Wireless Multi Input Multi Output (MIMO) channel simulation package

(Version 1.2)

Ali Tawfiq

Supervisors: Professor Kostas Plataniotis, Amir Aghaei

Multimedia laboratory, The Edward S. Rogers Sr. Dept of Electrical & Computer Engineering, University of Toronto

Contents

1. Overview of the Package

The package consists of seven modules each of which incorporates an important part of any simple communication system. The output of each module is cumulative and is used as an input to the next module in the system. The diagram below briefly shows how these modules are arranged:

Fig 1: The process in which the modules are used

The 'main' file is used to call each module accordingly. If all seven modules (the six above plus 'Mapping.m') are to be used in the program then they all must be used according to the process above, otherwise if the user wishes to exclude any of the modules and replace it with one of their own, it must be added to the main file accordingly.

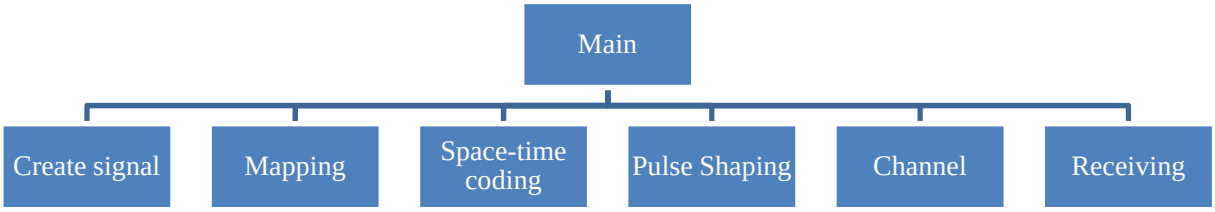


Fig 2: How the 'Main' file is used to call the different modules

Conceptually, this package implements the following simple communication system that allows the user to implement and simulate MIMO channels as well (The Transmitter and Receiver blocks are explained in more detail further in the manual):

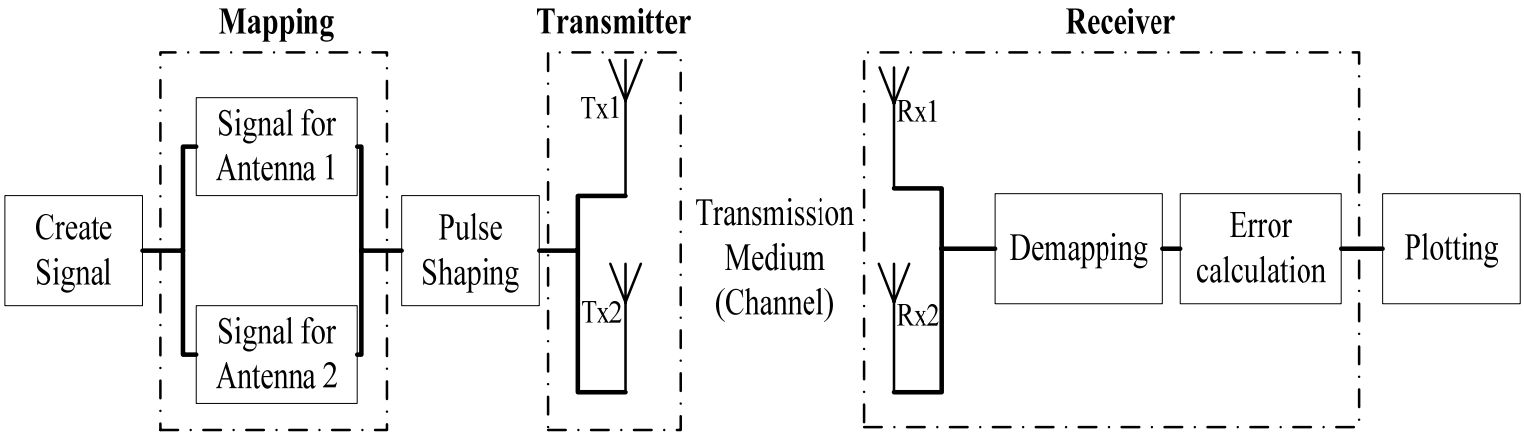


Fig 3: Simple communication system

2. Objects

In this package the different modules are implemented such that the inputs and outputs are organized in an 'object' oriented style. Each module will be fed with one or two objects, which includes certain parameters that need to be used in each specific module. The output data of each module is also generated as an object. This technique makes it easier for the user to locate and use any of the parameters in each different module. Below is a breakdown of the different objects and where and how each will be used.

2.1 Types of objects

2.1.1 Signal Parameters object (Fout_sig)

This object is first created in the "CreateSig.m" module, in which the user will get a chance to set the parameters used in creating the input signal, the pulse shaping filters, etc. Here is the list of the parameters that will be involved in this object:

Fout.sig Object

System Parameters	• **Mod:** Modulating technique • **M**: M-ary, binary (2), quad (4) or 2^n • **K**: Number of bits per symbol calculated from M • **Block**: String for block type (refer to 'Mapping.m') • **BlockL**: Length of blocks (refer to 'Mapping.m') • **Numtx**: Number of transmit antennas • **Numrx**: Number of receive antennas • **SNR**: a vector of SNR values • **Saving**: string equal to 'backup' to save parameters
Signal Parameters	• **L**: Length of signal • **itt:** number of iterations • **Px:** The power of signal (1 for PSK, different for QAMs)
Filter Parameters	• **r**: Roll-off-factor • **T**: Bit period • **dur**: Duration • **FltrType:** Type of filter to be used • **nsamp**: oversampling rate • **Delay**: Delay of the filter
Data Vectors	• **Out**: a zero vector to store final received signal • **InSig:** The input signal in bits • **S:** The split InSig, an array of Num_tx column vectors • **InSym**: The modulated Signal • **fltr**: The generated filter • **BER**: a zero vector to store the computed BER

2.1.2 Transmitted signal object (Fout_tx)

This object is first created in the "Transmit.m" file, in which the user will be creating the transmitted signals according to the specific transmit technique. This object will be used later on in the channel module.

Object Name	***Data Vectors***
Fout_tx:	• **tx:** Array of Signal to be transmitted from each antenna as columns o **tx1:** Signal to be transmitted from antenna 1 o **tx2:** Signal to be transmitted from antenna 2 ... o **tx[Numtx]:** Signal to be transmitted from antenna Numtx

2.1.3 Channel coefficients and received signal object (Fout_ch)

This object is first created in the "Channel.m" file, in which the user will be creating the channel coefficients and the received signals (after adding noise and channel coefficients) at the receive antennas.

