UNIVERSITY OF WESTMINSTER

Title:	**GMSK AND 8PSK MODEM SOFTWARE IMPLEMENTATION**
Author:	**Hashan Roshantha Mendis**
Supervisor:	**Katerina Christofylaki-Fines**
Date:	**April 2008**
Course:	**BSc. Network and Communications Engineering**

Third Year project thesis submitted
in partial fulfillment of the requirements of the degree of
Bachelor of Science
in the Department of Electronic Systems at the University of Westminster

Acknowledgment

I would like to thank my parents and sister, for their love and support throughout my Primary and Undergraduate education. I thank my extended family members for their kindness and generosity.

I would like to thank my thesis supervisor, Ms Katerina Christofylaki-Fines, for her enthusiasm, guidance, patience and her generosity in entertaining the massive amounts of queries which was subjected towards her during the affirmation of this thesis. Her encouragement, good ideas and excellent teaching made this project successful.

Ms Dragana Barjamovic for her advice on certain DSP related problems I encountered in this project and other academic areas. Her Signal Processing and Matlab based Laboratory work helped to reduce the Matlab learning curve. Dr. Mohammed Al-Janabi who was a pillar of support throughout this project, he constantly motivated us and reminded us of our deadlines and helped to complete this project on time.

I would also like to thank Ms Tehani Renganathan and Mr. Dasith Fernando for lending me some books which were of great value to my research in this project. Mr. Kamal Kouba and Mr. Nandan Das for their tips and suggestions which helped me to debug my Matlab code in more than one instance. I thank Mr. Hasula Dias who gave me advice when writing my report.

Lastly, I would like to thank my closest, fellow undergraduate colleagues, who assisted me in numerous ways throughout the period of my undergraduate study.

Abstract

Wireless and Mobile communications have grown rapidly over the past decade. Different techniques to boost data rates and to accommodate the ever increasing number of wireless subscribers have been the main concern of communication engineers worldwide. GSM is still the leading global wireless standard with over 70% of the worldwide market.

This documentation is concerned with the Digital modulation techniques used for the GSM and EDGE mobile communication systems. They are Gaussian Minimum Shift Keying and 8-Phase shift keying, respectively.

First the Modems are broken down into subsections and their key properties are examined. Next computer simulations of the Modems are implemented, taking into consideration the strict standards set by the 3rd Generation Partnership Project. The simulations will include scenarios where the Modulated signals are passed through a Multipath Fading channel and subjected to Additive White Gaussian Noise. Consequently the Bit Error rates are calculated at the output of the demodulators.

The results obtained were used to critically analyse the two modulation schemes in terms of Bit Error Rate Performance, power spectra and distortion of the signal due to Inter-symbol Interference.

List of Abbreviations

16-PSK	- 16-Phase Shift Keying
3GPP	- 3rd Generation Partnership Project
8PSK	- 8-Phase Shift Keying
ADC	- Analog to Digital Converter
AMPS	- Advanced Mobile Phone System
AWGN	- Additive White Gaussian Noise
BER	- Bit Error Rate
CDMA	- Code Division Multiple Access
CEPT	- European Conference of Postal and Telecommunications
DAC	- Digital to Analog Converter
DSP	- Digital Signal Processing
DTMF	- Dual Tone Multi-frequency
EDGE	- Enhanced Data Rates for GSM Evolution
EGPRS	- Enhanced GPRS
ETSI	- European Telecommunications Standards Institute
FIR	- Finite Impulse Response
FM	- Frequency Modulation
FSK	- Frequency Shift Keying
GF-8PSK	- Gaussian Filtered 8-Phase Shift Keying
GLPF	- Gaussian Low Pass Filter
GMSK	- Gaussian Minimum Shift Keying
GPRS	- General Packet Radio Service
GSM	- Global System for Mobile communications
IIR	- Infinite Impulse Response
IQ	- In-Phase and Quadrature
ISI	- Inter-Symbol Interference
LOS	- Line of Sight
MODEM	- Modulator/Demodulator
MS	- Mobile Station
MSK	- Minimum Shift Keying
NMT	- Nordic Mobile Telephone
NRZ	- Non-Return to Zero
OFDM	- Orthogonal frequency-division multiplexing
PDF	- Probability Distribution Function
PSD	- Power Spectral Desnsity
QPSK	- Quadrature Phase Shift Keying
RAM	- Random Access Memory
RC	- Raised Cosine
RCF-8PSK	- Raised Cosine Filtered 8-Phase Shift Keying
RMS	- Root Mean Square
SMS	- Short Message Service
SNR	- Signal to Noise Ratio
SOS	- Sum of Sinusoid
UMTS	- Universal Mobile Telecommunication System
VCO	- Voltage Controlled Oscillator
WiMAX	- Worldwide Interoperability for Microwave Access

CONTENTS

A software Compact Disk containing all computer code and a softcopy of this documentation is attached to the back of the report.

CHAPTER 1
INTRODUCTION

This technical report contains an in-depth physical layer analysis of two main wide spread cellular systems in the world today, namely GSM (Global System for Mobile communications) and EDGE (Enhanced Data Rates for GSM Evolution). A detailed study of the two modulation schemes used in each technology will be carried out. A simulation will be designed, implemented, and results will be compared with theoretical results. The Simulations strictly abide the 3GPP (3rd Generation Partnership Project) GSM/EDGE Radio Access Network (Modulation, Release 7) specifications. The two modulation schemes will be introduced to Additive White Gaussian Noise and Fading Channels (namely Rayleigh and Rician).

A critical analysis of the results obtained from the simulations will be carried out and explained. The performance evaluation of the simulated modulation methods is of the Monte-Carlo type, and will be mainly based on Bit Error Rate calculations by comparing input random bit sequences at the transmitter, and resulting sequences at the receiver. Individual simulation parts will be described by pseudo code and flow charts. The Matlab M-File scripts used for the simulation can be found in the appendix of this report.

To summarize this project will be divided into four main stages.

- Research of the Modulation schemes used in the GSM and EDGE mobile systems.
- Design the Simulation,
- Implement the Simulation using a suitable programming language
- Perform a critical analysis of the simulation results.

Note: A review of different literature on the GMSK and 8PSK Modems were made and can be found in Chapter 2 and Chapter 3 individually. This was done to maintain the structure/flow of this report, which would have been disrupted if the literature review was in one separate chapter alone.

1.1 OBJECTIVE

The main objective of this project is to gain an insight of the physical layer properties of the GSM and EDGE systems, specifically modulation. The knowledge gained in Digital Signal Processing and Communications during the course of the past two years will be used to simulate these modulation schemes. Therefore another important objective will be to confidently use communication theory to generate simulations in a particular programming language. Gaining confidence in writing simulations in an industry standard programming language is also one of my core objectives. It is also hoped that, this report and project will be of use to future research/student projects.

Even though GSM and EDGE are superseded by more advanced mobile communication systems such as UMTS (Universal Mobile Telecommunication System) and WiMAX

(Worldwide Interoperability for Microwave Access), they are still widely in use all around the world. The understanding of initial digital mobile communication standards is recommended before attempting to tackle 3rd and 4th Generation systems.

1.2 PROJECT WORK PLAN

The above tasks have been further divided into subtasks for ease of organisation. The Gantt chart in Figure 1.1 shows the project work plan schedule, clearly identifying these subtasks and their start and end dates. It gave a certain structure to the project, and it helped to be more organised by constantly reminding the deadline for the completion of each task.

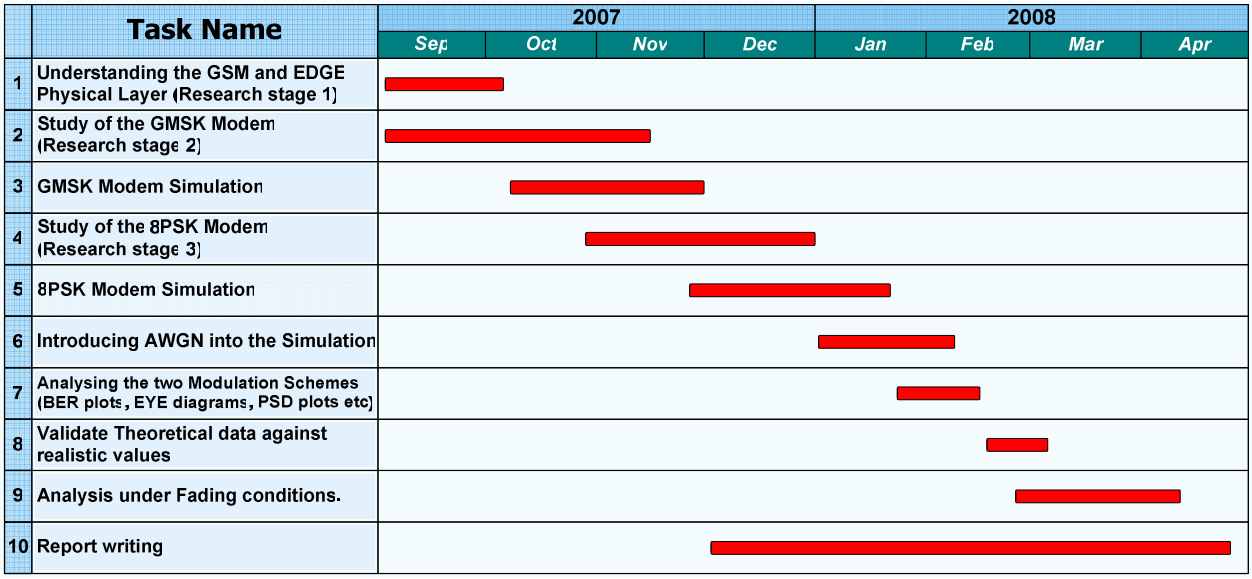

Figure 1.1 – Project work plan

The illustration below shows a block diagram of the simulation setup used in the project.

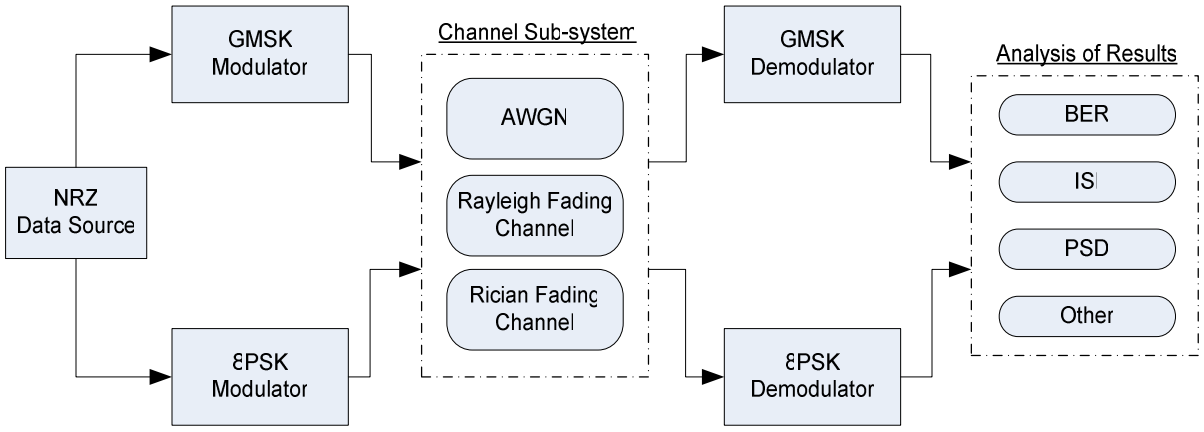

Figure 1.2 – Simulation setup

Each block of Figure 1.2 will be described in detail in the preceding chapters.

CHAPTER 2
THE GAUSSIAN MINIMUM SHIFT KEYING MODEM

Modulation is the process of converting digital data into analog form suitable for transmission. Demodulation is the process of converting the received analog data into digital form suitable for a computer or other digital system to understand. A Modem (Modulator/Demodulator) is a hardware structure which performs both these tasks.

This Chapter introduces the programming language that will be used to carry out the simulation and describes Gaussian Minimum Shift Keying (GMSK) which is the modulation scheme specified for GSM. The GMSK transmitter and receiver structures will be looked at and a description of all the processes in which the GMSK modem simulation is carried out will be given.

2.1 SELECTION OF A PROGRAMMING LANGUAGE

One of the important decisions faced at early stages of this project was selecting a suitable programming language for the simulation. There were several characteristics required:

- The language/tool had to be used in the industry at present.
- The language had to be fairly easy to learn
- Should give complete control over code and easy to debug.
- It should have lots of built in functions and libraries related to wireless communications.
- Have numerous plotting functions related to analysis of a communication system.

The following are a compilation of several programming languages which fit the above criteria

.SciLab :

Advantages:

- Free Open Source Matlab like Clone available for download.
- Used in several industries for signal processing, statistical analysis, image enhancement and fluid dynamics simulations etc.
- Supports an interactive mathematical shell
- Similar functionality as Matlab.
- Relatively new simulation and mathematical modelling tool.
- Can manipulate matrices and complex numbers.

Disadvantages:

- No toolboxes/libraries for Wireless Communication systems.
- Not many books published, therefore learning curve will be steeper compared to other languages.

- Certain operations will take a longer time than other programming languages
- Works only on the Windows platform.

C/C++ :

Advantages:

- Free compilers available for download.
- Widely used in the engineering community.
- Can be run on the Unix/Linux/Windows platforms
- Has a Communications and Signal Processing Library called IT++ which shows functionality similar to Matlab and Octave.
- C/C++ books can be found in any good bookstore or online.
- Better than Matlab, in handling computer memory does not cause 'out of memory' issues easily.

Disadvantages:

- Certain sections of code will be longer than when coding in Matlab/Octave.
- Does not support Matrices, only multidimensional arrays.
- Does not support an interactive mathematical shell
- Very Limited capability when it comes to plotting and visually analysing data.
- Difficult to learn than Matlab, has a steeper learning curve.

Python :

Advantages:

- Open source programming language. Compiler is free to download.
- Compatibility on Unix/Linux/Windows platforms.
- Lots of libraries related to signal processing (eg: Numpy, Scipy etc)
- Additional Library for Matlab-like plotting functions (eg: matplotlib)
- Lots of user support online, newsgroups and forums.
- Extensively used to develop mobile phone applications, but not as much for simulations.
- Multi-paradigm language.
- Can manipulate matrices, arrays and complex numbers.

Disadvantages:

- Certain sections of code will be longer to implement in python than in Matlab/Octave.
- Awkward notation and syntax a lot different to C/C++/Matlab.
- Relatively new scripting language, therefore not many books are published regarding python.

GNUOctave :

Advantages:

- Open Source programming language which has a Graphical User interface similar to that of Scilab and Matlab.
- Vast compatibility with Matlab.
- Many toolboxes and libraries related to DSP and Communications available.
- Not as much user support online as compared to C/C++ and Matlab, but has gained popularity amongst engineers over the couple of years.

Disadvantages:

- Not as much support for modelling communication systems as Matlab
- Certain plotting functions are much simpler and flexible in Matlab than Octave.
- Not widely used in the communication industry, but gradually catching up to Matlab.
- According to benchmarks Octave is a lot slower in processing matrices and other data than Matlab.

Matlab & Simulink :

Advantages:

- The standard stable and mature tool used in the DSP and Communications industry.
- Extensive support for plotting various graphs and a multitude of Communication systems analysis tools available.
- Simulinks' Communication Blockset gives the opportunity to the designer to focus more on the results than the design itself.
- Simulink offers many DSP and Wireless Communication related block sets.
- Matlab has many toolboxes related to Signal Processing and Communications which offer a wide variety of useful functions.
- A magnitude of online support as well as user-groups and many books have been published regarding Matlab illustrating how to use it as a simulation tool.
- The Syntax is fairly easy to learn, the documentation is easy to understand and very educational and the interface is very user friendly.
- Convenient notation for linear algebra.
- Impressive output formatting for matrices.

Disadvantages:

- Extremely Costly: over £1300 for the commercial product, over £80 for the student edition.
- Manipulation of large data sets, loops and conditional statements are faster when executed in C/C++ than in Matlab, as Matlab is an interpreted language while C/C++ are compiled languages.
- Plotting results for large matrices is also a slow process.
- Limitations on the size of the data set that can be used. Matlab consumes a large proportion of your physical as well as your virtual memory when it is handling large matrices and is always an issue for machines which are low on RAM(Random Access Memory)

Justification of the Selected Programming Language:

Matlab has its disadvantages when it comes to speed of execution and consumption of Memory (RAM), but in my opinion it is the best choice for this project. Its signal processing and communications toolboxes will be helpful when simulating the two modulation schemes. There are built in functions that can be used to generate eye-diagrams, power spectral density plots, bit error rate calculations, which will be very useful in the analysis stage of this project.

Simulink which is a Model-based design environment for engineering simulations was considered, because of its ease of use. The designer has only to drag and drop the function blocks and set its parameters correctly to implement a simulation. However to gain a better understanding of the communication system you are modelling, Matlab is a better solution. In Matlab you have more flexibility over the functions and code being used and execution time is faster compared to Simulink.

Therefore for the above mentioned reasons, Matlab was chosen as the programming language that will be used in this project.

2.2 GLOBAL SYSTEM FOR MOBILE COMMUNICATIONS (GSM)

GSM is a globally accepted digital mobile communication standard for digital cellular communication. It was established by the European Conference of Postal and Telecommunications Administrations (CEPT) in 1982 and later on was transferred to the European Telecommunications Standards Institute (ETSI) in 1989 [10]

Phase 1 of GSM was completed in 1990, Phase 2 in 1994 and Phase 2+ was completed in 1996 and standardised thereafter. GSM was a solution for the First Generation Analogue wireless telephone technology which was available at that time such as NMT (Nordic Mobile Telephone) and AMPS (Advanced Mobile Phone System). [12]

Some of the services offered by the GSM system are listed below [12,10]:

- Subscriber Services
  - Cell broadcast
  - Telephony(Voice calls)
  - Short Message Service (SMS)
  - Voice mail

- Supplementary Services (provided by the network) :
  - Call Forwarding
  - Call Barring
  - Call Waiting
  - Call Hold
  - Multiparty service
  - Calling Line Identification
  - International Roaming etc.

