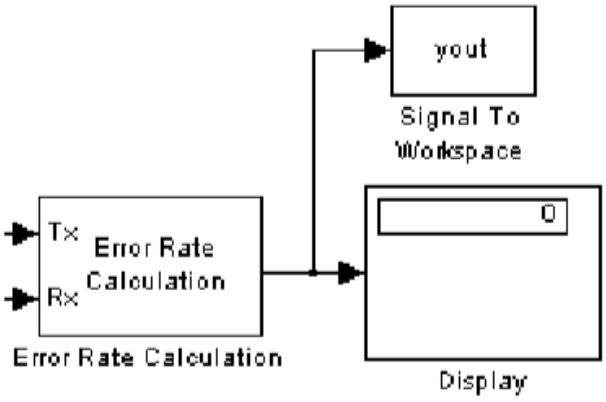
```
EbNo = 0; maxNumErrs = 100; maxNumBits = 1e8;
```

3 To ensure that BERTool uses the correct amount of noise each time it runs the simulation, open the dialog box for the AWGN Channel block by double-clicking the block. Set **Es/No** to `EbNo` and click **OK**. In this particular model, Es/N0 is equivalent to Eb/N0 because the modulation type is BPSK.

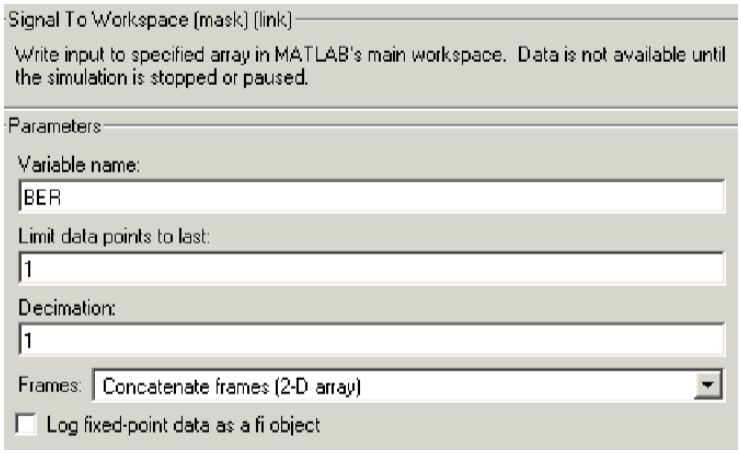
4 To ensure that BERTool uses the correct stopping criteria for each iteration, open the dialog box for the Error Rate Calculation block. Set **Target number of errors** to `maxNumErrs`, set **Maximum number of symbols** to `maxNumBits`, and click **OK**.

5 To enable BERTool to access the BER results that the Error Rate Calculation block computes, insert a Signal to Workspace block in the model and connect it to the output of the Error Rate Calculation block.

Note: The Signal to Workspace block is in Signal Processing Blockset software and is different from the To Workspace block in Simulink.



6 To configure the newly added Signal to Workspace block, open its dialog box. Set **Variable name** to `BER`, set **Limit data points to last** to `1`, and click **OK**.



7 (*Optional*) To make the simulation run faster, especially at high values of Eb/N0, open the dialog box for the Bernoulli Binary Generator block. Select **Frame-based outputs** and set **Samples per frame** to 1000.

8 Save the model in a directory on your MATLAB path using the file name `bertool_bpskdoc.mdl`.

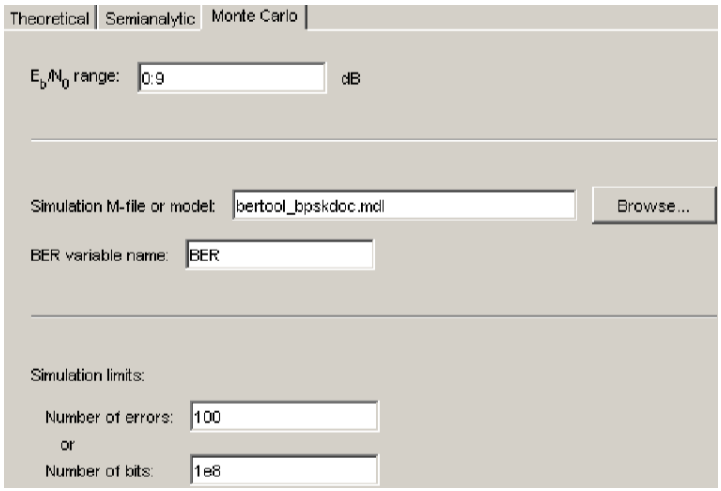
9 (*Optional*) To cause Simulink to initialise parameters if you reopen this model in a future MATLAB session, enter the following command in the MATLAB Command Window and resave the model.

```
set_param('bertool_bpskdoc','preLoadFcn',...
'EbNo = 0; maxNumErrs = 100; maxNumBits = 1e8;');
```

The `bertool_bpskdoc` model is now compatible with BERTool. To use it in conjunction with BERTool, continue the exercise by following these steps:

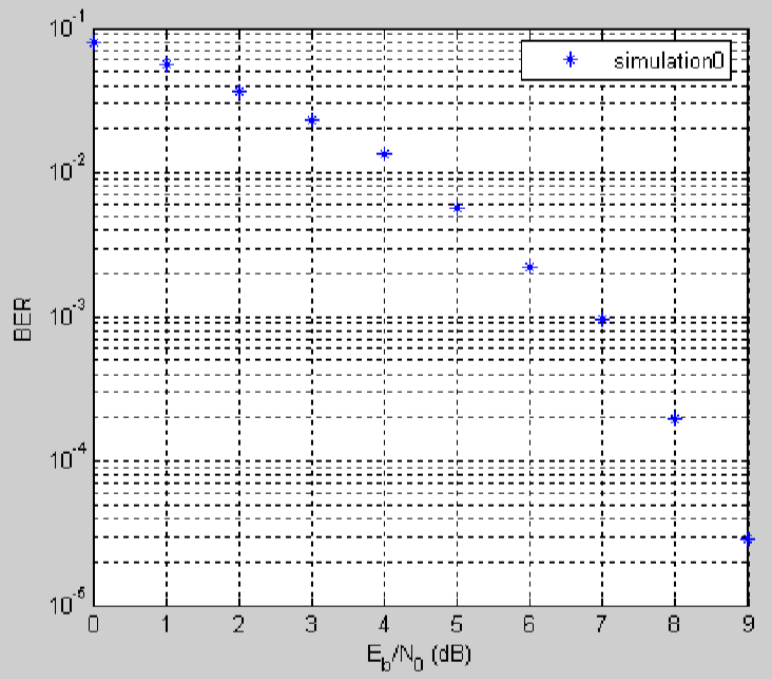
10 Open BERTool and go to the **Monte Carlo** tab.

11 Set parameters on the **Monte Carlo** tab as shown in the following figure.

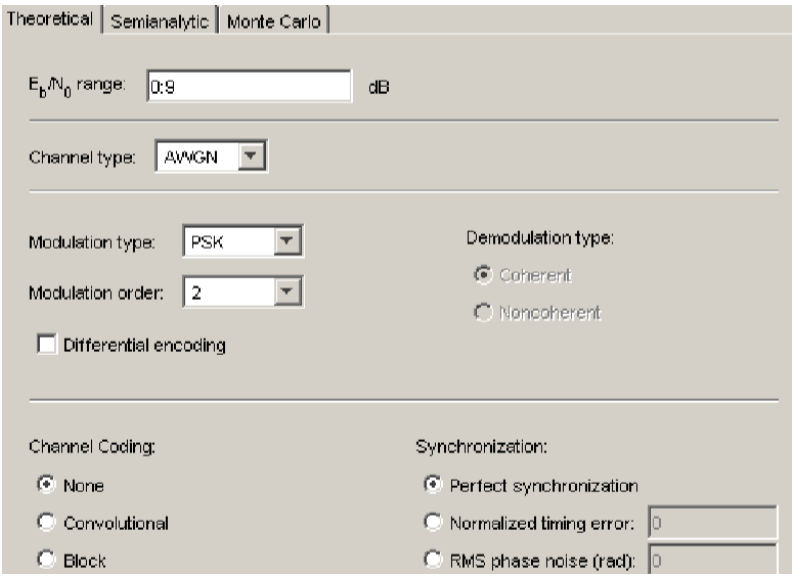


12 Click **Run**.

BER Tool spends some time computing results and then plots them.