Object Name	***Data Vectors***
Fout_ch:	• **Ch:** Array with channel coefficients as column vectors o **ch1:** Channel coefficients between Tx1 &Rx1 o **ch2:** Channel coefficients between Tx1 &Rx2 ... o **ch[Numtx * Numrx]:** Channel coefficients between Tx[Numtx] & Rx[Numrx] • **R:** A 3-dimensional array of the signals to be received at the receive antennas. The third dimension represents the number of the received antenna. o **R1:** Array of received signals at Rx1 each column representing different SNR value. o **R2:** Array of received signals at Rx2 each column representing different SNR value. **...** o **R[Numrx]:** Array of received signals at Rx[Numrx] each column representing different SNR value.

2.2 Objects usage

The objects described in section 2.1 will be used as follows:

	Fout_sig	Fout_tx	Fout_ch
CreateSig.m	O	n/a	n/a
Mapping.m	I/O	n/a	n/a
PulseShaping.m	I/O	n/a	n/a
Transmit.m	I	O	n/a
Channel.m	I	I	O
Receive.m	I/O	n/a	I
Plotting.m	I	n/a	n/a

n/a: Not used **I:** Input **O:** Output

3. The Main File

As part of the simulation package, the 'Main' file is the module responsible for calling the corresponding modules accordingly. As implemented in this package, the 'Main.m' file calls out the different modules of the package in the appropriate order.

In order to run a simulation, the user must only perform the following command in matlab:

```
Main
```

The 'Main.m' file will be initiated and will then start performing the simulation by calling out the different modules in organized fashion. Within the main file implemented is a loop that will run for 'itt' times. This iterative loop operates to loop around the following three functions; Transmit, Channel, and Receive. The package is designed to transmit the generated signal in smaller pieces, due to memory constraints. By doing so the simulation allows for the least memory usage when running and yields faster output time.

What needs to be changed?

In order to test/ simulate different MIMO transmission schemes the user needs to **only** edit 'Transmit.m' and 'Receiver.m' accordingly to reflect the new scheme. These two modules currently correspond to a 2x2 MIMO system. If the user wishes to change other system parameters such as Modulation scheme, pulse shaping, etc, the user has complete control of these features in the 'CreateSig.m' module.

Compute Time

The time required for a simulation to complete is depending primarily on the following parameters: **Length, itt, T, and SNR range.** While these parameters generally affect compute time, machine processor and available memory are the two main constraints.

Results

Once the simulation is completed, the main file then saves an output file that includes the simulation results. The result will be saved in a matlab file uniquely named for each simulation as follows [Output "Modulation Type"-"Filter Type" ("Length", "Iteration").mat]. Thus if the package is simulating a BPSK with a square root raised cosine filter, length of 1000 and 100 iterations then the output file will be named as follows:

```
Output BPSK-sqrtrcos (1000, 100).mat
```

This will allow the user to review the results once the simulation is complete. The '.mat' file will be saved in the subdirectory 'Output' and will contain the following parameters:

- BER: the bit error rate for all iterations
- InSig: The randomly generated binary signal
- InSym: The modulated input signal
- Output: The resulting received signal once the transmission is complete.

4. Modules

4.1 Create Signal

This module is used to set up the different variables needed for the communication system. The user must change the value of each variable as required in the specific simulation needed to be run.

Input	Output	Hierarchy
• **n/a**	• **Fout_sig**	**CreateSig.m** **Setting Variables** **Generate Signal** **partition the signal**

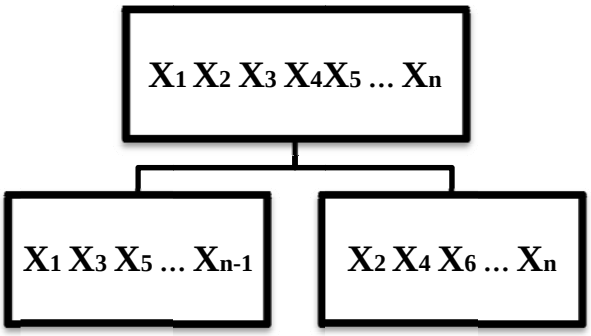
Hierarchy explained:

- **Setting variables**
  The different global variables needed within the communication system are set here. The user must change these values depending on the type of simulation being conducted. Some important variables such as the length of the signal, number of iterations, number of transmit and receive antennas, bit period, symbol frequency, etc must be set in order to have a functioning communication system. The user might also wish to add variables other than what is already available.

- **Generate signal**
  The input signal is randomly generated here using a random number generator. It generates a stream of bits (0s and 1s). The size of the signal will be equal to $L \times itt \times k$. The signal generated here will be the parameter 'InSig' and can be used later on in the program. The User can substitute a signal of their own if they desire, the signal must be a *column vector* of bits.

- **Partition the signal**

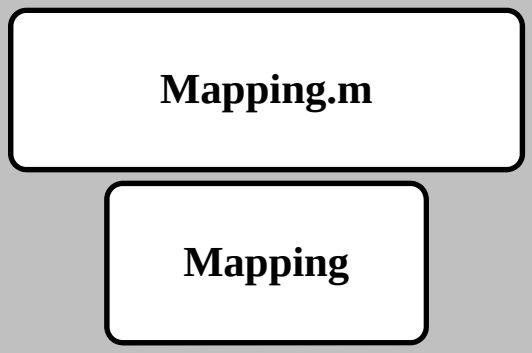
  Once the signal is generated it must be partitioned according to the transmission technique being used. For example if the simulation implemented is for a 2x2 Alamouti transmission scheme, then the input signal will be partitioned into two signals as follows:



4.2 Mapping

This module provides various selection of mapping techniques. For this specific MIMO project only a few modulation techniques are operational, for the complete list of implemented techniques refer to table on page 12. This module generates the modulated signal and the power of the signal and saves them to the object 'Fout_Sig'.

Note: The demodulation must be implemented by the user in the 'receiver.m' file.

Input	Output	Hierarchy
• **Fout_sig**	• **Fout_sig**	

Hierarchy explained:

- **Mapping**
  This function implements a case hierarchy; depending on the input of the modulating technique 'Fout_sig.Mod' the function will modulate the input signal accordingly. The Function will also return the power of the signal, to be later used in the noise generation.