The table below shows the important characteristics of the GSM Physical Layer.

Operating bands	900Mhz and 1800Mhz (Europe) 1900Mhz and 850Mhz (US & other parts of the world)
Bandwidth	25Mhz
Carrier Frequency	200KHz
Multiple Access Method	Time Division Multiple Access (with 8 time slots, 0.577ms each)
Modulation scheme	Gaussian Minimum Shift Keying (GMSK, BT = 0.3)
Over the air bit rate	270.833kbps
Channel Coding	Convolution Coding (18.9kbps)
Speech Coding	Regular Pulse Excitation with Long Term Prediction (at 13kbps)

Table 2.1 – GSM Physical Layer characteristics

2.3 THE GAUSSIAN MINIMUM SHIFT KEYING TRANSMITTER

GMSK is a type of constant-envelope Frequency Shift Keying (FSK) method where the frequency modulation is a carefully performed phase modulation. Its precursor is Minimum Shift Keying (MSK), which does not use the Gaussian pre-modulation pulse filter that GMSK uses.

For transmission of mobile radio signals through fading channels certain characteristics are required from the modulation method used. High immunity to noise and interference, ease of implementation and most importantly compact power spectrum. By using a pre-modulation filter it is possible to reduce the spread of the power spectrum drastically.

However, according to [11],

> *"It should be noted that since the Gaussian pulse-shaping filter does not satisfy the Nyquist criterion for Inter-Symbol-Interference cancellation, reducing the spectral occupancy creates degradation in performance due to increased ISI"*

Thus the trade-off of having more compact power spectrum is that the Gaussian pre-modulation filter spreads the signal pulse and introduces ISI.

2.3.1 The Gaussian Pulse Shaping filter

The Gaussian shaped pulse filter is the key element for generating GMSK signals which have a compact power spectrum compared to MSK. In accordance to the specification, GSM uses a Gaussian pulse shaping filter with $BT = 0.3$. Where;

B = the 3db bandwidth of the filter and,
T = bit duration.

According the 3GPP specification [1], the Impulse response of the Gaussian filter is given below;

$$h(t) = \frac{\exp\left(\frac{-t^2}{2\delta^2 T^2}\right)}{\sqrt{(2\pi)}.\delta T} \qquad (1.1)$$

$$\delta = \frac{\sqrt{\ln(2)}}{2\pi BT} \quad \text{and BT = 0.3 for GSM} \qquad (1.2)$$

The Differential coded NRZ data pulses are sent through h(t) and thereby they are convolved, giving the Gaussian filter response;

$$g(t) = h(t) * rect\left(\frac{t}{T}\right) \tag{1.3}$$

Where (*) denotes convolution. And the function *rect(t)* is defined as (1.4)

$$rect\left(\frac{t}{T}\right) = \begin{cases} 1/T, & for|t| < \frac{T}{2} \\ 0, & otherwise \end{cases} \tag{1.4}$$

According to Appendix D of the LINK PCP [2], this Impulse response of the filter can also be written as:

$$g(t) = \frac{1}{2T}\left[Q\left(2\pi BT\frac{t-T/2}{T\sqrt{\ln(2)}}\right) - Q\left(2\pi BT\frac{t+T/2}{T\sqrt{\ln(2)}}\right)\right] \tag{1.5}$$

Where Q(t) is a the Q-function,

$$Q(t) = \int_t^\infty \frac{1}{2}\exp\left(\frac{-\tau^2}{2}\right)d\tau \tag{1.6}$$

The Gaussian filter should be implemented in such a way that the phase change between a 1 and a -1 (or 1 and 0) should be $\frac{\pi}{2}$. Therefore the overall filter should be truncated and scaled by multiplying by a constant *K* such that the following equation is satisfied [2].

$$\int_{-T}^{T} Kg(t)dt = \frac{\pi}{2} \tag{1.7}$$

2.3.2 GMSK Modulator Design

There are many ways to generate a GMSK signal. This section will discuss 3 main methods which are used.

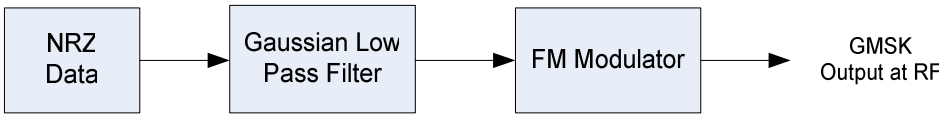

Figure 2.1 – GMSK signal generation using a FM Modulator

As Figure 2.1 shows a certain type of GMSK modulator design which incorporates Frequency Shift Keying. The FM Modulator block shown, uses a VCO (Voltage Controlled Oscillator) to change the frequency of the signal according to the respective bit change. One disadvantage which this type of design faces is that the frequency deviation factor that the VCO is locked onto may change/drift over time and temperature.

Another method being used is the Quadrature base-band method and is shown in Figure 2.2

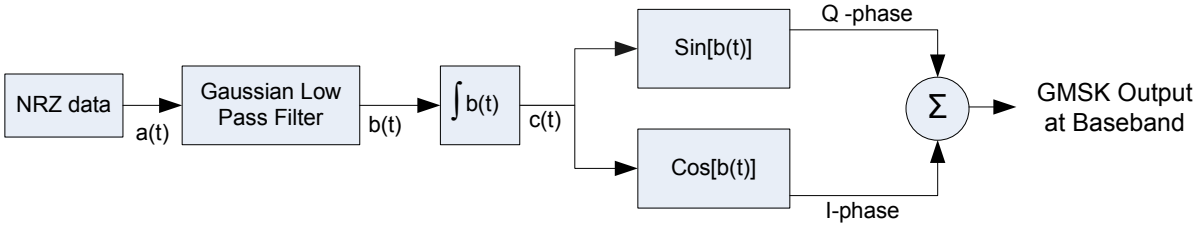

Figure 2.2 – GMSK signal generation using the Quadrature method

As Figure 2.2 illustrates, the NRZ data pulses are passed through a Gaussian LPF and then integrated, with respect to time (from t to ∞) to give the function *c(t)* which is the GMSK phase. Once the function *c(t)* is obtained the sine and cosine of this is taken to obtain the Q and I phase components of the signal respectively.

The third method uses a look-up table which compares the current symbol and the previous symbol and determines which phase transition to use. The 'CommsDesign' website gives a detailed description of this method. Figure 2.3 is a flow chart similar to the one found in their web site [20].

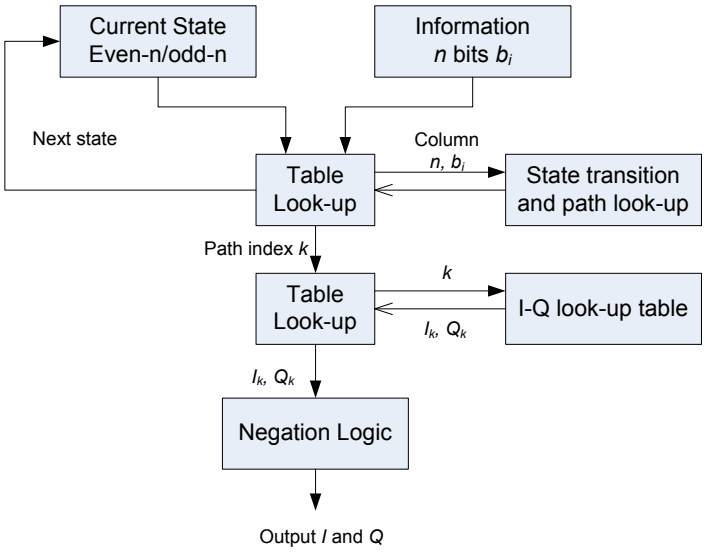

Figure 2.3 – GMSK signal generation using lookup table, excerpt from [20]

The design illustrated in Figure 2.3 has an advantage over other methods, because it uses the least amount of computational cost and uses less memory as opposed to the other two methods and is considered a practical choice for mobile phones.

One important factor is that all three methods produce the same GMSK signal.

After an analysis of the three methods, it was decided that the simplest method to simulate in Matlab was the Quadrature base-band method illustrated in Figure 2.2.

2.4 THE GAUSSIAN MINIMUM SHIFT KEYING RECEIVER

Coherent demodulation in GMSK systems is quite popular. *Coherent demodulation* means the process of recovering the original signal from a modulated signal using a synchronized oscillator; this is also called *synchronous detection* [13, 14]. Coherent detection requires both frequency and phase synchronization and is usually implemented by using *phase-locked loops*.

Non-coherent demodulation techniques do not require knowledge of the reference phase, eliminating the need for phase-lock loops, local oscillators, and carrier recovery circuits.

According to [8] Coherent demodulation leads to a better Bit error rate in the presence of AWGN, but results in irreducible error rates for fast fading Rayleigh type mobile channels.

The next section will discuss some common types of coherent and non-coherent demodulator designs.

2.4.1 GMSK Demodulator Design

Figure 2.4 is an illustration from Jeffrey Laster's PhD thesis [8] which shows the structure of a common coherent demodulator used in GMSK. This type of demodulation is closely linked with the Quadrature MSK demodulation, which uses two parallel Quadrature and In-phase channels and a Local Oscillator locked on to the carrier frequency, r(t) is the received modulated GMSK signal.

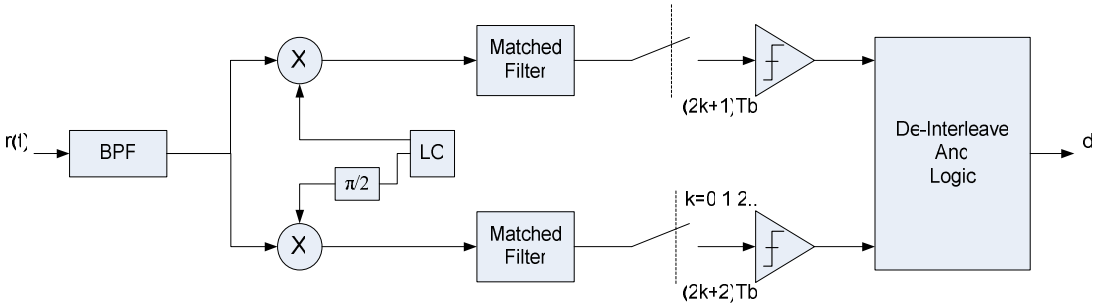

Figure 2.4 – General block diagram of a coherent GMSK Demodulator, excerpt from [9]

Another coherent GMSK demodulation block diagram given in [4] is shown in Figure 2.5. This method of detection also separates the received signal r(t) into Quadrature and In-phase components and performs phase measurement calculation, showed by the arctan(I/Q) block. The phase of the signal is then differentiated, and this results in positive and negative signal values. These values are then down-sampled and sent to the decision maker whose output results in NRZ bits. This form of demodulation is closely linked to FM demodulation, which is theoretically correct because GMSK is a type of frequency modulation as explained at the start of this chapter.

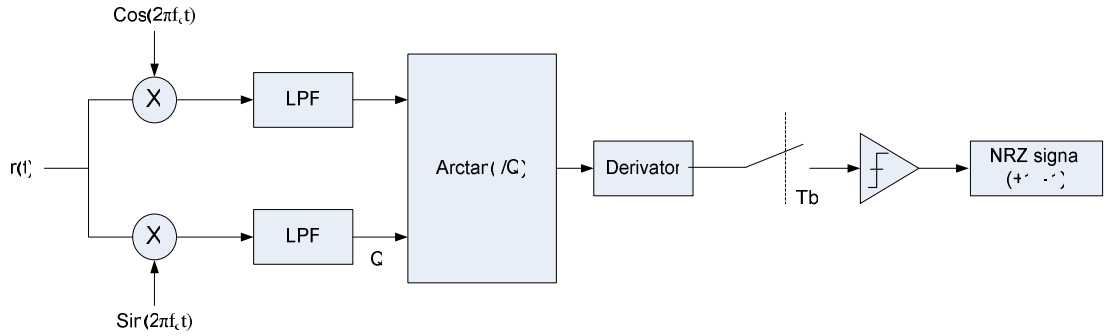

Figure 2.5 – Coherent GMSK Demodulation, excerpt from [4]

Many non-coherent demodulation methods have been discussed in [8], one of them is the *Discriminator* type demodulator, shown in Figure 2.6. A discriminator is a common type of FM detector circuit which converts frequency changes in a signal to amplitude changes. The *Limiter* block is used to replace the constant envelope property to the received corrupted signal.

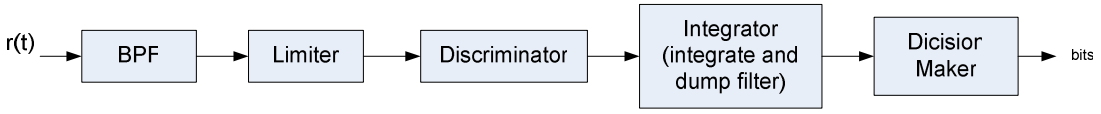

Figure 2.6 – Non-coherent, Discriminator Type Demodulator

Another type of non-coherent demodulator is illustrated in Figure 2.7. This type of demodulator is called a Differential receiver and is extensively discussed in [8] and [9]. It compares the phase of the received signal at two different time intervals, usually multiples of the bit duration, and makes a decision based upon the two values.

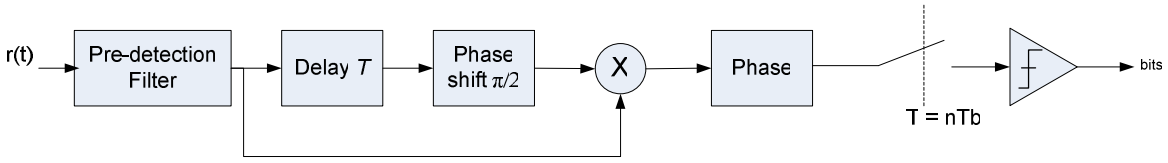

Figure 2.7 – Non-coherent Differential receiver, excerpt from [11]

Other types of non-coherent demodulators include the *Direct Conversion demodulator* which is discussed in [8] and the *Maximum-Likelihood Sequence Estimation (MLSE)* using trellis diagrams or Viterbi detection discussed in [17].

After researching into the different types of GMSK demodulators mentioned above, the design illustrated in Figure 2.5, the model by [4] was chosen for my simulation. It may not be the most practical method of demodulating GMSK signals, but compared to the rest of the methods, it seemed the easiest to implement in a simulation.

2.5 SOFTWARE DESIGN TECHNIQUES AND TERMINOLOGY

Optimising the Simulation Execution Speed:

A few software design techniques were taken into consideration when writing the matlab code.

Code Optimisation:

Matlab code was vectorized when ever possible to improve speed of execution. Code iteration using 'for' and 'while' loops were avoided as much as possible.

Consider the following code example:

```
for i = 1:1:10
      y(i) = 2^i;
end
```

The above code can be replaced by a much faster code:

```
i = 1:1:10;
y = 2.^i;
```

Even though this may seem trivial, matlab performs faster when using matrices instead of arrays.

Pre-allocation:
Matrices used are pre-allocated whenever possible. This proved much faster when resizing matrices. The code below pre-allocates a matrix with zeros.

```
x = zeros(1,1000);
```

Usage of functions:

In Figure 2.8 the GMSK modem simulation is shown as divided into separate functions, and called by the 'main' m-file, without using one contiguous chunk of matlab code. Use of functions makes it easier to debug. Also according to the Matlab User guide, it proves faster in execution, because Functions are compiled into pseudo-code and loaded into memory once. Therefore additional calls to a function are faster.

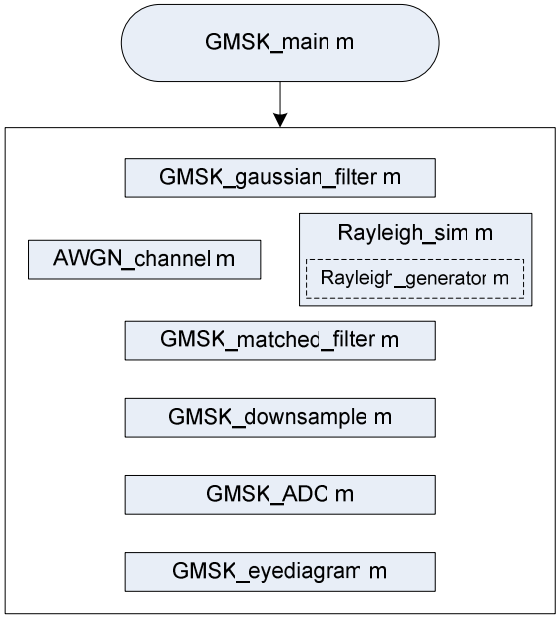

Figure 2.8 – GMSK simulation block diagram

Important Matlab Terminology:

Scalar
Any individual real or complex number is represented in Matlab as a 1-by-1 matrix called a scalar value.

Vectors
Matlab uses 'Vectors' instead of arrays. Vectors are similar to arrays, it is a collection of scalars but their first element has an index of 1, where as the first element of an array has an index of '0'.

Matrix
A Multidimensional Vector has the same concept of a Multidimensional Array, but in Matlab they are termed 'Matrices'. Matrices can be a mixture of any data type (i.e. integer, character, double, etc).

Scripts
Matlab lets its users to write scripts and for these scripts to be executed as programs. Scripts are not compiled, they are only interpreted. 'Functions' are also basically scripts which are called by a main script.

Workspace
The workspace is part of the environment in Matlab, in which variables that are being used are stored temporarily. Users of Matlab can view the vectors and other variables being used in their scripts, they can view the different attributes in the workspace variables as well (i.e. size, length, data type etc.)

2.6 GAUSSIAN MINIMUM SHIFT KEYING MODEM SIMULATION

This section of the report describes the simulation procedure of the GMSK modulator and demodulator. Appendix [B] of this report contains the source code used for the simulation. Detailed description of each custom made function, its input parameters and its outputs, are all described inside the code, as comments.