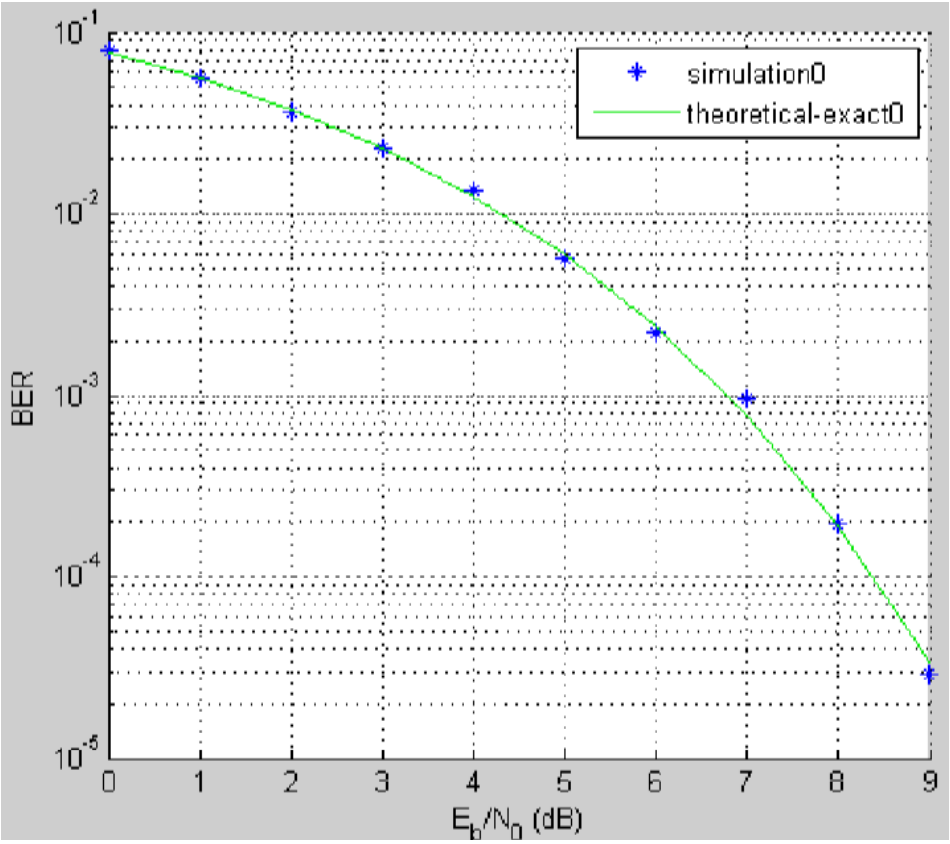


13 To compare these simulation results with theoretical results, go to the **Theoretical** tab in BERTool and set parameters as shown below.



14 Click **Plot**.

BERTool plots the theoretical curve in the BER Figure window along with the earlier simulation results.



Milestone: Demonstrate your code and the output to a demonstrator.

Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1. **2.**

Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.

Code: Please write your code here. Do not forget to include comments.
Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

EXERCISE 6: Eye Diagram

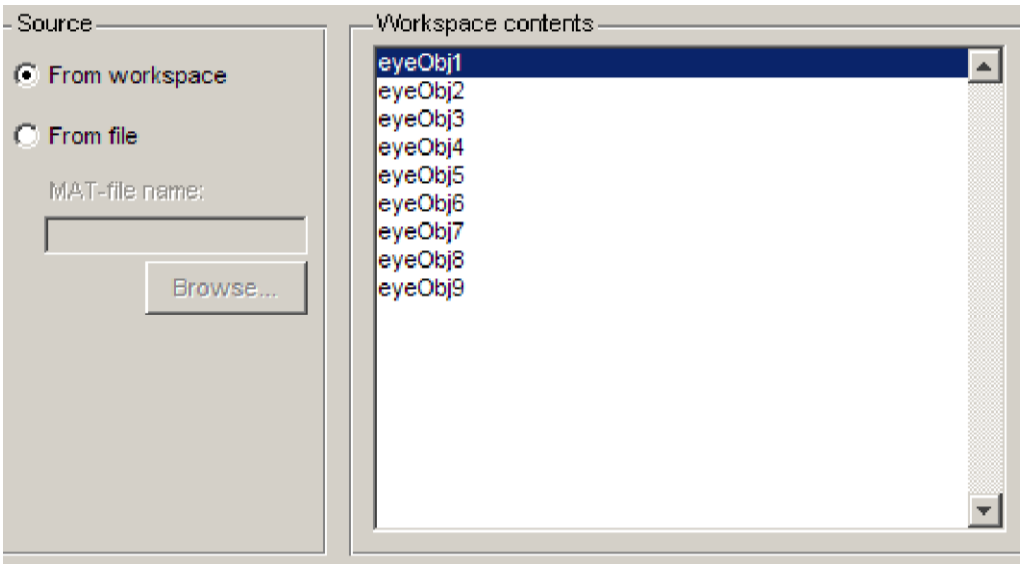
This exercise provides a step-by-step introduction for using `EyeScope` to import eye diagram objects, select and change which eye diagram measurements `EyeScope` displays, compare measurement results, and print a plot object.
MATLAB software includes a set of data containing nine eye diagram objects, which you can import into `EyeScope`. While `EyeScope` can import eye diagram objects from either the workspace or a MAT-file, this introduction covers importing from the workspace. `EyeScope` reconstructs the variable names it imports to reflect the origin of the eye diagram object.

1 Type load `commeye_EyeMeasureDemoData` at the MATLAB command line to load nine eye diagram objects into the MATLAB workspace.

2 Type `eyescope` at the MATLAB command line to start the `EyeScope` tool.

3 In the `EyeScope` window, select **File > Import Eye Diagram Object**.

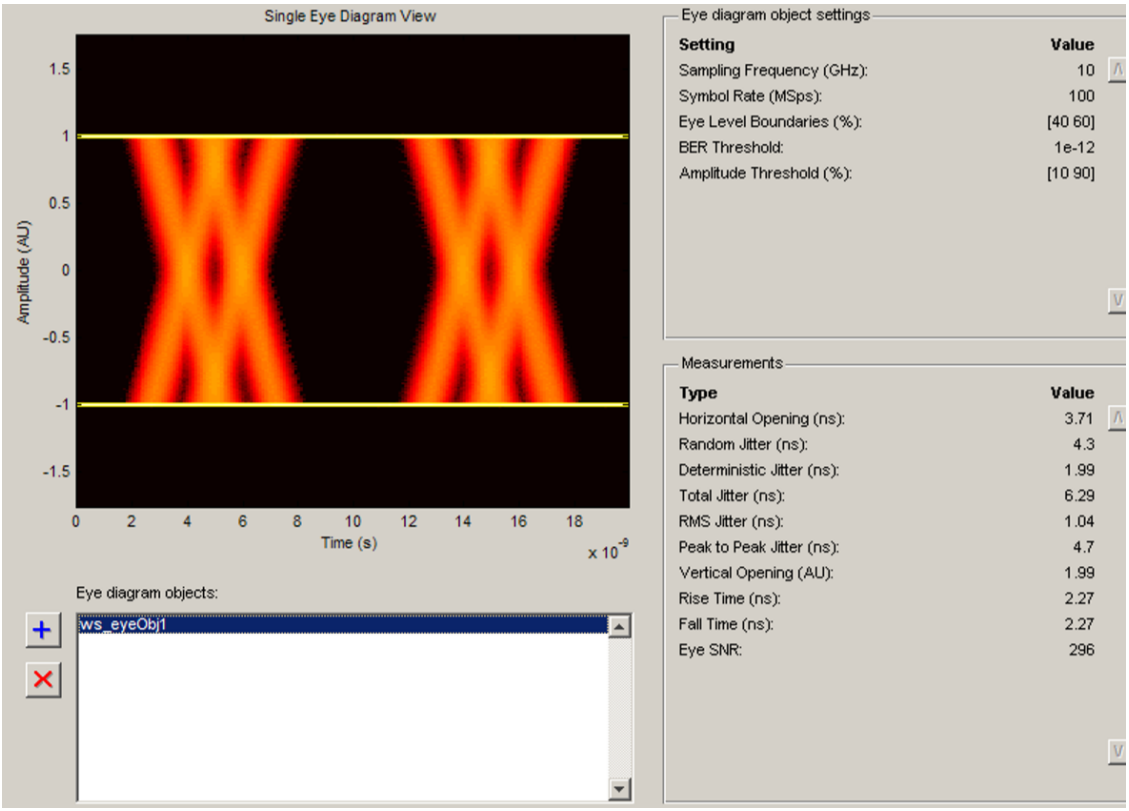
The **Import eye diagram object** dialog box opens.



In this window, the **Workspace contents** panel displays all eye diagram objects available in the source location.

4 Select **eyeObj1** and click **Import**. `EyeScope` imports the object, displaying an image in the object plot and listing the file name in the **Eye diagram objects** list.

Note: Object names associated with eye diagram objects that you import from the work space begin with the prefix *ws*.



Review the image and note the default **Eye diagram object settings** and **Measurements** selections.

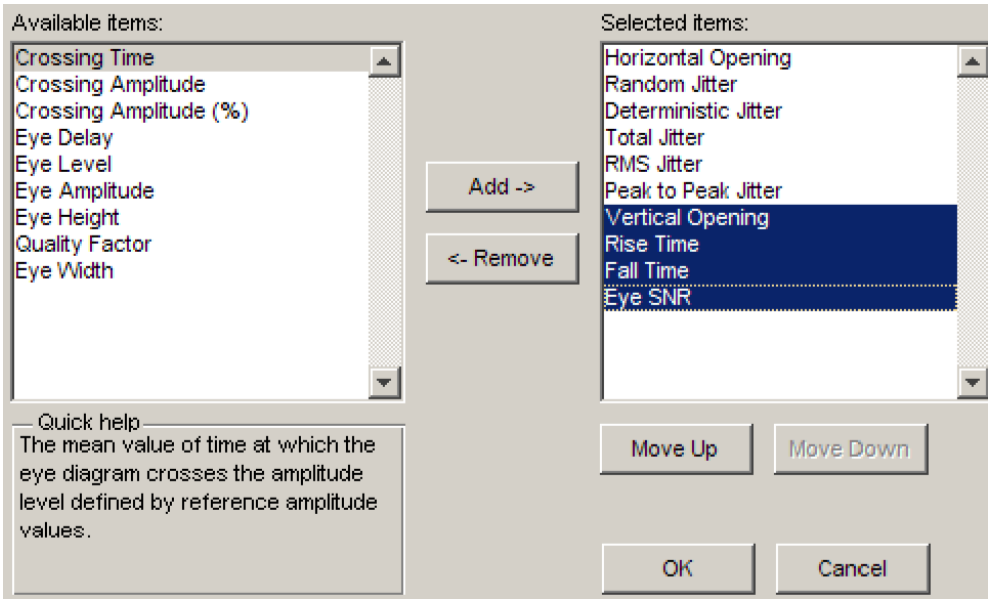
5 In the `EyeScope` window, click the **Import** button.