Notes:

This function has one major loop which allows for the modulation of different partitioned signals in the 'InSig' parameter. Considerable amount of time and memory will be saved by modulating the two signals without the need to call the function twice.

Modulation Techniques

Apart from the fully implemented modulation techniques, this file contains many other techniques that are not fully implemented. Each of the cases calls other functions found in the folder 'Modulation'; the user can understand how each modulation technique is implemented by inspecting those functions. Below is a table that lists the fully implemented modulation techniques that do not require *training blocks*: [1]

Modulating Technique
BPSK
QPSK
8PSK
16PSK 16QAM 64QAM

The table below summarizes the implemented modulating techniques that require training blocks and their corresponding block string input (to be set in 'CreateSig.m'): [2]

Modulating Type	'Block' Input	
DBPSK	"0 Symbol"	"Every L Symbols"
DQPSK	"0 Symbol"	"Every L Symbols"
8DPSK	"0 Symbol"	"Every L Symbols"
16DPSK	"0 Symbol"	"Every L Symbols"
pi2DBPSK	"0 Symbol"	"Every L Symbols"
pi4DQPSK	"0 Symbol"	"Every L Symbols"
Delayed_DBPSK	"0 Symbol"	"Every L Symbols"
Delayed_DQPSK	"0 Symbol"	"Every L Symbols"
AsymBPSK	-	-
AsymQPSK	-	-
AsymDBPSK	"0 Symbol"	"Every L Symbols"
AsymDQPSK	"0 Symbol"	"Every L Symbols"
AsymBPSK_BPSK	-	"Every L Symbols"
AsymQPSK_QPSK	-	"Every L Symbols"
AsymDBPSK_DBPSK	-	"Every L Symbols"
AsymDQPSK_DQPSK	-	"Every L Symbols"

Note: If using constellations other than Binary, the input signal **must** be sized accordingly to generate corresponding symbol size.

Quadrature	Input signal must be twice the desired size
8-Mary	3*length(InSig)
16-Mary	4*length(InSig)
64 -Mary	6*length(InSig)

4.3 Pulse Shaping

This module is used to create a filter that is later applied to the signal. This module implements four different types of Square Root Raised Cosine filter. The Filter Parameters are set in 'CreateSig.m'; the 'FltrType' parameter specifies the type of filter to be generated.

Input	Output	Hierarchy
• **Fout_sig**	• **Fout_sig**	**PulseShaping.m** **Filter Variables** **Generating Filter**

Hierarchy explained:

- **Filter Variables**
  The filter types implemented in this module require the following basic input parameters: ***Duration, dt (1/Fs), Rolloff, and Period (1/Fd)***

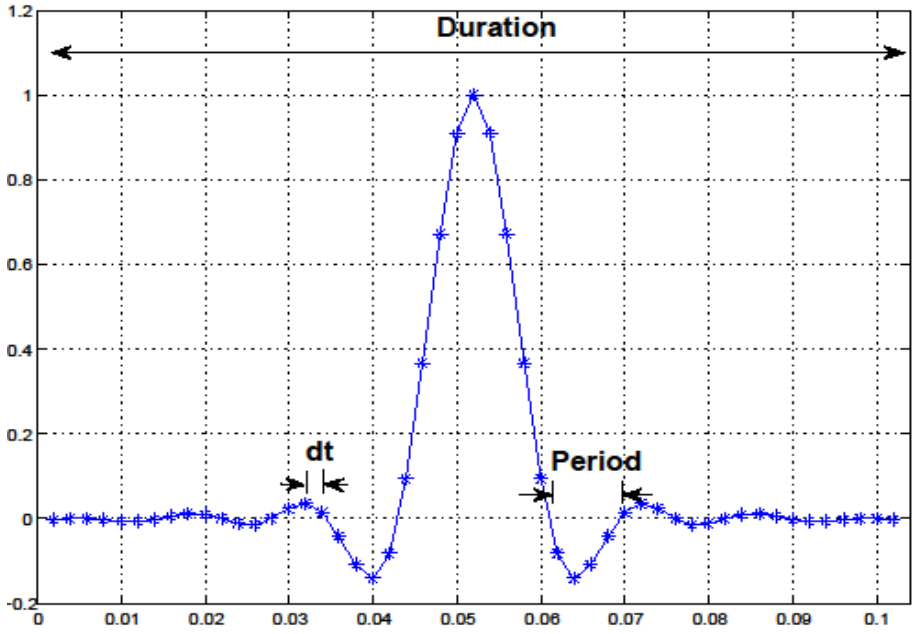


Fig 4: Pulse Shaping Filter Parameters.

The sample filter generated above has *duration* equal to 0.1, *period* equal to 0.01, *dt* equal to 0.01/5 and *rolloff* equal to 0.5. Also computed in this module the delay of the filter, which equals to **Duration/dt;** which is to be used in the receiver when retrieving the original message.

- **Generating Filter**

  Implemented in this module a case hierarchy which allows the user to use different types of pulse shaping filters as required for the simulation. The input parameter '**FltrType'** initialized in the *'CreateSig.m'* file is used to decide which of the following filters to be used for pulse shaping:

  - **'sqrtrcos':** Generates a Square Root Raised Cosine Filter.
  - **'sqrttrrcos':** Generates a Truncated Square Root Raised Cosine Filter.
  - **'sqrtmrcos':** Generates the Modified Square Root Raised Cosine Filter.
  - **'sqrtsfrcos':** Generates a Shifted Square Root Raised Cosine Filter.