2.6.1 GMSK Modulator Implementation

Bit Stream Generation

The first step in the simulation is to generate a stream of bits. Bipolar signalling will be used, because it offers a much more tolerance to noise than the unipolar counterpart. The following parameters were used for the bit stream:

- Bit duration $T_b = 1$ second.
- Samples per bit = 36. (i.e. sampling rate = fs = 36Hz);
- Type of coding = Bipolar Non-Return to Zero (NRZ)(i.e. ±1)

The Matlab function '`randsrc`' was used to generate bits. The '`kron`' function was used to up-sample the bits. The '`randsrc`' function generates a m-by-n matrix of random -1's and 1's with equal probability. The '`kron`' function performs a 'Kronecker Product' between two matrices. The following example shows how it is used to sample the bits. (⊗ - denotes a 'Kronecker Product')

$$[-1\ -1\ 1\ -1] \otimes [1\ 1] = [-1\ -1\ \ -1\ -1\ \ 1\ 1\ \ -1\ -1]$$

The following two lines of code, generates the random NRZ bits and samples them by 36. Lines following the '%' symbol indicate a comment in the matlab script.

```
      m = randsrc(1,10000); % produces 10000 random -1's and 1's.
      rect = kron(m,ones(1,36)); % use the kron function to up-sample the
bits.
```

As a result, '`rect`' will be a 1x36000 matrix which is made up of -1s and 1s sampled 36 times. For the purpose of clearly illustrating the GMSK characteristics, a fixed bit stream will be considered.

The bit stream that will be used for the demonstration is:

m = [-1 -1 -1 1 1 -1 1 1 1 1 -1 1 -1 1 -1 -1]

Figure 2.9(a) and Figure 2.9(b) shows the plot of the fixed bipolar NRZ data before and after sampling.

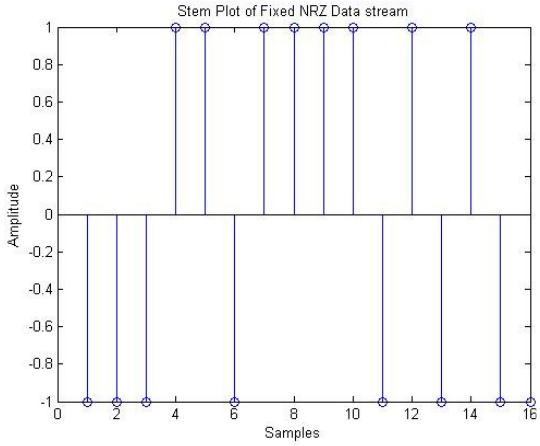

Figure 2.9(a)
Bipolar NRZ data , before sampling

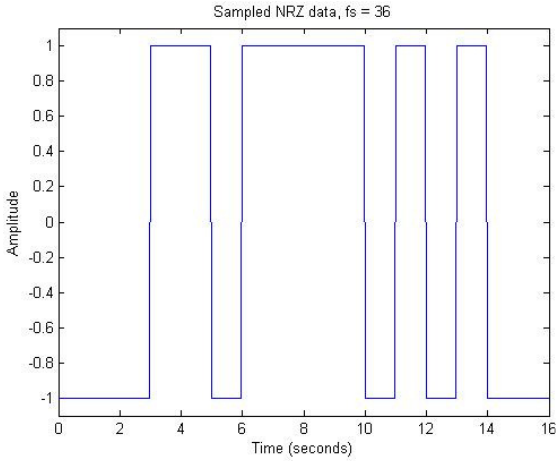

Figure 2.9(b)
Bipolar NRZ data, after sampling

Pulse Shaping

Next the data is passed through a Gaussian Pulse shaping filter. The equation given in [11] will be used for the Gaussian Filter which combines the 'δ' term in (1.2) into the expression (1.1), yielding the same result. One of the key requirements that the filter had to satisfy was to ensure a phase change of $\pi/2$ for every bit change. This was implemented by scaling the filter impulse response coefficients in the following manner:

```
% Definition of the Gaussian Filter, BT = 0.3
 Tb = 1; B = 0.3/Tb;
 t = -1.5:Tb/samples:1.5;
 h = (B.*sqrt((2*pi)/log(2))).*exp((-1*(((2*pi^2)*(B.^2))).*t.^2)./log(2));


% Filter scaling - to ensure phase change of pi/2.
→ K = pi/2/sum(h);
   gfilter = K*h;
   gfilter = gfilter./sqrt(sum(gfilter)); %normalize for unit gain.
```

In the above piece of code, the filter coefficients for the Gaussian filter with BT=0.3 are held in the vector '`h`' and the scale factor is defined by the integer '`K`'. The final resultant scaled down coefficients are held in the vector '`gfilter`'.

Figures 2.10(a) and 2.10(b) illustrate the Gaussian filter for a range of BT values in both Time and Frequency domain.

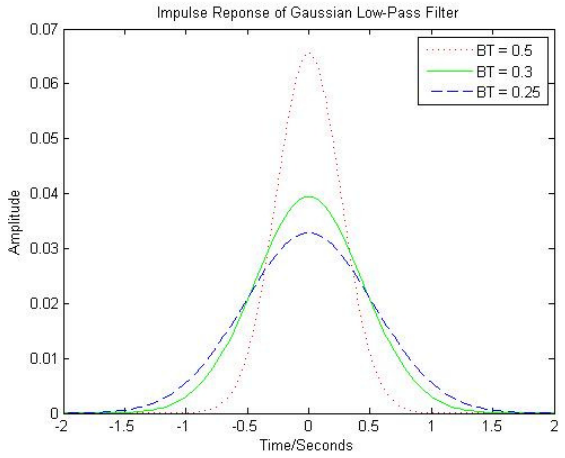

Figure 2.10(a) – Gaussian LPF in Time domain

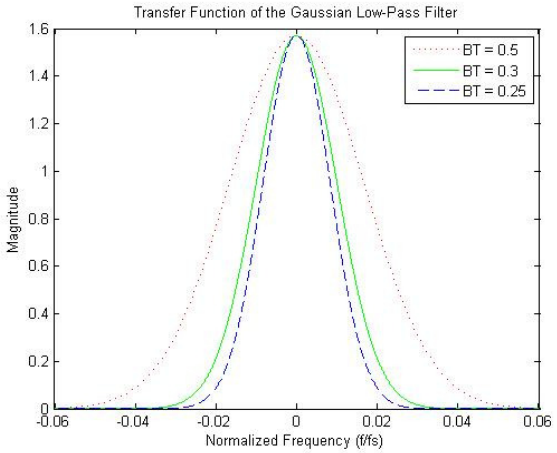

Figure 2.10(b) – Gaussian LPF in Frequency domain

Figure 2.10(a) shows the time domain representation of the Filter. Negative time values have been shown here for illustration purposes only. It can be seen the filter takes the form of a Gaussian bell shape. The filter has no zero crossings, even though the plot shows the impulse response between -2 and 2 seconds.

In time domain, it can be seen that, as the BT product value increases, the width of the filter, in time domain decreases.

A Frequency domain representation shown in Figure 2.10(b) was obtained in matlab, using the piece of code below:

```
N=2^16;
nu=[-0.5:1/N:0.5-1/N];
h = abs(fftshift(fft(gfilter_1,N)));
```

The '`fft`' command performs a Fast Fourier Transform using N number of Discrete Fourier Transform points. The '`fftshift`' command is used to obtain both the sidebands of the filter, while the 'abs' command is used to obtain the magnitude response. In frequency domain, the filter bandwidth is directly proportional to the BT value.

It was stated in [20] website that the length of the Gaussian filter with BT=0.3 must spread over at least three bit periods.

Taking into consideration that the bit duration is 1 second and 36 samples per second, the filter was designed to have 109 samples (55^{th} sample represents the peak of the filter) and to stretch from -1.5 seconds to 1.5 seconds as shown in the piece of code before. Figure 2.11 displays the impulse response of the filter, with the x-axis showing the number of samples.

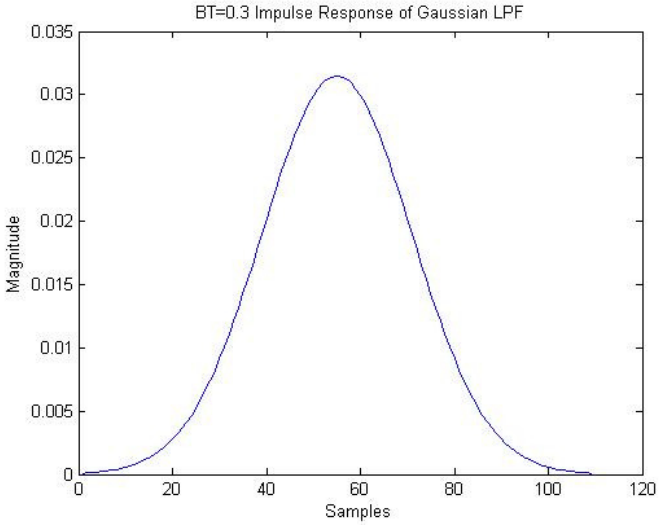

Figure 2.11 - Gaussian Low Pass Filter (x-axis displays samples)

The filtered data can be seen in Figure 2.12. There are amplitude fluctuations visible; this is due to the overlapping of other filtered symbols. At these points, the amplitudes from other filtered symbols add up and result in decreases and increases in amplitudes. This kind of occurrence results in Inter-Symbol-Interference introduced into the system, which will be discussed in detail later in this chapter. The x-axis in the Figure 2.12 represents samples. And as it can be seen, the total amount of samples in the filtered data is now 685.

Note: The Preliminary analysis was chosen to be performed in samples instead of time domain, as it made it easier to write the code, to debug and to understand the results in matlab plots.

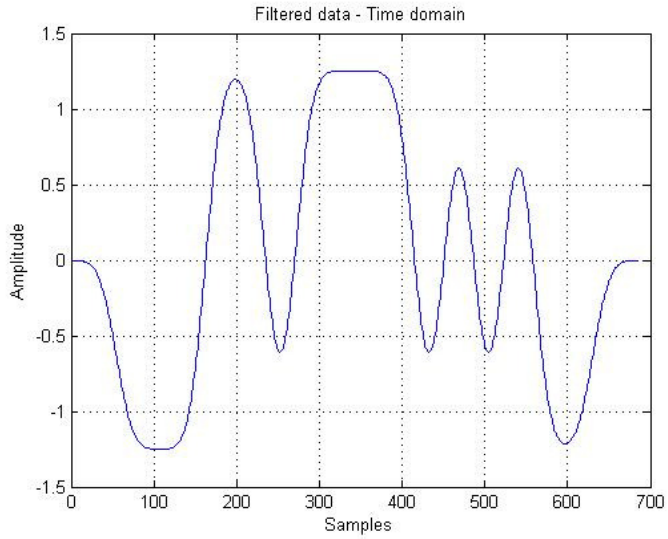

Figure 2.12 – Gaussian Filtered NRZ data using the 'conv' command

The filtering of the Gaussian filter and the data was performed using convolution. As it is well known, to filter a signal in time domain, you have to convolve the impulse response of the signal and the filter together and respectively in frequency domain you have to multiply the transfer function of the signal and the filter. It is known that after convolution,

$$\text{Length of Filtered Data} = \text{Length of Signal} + \text{Length of Filter} \qquad (1.8)$$

The same theory applies in matlab when using the 'conv' function. The use of this function can be seen below:

```
% create gaussian low pass filter(defined in the gaussian_filter.m)
    gaussfilter = GMSK_gaussian_filter(Tb,samples);

% pass message signal through Gaussian LPF
    m_filtered = conv(gaussfilter,rect);
```

where, '`gaussfilter`' holds the filter coefficients and 'rect' holds the sampled NRZ bits. A detailed explanation of the function '`GMSK_gauss_filter`' can be found in *Appendix C1*. The matrix '`m_filtered`' holds the filtered data and the its plot is shown in Figure 2.12.

Another matlab command exists to filter data with a digital filter; this command is 'filter'. It's use is shown below :

```
% pass message signal through Gaussian LPF
    m_filtered = filter(gaussfilter,1,rect);
```

The parameter '1' denotes the denominator coefficients, and since the Gaussian filter is a Finite Impulse Response (FIR) the denominator coefficient is set to 1, and the numerator coefficients are held by the vector '`gaussfilter`'. This command however did not give valid results, as it lost the last two bits in the data sequence after filtering. One solution to this problem would have been to shift the filter to the right before filtering, but it was chosen to use the '`conv`' command instead. As Figure 2.13 shows, one advantage in using the '`filter`' command is that, it does not result with a matrix longer than the filtered data.

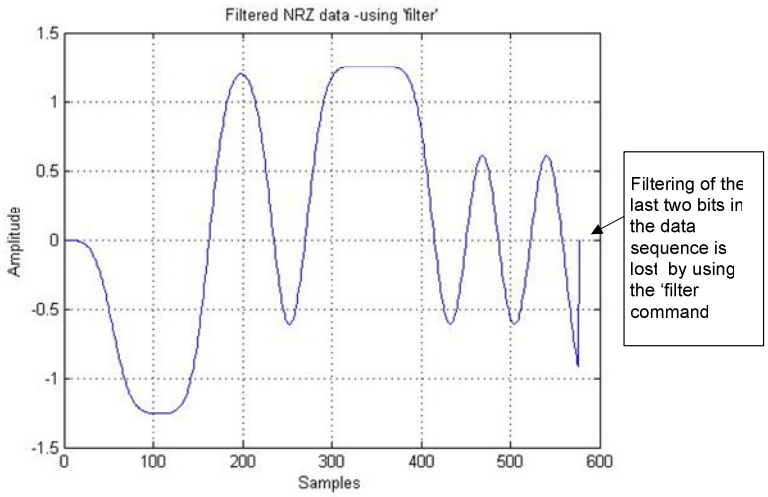

Figure 2.13 – Filtered data using the 'filter' command

Integration

The filtered data is then integrated to obtain the area below the signal. The matlab function '`cumsum`' was used to perform the integration. The '`cumsum`' function basically performs a cumulative summation of the input vector. The piece of code below performs the same operation of the '`cumsum`' function.

```
% hand coded 'cumsum'
 msg_int =[]; % empty matrix to store result
 for k = 1:length(m_filtered)
     msg_int(k) = sum(m_filtered(1:k)); % take sum up to every k samples.
 end
```

At each execution of the loop shown above, the summation of the signal is taken up to the k^{th} sample, where the integer '`k`' increases from 1 to length of the filtered data vector.

However it was easier to use the '`cumsum`' function, after understanding its functionality. Figure 2.14 displays the plot of the filtered and integrated data; this was the exact same plot obtained by running the above script.

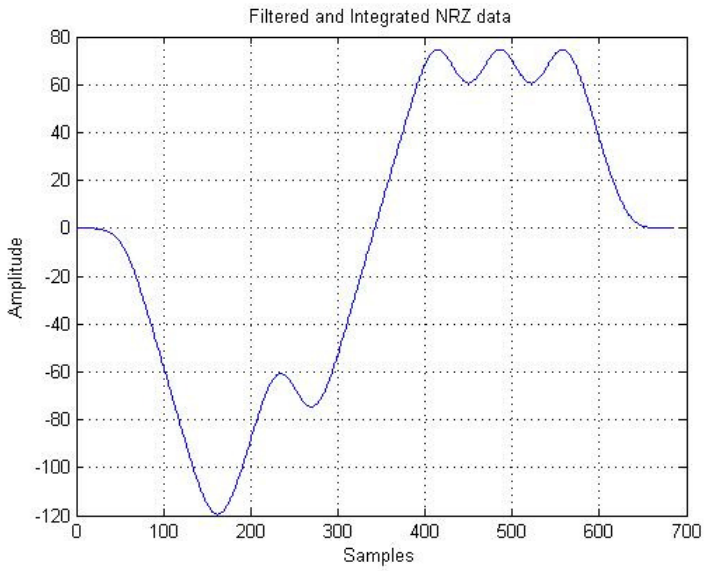

Figure 2.14 – Integration of the filtered NRZ data, using the 'cumsum' function

In-Phase and Quadrature Channel Separation

As Figure 2.2 illustrates the final step in the GMSK transmitter is splitting the In-Phase (I) and Quadrature (Q) Channels of the Signal. Since this is a Baseband simulation, there will be no carrier introduced. The I and Q channels are separated as follows:

```
m_filtered2_real = cos(m_filtered1);     % In-Phase Channel
m_filtered2_imag = sin(m_filtered1);     % Quadrature Channel
```

The '`sin`' and '`cos`' functions are built-in matlab functions. The '`cos`' and '`sin`' functions in this instance is used to calculate the cosine and sine of every value in the matrix '`m_filtered1`' ('m_filtered1' is the matrix which contains the filtered and integrated GMSK signal).

The resulting In-phase channel is then saved in the matrix '`m_filtered2_real`' as the Real component of the complex signal and likewise the Quadrature channel is saved in '`m_filtered2_imag`' as the Imaginary component of the signal. A plot of the In-phase and Quadrature channels can be seen in Figure 2.15(a) and Figure 2.15(b).