6 From the Import eye diagram object window, click to select `eyeObj5` then click the **Import** button.

- The `EyeScope` window changes, displaying a new plot and adding `ws_eyeObj5` to the **Eye diagram objects** list. `EyeScope` displays the same settings and measurements for both eye diagram objects.
- You can switch between the eye diagram plots `EyeScope` displays by clicking on an object name in the Eye diagram object list.
- Next, click **ws_eyeObj1** and note the `EyeScope` plot and measurement values changes.

7 To change or remove measurements from the `EyeScope` display:

- Select **Options > Measurements View**. The **Configure measurement view** shuttle control opens.



- Hold down the <Ctrl> key and click to select Vertical Opening, Rise Time, Fall Time, Eye SNR. Then click **Remove**.

8 From the left side of the shuttle control, select **Crossing Time** and **Crossing Amplitude** and then click **Add**. To display EyeScope with these new settings, click **OK**. EyeScope's **Measurement** region displays Crossing Time and Crossing Amplitude at the bottom of the Measurements section.

9 Change the list order so that **Crossing Time** and **Crossing Amplitude** appear at the top of the list.

- Select **Options > Measurements View**.
- When the **Configure measurement view** shuttle control opens, hold down the <Ctrl> key and click to select **Crossing Time** and **Crossing Amplitude**.
- Click the **Move Up** button until these selections appear at the top of the list. Then, click **OK**

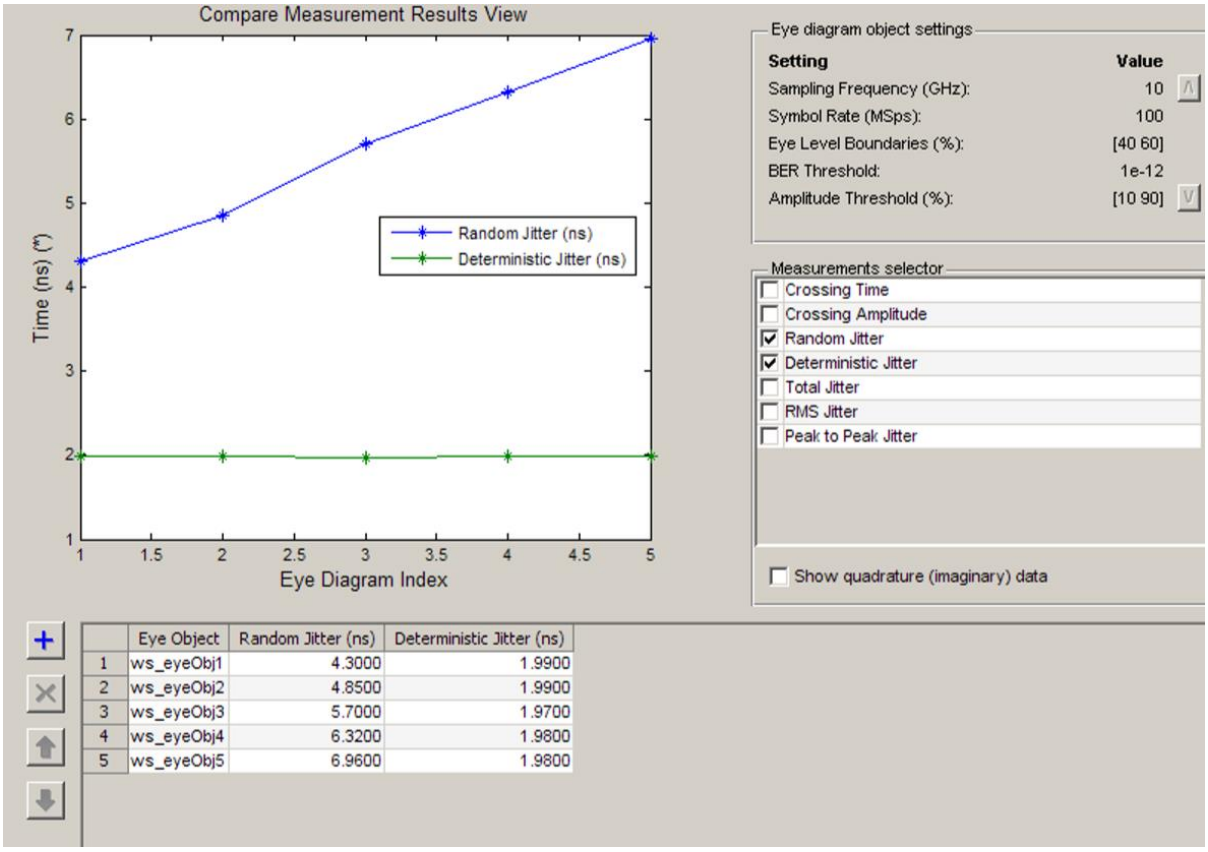
10 Select **File > Save session as** and then type a file name in the pop-up window.

11 Import **ws_eyeObj2, ws_eyeObj3**, and **ws_eyeObj4**. EyeScope now contains eye diagram objects 1, 5, 2, 3, and 4 in the list.

12 Select **ws_eyeObj5** and click the **delete** button.

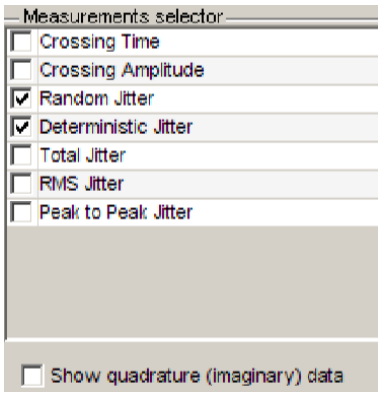
13 Click **File > Import Eye Diagram Object** and select **ws_eyeObj5**.

14 To compare measurement results for multiple eye diagram objects, click **View > Compare Measurement Results View**.



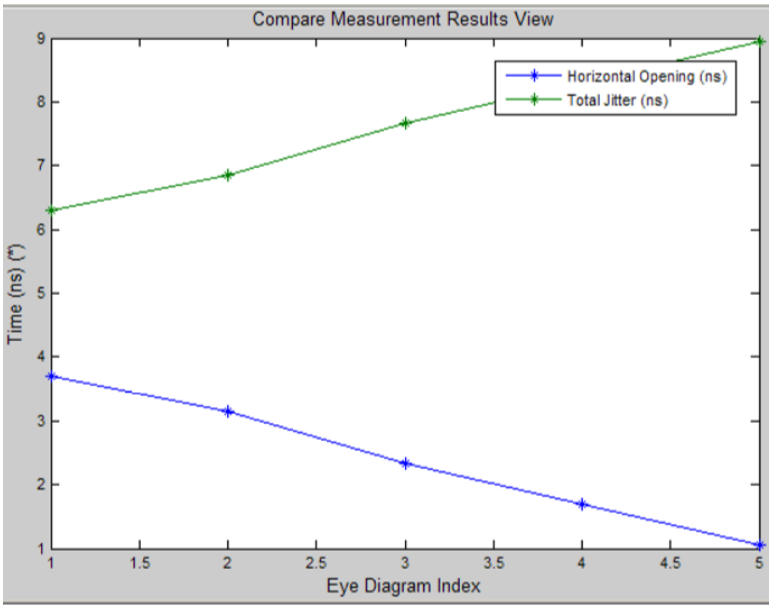
In the dataset, random jitter increases from experiment 1 to experiment 5, as you can see in both the table and plot figure.

15 To include any data from the Measurements selection you chose earlier in this procedure, go to the **Measurement selector** and select, the **Total Jitter** check box. The object plot updates to display the additional measurements.



You can also remove measurements from the plot display. In the **Measurements selector**, select the check boxes for **Random Jitter** and **Deterministic Jitter**. The object plot updates, removing these two measurements.

16 To print the plot display, select **File > Print to Figure**. From **Figure** window, click the print button.



Milestone: Demonstrate your code and the output to a demonstrator.

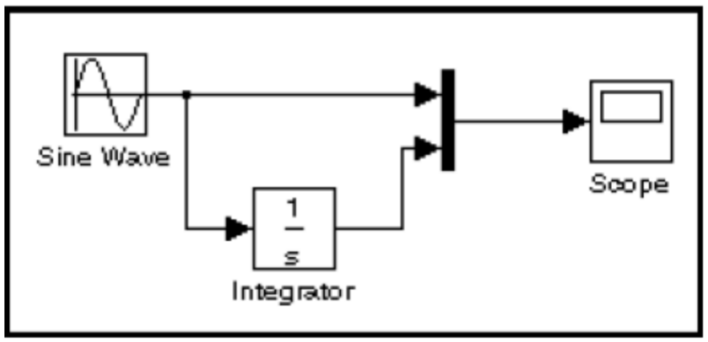
Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1.
2.

Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.