  The Corresponding functions for generating the filters are located in the *'Pulse Shaping'* directory. For more information regarding the above filters, refer to the paper titled "Pulse Shaping for Differential Offset-QPSK." [3] The Generated Filters are then Normalized and saved in the Data vector **'fltr'.**

4.4 Space-Time Coding

This module implements a transmit technique in which the partitioned signal is processed and filtered. The user can implement any transmit scheme within this module and must correspondingly implement the same scheme on the receiver. [4] [5] The current file implements the Alamouti transmit scheme for a 2x2 MIMO system. [6]

Input	Output	Hierarchy
• **Fout_sig** • **ii: Iteration count**	• **Fout_tx**	**Transmit.m** **Transmitted sequence for each antenna** **Applying the filter**

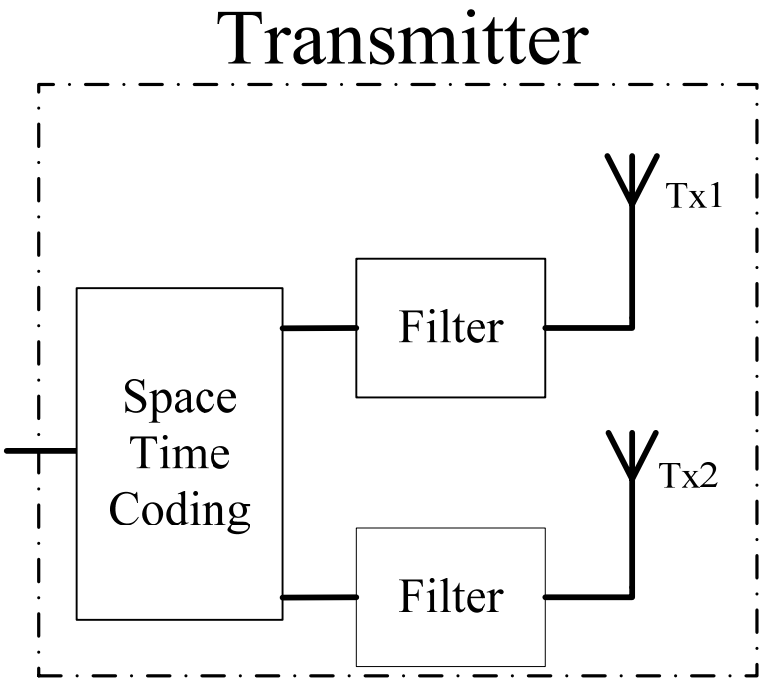


Fig 5: Transmitter Block Diagram

Hierarchy explained:

- **Transmitted Sequence for each antenna**
  The input signal was partitioned in 'CreateSig.m' into 'Num_tx" parts. This function is called within the iterative loop in the Main file. The idea behind this is to allow for faster transmission by transmitting the full signal by parts. Each iteration only transmits 'L' bits simultaneously, thus when 'itt' iterations are completed the entire signal would have been transmitted.

- **Applying the filter**
  Once the signal has been partitioned into the required number of transmit sequences, the signal is fed through the filter that was created in the 'PulseShaping.m' file. The signal is first upsampled and zeros are added at the end to accommodate for the delay in the filter. Once that is completed, the sequences are saved in the array 'tx' each antenna representing a column vector to be used in the following modules.

4.5 Channel

This module acts as the "transmission medium" for the communication system. This is the center piece of the package and in this module the user can test/ simulate different coefficient generation methods, noise structures, encoding and transmitting signal schemes.

Input	Output	Hierarchy
• **Fout_sig** • **Fout_tx** • **ii: Iteration count**	• **Fout_ch**	**Channel.m** **Generate channel Coefficients** **Generate Noise** **Encode and transmit signal**

Hierarchy explained:

- **Generate Channel Coefficients**
  First the module generates the channel coefficients that will be used for each transmission path, c***h0, ch1, ch2, and ch3*** as shown below:

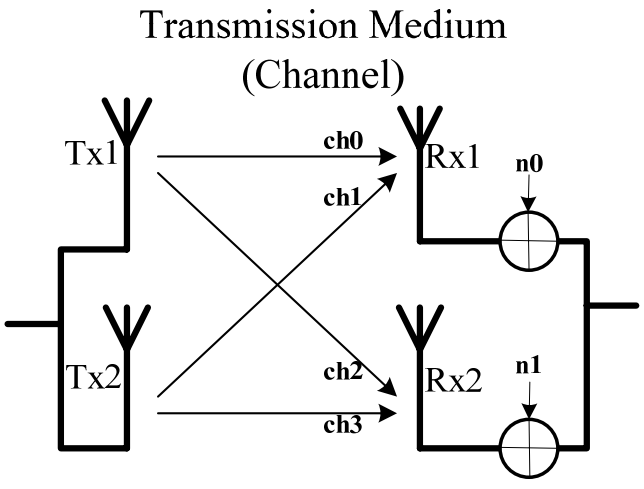


Fig 6: Transmission Medium

Included in the package are two functions that generate random Rayleigh fading coefficients using two methods. The first function 'Rayleigh.m" uses jakes model and the method of summing sinusoids to generate the Rayleigh fading coefficients. The other file named 'ricefad.m' uses Inverse discrete Fourier transform technique to create Rayleigh and Rician fading coefficients.

Both files require the user to input the size of the desired number of coefficients; this will be the length of the signal to be transmitted. As well, the user must input the sampling frequency and the bit period. The sets of coefficients generated are stored in the channel object "ch" and will used later on in the receiving algorithm.

- **Generate Noise**

  In this module, random noise (Additive White Gaussian Noise) is generated and added to our transmitted sequences that are being transmitted from each antenna. According to the Signal to Noise Ratio (SNR) range that has been provided in 'CreateSig.m', the simulation will run for all the input values of SNR. The SNR value is used to calculate the noise variable needed to create the random receiver noise. This part of the module generates a random noise that is 10x larger than the required sequence length, and further randomly select a vector from this sequence; by doing so we allow for less error within the MATLAB random generator and minimize the correlation between the coefficients of the random noise.

- **Encode and transmit Signal**

  The two for loops in the section encode the signals to be received at the receive antennas by accounting for channel coefficients and noise accordingly. The Final received signals are saved in the object 'ch', variable 'R' in order to be used in the next module as follows:

  R1= Ch1*Tx1 + Ch2*Tx2 + ... + Ch (Numtx) *Tx (Numtx)
  R2= Ch1 (Numtx+1)*Tx1 + Ch (Numtx+2)*Tx2 + ... + Ch (Numtx*2)*Tx (Numtx*2)
  ...
  R (Numrx) =.....

4.6 Receiving

The receiver in a communication system implements a decoding and decryption scheme that is designed in parallel with the transmit scheme; by doing so the maximum likelihood detector in the receiver will try to retrieve the original signal with the least error.