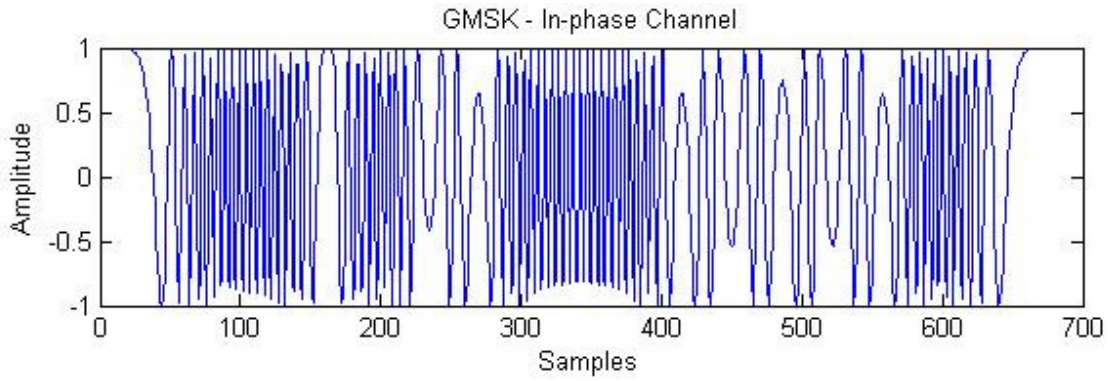

Figure 2.15(a) – GMSK In-Phase Channel (Baseband)

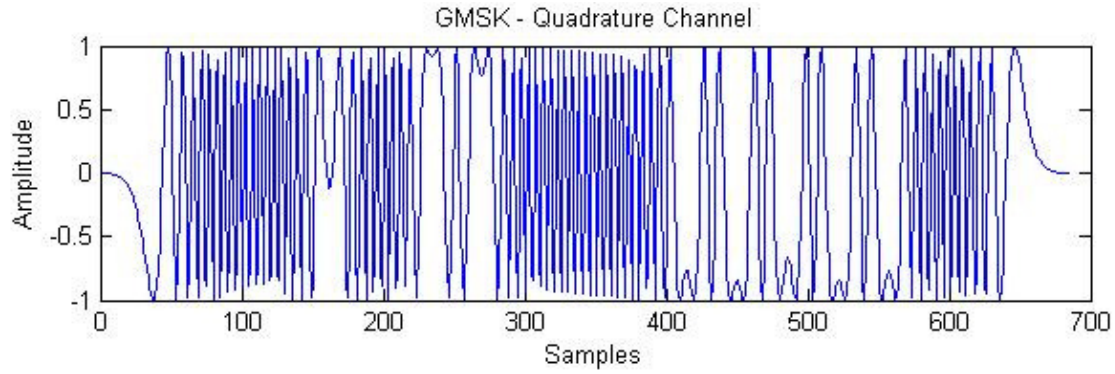

Figure 2.15(b) – GMSK Quadrature Channel (Baseband)

It is interesting to see the relationship of the In-Phase and Quadrature channels of a complex signal by these two plots. Where there is high amplitude in the Real component of the signal, there is low amplitude in the Imaginary component. Also since GMSK is a continuous phase modulation technique, it is visible that there are no sudden phase discontinuities in the I and Q signals.

To further investigate the relationship between the I and Q components of the GMSK signal, a plot of the '`m_filtered2`' matrix was generated. This matrix contains the complex vectors of the GMSK signal. When plotting a complex vector using the matlab command '`plot`' the plot function operates as, '`plot(real(Y),imag(Y))`' meaning the it plots the real component against the imaginary component. Figure 2.16(a) shows the complex GMSK Baseband signal being plotted using the 'plot' function.

Figure 2.16(b) shows the 3D representation of the 2D plot shown in Figure 2.16(a). Figure 2.16(b) helped to understand why the GMSK signal 2D plot was shaped in such a way. The 3D plot was obtained using the 'plot3' function in matlab, whose input parameters are as follows:

```
plot3(m_filtered2_real,m_filtered2_imag,(1:1:length(m_filtered2)));
```

Rea Component — Imaginary Component — Samples

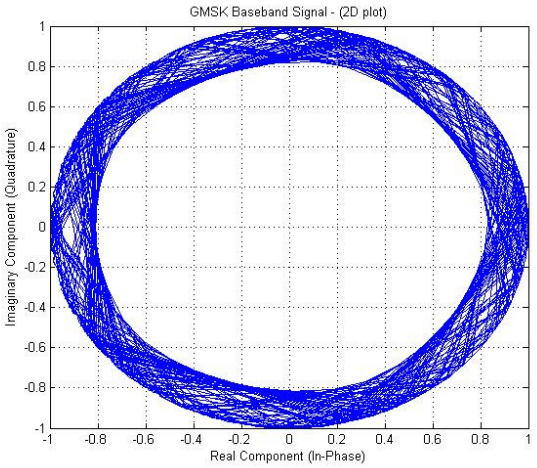

Figure 2.16(a) – GMSK Baseband 2D plot

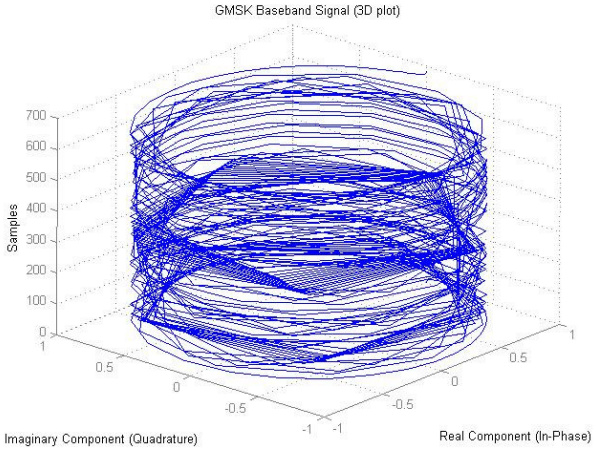

Figure 2.16(b) – GMSK Baseband 3D plot

It can be seen from Figure 2.16(a) that the lines do not cross the centre of the plot. This is mainly because, GMSK is a continuous phase modulation scheme, and therefore it reduces the sudden $\pm 90^{o}$ phase jumps from symbol to symbol which can be seen in other modulation schemes such as QPSK or BPSK.

This can be further clearly seen from the GMSK constellation diagram, which was plotted using the '`scatterplot`' function. The '`scatterplot`' function performs a similar operation to the 'plot' function when given a complex matrix, except it does not join each individual point with a line. It plots a "•" for each point, taking the Real component as the X-Coordinate and the Imaginary component as the Y-Coordinate.

Figure 2.16(c) shows a '`scatterplot`' for the GMSK signal ('m_filtered2'). There appeared to be breaks in the circular shape at certain points shown in Figure 2.16(c). This was because of the small amount of bits used in the simulation (16 NRZ bits). The amount of bits was

increased to 10000 and another '`scatterplot`' was obtained and can be seen in Figure 2.16(d). Figure 2.16(d) appeared to be clearer in the definition of the GMSK signal.

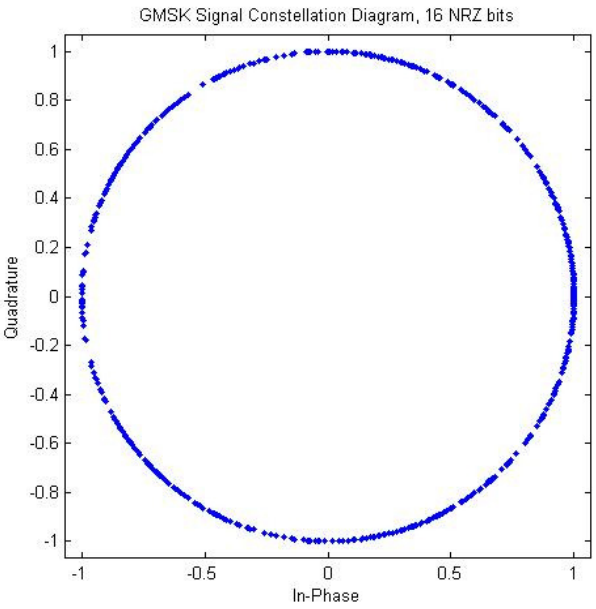

Figure 2.16(c)
GMSK Signal Constellation, using `Scatterplot`
(for 16 bits)

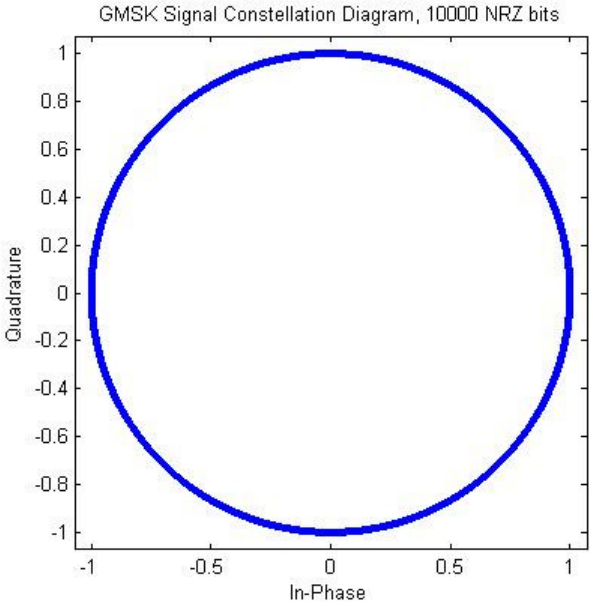

Figure 2.16(d)
GMSK Signal Constellation, using `Sacatterplot`
(for 10,000 bits)

As seen from Figures 2.16(c) and 2.16(d) the constellation diagram for a GMSK signal resembles a circle at constant amplitude around the origin. And the key measurement for this type of modulation is phase measurement, as the amplitude does not change. This type of constellation diagram can also be seen in other Frequency Modulation type schemes. If there is an amplitude change in the system, this circle will change into an elliptical shape.

Successful implementation of GMSK Modulation is verified by the above constellation diagrams and, I and Q channel plots shown in Figures 2.15(a) and 2.15(b).

2.6.2 GMSK Demodulator Implementation

The demodulation of the GMSK signal was the most demanding of the tasks in the entire project. Figure 2.5 in Section 2.4 of this report illustrates the basic Coherent method of demodulating GMSK signals, introduced by [4]. This method of demodulation by [4] was identified as frequency demodulation. And in general terms the process can be broken down as follows:

1. Filter the received signal using a Low Pass Matched Filter. Using the same definition as the Gaussian Low Pass Filter at the transmitting end. (perform convolution)
2. Calculate the Phase of each sample of the signal. The GMSK signal at this point is a set of complex samples. The angle or phase of each complex vector should be obtained.
3. Perform differentiation of the calculated phase values. (opposite of the integration performed at the transmitter)
4. The differentiated signal is still in continuous time; therefore it must be sampled appropriately to convert to discrete time as before.
5. Perform a decision checking routine to obtain (1's and -1's from the discrete samples)

Matched Filtering

Matched filtering is performed to maximise the signal-to-noise ratio at the receiver in the presence of Additive White Gaussian Noise.

A Low pass Gaussian filter, similar to the one used at the modulator will be used at this point as the matched filter. Initial simulations were performed using the same BT value (BT = 0.3) as the pre-modulation filter. This however did not give accurate results (excessive bits in error at the output when AWGN was introduced). After a couple of simulation runs it was only logical to briefly analyse the filtering process in frequency domain. This was to ensure that the low pass filter which was being used had a correct cut-off frequency. More details on this investigation are included in *Section 2.6.3*.

A few but significant changes were made to the initial design of the matched filter:

- Sampling rate used in the pre-modulation filter was 36Hz. Sampling rate used for the Matched Filter is **7Hz**. High sampling rate was creating excessive bits in error at the output of the demodulator. Also the shape of the filtered I and Q channels were quite different to the transmitted I and Q channels, in the presence of zero AWGN. This was the main reason to lower the sampling rate.

- The Gaussian Matched Filter has **BT = 0.75**. This makes the Filter wider in frequency. Section 2.6.3 shows a frequency domain analysis showing a Gaussian matched filter with BT=0.3 is too narrow, and results in loss of some higher frequency components.

- The above two parameter changes to the filter, makes it loose its Gaussian shape in time domain, but retains the Gaussian shape in frequency domain (Figure 3.7(b))

The original design of the Matched filter was exactly the filter used for at the modulator, and is shown in Figure 2.11. A plot of the Matched filter which will be used for the simulation is shown in Figure 2.17(a) and Figure 2.17(b). The Matched filter will have 22 samples in total meaning the received signal will be 22 samples longer after convolution.

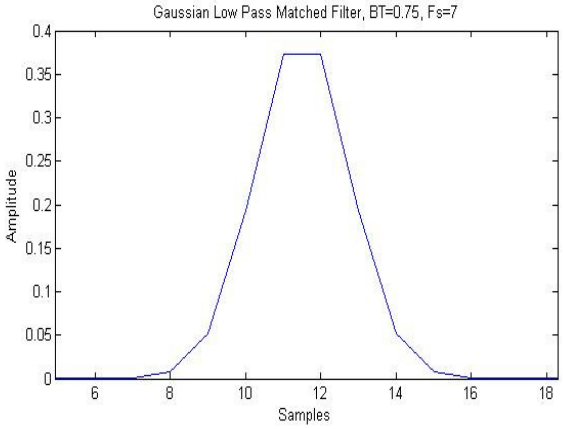

Figure 2.17(a)
Gaussian Matched Filter (BT=0.75, Fs=7)

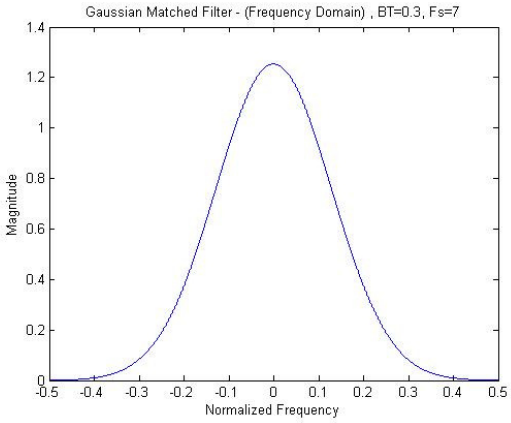

Figure 2.17(b)
Gaussian Matched Filter (BT=0.3, Fs=36), Frequency Domain

The matlab code that was used for the matched filter was placed inside a different function '`GMSK_matched_filter()`' to separate the matched filter functionality from the pre-modulation filter, and to be able to change the code when necessary.

The I and Q channels were separately passed through the matched filter. The following two lines perform the filtering, '`filt_noisy_real`' and '`filt_noisy_imag`' will contain the resulting filtered I and Q channels respectively:

```
filt_noisy_real = conv(matchfilter,noisy_real);
filt_noisy_imag = conv(matchfilter,noisy_imag);
```

Figure 2.18 illustrates what happens to the I and Q GMSK signals after they are Matched Filtered using a Gaussian Filter of BT=0.3.

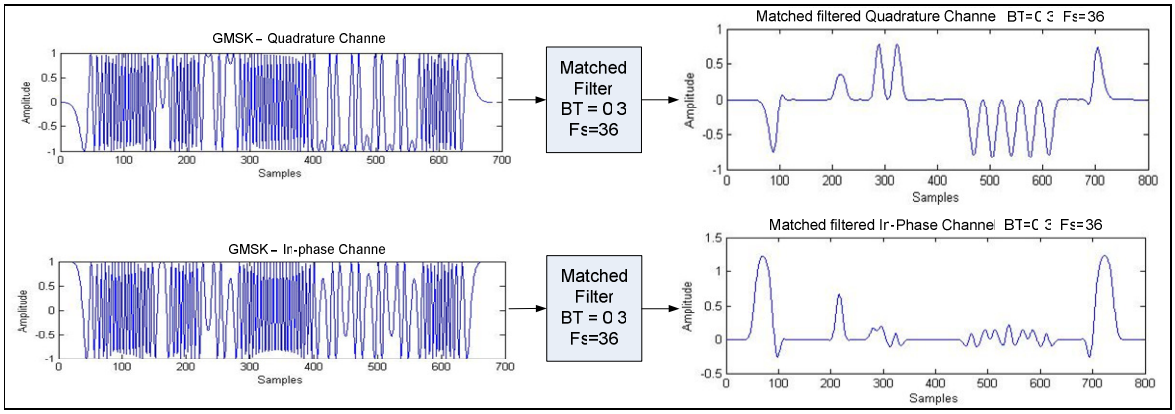

Figure 2.18 – Effect of Matched Filtering (using Gaussian Filter, BT=0.3, Fs = 36)

The matched filter has altered the shape of the time domain representation of the GMSK signal. This in turn would affect the decision making process in the demodulator because phase information in the signal is lost.

The above Matched filtering was performed using a Gaussian Filter of BT = 0.75 (i.e. higher cut-off frequency, larger bandwidth). The results obtained were much better than the previous case when a BT product of 0.3 was used. The results can be seen in Figure 2.19.

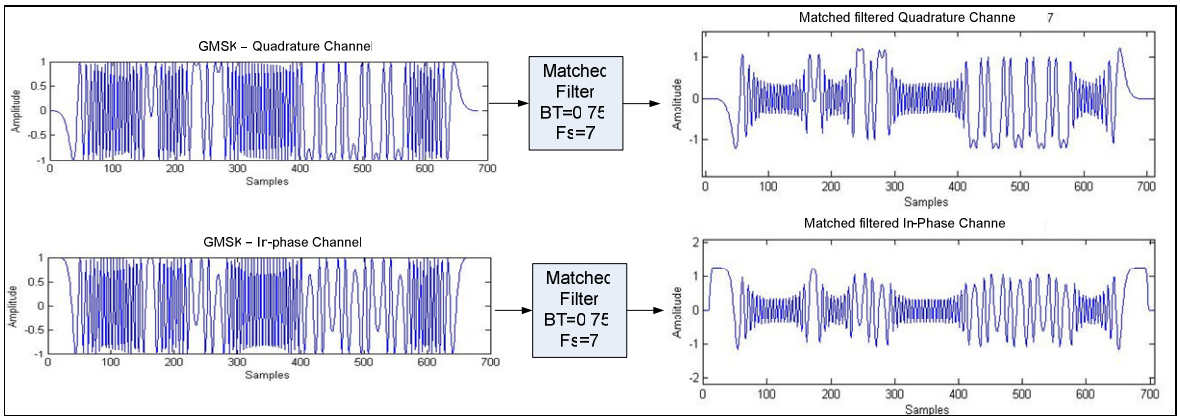

Figure 2.19 - Effect of Matched Filtering (using Gaussian Filter, BT=0.75, Fs=7)

It was shown in [6] that the optimum choice for a Gaussian receiver filter is BT=0.3, but in [7] it is shown the ideal pre-detection filter has BT=0.63. However according to the facts at hand, it was decided to use the Gaussian Filter with BT=0.75 for the simulations.

Phase Calculation of the Received Signal

The next step in Demodulation is to obtain the phase of each sample of the signal. Two important Matlab functions will be used for this task.