Code: Please write your code here. Do not forget to include comments.
Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

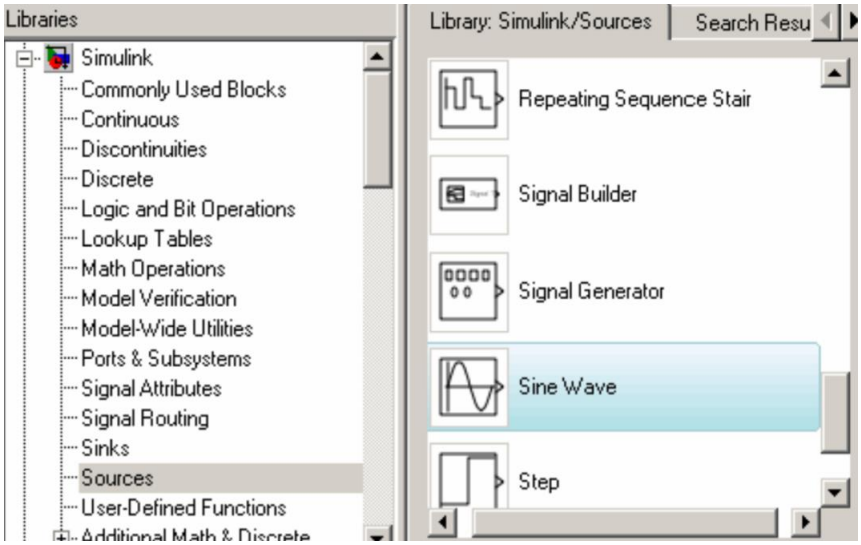
EXERCISE 7: Simulating a Simple Model

Here we learn and describe how to construct a simple model using Simulink software and how to simulate that model. The basic techniques you utilise to construct and simulate this simple model are the same as those for more complex models. The model described in this exercise integrates a sine wave and displays the result along with the original wave. When completed, the block diagram of the model should look as



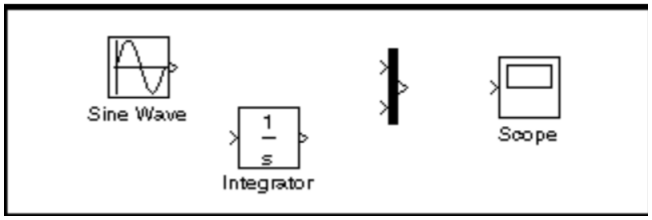
Follow these steps to construct the model:

- Click the Simulink icon in the MATLAB toolbar to enter Simulink or click the MATLAB **Start** button, then select **Simulink** > **Library** Browser.
- The Library Browser appears. It displays a tree-structured view of the Simulink block libraries installed on your system. You build models by copying blocks from the Library Browser into a model window.
- Select **File** > **New** > **Model** in the Simulink Library Browser to construct a new model. An empty model window must appear.
- To construct a model, you first copy blocks from the Simulink Library Browser to the model window. To create the simple model in this exercise, you need four blocks:
    - **Sine Wave** – To generate an input signal for the model
    - **Integrator** – To process the input signal
    - **Scope** – To visualise the signals in the model
    - **Mux** – To multiplex the input signal and processed signal into a single scope
- To add blocks to your model:
    - Select the **Sources** library in the Simulink Library Browser. The Simulink Library Browser displays the Sources library.

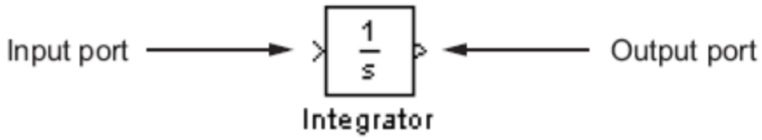


- Select the **Sine Wave** block in the Simulink Library Browser, then drag it to the model window. A copy of the Sine Wave block appears in the model window.

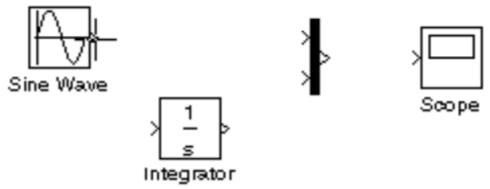
  - Select the **Sinks** library in the Simulink Library Browser.
  - Select the **Scope** block from the Sinks library, then drag it to the model window. A Scope block appears in the model window.
  - Select the **Continuous** library in the Simulink Library Browser.
  - Select the **Integrator** block from the Continuous library, then drag it to the model window. An Integrator block appears in the model window.
  - Select the **Signal Routing** library in the Simulink Library Browser.
  - Select the **Mux** block from the Sinks library, then drag it to the model window. A Mux block appears in the model window.
- Before you connect the blocks in your model, you should arrange them logically to make the signal connections as straightforward as possible.



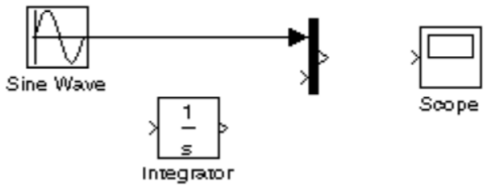
- After adding blocks to the model window, you must connect them to represent the signal connections within the model. Notice that each block has angle brackets on one or both sides. These angle brackets represent input and output ports:
    - The > symbol pointing into a block is an *input port*.
    - The > symbol pointing out of a block is an *output port*.



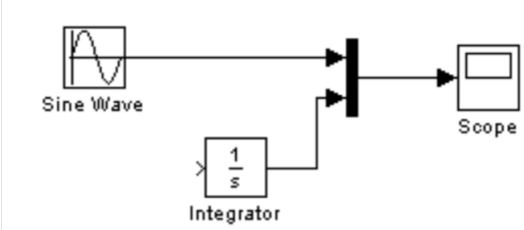
- To draw a line between two blocks:
    - Position the mouse pointer over the output port on the right side of the Sine Wave block. Note that the pointer changes to a cross hairs (+) shape while over the port.



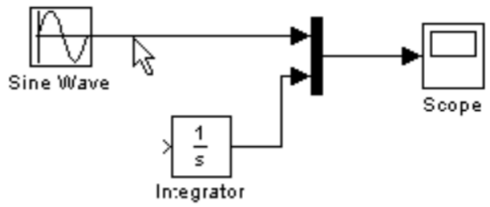
    - Drag a line from the output port to the top input port of the Mux block. Note that the line is dashed while you hold the mouse button down, and that the pointer changes to a double-lined cross hair as it approaches the input port of the Mux block. Release the mouse button. The software connects the blocks with an arrow that indicates the direction of signal flow.



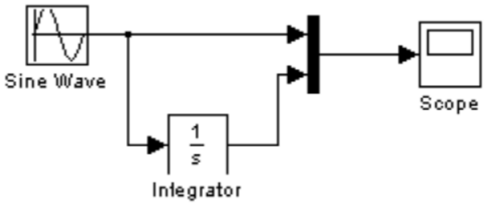
- The model should look like the following figure after making other connections:



- The model is almost complete. To finish the model, you must connect the Sine Wave block to the Integrator block. This final connection is somewhat different from the other three. Because the output port of the Sine Wave block already has a connection, you must connect this existing line to the input port of the Integrator block. The new line, called a *branch line*, carries the same signal that passes from the Sine Wave block to the Mux block. To weld a connection to an existing line:
    - Position the mouse pointer *on the line* between the Sine Wave and the Mux block.

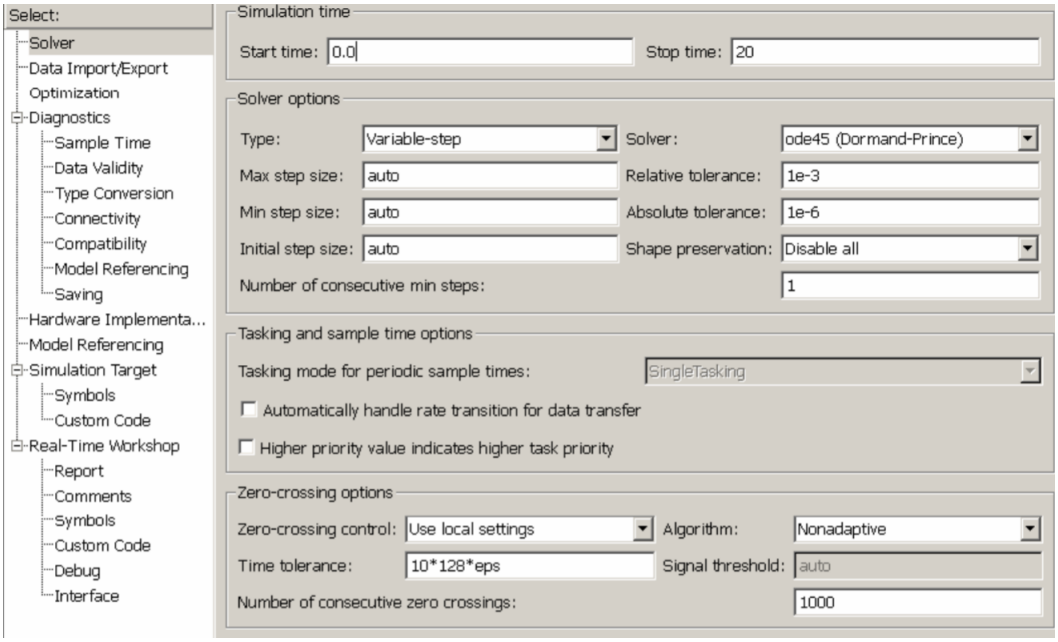


- Press and hold the **Ctrl** key, then drag a line to the Integrator block's input port. The software draws a line between the starting point and the input port of the Integrator block as shown below.