Input	Output	Hierarchy
• Fout_sig • Fout_ch • ii: Iteration count	• Fout_sig	Receiver.m Apply Filter and downsample Combiner/ Likelihood detector Demodulating Bit Error calculation

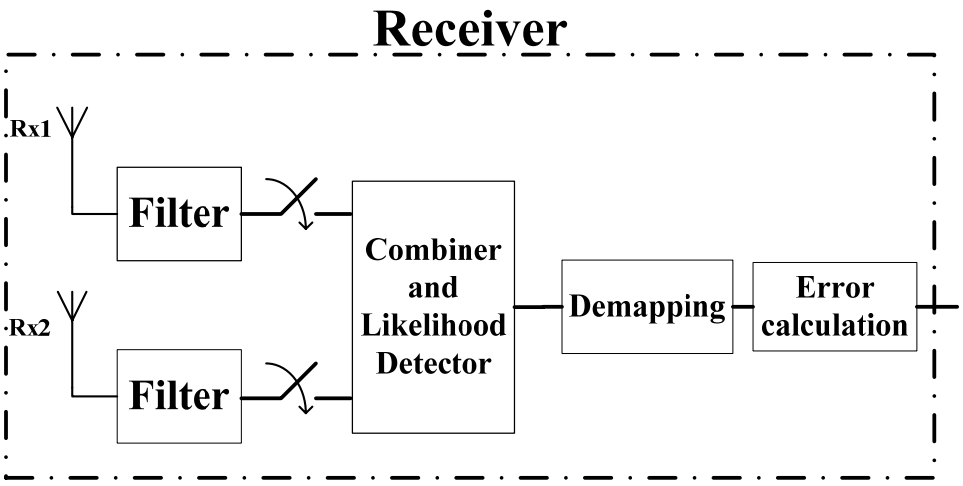


Fig 7: Receiver Block diagram

Hierarchy explained:

- **Applying the filter and down sampling**
  The same pulse shaping filter that was used in the transmitter is applied once again to the received signal (the variable 'fltr'). For each received signal on each receive antenna we apply the filter and down sample the received signal to its original length. In order to account for the delay in the filter, the signal is truncated by an amount 2*Delay (from the beginning and the end of the signal). At this point the received signal is the same size as the original unsampled and unfiltered signal.

- **Combiner and Likelihood detector**
  Since this is a MIMO system and depending on the scheme we are using, the signal received on each antenna need to be rearranged and combined and passed to a maximum likelihood detector. The likelihood detector implemented in the package decodes for the Alamouti 2x2 scheme, since this scheme was used in the transmitter module. The user must change the combiner and the likelihood detector accordingly when simulating different transmission schemes.

- **Demodulating**
  Once the final signals are rearranged and estimated they need to be demodulated. The user must implement within this code a demodulating technique that corresponds to the modulating technique used in 'Mapping.m'. In the sample simulation, implemented is MATLAB's predefined phase shift keying demodulating function since the BPSK function was used to modulate the input signal.

- **Bit Error Calculation**
  The primary goal of the simulation is to be able to find how efficient and accurate the scheme, the noise structures and the channel coefficient are in this communication system. This is done by comparing the input signal to the final received signal on a bit-by-bit base. This comparison will yield the Bit Error Rate and will save the output in 'BER' Accordingly.

4.7 Plotting

This module acts like the data processing and assessing. It currently generates a graph that shows the results of the simulation. The current implemented graph technique takes an average of the entire Bit Error Rate for all iterations and plots it against the Signal to Noise ratio range. The user has the ability to use the results as they wish since the output is saved in the '.mat' file which has stored the input and the output results of the simulation.

Input	Output	Hierarchy
• **Fout_sig**	• **n/a**	**Plotting.m** **Generating the Graph**

Hierarchy explained:

- **Generating the graph**

The current module generates a semilog of BER Vs SNR and saves the graph in the current directory for further reference. The graph is saved as a matlab '.fig' format and '.eps' format uniquely named for each simulation as follows ["Modulation Type"-"Filter Type" ("Length", "Iteration")]. Thus if the package is simulating a BPSK with a square root raised cosine filter, length of 1000 and 100 iterations then the graph files will be named as follows:

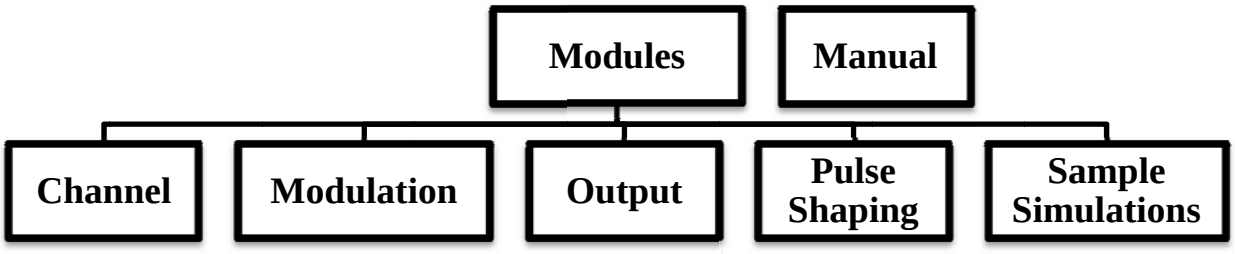
BPSK-sqrtrcos (1000, 100).fig

BPSK-sqrtrcos (1000, 100).eps

These graphs will be saved in the subdirectory 'Output' once the simulation is completed.

5. Directories Explained

The package includes many other functions and files beyond the 7 major modules. For that matter, these files were arranged in directories accordingly since the majority of them are indirectly used and do not require any editing from the user. The directories and their contents are arranged and described as follows:



Modules

This is the parent directory, when running the simulation in MATLAB this should be the working directory. This directory contains the following files:

- **Main.m**
- **CreateSig.m**
- **Mapping.m**
- **PulseShaping.m**
- **Transmit.m**
- **Channel.m**
- **Receiver.m**
- **Plotting.m**
- **Readme.txt**

Channel

This directory contains files that are used to generate the channel coefficients in the 'Channel' module. Currently available are two methods for generating the channel coefficients explained below:

- **Rayleigh.m:** This function is used to generate Rayleigh fading channel coefficients.
- **Ricefad:** This function is used to generate correlated Rayleigh variates by Discrete Fourier Transform. [7]

Modulation

Included in this directory are all the functions that are used in the ‘Mapping.m’. Since Mapping implements a case hierarchy, each cases corresponding to a different modulation technique, the functions here are individually called out when required from ‘Mapping.m’. It is important for the user to understand how each technique is being implemented, thus reference to these files will be needed from time to time.

Output

This directory will be used to save the output of the simulations. Each simulation will generate 3 files, two of which are the generated plots; the third is the ‘.mat’ file. The uniquely named files will be saved in this directory after each simulation.