The '`angle`' function was used in matlab to obtain the phase of a complex vector. The function returns the angle of the complex vector in radians, and the angle will lie between $\pm\pi$. The '`unwrap`' function is used to correct the phase angles by adding multiples of $\pm 2\pi$.It is used to ensure that as a complex number travels smoothly counter clockwise around the complex plane, rather than having the phase suddenly jump from 179.9° to -179.9° it is unwrapped to go from 179.9° to 180.1°.

The two functions are used together as follows:

```
phase = unwrap(angle(filt_noisy_real+filt_noisy_imag*j));
```

Where '`filt_noisy_real+filt_noisy_imag*j`' was used to create the complex vector from the real and imaginary components of the received GMSK signal. A plot of the retrieved

phase of the signal can be seen in Figure 2.20. This plot is identical to Figure 2.14 (integrated data at the modulator), in terms of amplitude and shape, which was the objective of the phase calculation.

The vector '`phase`' however holds more samples (707 samples) than the matrix used in the modulator to hold the integrated data '`m_filtered1`', this is due to the convolution performed when using the pre-detection filter.

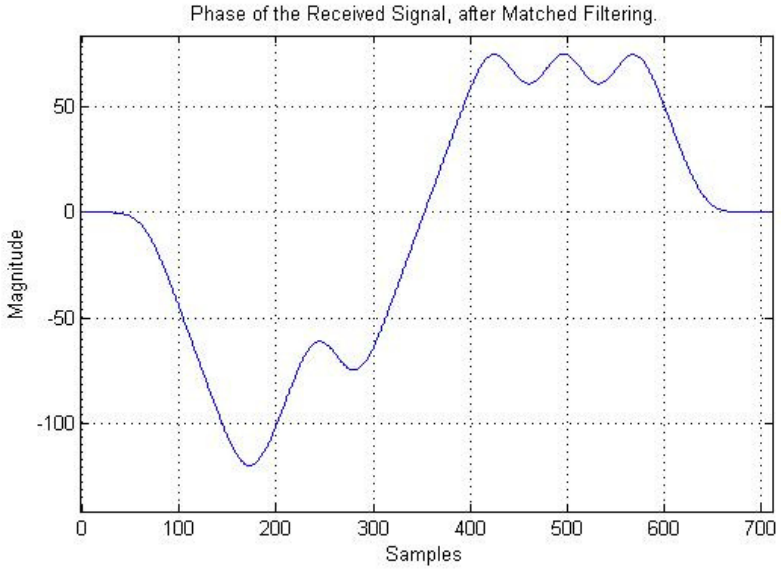

Figure 2.20 – Phase of the Received Signal, after Matched Filtering

Differentiation of the Phase Values

The next step of the demodulator is to differentiate the phase of the received signal. At the modulator, the data was integrated, so it is a logical design to differentiate the received signal at the demodulator. The objective at this point is to obtain the exact data *prior* to integration (just after pulse shaping) at the transmitter/modulator, which is shown in Figure 2.13. Matlab has a built in Differentiation function called '`diff`' which calculates an approximate derivative of the input matrix.
According to the Matlab Documentation ('Help'):

"e.g.: Y = diff(X) calculates differences between adjacent elements of X".

The '`diff`' function was used in the simulation as follows:

```
derivative = diff(phase); % obtain the derivative of the signal.
derivative = [phase(1) derivative]; % add an extra sample to the front.
```

The '`diff`' function returns a vector, which is one element short than the input vector '`phase`'. This was overcome by inserting the first element of the calculated phase values to the front of the '`derivate`' vector. The vector '`derivative`' now holds the differentiated '`phase`' vector.

Figure 2.21 shows a plot of the derivate of the phase.

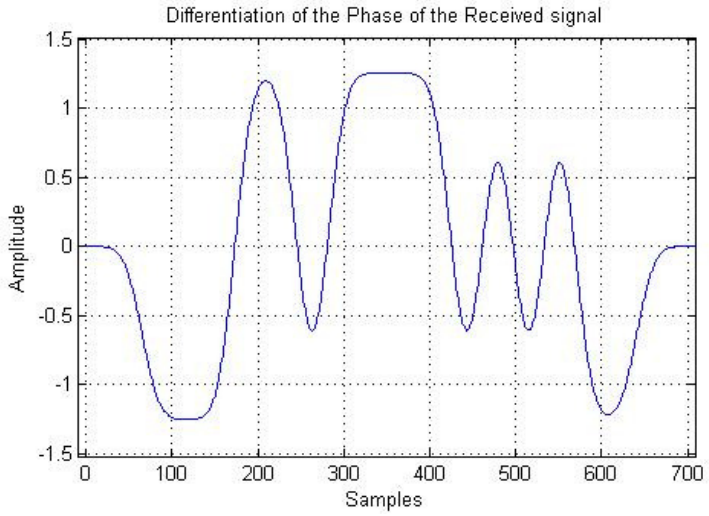

Figure 2.21 – Derivative of the phase, of the received signal

Sampling at the Receiver – Continuous Time to Discrete Time conversion

The sampling is performed on the vector '`diff`' (plotted in Figure 2.21), to convert the continuous time signal into discrete time. The main problem faced at this point is to deal with the extra samples introduced to the signal when convolving with the Pulse shaping Filter and the Matched Filter. The pulse shaping filter is 109 samples long, Matched Filter was 22 samples long, and the vector which was going to be sampled ('`derivative`') is 707 in length.

Therefore;

```
Length of data payload = 707 – (109 + 22) = 576
Original sampling rate = 36 samples per bit
∴ 576/36 = 16 bits (= length of original message)
```

The function written for this purpose is as follows:

```
% Function parameters:
% ---------------------
% start - begin sampling from this sample (leave out(start-1) samples)
% en    - number of samples to leave out at the end
% sps   - down sampling rate
% derivative - input signal to be sampled.

function result = GMSK_downsample(start,en,sps,derivative)
    i = start:sps:length(derivative)-en;
    temp = derivative(i);
    result = temp;
end
```

The vector '`i`' holds sampling points starting from the start integer provided '`start`', and increments with steps of 36 until the end is reached which is given by

'`length(derivative)- en`' (e.g. 707 – 71 = 636). For this task the use of 'for' and 'while' loops were avoided to improve the speed of execution. The temporary vector '`temp`' is used to store these samples before returning them to the main script.

The function was called from the main script as follows:

```
rx_dwnsmpled = GMSK_downsample(70,71,samples,derivative);
```

The initial design was to leave out the first 109 and the last 23 samples and obtain every 36th sample (because original sample rate used was 36 samples per bit) from the vector. The '`end`' integer was chosen to be 23, and not 22 because if 22 was used the resulting samples would be 17 samples long (one extra sample), therefore 23 was chosen as the '`end`' integer, which stops the sampling procedure at the 684th sample. This however proved incorrect because about 50% of bits were in error after sampling and decision checking.

The next design procedure was to leave out the first 66 samples and the last 66 samples and to obtain every 36th sample as before. (66+66 = 109 + 23, equal amount of samples left out from both ends of the vector). This again did not work because about 20% of bits were in error.

My next attempt was to increase the amount of samples being left out from both ends. This was increased to:`[start = 70 and en = 71]`, which gave zero bits in error at the output and also resulted in a much better Bit Error Rate curve, which can be seen in Chapter 4.

Figure 2.22 illustrates the sampling process much clearer, and Figure 2.23 shows a plot of the down sampled received signal; the output of the sampling processes.

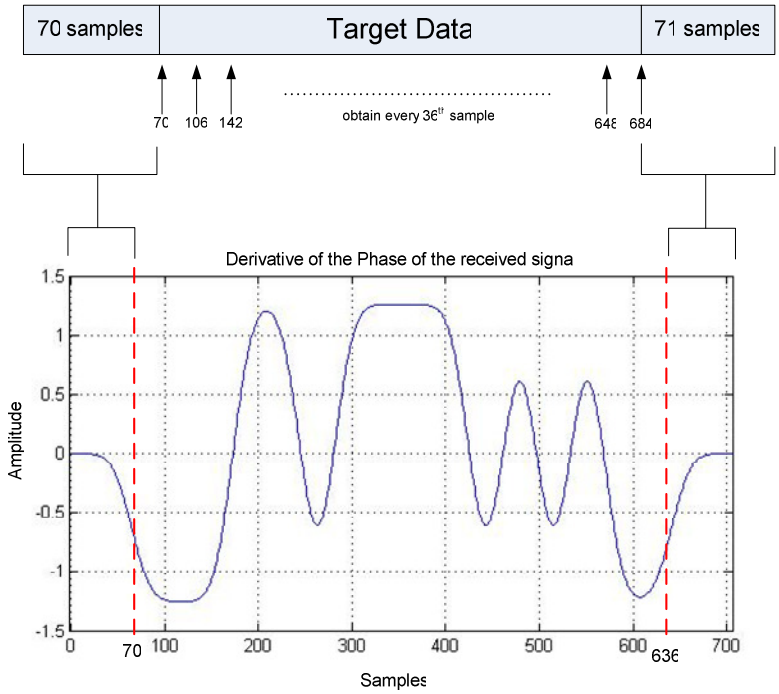

Figure 2.22 – Down sampling process at the demodulator

As Figure 2.22 illustrates, the first 70 samples and the last (707-636=71) samples will be left out and only the middle 566 samples will be used for the sampling process.

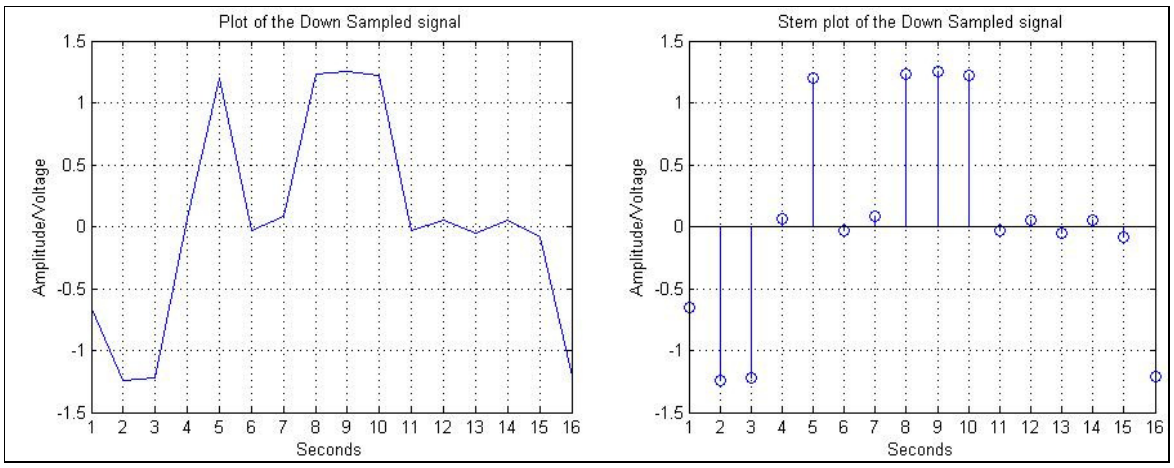

Figure 2.23 – Output of the sampling process

Figure 2.23 shows the output of the sampling process, resulting in 16 different samples (one sample for every second). In realistic terms, this represents different voltage levels (analog), which has to be turned into bipolar data by the decision checking procedure.

Decision Making Procedure – Analog to Digital Converter

The final step in the demodulator is to convert the different voltage levels to Bipolar Data, which should be equal to the NRZ bipolar data transmitted at the modulator. Figure 2.24 shows a flow chart describing this decision checking procedure.

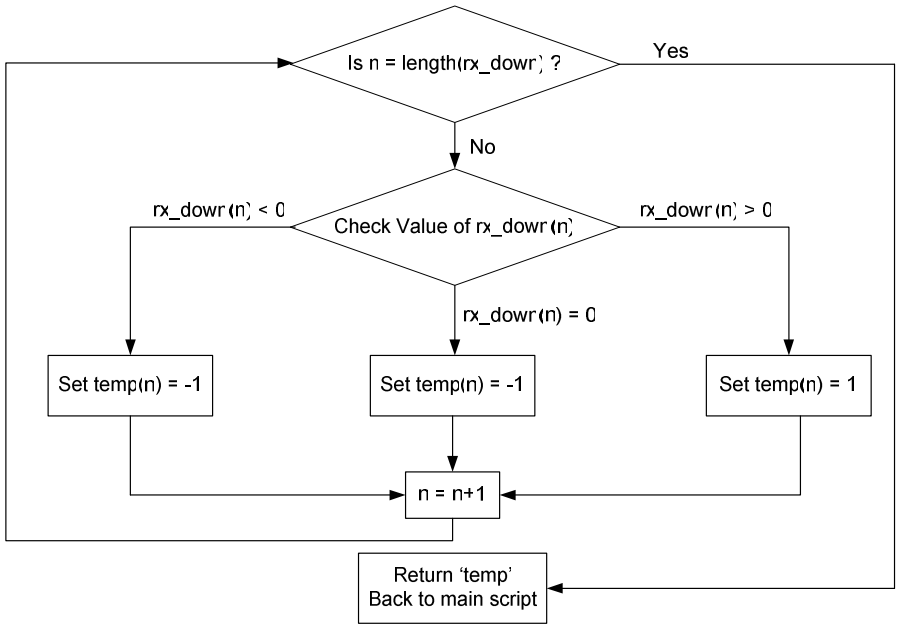

Figure 2.24 – Flow chart of the Decision checking process

A separate function was written for this process, called '`GMSK_ADC`' and can be seen in the *Appendix C1* section of this report. The only parameter that is passed to this function is the vector '`rx_downsampled`' which holds the sampled values.

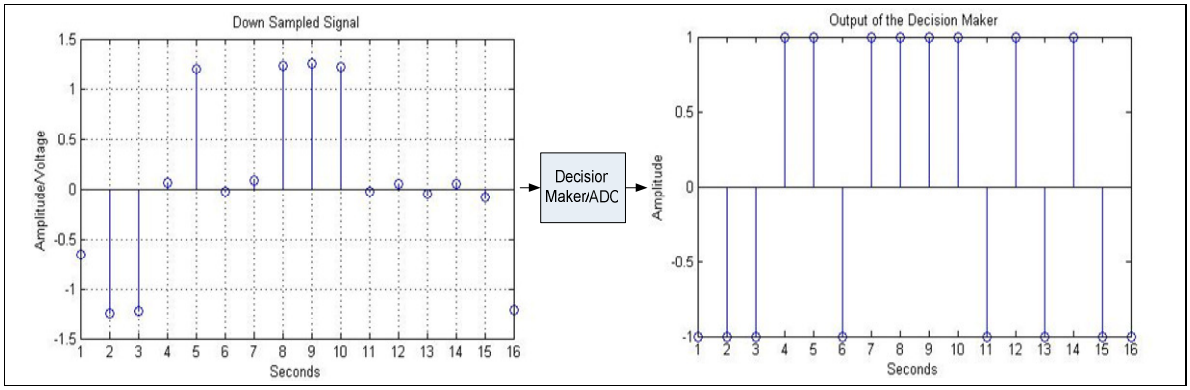

Figure 2.25 – Output of the Decision checking/Analog to digital converter

Figure 2.25 shows a stem plot of the output of the decision making process. The main goal of the demodulator has been completed, because the above plots verify that the original bits have been successfully recovered.

2.6.3 Demodulator Issues

A few problems were faced when simulating the GMSK Demodulator. It was chosen to list them out, and briefly explain steps taken to solve them as logically as possible.

- **Original Matched Filter design was creating bits in error at the output of the demodulator**.

  Conclusion: The initial design of the matched filter was not able to completely retrieve the transmitted signal at the receiver.

  If no noise was introduced to the system, the Low Pass Matched filter should be wide enough able to obtain the original spectrum. As Figure 2.18 showed before, the time domain representation of the transmitted signal was distorted after matched filtering. A spectrum analysis was carried out and its results are displayed in Figure 2.26, in which the x-axis denotes the *Normalized frequency ($\nu = F/Fs$, where Fs=36Hz)*.

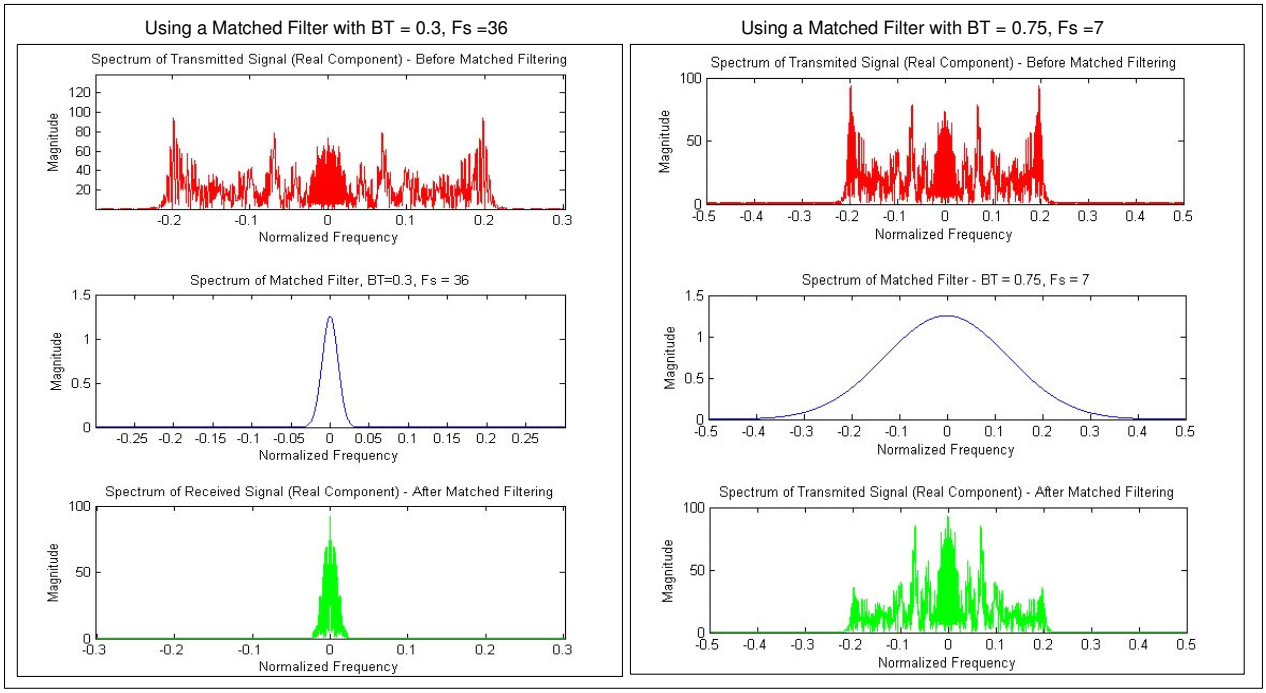

Figure 2.26 – Spectrum analysis (Before & After, using a Matched Filter of BT =0.3)

Solution: It is clear from Figure 2.26, that when a Matched Filter with BT = 0.3 was used, past of the high frequencies are filtered out, resulting in loss of data. A Filter with BT = 0.75 manages to completely obtain the original transmitted spectrum, without any losses. The BT value was increased in steps of 0.2 and the sampling frequency for the Filter was reduced in steps of 2 to obtain an accurate result, until BT = 0.75 and Fs = 7 was reached.