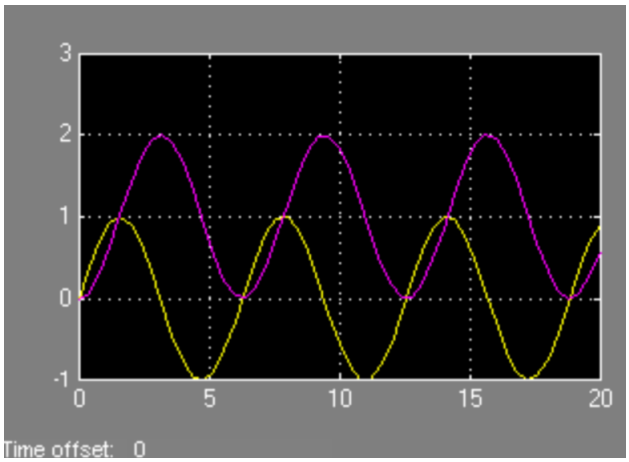
- After you complete the model, you should save it for future use. To save the model:
    - Select **File** > **Save** in the model window.
    - Specify the location in which you want to save the model.
    - Enter simple_model in the **File name** field.
    - Click **Save**.

  The software saves the model with the file name simple_model.mdl.
- You can now simulate the system and visualise the results as follows.
- Set simulation options such as the start and stop time, and the type of solver that Simulink software uses to solve the model at each time step. You specify these options using the Configuration Parameters dialog box.
- To specify simulation options for the sample model:
    - Select **Simulation** > **Configuration Parameters** in the model window. The software displays the Configuration Parameters dialog box.



- Enter 20 in the **Stop time** field.
- Click **OK**.

- Now you are ready to simulate your sample model and observe the simulation results. To run the simulation:
  - Select **Simulation** > **Start** in the model window. The software runs the model, stopping when it reaches the stop time specified in the Configuration Parameters dialog box.
  - On computers running the Microsoft® Windows® operating system, you can also click the **Start simulation** button and **Stop simulation** button in the model window toolbar to start and stop a simulation.
  - Double-click the **Scope** block in the model window. The Scope window displays the simulation results.



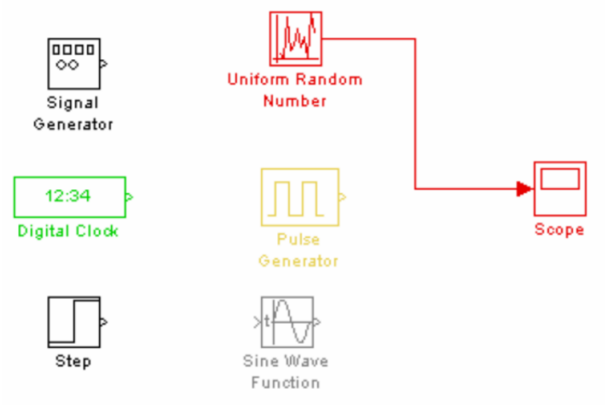
Milestone: Demonstrate your code and the output to a demonstrator.

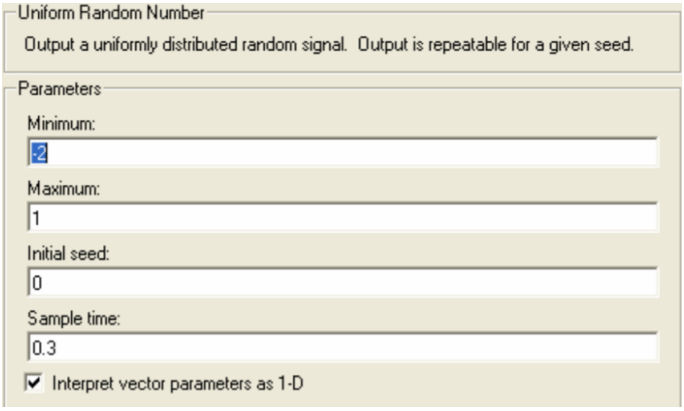
Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1. **2.**
Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.

Diagram: Please show your system diagram here. Do not forget to include comments.
Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

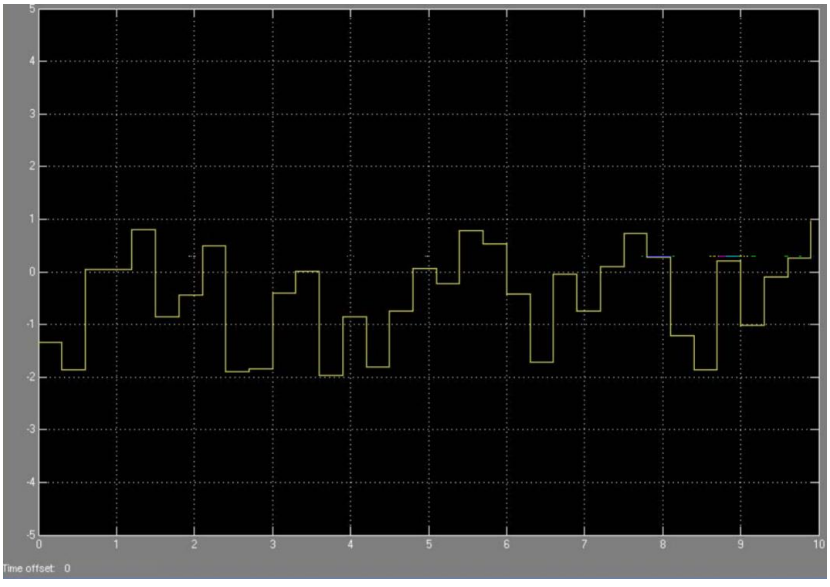
EXERCISE 8: Understanding Simulink blocks

- Study following Simulink blocks using scope: signal generator, digital clock, step, uniform random number, pulse generator, and sine wave function. The use of uniform random number is outlined below:





  - After these arrangements, start the simulation and observe the result.
  - The figure below indicates a sample simulation result.



- Do similar steps for other sources, that is, signal generator and scope, digital clock and scope, step and scope, pulse generator and scope, and sine wave function and scope. For the values of the 'Block Parameters' of the sources, choose your own.

- Follow the steps below to design a multiplier with a square-wave input.
  - Start a Simulink session.
  - Copy the pulse generator block by dragging it to your model. Using the left mouse button, double-click on the pulse generator block. A new window appears that displays all the properties of your selected block. Specify the model fields as follows.
    - **Period**: 1
    - **Pulse width**: 50%
    - **Amplitude**: 0.7
  - Next, multiply the output of the signal generator by a gain. You can use a gain block from the math library by placing it in the design window. Double-click on the gain block to open the property window and set the gain to 0.9.
  - Connect the blocks.
  - To analyse the output, drag the Scope block from Sinks of the Simulink block library. Double click the Scope block to open the Scope window. Specify the Scope axes as follows.
    - **Ymax**: 1.5
    - **Xmax**: −1.5
    - **Time range**: 10
  - Select the Simulation menu and then parameters to open the Simulation control window. Set the simulation parameters as follows.
    - **Start time**: 0.0
    - **Stop time**: 10.0
    - **Solver options Type**: Fixed step discrete (no continuous states)
    - **Fixed step size**: 0.01
  - Select the Simulation menu and then run the simulation by clicking Start.
  - Save your model file as a Matlab mdl-file.

Milestone: Demonstrate your code and the output to a demonstrator.

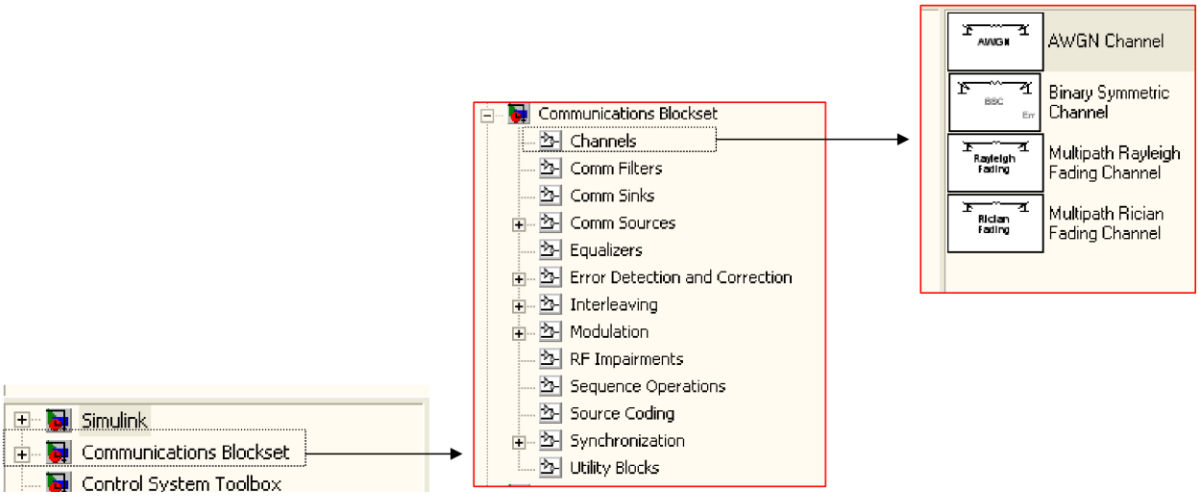
Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1. **2.**

Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.
Diagram: Please show your system diagram here. Do not forget to include comments.

Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

NOTE on Using Communication Toolbox

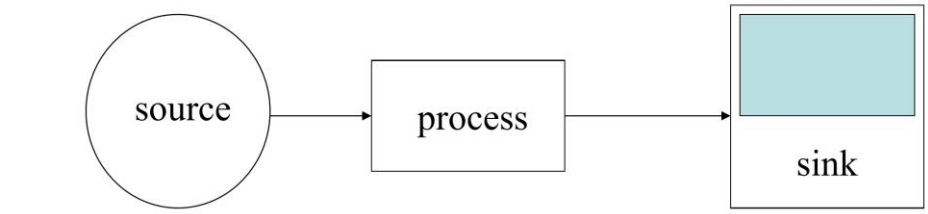
Open the Communication Toolbox located in Blocksets&Toolboxes!Comm Tbx Library. This shows all the elements available to a communication system. Open several of the blocks in order to see what functions are available; for example, look at Modulation and Analog Modem. Within Analog Modem, there are several modulation techniques. Try experimenting with some of the blocks to familiarise yourself with what is available. If you need further help, there is an on0line tutorial in PDF form for the Communications Toolbox that provides descriptions of the available functions and blocks.



EXERCISE 9: Analysing data via Simulation

Simulink has three classes of blocksets:

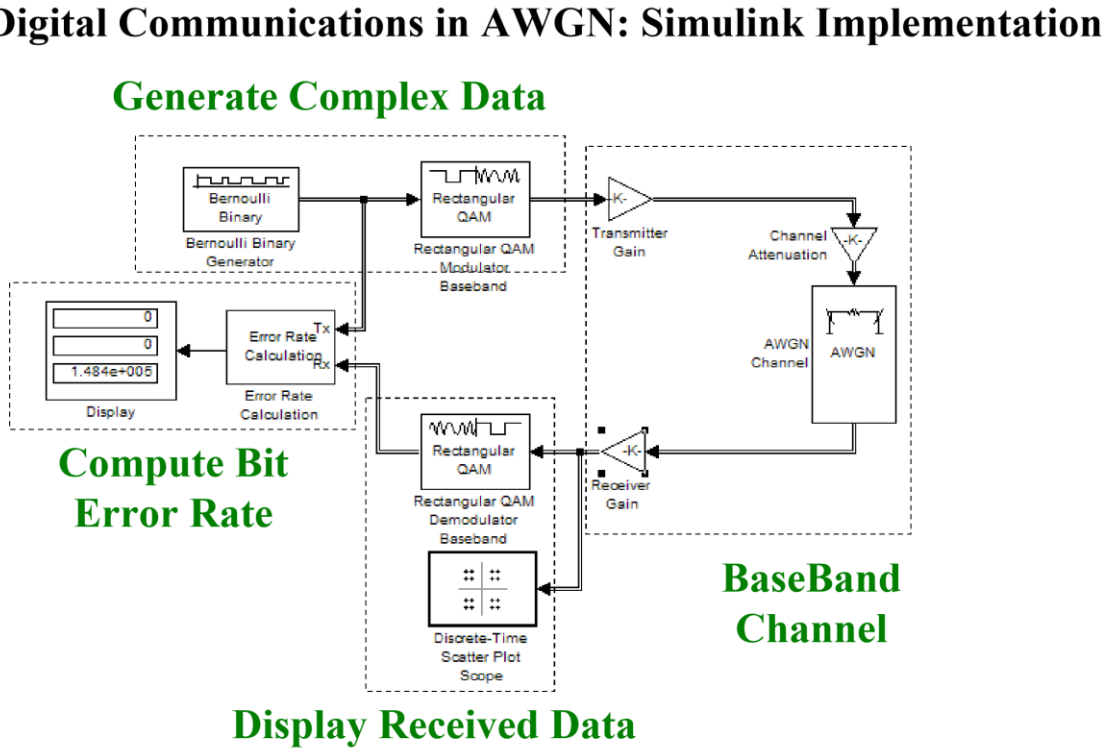
- sources (outputs only)
- processes (inputs/outputs)
- sinks (inputs only)



- generate data
- read data
- import files …

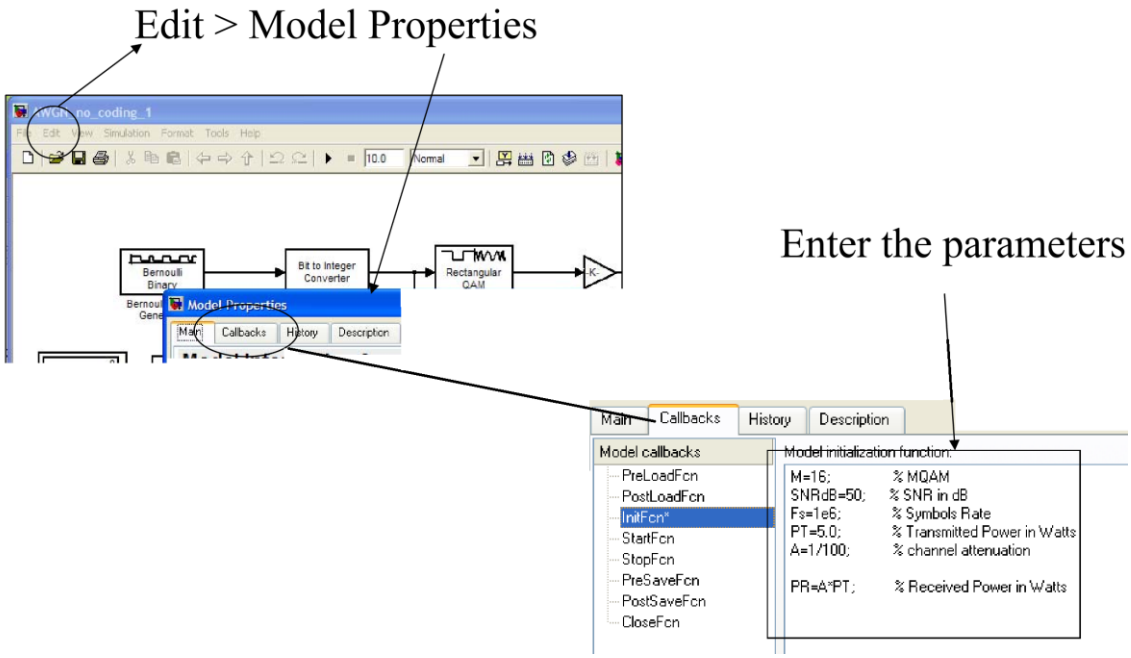
- display results
- plots, scopes…
- send data to devices