Pulse Shaping

The directory ‘Pulse Shaping’ contains the functions that are used to generate the pulse shaping filters. As mentioned in section 4.3, four types of pulse shape filters are implemented and the files within this directory correspond to these filters.

Sample Simulations

This directory includes a set of plots and output (‘.mat’) files that were generated during the testing phase of this package. These plots have been provided to give an example of the expected output from the simulations. The user can open the output files in MATLAB to examine the contents provided to the user once the simulation is complete.

Manual

Included in this directory is the PDF version of this manual.

6. Performance Comparison

In order to demonstrate the possible performance difference between the different pulse shaping techniques, three sets of simulations were conducted for the following three Square Root raised Cosine pulse shape filters: ***sqrtrcos* (normal), *sqrttrrcos* (truncated), and *sqrtmrcos* (modified).** *Alamouti's 2x2* transmission technique is implemented in the simulation. The three simulations were conducted with *length* equal to 5000, *iteration* count equal to 500, *SNR* range of 0 to 12 and BPSK *modulation.* The filters in the three simulations had *period* equal to 0.01, *duration* equal to 0.05, *rolloff factor* equal to 0.25 and an *oversampling* rate of 4.

The performance results were plotted and compared as shown below:

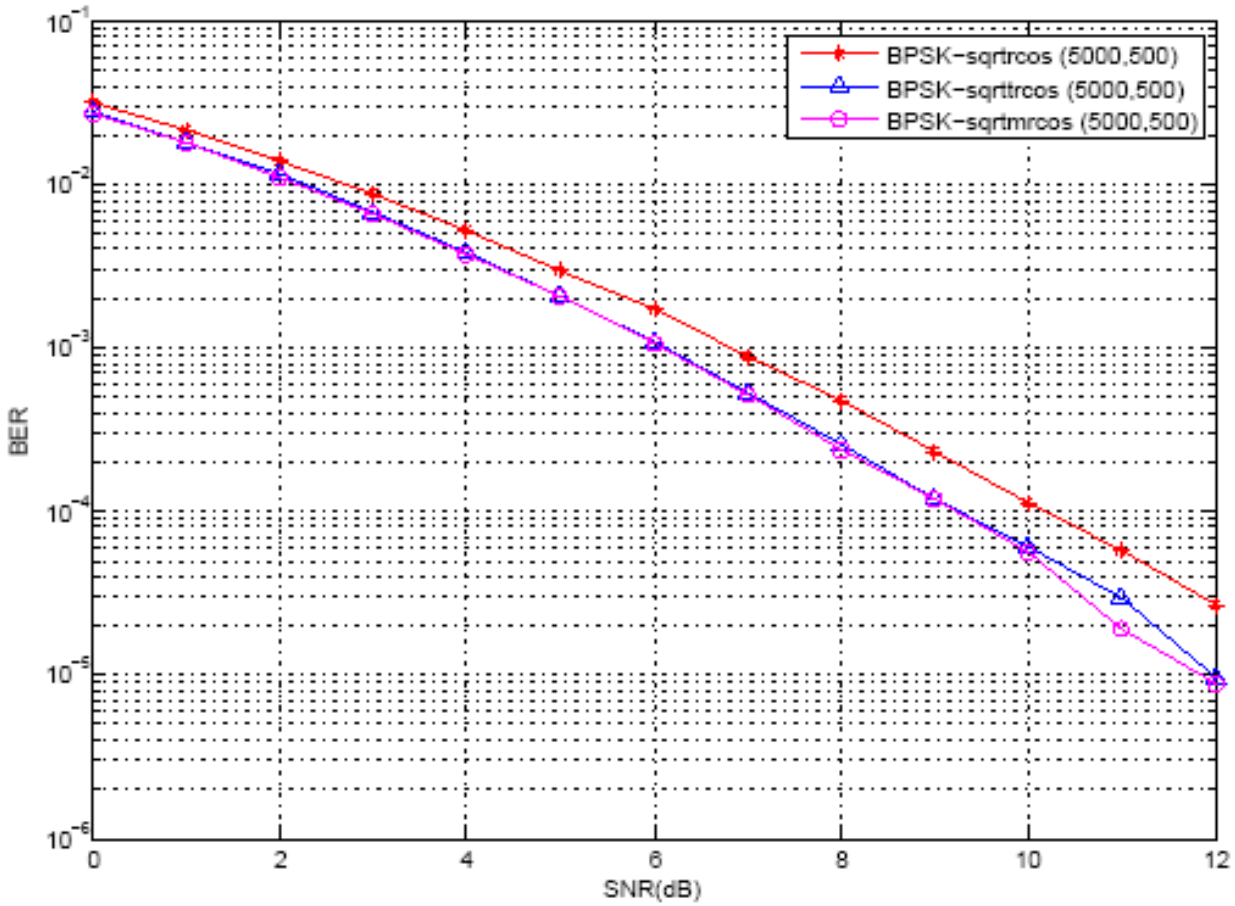


Fig 8: Performance Comparison between Different Pulse shaping Filters

The result shown in Figure 8 indicates that the truncated Raised Cosine and the Modified Raised Cosine produce similar performance; while the two perform better than the normal Square Root Raised Cosine. *Sqrttrrcos & sqrtmrcos* provide a 1dB performance boost than *sqrtrcos* at Signal to Noise Ratio (SNR) above 4dB. This comparison is a simple example of how this package can be used to test and analyze wireless transmission schemes.

7. References

[1] Arash Mirbagheri, K.N. Plataniotis, S. Pasupathy, `An enhanced widely linear CDMA receiver with OQPSK modulation' , IEEE Trans. on Communications, vol. 54, no. 2, pp. 261-272, February 2006.

[2] S.W.L. Poon, K.N. Plataniotis, S. Pasupathy, `Superimposed asymmetric modulation in narrow-band fading channels with orthogonal codes' , IEEE Trans. on Wireless Communications, vol. 5, no. 6, pp. 1260-1265, May 2006.

[3] A.C.C.C. Lam, A. Elkhazin, S. Pasupathy, K.N. Plataniotis, `Pulse shaping for differential offset-QPSK' , IEEE Trans. on Communications, vol. 54, no. 10, pp. 1731-1734, October 2006.