It was hard to resolve as to why the GMSK spectrum had such a wide bandwidth (0.25ν). Spectrum analysis into the modulator verified that when taking the cos( ) and

sin( ) of the GMSK signal to obtain the In-Phase and Quadrature components respectively (shown in Figure 2.15(a) and 2.15(b)), at the modulator, resulted in an expansion (spreading) to the spectrum. This is why the GMSK spectrum was over 0.2ν wide.

There is a major drawback in using a matched filter with a large bandwidth. If there are noise components between 0.2ν and 0.3ν, they will not be completely filtered out, due to the shape of the Matched Filter. This proved to be a disadvantage when the GMSK signal was passed through a fading channel (this can be seen from the results obtained in Chapter 5).

- **Incorrect Phase calculation at the Demodulator.**
  Conclusion: The Matlab function '`angle`' calculates the phase of a complex vector and returns the result within the range of ±π. Angles which were over these ranges were not handled by this function, and resulted in phase jumps from -π to +π and not ±2π. The Matlab function '`atan`' calculates the phase (arctangent) of a complex vector and returns the result within the range of ±π/2. Initial runs of the simulation were carried out using '`atan`' and this proved unsuccessful, because there were many bits in error.

  Solution: After reading the Matlab help and searching through different DSP/Matlab forums and the Matlab File Exchange website, the problem was resolved by the use of the '`unwrap`' function which was used in conjunction with the '`angle`' function to correct the phase jumps from -π to +π.

- **The vector containing the GMSK Matched Filtered signal at the demodulator had a length much longer than the original up sampled message at the modulator.**

  Conclusion: The Gaussian Pulse shaping and Matched filtering which were performed used convolution. The resulting signal, after the convolution has length, given by expression (1.8).

  Solution: The Gaussian Pulse shaping filter's length is 109 samples, and the Matched Filter's length is 22 samples. Therefore when down sampling at the demodulator (109+22 = 131) samples or more had to be left out (chopped off) to obtain the correct length. Figure 2.22 explains how this was performed. When 70 and 71 samples were left out from both ends of the signal, before down sampling, there were no bits in error at the output of the demodulator.

2.6.4 Preliminary Tests without Additive White Gaussian Noise

The preliminary tests on the GMSK simulation were carried out without the introduction of Additive White Gaussian Noise.

The Matlab communication toolbox has a function called '`symerr`'. This function compares two matrices element by element, and returns the amount of elements which are different to each other, as a scalar number and as a ratio.

Example 1:

```
x = [1 2 3 4 5 6 7 8 9 10];
y = [1 2 3 5 6 4 7 8 10 9];
[num,rat] = symerr(x,y)

num =
             5

rat =
        0.5000
```

Example 2:

```
x = [1 -1 1 -1 1 1 1 -1];
y = [1 1 1 -1 1 1 -1 -1];
[num,rat] = symerr(x,y)

num =
             2

rat =
        0.2500
```

Example 2 shows two vectors which contain elements which are either -1 or 1. Here, '`symerr`' is used to obtain the number of elements in error, and to obtain the ratio between the elements in error and the total number of elements. The ratio which is returned by the '`symerr`' function is equal to the Bit Error Rate (BER), when the two input vectors are bits transmitted, and the bits recovered at the demodulator output. The function was used as follows:

```
[num,rat] = symerr(m,rx_digital)
```

Where '`num`' will contain the amount of elements in error, '`rat`' will contain the ratio between the number of elements in error and the total number of elements in the two vectors. '`m`' vector contains the original message (random ±1s) and the vector '`rx_digital`' contains the output of the demodulator.

For the first test, 10 random Bipolar (±1's) were entered into the modulator. This resulted in an error within the '`symerr`' function, which said the input vectors had different lengths. This meant that the Down Sampling process carried out in the demodulator had a bug in it, it was producing more bits than the original message. This problem was discussed in detail in Section 2.6.3, and a method of overcoming this problem was introduced.

For the second test, 50 random Bipolar (±1's) were entered into the modulator. Initially the '`symerr`' function showed that over 50% of the bits were in error. This was due to the incorrect Matched Filtering which was used and was discussed in detail in Section 2.6.3. Figure 2.27 shows a

After changing the matched filter parameters, the simulation was executed again for 100 random bipolar bits. For this second simulation run, there were 30% of bits in error. After some investigation, and checking each subsection of the Matlab scripts, it was experimentally discovered that the phase of the received signal was calculated incorrectly. This problem was explained in Section 2.6.3 and ways of solving it were introduced.

Further tests were performed for 1000 and 10,000 bipolar bits. When simulation runs were attempted for 10^6 = 1,000,000 number of bits Matlab produced the following error:

"`Out of memory. Type HELP MEMORY for your options.`"

Simulations were executed in Computers running Windows XP and 2GB of RAM and over 2GB of virtual memory. Using a sampling factor of 36 samples per bit, would also be liable for this memory problem, because this makes the sampled vector $36*10^6$ in length.

The Matlab help files provide some methods that can be used to overcome this memory problem in Matlab, but it involves using complex memory management functions in Matlab, such as defragmenting the memory, compressing the variables in the workspace etc. which delay the execution time, and these methods may not always prove successful.

After a couple of test simulation runs, it was found out that the maximum amount of bits that can be simulated in Matlab (without using extra memory optimising functions), is 100,000 (10^5). This will be statistically sufficient to get a BER of 10^{-4} (i.e. 1 in 100,000 bits in error).

Multiple simulation runs were produced up to 100,000 bits with zero BER (without introducing noise). This implied that further simulations regarding scenarios with AWGN and Multipath Fading can now be carried out.

CHAPTER 3
8-PHASE SHIFT KEYING MODEM

This Chapter describes the EDGE mobile system and 8-Phase Shift Keying which is the modulation scheme specified for EDGE. The 8PSK transmitter and receiver structures used for EDGE will be looked at, and a description of all the processes in which the 8PSK modem simulation is carried out, will be given.

Simulations were also performed for a traditional 8-PSK Modem, which uses a different type of Filtering to the EDGE system. This was done to investigate the specifications set by the 3GPP and to compare the EDGE modem against a traditional 8PSK modem. Details of which will be given in this chapter.

3.1 ENHANCED DATA RATES FOR GSM EVOLUTION (EDGE)

EDGE is a digital mobile technology with increased speeds compared to GSM. EDGE can be considered as the transition to 3rd Generation mobile communication systems. EDGE offers three times the data rate GSM does, therefore meaning it can accommodate three times more subscribers than GSM or triple the data rate per subscriber. EDGE was introduced into the GSM networks around 2003, initially in North America. EDGE uses similar multiple access methods (TDMA) and the same carrier frequency (200KHz) as GSM but uses a different modulation scheme, therefore certain software and hardware changes will be made to the GSM physical layer. Subscribers have to obtain new Mobile terminals to experience the enhanced services.

The main difference between the GSM and EDGE systems is the modulation scheme used in EDGE, which is *8-Phase Shift Keying* (8PSK). This modulation scheme offers higher data rates over the air interface (812.5 Kbps). This modulation scheme will be discussed in greater detail in a separate chapter of this report, as it is one of the main focuses of this project.

The new modulation scheme (8PSK) will not replace the already existing GMSK modulation scheme but will rather co-exist with it. With 8PSK it is possible to provide higher data rates with somewhat reduced coverage, but GMSK will be used for wide area coverage.

The EDGE subscribers are offered all of the services already provided in GSM, plus the extra wireless data services mentioned below:

- Wireless multimedia (transmission of pictures, video, data, music etc)
- Web based email
- High speed internet access.
- Voice over internet
- Broadcasting
- Document and information sharing.

3.2 THE 8-PHASE SHIFT KEYING TRANSMITTER

When ETSI (European Telecommunications Standards Institute) began work on the EDGE standard, they were looking for a modulation scheme which would increase the raw data rate given by GMSK. This improvement had to be made within the same spectrum, because EDGE and GSM would be using the same spectrum, where the EDGE data slots are mixed with the GSM voice slots. 8PSK was the modulation scheme chosen by ETSI for EDGE. The mobile handset can distinguish between the two signal formats, without additional signalling through a process called 'blind detection'.

8PSK transmits 3 bits per symbol, as seen in the constellation diagram (Figure 3.1), whereas GMSK transmits only 1 bit per symbol. The symbol rate is the normal symbol rate (270.833ksym/s) which corresponds to 3*270.833kbps (i.e. 812.5kbps).

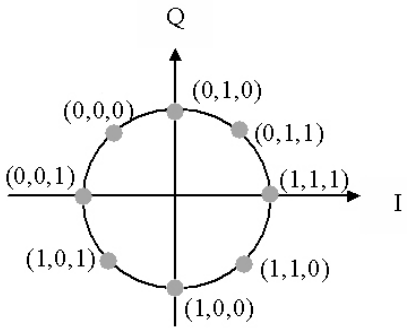

Figure 3.1 - 8PSK symbol constellation diagram

3.2.1 Symbol Mapping and Rotation

According to the 3GPP specification, TS 45.004 v7.0 [1], the modulation format is specified as follows;

First the modulating bits are Gray Coded (adjacent symbols have a single digit differing by 1) and mapped in groups of three to 8PSK symbols, by the rule:

$$s_i = e^{j2\pi l/8} \qquad (1.9)$$

Where the *l* is given specified in Table 3.1:
(Note: when Bipolar signalling is used, the zeros in Table 3.1 will be '-1'):

Modulating bits d_{3i}, d_{3i+1}, d_{3i+2}	Symbol parameter l
(1,1,1)	0
(0,1,1)	1
(0,1,0)	2
(0,0,0)	3
(0,0,1)	4
(1,0,1)	5
(1,0,0)	6
(1,1,0)	7

Table 3.1 – 8PSK Symbol Mapping

In Table 3.1, if Bipolar NRZ signalling is used, the zeros will be represented as a '-1'.
(e.g. 0 1 0 → -1 1 -1)

A problem occurs when the modulating symbols crosses the origin or zero position in the constellation diagram. This affects the power amplifier at the transmitter and receiver ends, where a dynamic range becomes a requirement. To solve this issue, the symbols are rotated continuously by 3π/8 radians per symbol before pulse shaping. This procedure is discussed in detail in the Agilent Technologies' Application note on "EGPRS Test" [3].

The rotated symbols are defined as:

$$\hat{s}_i = s_i \cdot e^{ji3\pi/8} \tag{2.0}$$

This rotation of the symbols results in a 16PSK constellation, which is further, illustrated in section *3.3 Modem Simulation.*

3.2.2 8PSK- Pulse Shaping Filter

Before the Transmitted signal is sent out to the channel, a pulse shaping filter is used to limit the bandwidth.

According to the 3GPP Technical specification Document [1], the pulse shaping filter, used for EDGE the mobile system is a Linearized Gaussian filter, which has an impulse response:

$$c_0(t) = \begin{cases} \prod_{i=0}^{3} S(t+iT), \ for\ 0 \le t \le 5T \\ 0, \ else \end{cases} \tag{2.1}$$

where,

$$S(t) = \begin{cases} \sin(\pi \int_0^t g(t')dt'), \text{ for } 0 \le t \le 4T \\ \sin(\frac{\pi}{2} - \pi \int_0^{t-4T} g(t')dt'), \text{ for } 4T < t \le 8T \\ 0, \text{ else} \end{cases} \tag{2.2}$$

and,

$$g(t) = \frac{1}{2T}\left(Q(2\pi \cdot 0.3 \frac{t-5T/2}{T\sqrt{\log_e(2)}}) - Q(2\pi \cdot 0.3 \frac{t-3T/2}{T\sqrt{\log_e(2)}}) \right) \tag{2.3}$$

where Q is the Q-function defined as,

$$Q(t) = \frac{1}{\sqrt{2\pi}} \int_t^\infty e^{-\frac{\tau^2}{2}} d\tau \tag{2.4}$$

resulting in the base-band signal,

$$y(t') = \sum_i \hat{s}_i \cdot c_0(t' - iT + 2T) \tag{2.5}$$

The method of simulating the pulse shaping filter, using the expressions (2.1) to (2.5) is complicated in lengthy. Therefore the expression founded by *Tropian Inc*, in the paper titled *"Implementation Effects on GSM's EDGE Modulation"* [5] was used in the simulation instead. This approximation for the EDGE pulse shaping filter is valid for all values of *t* and does not involve integrated Q-functions. The impulse response of the approximate filter is shown below:

$$p(t) = \exp\left(-1.045\left(\frac{t}{T}\right)^2 - 0.218\left(\frac{t}{T}\right)^4 \right) \tag{2.6}$$

This approximation is about 0.25% of the peak pulse value, and the RMS error had been measured to be 0.15% by *Tropian Inc*.

This expression will be implemented in the simulation, and will be used as to filter the 8PSK signals.

3.2.3 8PSK Modulator Design

The basic architecture of the 8PSK Modulator is straight forward and is illustrated in Figure 3.2. Each one of these blocks were described in the previous sections (Section 3.2.1 and Section 3.2.2)

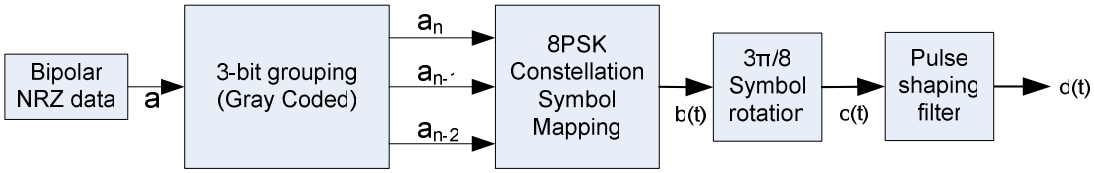

Figure 3.2 – 8PSK signal generation

The Bipolar NRZ (+1,-1) data are first grouped into three Gray coded bits according to Table 3.1 and are then mapped onto the respective symbols. The symbols are then rotated by $3\pi/8$ radians. And finally the rotated symbols are convolved with the pulse shaping filter defined by expression (2.6).

The signal *d(t)* will be a *complex base-band* signal because the resultant signal *b(t)* from the Constellation mapping is a complex signal having a real and imaginary components. However in practice the carrier is applied to the base-band signal (i.e. $d(t).\cos(2\pi f_c t)$) just after pulse shaping.

3.2.4 8PSK Demodulator Design

The demodulator design that will be used in this project takes the complete opposite form of the Modulator. The Demodulator is basically operating the modulation procedures backwards, and is illustrated in Figure 3.3.

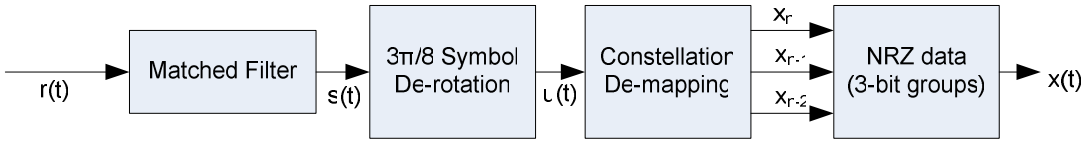

Figure 3.3 – 8PSK signal demodulation

Firstly the received signal is passed through a low pass matched filter. This matched filter is defined by the same expression used for the pulse shaping filter (2.6). The resulting signal *s(t)* is then de-rotated, to obtain the original 8PSK constellation (illustrated in the next Section 3.3). The de-rotated symbols are defined as:

$$u_i(t) = \frac{s_i(t)}{e^{\frac{ji3\pi}{8}}} \qquad (2.7)$$

Every i^{th} symbol of *s(t)* is divided by $e^{\frac{ji3\pi}{8}}$ to obtain the original symbol. Where, *i* denotes the sample position in *s(t)* , the resultant signal after matched filtering.

The de-rotated symbols are then de-mapped into three bit groups. This is done by inspecting the phase of the input signal. If the phase of the signal lies between certain limits then the output of the de-mapping produces three bits which were originally mapped onto that phase.

As an example, it can be seen from Figure 3.4; when the phase of the input signal is between (π/8) and (3π/8) the output of the decision maker/symbol de-mapping results in '011'.