1. Design the following system.

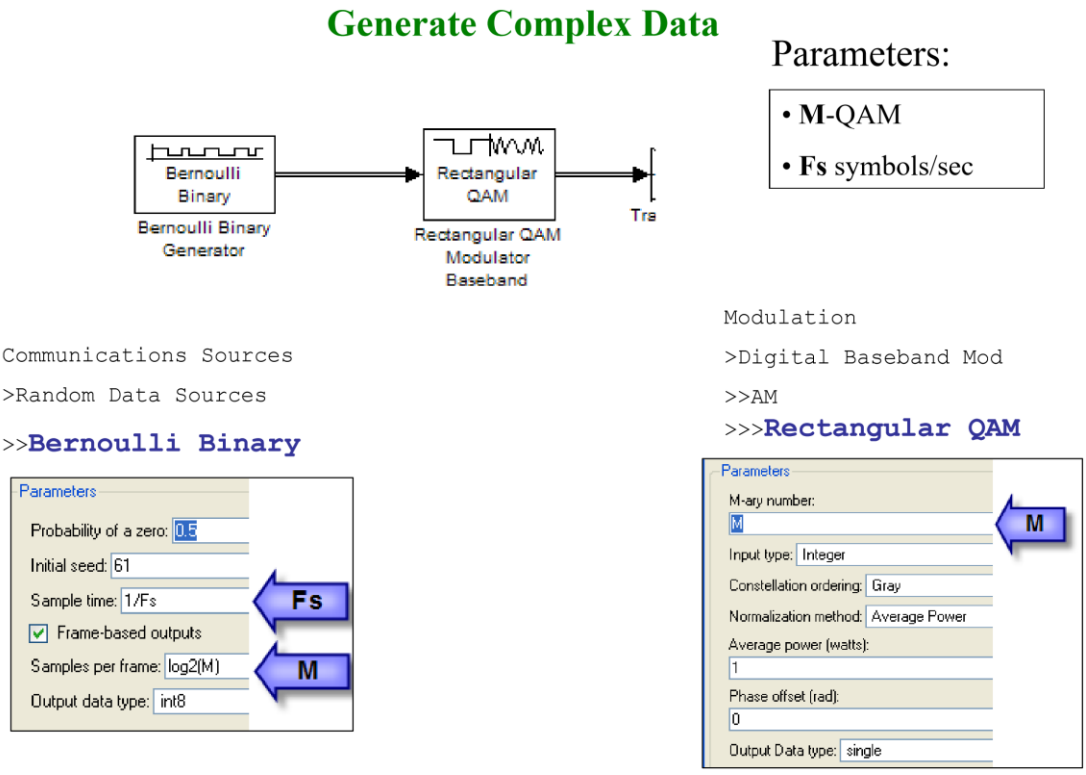


Simulink Model: **AWGN_no_coding.mdl**

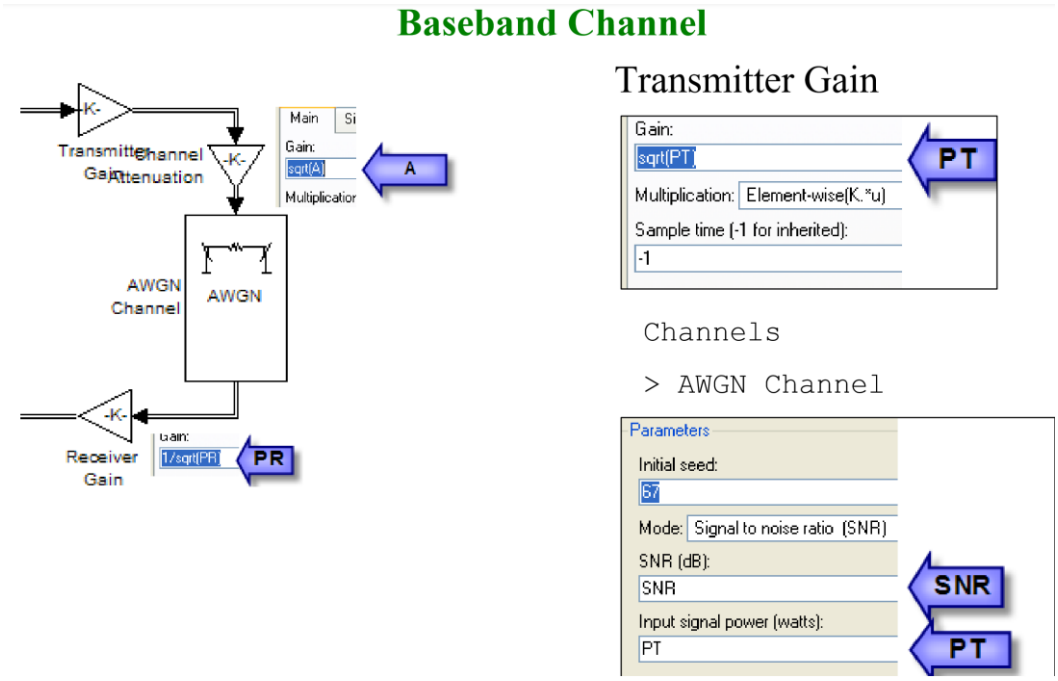
2. Utilise the following channel characteristics.



3. Use the following parameter values to generate complex data.

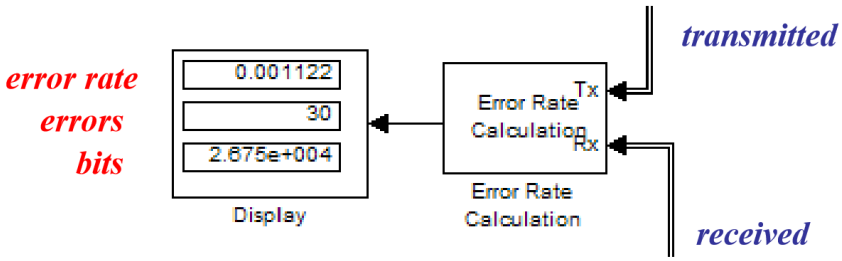


4. Now, include the following parameter values for an appropriate transmitter gain and implementing an AWGN channel.



5. Analyse the BER of the designed system.

Analysis of Data by Computer Simulation



Blocks:

- Comms Sinks > Error Rates Calculation
- Simulink > Sinks > Display

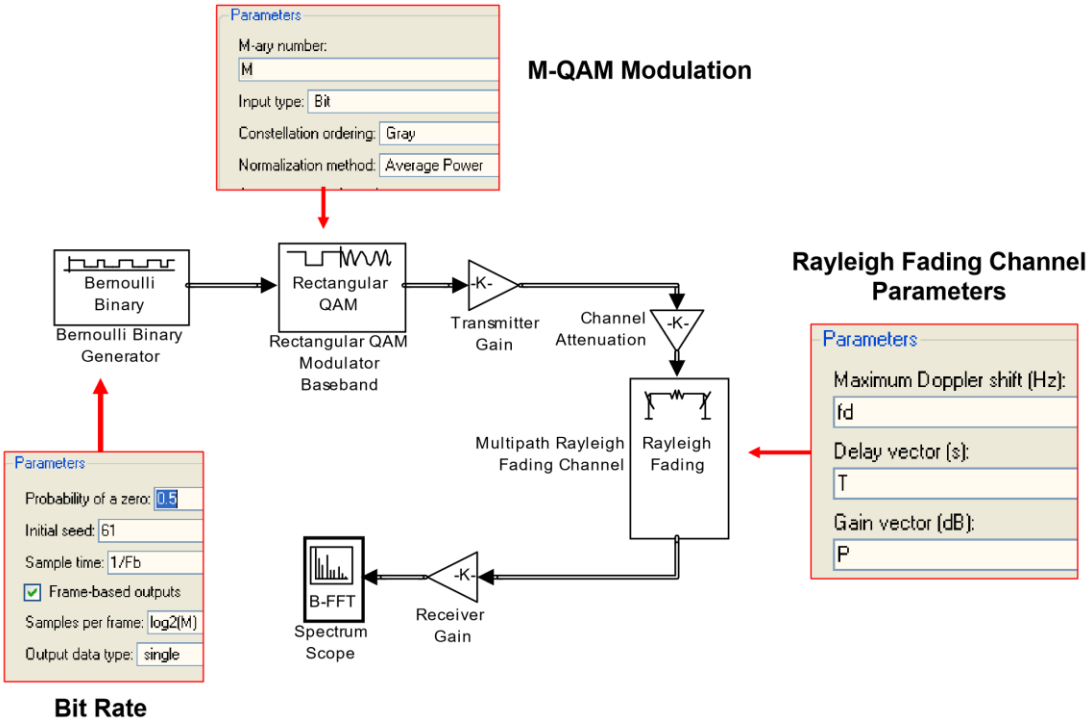
Milestone: Demonstrate your code and the output to a demonstrator.

Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1. **2.**
Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.

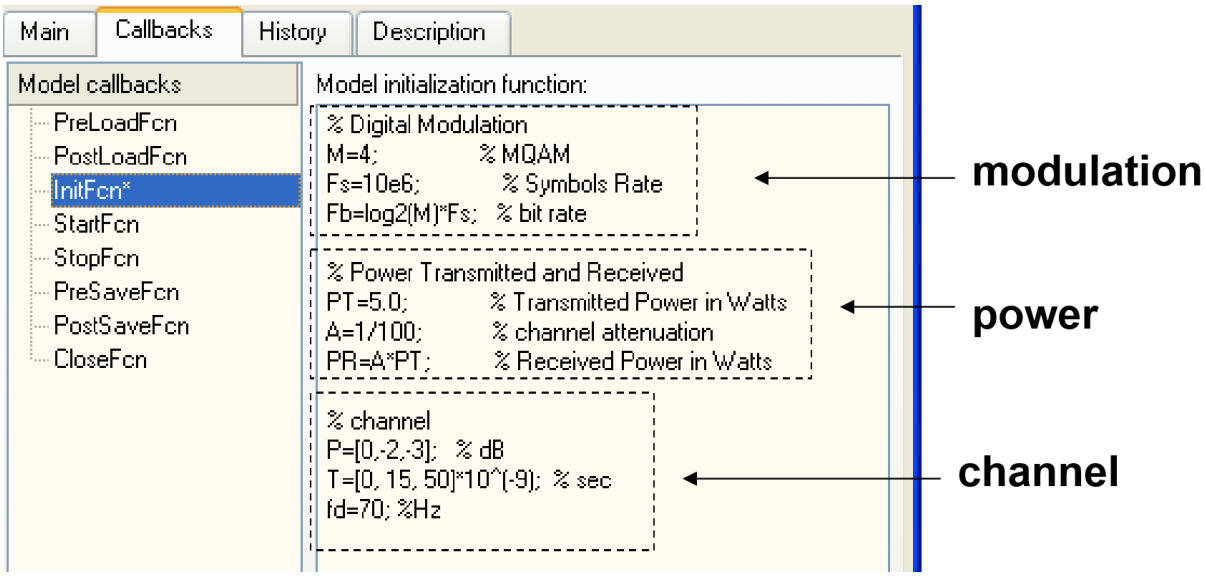
Diagram: Please show your system diagram here. Do not forget to include comments.
Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