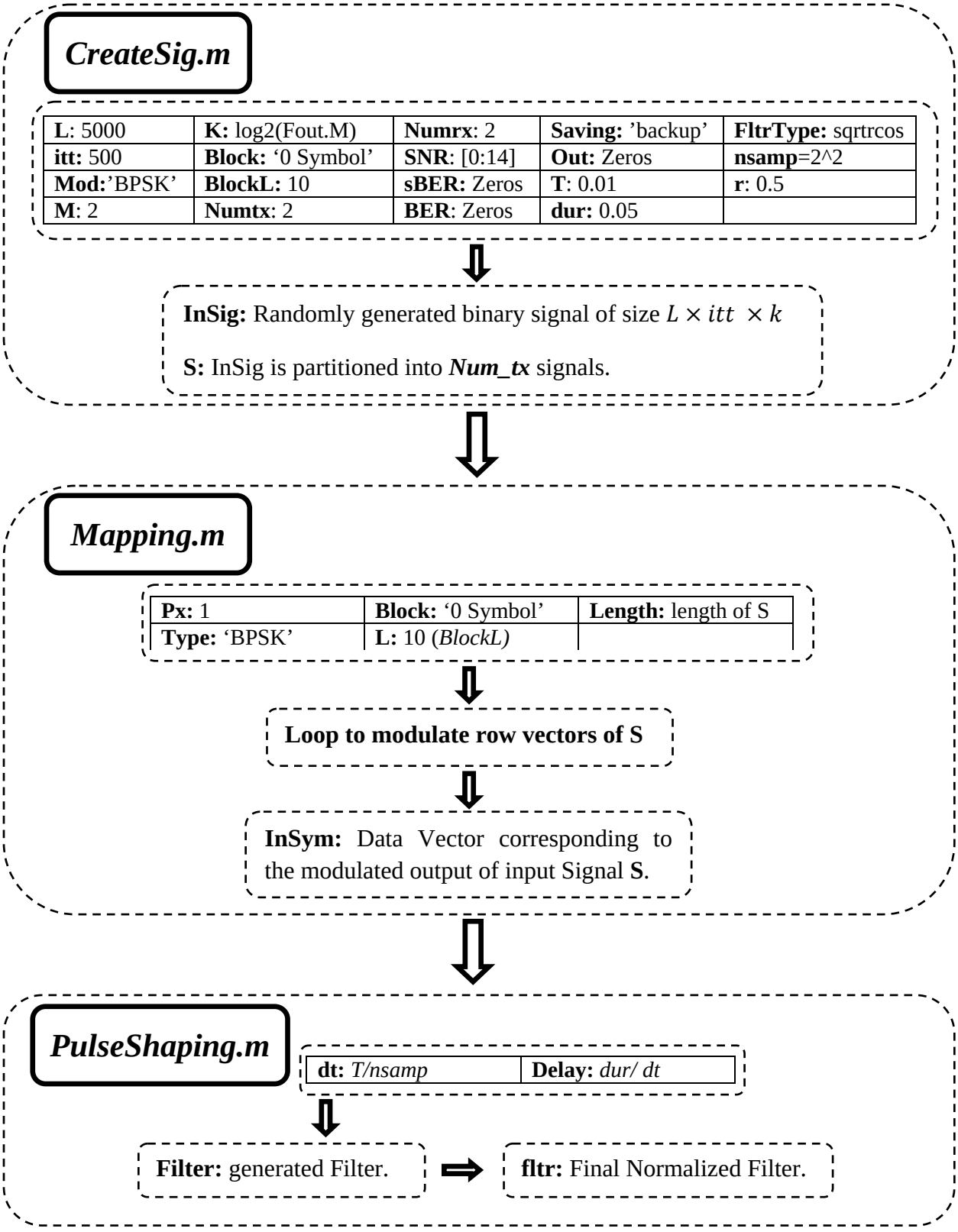
[4] S. Lam, K.N. Plataniotis, S. Pasupathy, `Self-matching space-time block codes for matrix Kalman estimator based ML detector in MIMO fading channels' , IEEE Trans. on Vehicular Technology, vol.56, no 4 II, pp. 2130-2142, July 2007.

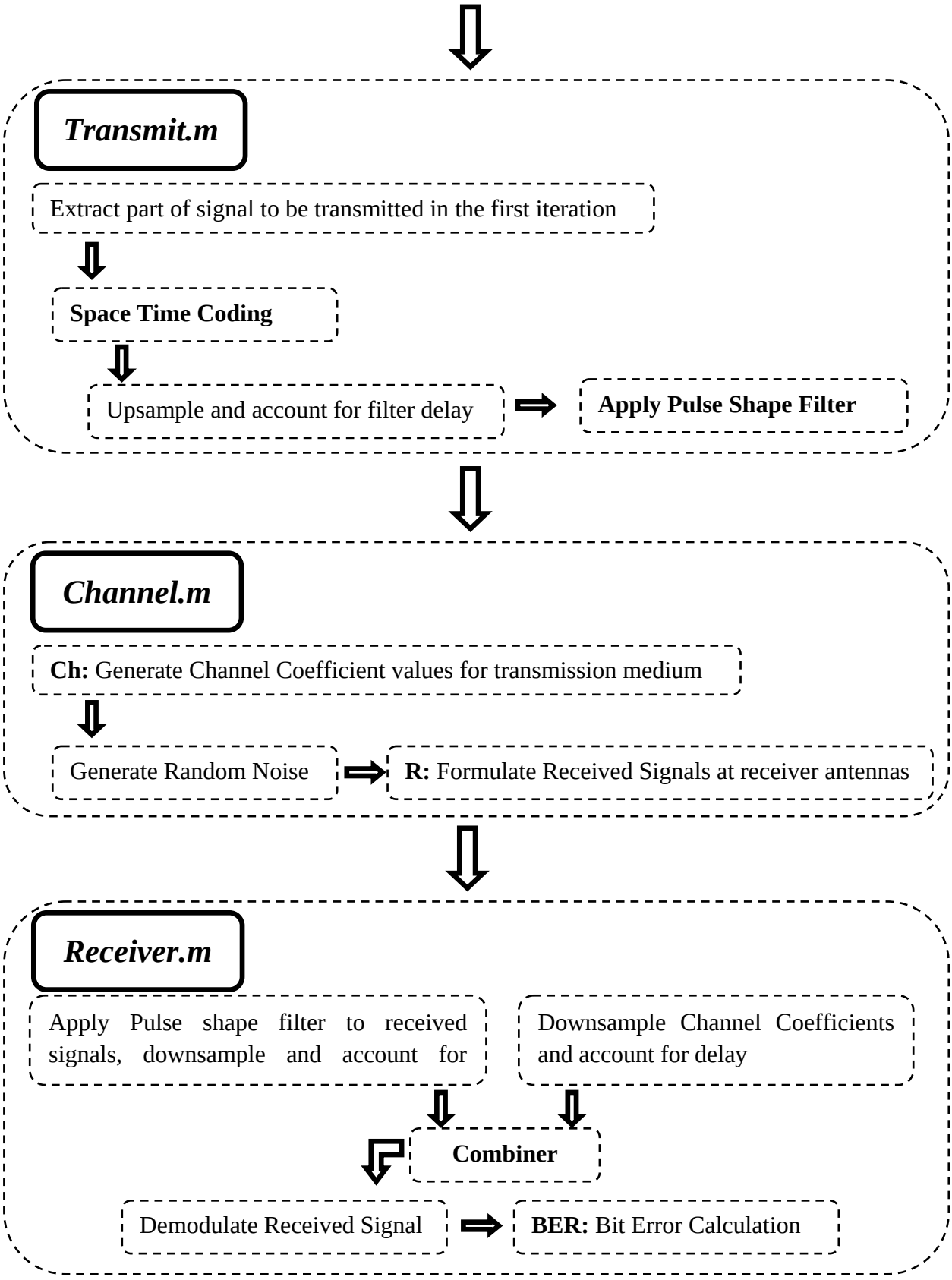
[5] A. Elkhazin, K.N. Plataniotis, S. Pasupathy, `Reduced dimension MAP turbo-BLAST detection' , IEEE Trans. on Communications, vol. 54, no. 1, pp. 108-118, January 2006.