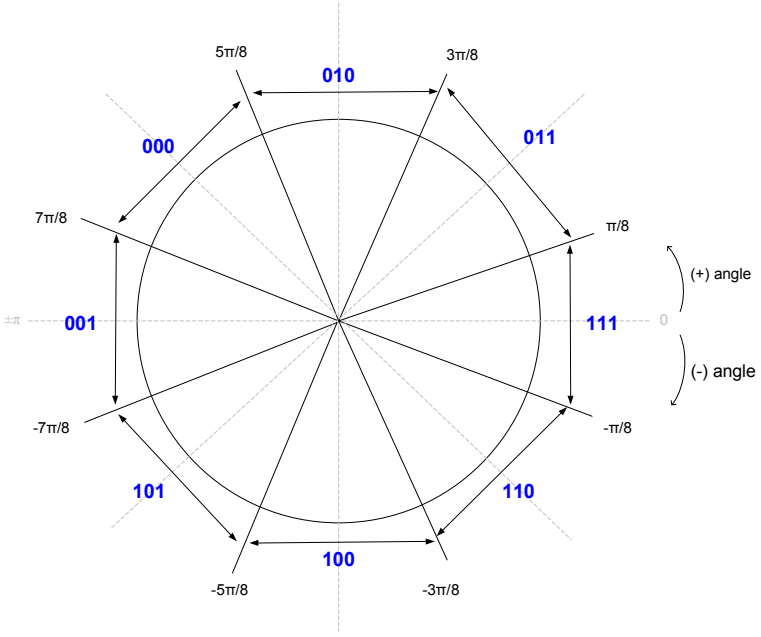

Figure 3.4 – Constellation de-mapping

Note: If the original bits that were transmitted, used Bipolar NRZ signalling, then the constellation de-mapping or decision making procedure should replace zeros with '-1's.

For example, if the phase of the received signal takes a value between, (-3π/8) and (-5π/8), then the output 3 bit group should be '1 -1 -1'.

Finally, the stream of 3-bit groups should be combined together to form a stream of bits and compared with the original transmitted bits, to calculate the Bit Error Rate.

3.3 8-PHASE SHIFT KEYING MODEM SIMULATION

This section of the report describes the simulation procedure of the 8PSK modulator and demodulator. The Appendix of this report contains the source code used for the simulation. A full description of each sub-function written is included in the code as comments.

The 8PSK simulation contained many small sub operations that were implemented by separating them into user defined sub-functions. Figure 3.5 shows the sub-functions that were created for the 8PSK simulation.

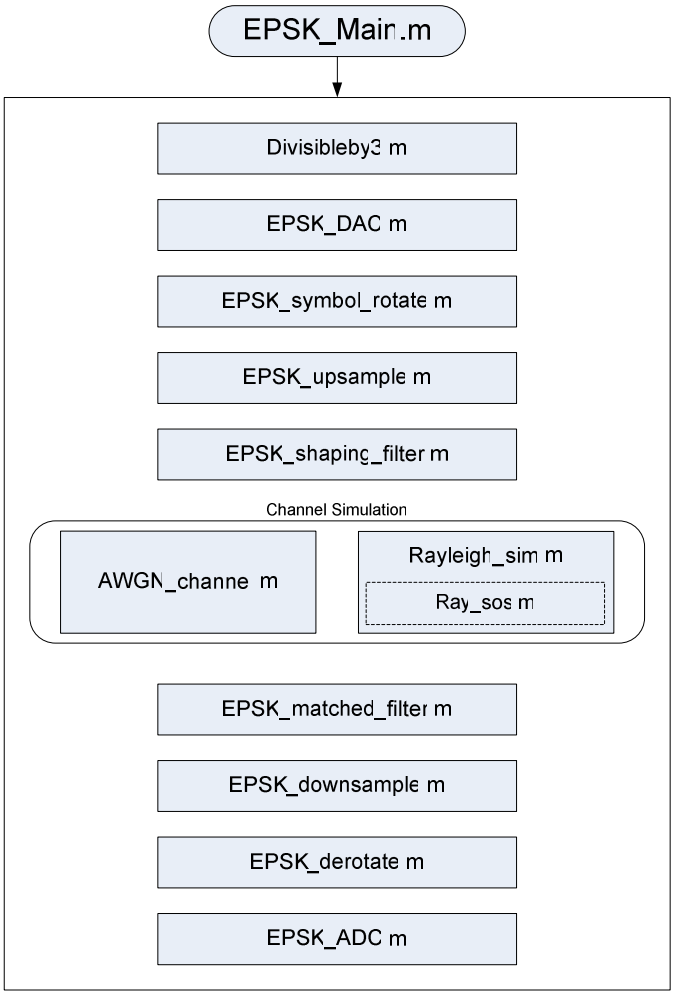

Figure 3.5 – 8PSK Modem Simulation Sub-functions

Simulations were implemented for two types of 8PSK modems.

- One which used a ***Gaussian pulse shaping filter***, and an equal Matched Filter.
- And the second type of 8PSK modem used a ***Raised Cosine filter***, for the pulse shaping, and Matched Filtering.

This was done, to investigate the difference between the traditional 8PSK modem, and the 8PSK modem specified by the 3GPP TS 45.004 v7.0 standards. A detailed comparison of the two significantly different techniques will be discussed in Chapter 4 and 5.

3.3.1 Modulator Implementation

Certain Matlab functions and techniques which were used for the GMSK simulation will be used for the 8PSK simulation as well. Therefore explanations of these functions will be avoided and only the relevant results will be discussed.

Bit Stream Generation

The first step in the simulation is to generate a stream of bits. The same parameters that were used for the GMSK simulation was used here, to be consistent in the two simulations (similar parameters will help accurate analysis in later stages of the project). The parameters are as follows:

- Bit duration $T_b = 1$ second.
- Samples per bit = 36. (i.e. sampling rate = fs = 36Hz);
- Type of coding = Bipolar Non-Return to Zero (NRZ)(i.e. ±1)

The '`randsrc`' function was used exactly the same way they were used in the GMSK simulation to generate the data (refer *Appendix C1* for the source code). The bits will be sampled and converted into a continuous time signal just before pulse shaping.

To understand the 8PSK modulation, it is better to use a larger number of bits. This is because certain plots such as the constellation plots and the symbol rotation are much more visible when a high number of bits are being simulated. Therefore 1000 random bits will be simulated.

Figure 3.6 shows the first 60 of the 1000 bits that were generated. Note that the bits are still discrete time signals and are converted into continuous time signals jest before filtering.

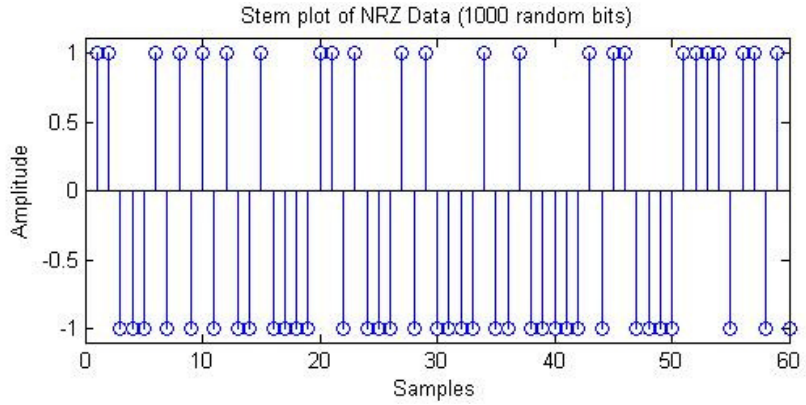

Figure 3.6 – Random Bipolar data bits (input to the 8PSK modulator)

A function was made called '`divisibleby3`' which takes in the generated data bits and checks if they are divisible by 3. This is done to ensure that the generated bits can be grouped into 3 bit groups before being mapped onto the 8 different symbols (refer Figure 3.1 for 8PSK symbol constellation). Figure 3.7 illustrates this function which was written to ensure the number of bits is divisible by three.

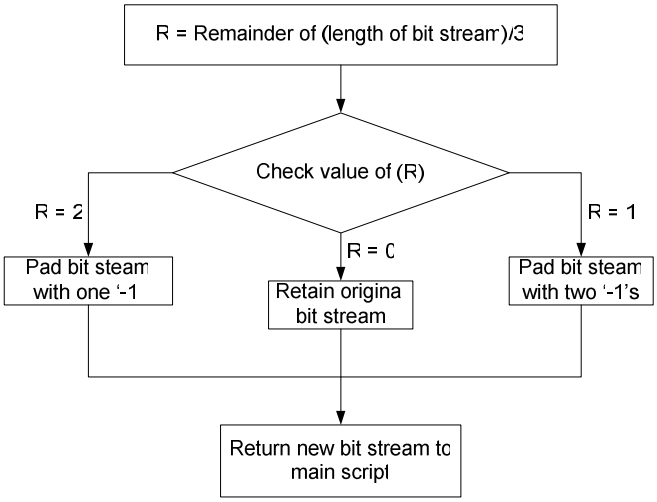

Figure 3.7 – Flow chart of the '`divisibleby3`' function

For example if 1000 bits are generated, they cannot be grouped into 3 bit groups because one bit will be left remaining (1000/3 = 333+1/3). For such a number of bits, the function pads the original bit stream with two '-1's to make the total number of bits 1002 (i.e. 1002/3 = 334).

Symbol Mapping (Digital to Analog Conversion)

The generated bit stream (of length 1002 bits, after the '-1' padding) is now mapped onto the 8PSK symbols according to expression (1.9) and Table 3.1.

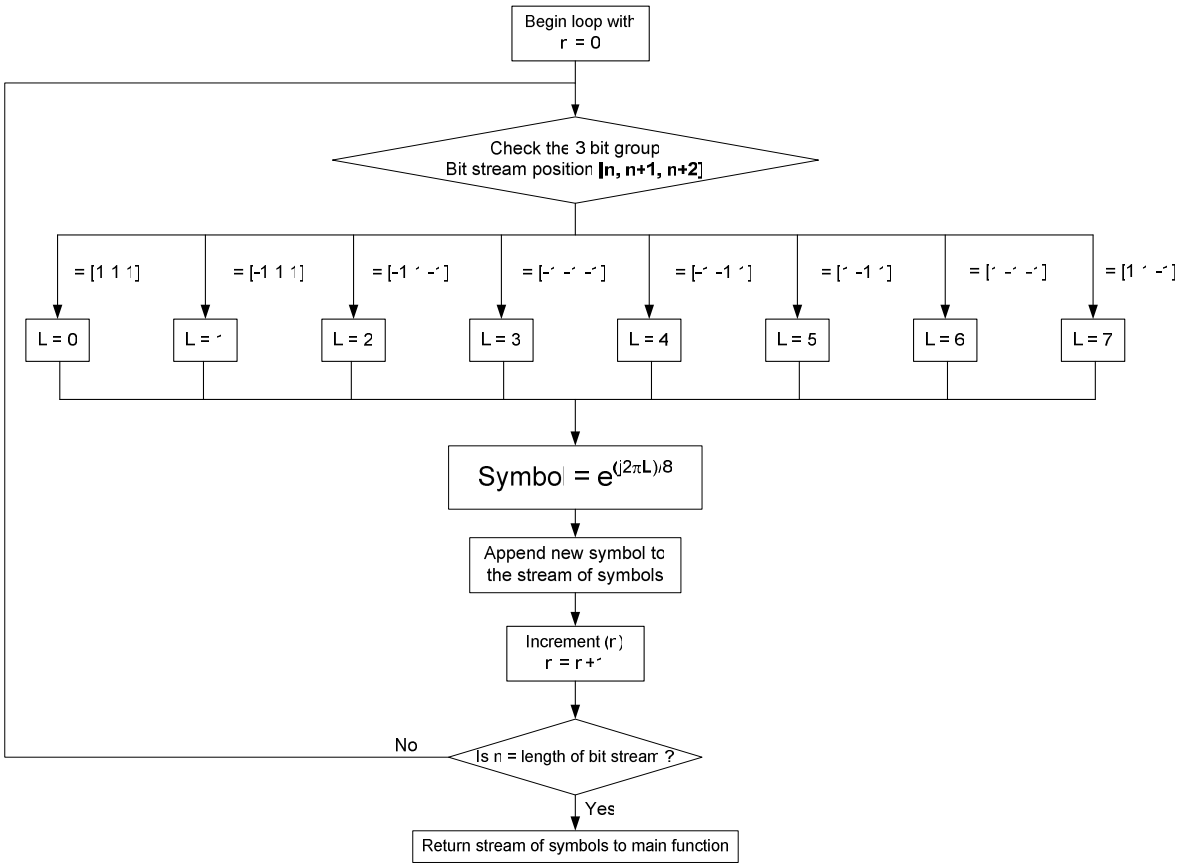

Figure 3.8 – Flow chart of the 8PSK constellation Symbol Mapping Process

The process in which this was performed is described by Figure 3.8 and the code can be found in *Appendix C2.1*. Three bits at a time is compared according to Table 3.1 and appropriate symbols are substituted instead of the bits. The symbols which is referred to here are complex numbers each having a real and imaginary component.

A separate function was written to perform this process and was named '`epsk_DAC`'. The resulting symbol stream will be 334 symbols in length (i.e. 1002/3 = 334), and is stored in a vector named '`s`'.

Figures 3.8(a) and 3.8(b) shows a line plot and scatter plot of the vector '`s`' respectively. These plots represent the 8PSK symbol IQ constellation diagram shown in Figure 3.1, and thus verify that the symbol mapping was performed successfully.

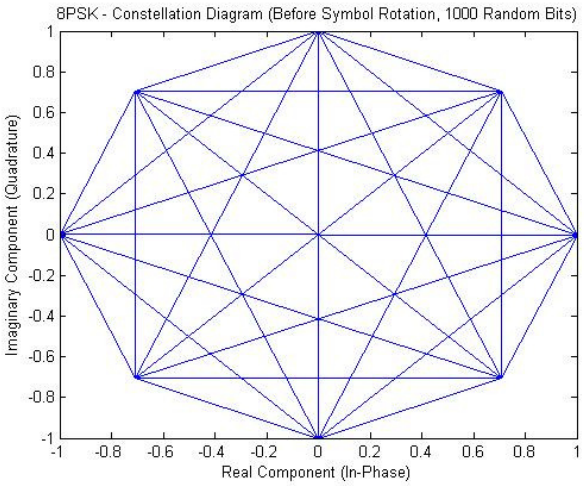

Figure 3.9(a)
Line plot of the 8PSK symbol stream (after symbol mapping and before symbol rotation)

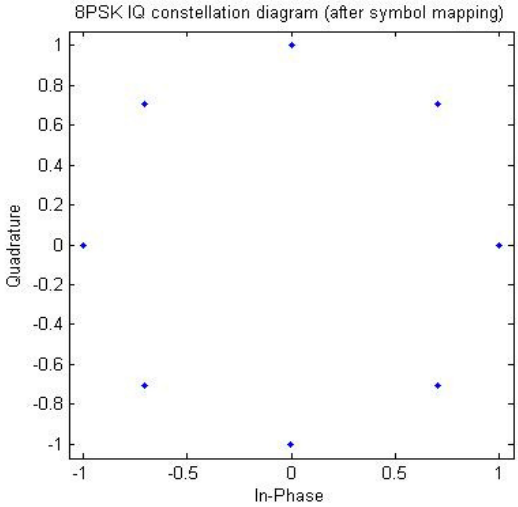

Figure 3.9(b)
Scatter plot of the 8PSK symbols (prior to symbol rotation)

As explained before in the GMSK simulation, the line plot is merely a plot where the dots in Figure 3.8(b) are joined together using a straight line.

Notice the lines cross the center of the origin of the plot in Figure 3.9(a). Some engineers consider this an issue when building power amplifiers and thus symbol rotation is performed.

These two figures indicate that the Real and Imaginary components of the 8PSK signal (prior to symbol rotation) will have 4 different voltage/magnitude levels each.

Table 3.2 explains the 4 different amplitude levels and the symbol mapping that took place, before symbol rotation. The four levels which the I and Q component will have are, ±1 and ±0.7071, and these represents different voltage levels.

The number of amplitude levels in the I and Q components are directly related to the number of modulation symbols in the constellation diagram. For example, QPSK has 2 levels of amplitude in each of its I and Q component.

To verify this observation, a plot of the individual real and complex components of the symbols were obtained, and is shown in Figure 4.0(a) and Figure 4.0(b). The symbols are still in discrete time and therefore stem plots were obtained. Only the first 20 symbols were plotted out of the total 334, but the distinct 4 levels of magnitude can be seen.

Symbol (S_i =)	In-phase component (voltage)	Quadrature component (voltage)
$e^{j2\pi(0)/8}$	1	0
$e^{j2\pi(1)/8}$	0.7071	0.7071
$e^{j2\pi(2)/8}$	0	1
$e^{j2\pi(3)/8}$	-0.7071	0.7071
$e^{j2\pi(4)/8}$	-1	0
$e^{j2\pi(5)/8}$	-0.7071	-0.7071
$e^{j2\pi(6)/8}$	0	-1
$e^{j2\pi(7)/8}$	0.7071	-0.7071

Table 3.2 – Different I and Q levels after 8PSK symbol mapping

Symbol Rotation

As Figure 3.2 describes, the next step in the 8PSK signal generation is to rotate the symbols in order to prevent the symbols from crossing the origin, as shown in Figure 3.8(a).

This procedure is mathematically expressed in expression 2.0, and is implemented according to the flow chart shown in Figure 3.10. (Source Code can be found in *Appendix C2.1*)

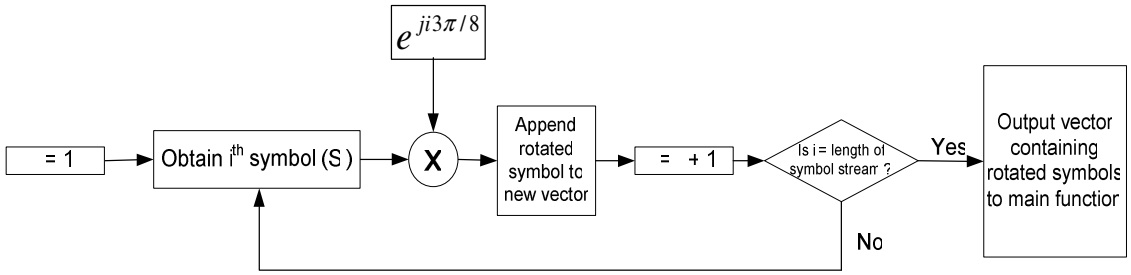

Figure 3.10 – Flow chart of the Symbol Rotation process.

As Figure 3.10 illustrates, each element in vector '`s`' (the vector containing the 8PSK symbol) is multiplied by $e^{ji3\pi/8}$, where the 'i' is incremented by 1 for each element in the vector. This loop is continued until i = the length of the vector.