EXERCISE 10: Realising Power Spectrum

1. Design the following System.

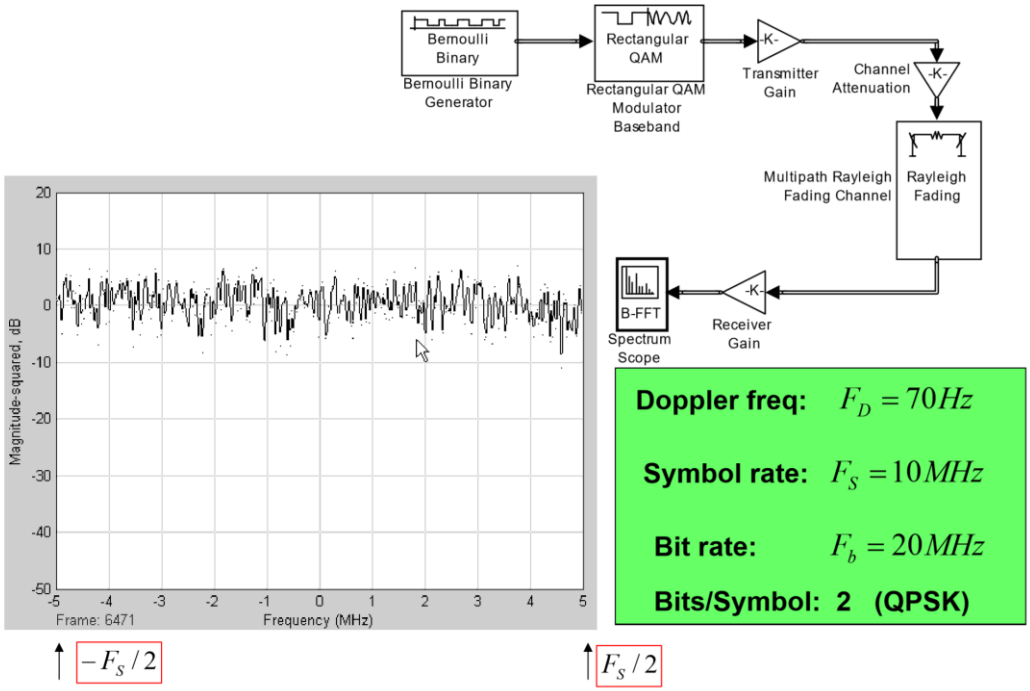


2. Use the following channel characterisation.



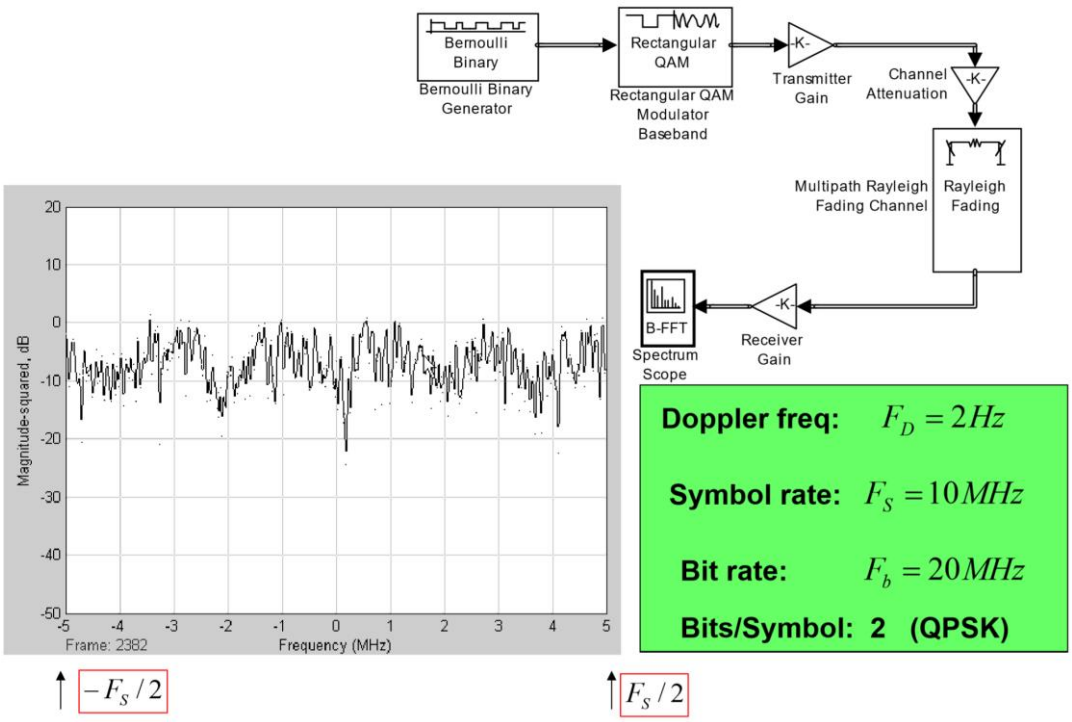
3. Realise the received power spectrum.

Typical Received Power Spectrum



4. Now, change the parameter configurations as shown below and observe the difference in the received power spectrum.

Similar Example, different parameters



Milestone: Demonstrate your code and the output to a demonstrator.

Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1. **2.**
Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.

Diagram: Please show your system diagram here. Do not forget to include comments.
Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

EXERCISE 11: Test Yourself 1

Create an array of cells, explain each function, and create your own cell placement.

Milestone: Demonstrate your code and the output to a demonstrator.

Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1. **2.**
Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.

Code: Please write your code here. Do not forget to include comments.
Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

EXERCISE 12: Test Yourself 2

Model and simulate a multiple-input multiple-output (MIMO) channel with 'Nt' transmit antennas and 'Nr' receive antennas and analyse it to plot the system capacity. Demonstrate multiple combinations of the 'Nt' and 'Nr'.

Milestone: Demonstrate your code and the output to a demonstrator.

Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1. **2.**
Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.

Code: Please write your code here. Do not forget to include comments.
Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

EXERCISE 13: Test Yourself 3

Model and simulate a Rayleigh fading channel and a Rician fading channel and plot their density functions.

Milestone: Demonstrate your code and the output to a demonstrator.

Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1. **2.**
Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.

Code: Please write your code here. Do not forget to include comments.
Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

EXERCISE 14: Test Yourself 4

Model and simulate a free-space propagation loss. *(Hint: The simplest wave propagation case is that of direct wave propagation in free space.)*

Milestone: Demonstrate your code and the output to a demonstrator.

Objectives: Please write at least two points mentioning the purpose of this lab exercise.
1. **2.**
Procedure: Please briefly describe the exercise procedure. Also provide schematic diagram, output, figure and/or table, if any.

Code: Please write your code here. Do not forget to include comments.
Learning Outcomes: Please mention at least two things that you have learnt while performing this exercise.
1. **2.**

SOLUTIONS

11.

```
x_hexagon=[-1 -0.5 0.5 1 0.5 -0.5 -1];
y_hexagon=[0 -sqrt(3)/2 -sqrt(3)/2 0 sqrt(3)/2 sqrt(3)/2 0];

N=10;
M=10;

figure(1)
hold on
for nn=0:N
    for mm=0:M
        plot(x_hexagon+3*nn,y_hexagon+sqrt(3)*mm)
    end
end
for nn=0:N-1
    for mm=0:M-1
        plot(x_hexagon+1.5+3*nn,y_hexagon+sqrt(3)/2+sqrt(3)*mm)
    end
end
hold off
axis equal
```

12.

```
clear all
clc
%Shannon capacity
snr=0;
for i = 1:10
snr = snr +2;
c=(log(1+10^(snr/10)))/log(2);
x(i)=snr;
y(i)=c;
end
figure
plot(x,y,'-','LineWidth',1.5)
hold on

% capacity of MIMO Link with NR=2, NT=2
NR=2;
rand('state',456321)
snr=0;
for i=1:10;
snr=snr+2;
for j=1:10000;
c(j)=(NR*log(1+(10^(snr/10))*abs(normrnd(0,1)))/log(2));
end
yy(i)=mean(c);
xx(i)=snr;
end
plot(xx,yy,':','LineWidth',1.5)

% capacity of MIMO Link with NR=3, NT=3
NR=3;
rand('state',456321)
snr=0;
for i=1:10;
snr=snr+2;
for j=1:10000;
c(j)=(NR*log(1+(10^(snr/10))*abs(normrnd(0,1)))/log(2));
end
yy(i)=mean(c);
xx(i)=snr;
end
plot(xx,yy,'-.','LineWidth',1.5)

% capacity of MIMO Link with NR=4, NT=4
NR=4;
rand('state',456321)
snr=0;
for i=1:10;
snr=snr+2;
for j=1:10000;
c(j)=(NR*log(1+(10^(snr/10))*abs(normrnd(0,1)))/log(2));
end
yy(i)=mean(c);
xx(i)=snr;
end
plot(xx,yy,'--','LineWidth',1.5)

xlabel('SNR(dB)')
ylabel('Capacity (bit/s/Hz)')
grid on
```

```
legend('Shannon Capacity','MIMO, NT=NR=2','MIMO, NT=NR=3','MIMO,
NT=NR=4',2)
title('MIMO Capacity')
print -deps -tiff -r300 capmimo
```

13.

Rayleigh fading pdf plot MATLAB code:

```
sig=1; %RMS value of received voltage signal
r = 0:0.01:10; %the range of value of 'r'
pdf=(r/sig.^2) .* exp(-r.^2/2*(sig.^2)); % the Rayleigh PDF figure;
plot(r,pdf,'b') %Plotting the Rayleigh PDF
title('Rayleigh PDF Plot');
grid on;
xlabel('r');
ylabel('pdf');
```

Rician fading pdf plot MATLAB code:

```
r=0:0.01:10;
v=5;
I0=0.55;
sig = 2; %the range of value of 'r'
pdf=(r./sig.^2) .* exp(-r.^2+v.^2./2*(sig.^2)).*I0.*(r.*v./(sig.^2)); % the
rician PDF
figure;
plot(r,pdf,'b') %Plotting the rician PDF
title('Rician PDF Plot');
xlabel('r');
ylabel('pdf');
grid on;
```

14.

```
% Free Space Propagation Loss
clc;
close all;
clear all;

f = [1000 2000 3000];
c = 300;
d = 1:1:10000;
Lp1 =20*log10((4*pi*d*f(1))/c);
Lp2 =20*log10((4*pi*d*f(2))/c);
Lp3 =20*log10((4*pi*d*f(3))/c);

figure(1);
plot(d,Lp1,'b',d,Lp2,'r',d,Lp3,'g');
xlabel('x--> D (distance in Meter)');
ylabel('y--> Lp (path loss in dB)');
title('Free space model');
hold on;
legend('f=1000MHz' , 'f=2000MHz' , 'f=3000 MHz');
grid on;
```