[6] Alamouti, Siavash M, “A Simple Transmit Diversity Technique for Wireless Communications:, IEEE JOURNAL ON SELECT AREAS IN COMMUNICATIONS, VOL. 16, NO. 8, OCTOBER 1998, 1451-1458

[7] D.J. Young and N.C. Beaulieu, "The Generation of Correlated Rayleigh Random Variates by Discrete Fourier Transform", IEEE Transactions on Communications, vol. 48, pp. 1114-1227, July 2000.

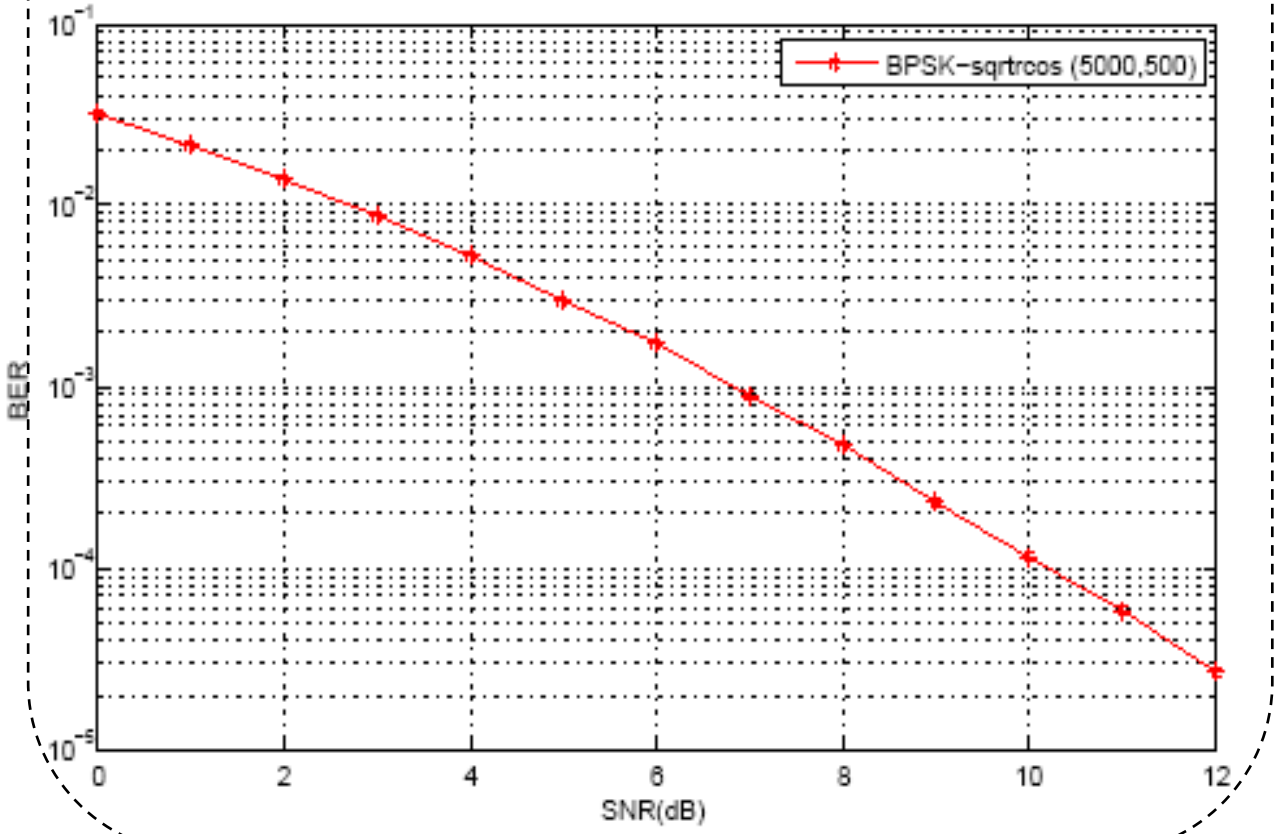
Appendix A: Sample Simulation/ Simulation Workflow




Transmit.m
Extract part of signal to be transmitted in the first iteration
Space Time Coding
Upsample and account for filter delay
Apply Pulse Shape Filter
Channel.m
Ch: Generate Channel Coefficient values for transmission medium
Generate Random Noise
R: Formulate Received Signals at receiver antennas
Receiver.m
Apply Pulse shape filter to received signals, downsample and account for
Downsample Channel Coefficients and account for delay
Combiner
Demodulate Received Signal
BER: Bit Error Calculation

⇩

Plotting.m


BPSK−sqrtrcos (5000,500)
BER
SNR(dB)
10^-1
10^-2
10^-3
10^-4
10^-5
0
2
4
6
8
10
12