Figure 3.11 shows result of the first 20 symbols, after they have been rotated. The Figure clearly shows there were four levels of amplitude before rotation and eight levels after rotation.

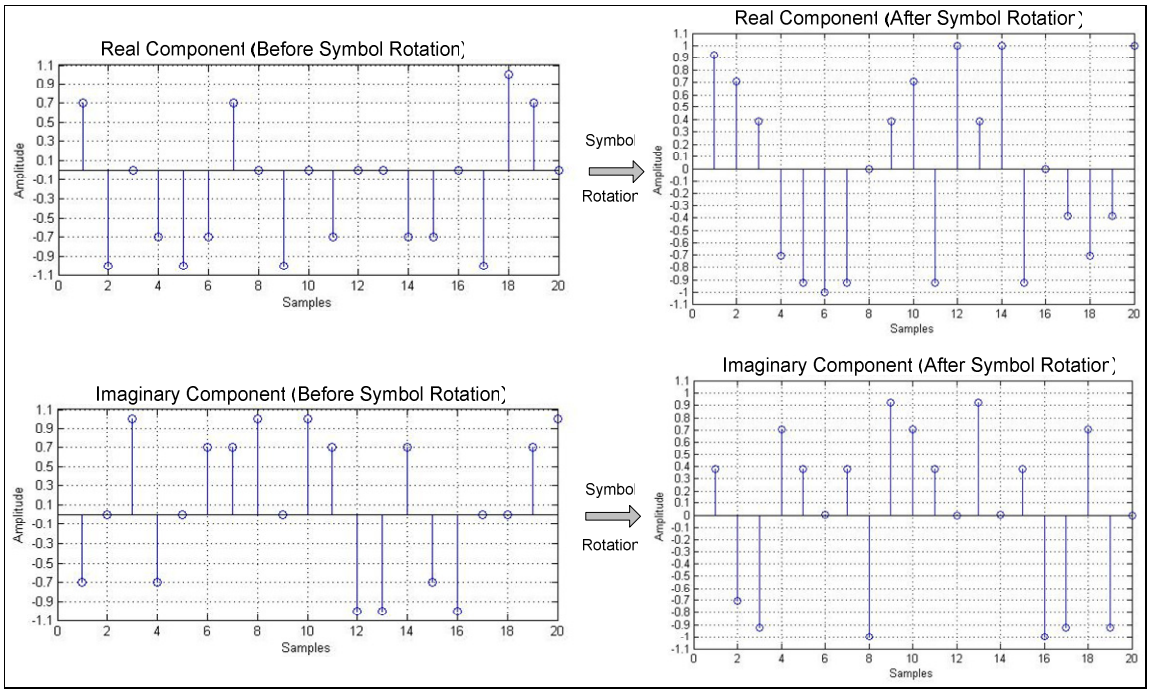

Figure 3.11 – Symbol Rotation of Real & Imaginary Components (first 20 symbols)

Figure 3.12(a) and 3.12(b) shows a line plot and scatter plot of the rotated symbols respectively.

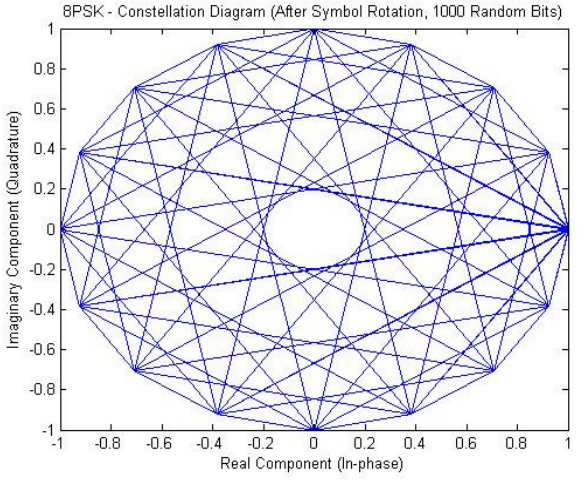

Figure 3.12(a)
Line plot of the 3π/8 rotated 8PSK symbol l stream

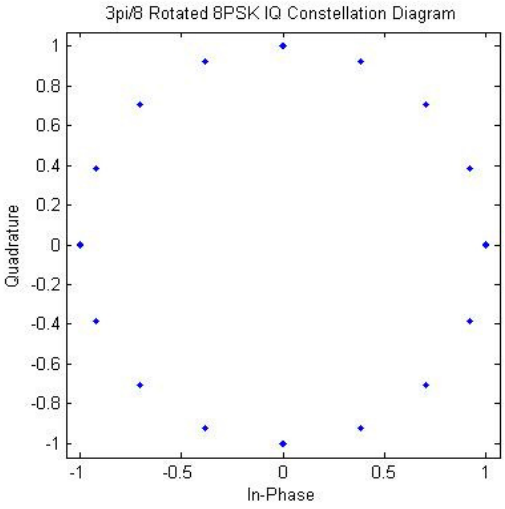

Figure 3.12(b)
Scatter plot of the 3π/8 rotated 8PSK symbol stream

Figures 3.12(a) clearly shows that rotating the symbols by 3π/8 has solved the issue of the symbols crossing the origin, because now there is an empty space in the centre of the constellation diagram. This ensures envelope of the signal will not go to zero.

Due to the rotation of the symbols, there are 16 different symbols in the rotated constellation diagram, which is equivalent to the constellation diagram of 16-PSK. Table 3.3 shows the different levels of the Real (In-phase) and Imaginary (Quadrature) components of the signal after rotation. There are 16 different symbols which correspond to the 16 different points in the IQ constellation diagram shown in Figure 3.12(b).

Symbol (S_i =)	**In-phase component (voltage)**	**Quadrature component (voltage)**
$e^{j(0)3\pi/8}$	1.0000	0.0000
$e^{j(1)3\pi/8}$	0.3827	0.9239
$e^{j(2)3\pi/8}$	-0.7071	0.7071
$e^{j(3)3\pi/8}$	-0.9239	-0.3827
$e^{j(4)3\pi/8}$	-0.0000	-1.0000
$e^{j(5)3\pi/8}$	0.9239	-0.3827
$e^{j(6)3\pi/8}$	0.7071	0.7071
$e^{j(7)3\pi/8}$	-0.3827	0.9239
$e^{j(8)3\pi/8}$	-1.0000	0.0000
$e^{j(9)3\pi/8}$	-0.3827	-0.9239
$e^{j(10)3\pi/8}$	0.7071	-0.7071
$e^{j(11)3\pi/8}$	0.9239	0.3827
$e^{j(12)3\pi/8}$	0.0000	1.0000
$e^{j(13)3\pi/8}$	-0.9239	0.3827
$e^{j(14)3\pi/8}$	-0.7071	-0.7071
$e^{j(15)3\pi/8}$	0.3827	-0.9239

Table 3.3 - Different I and Q levels after 8PSK symbol rotation

Discrete time to Continuous time conversion (Sampling)

The next step in the modulator is to sample the rotated symbols. The symbols will be sampled at 36Hz, meaning 36 samples will be used to represent one rotated symbol. The same matlab function '`kron`' which was used in the GMSK simulation will be used in the similar manner to convert the samples into continuous time.

Even though this is a trivial task in matlab, a separate function was written and was named '`epsk_upsample`'. (The source code can be found in *Appendix C2.1*).

Figure 3.13 shows the result of the Digital to Analog conversion process zoomed in to the first 20 symbols. The different voltage/amplitude levels are more visible now. The vector containing these sampled continuous time symbols was named '`s3`' and its length was 12024 samples long (i.e. 334*36 = 12024). The x-axis in Figure 3.13 represents the time domain, and shows symbol duration of 1 second (36Hz = 36 samples per second).

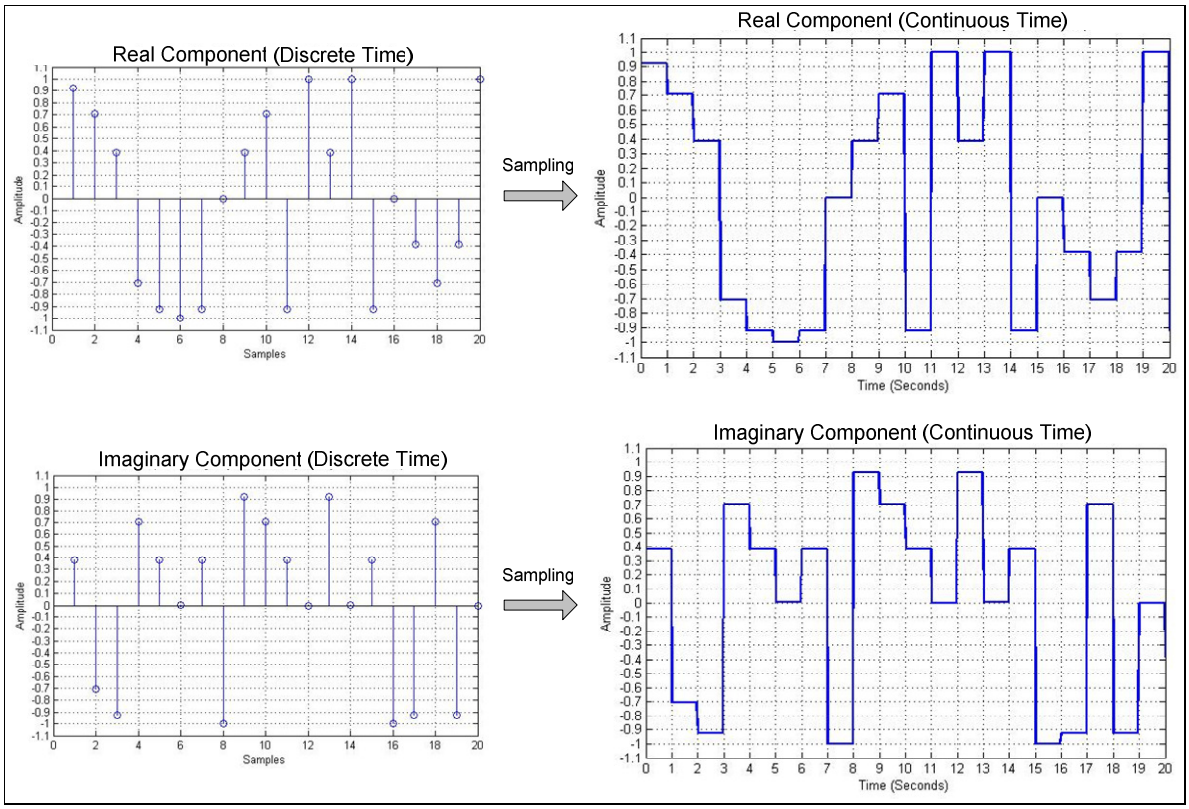

Figure 3.13 – Discrete Time to Continuous Time conversion

The Pulse Shaping Filters

The sampled symbols which are shown in Figure 3.13 are a combination of rectangular signals, each with length 1 second (36 samples) and different heights. In frequency domain, these 'rects' Fourier transform to a combination of 'sinc' signals, each overlapping the other and having a wide spectrum.

This spectrum is narrowed to be able to fit into the allocated bandwidth given for GSM transmission (200 kHz). This is an important fact, because even though we are now performing 8PSK modulation for EDGE, RF engineers still have to manage with the allocated bandwidth available for GSM.

Therefore the spectrum has to be filtered and in time domain the rectangular pulses have to be convolved with a filter. The 8-PSK signal with raised-cosine filtering does not fit within 200 kHz of bandwidth.

As mentioned at the beginning of Section 3.3, simulations will be performed for two kinds of Pulse shaping filters.

- A Gaussian shaped filter specified in the 3GPP EDGE standards [1].
- A Raised Cosine Filter.

The Gaussian Shaped Low Pass Filter (GLP Filter)

The Gaussian shaped filter was implemented using expression (2.6), as follows:

```
t = (-5*T/2:T/sps:5*T/2);
approx_filt = exp((-1.045*(t/T).^2) - (0.218*(t/T).^4));
```

The filter coefficients are stored in the vector named '`approx_filt`' (approximate filter). The code was placed in a separate function called '`epsk_shaping_filter`' to maintain the functional programming style maintained throughout all the simulations. The following line of code was added to the function, to normalize the filter coefficients, to ensure that the filter has total power of unity, thus it will not change the power of the signal after convolution.

```
%normalize filter gain.
approx_filt = approx_filt/sqrt(sum(approx_filt));
```

The sampling rate used for the filter is 21Hz ('`sps`'=21), and the filter was designed to stretch over 3 symbol periods, similar to the Gaussian filter used for GMSK, BT=0.3. The total length of the Filter was 106 samples long.

Figure 3.14 shows the Gaussian Filter that was used to filter the symbols, super-imposed on the Gaussian Filter used in the GMSK modem. There is a vast difference in peak height between the Gaussian Filter used for 8PSK and the one used for GMSK. The Gaussian Low pass filter used in the 8PSK modulator had a much larger gain; this was partly due to the scaling factor used in the GMSK Filter described in Section 2.6.1.

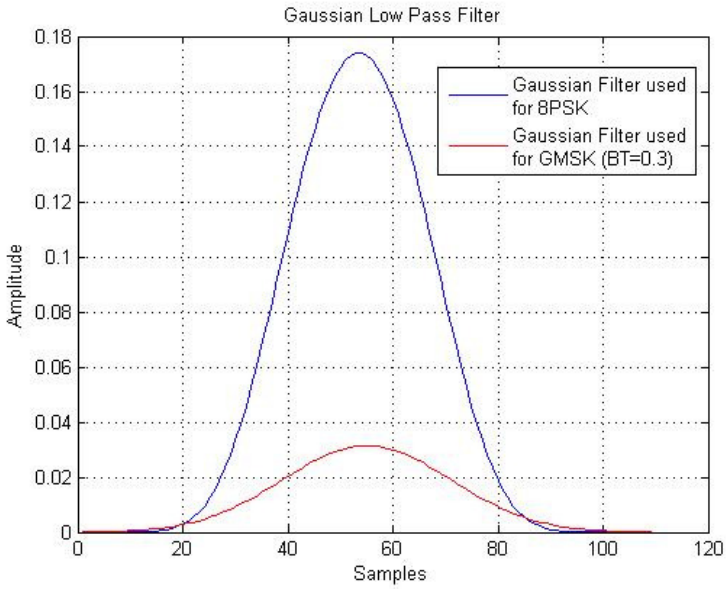

Figure 3.14 – Comparison of Gaussian Filters used for 8PSK and GMSK

The Raised Cosine Filter (RC filter)

Using a Raised Cosine filter has one main advantage which is it reduces Inter-Symbol-Interference (ISI) which will be discussed in detail in Chapter 4. The impulse response of a raised cosine is defined by the following expression:

$$h(t) = \frac{\sin(\pi t / T)}{\pi t / T} \cdot \frac{\cos\left(\frac{\pi \beta t}{T}\right)}{1 - \frac{4\beta^2 t^2}{T^2}} \qquad (2.8)$$

Where $0 \leq \beta \leq 1$ is the roll off factor, which is a measure of the excess bandwidth over the Nyquist limit.

The raised cosine filter has a property of having its impulse response zero at integer multiples of the symbol duration (*nT*), except when *n=0*. This is used at an advantage in communication systems, because if the transmitted waveform is correctly sampled at the receiver, the original symbols can be retrieved completely.

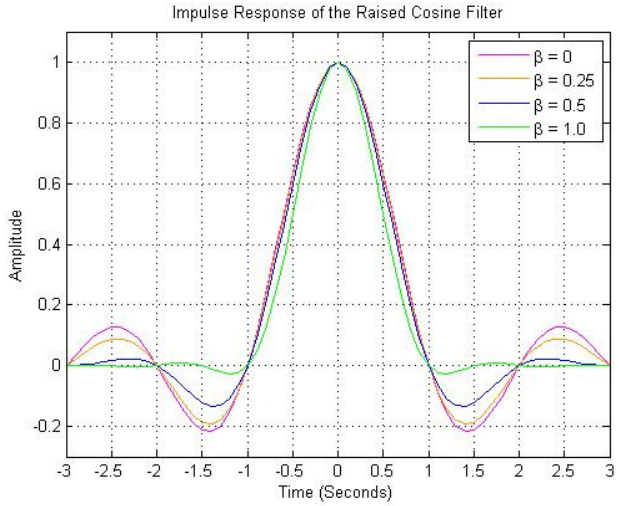

Figure 3.15(a)
Impulse response of the Raised cosine Filter

Figure 3.15(a) shows the impulse response of raised cosine filters for different values of β.

When β = 0 the raised cosine filter converges to a 'sinc' function.

As can be seen, the time-domain ripple level increases as β decreases.

Higher values of roll off factor make the filtering more tolerant against ISI.

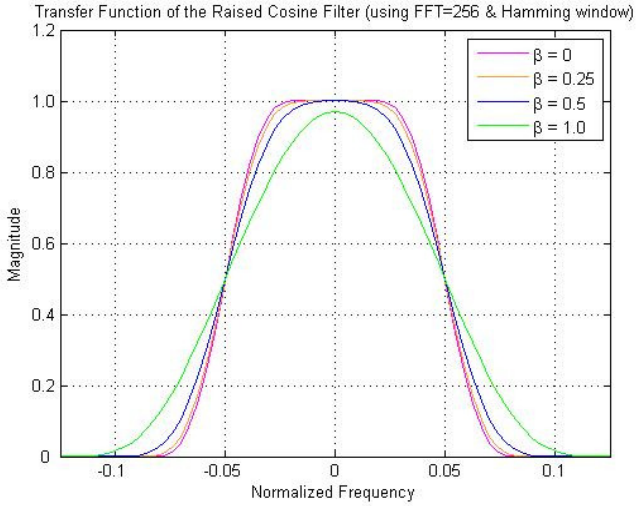

Figure 3.15(b)
Approximate Transfer function of the Raised cosine Filter

A Fast Fourier Transform (FFT) of these filters were produced with Hamming windowing to obtain an approximation for the transfer function of the filters, and results are shown in Figure 3.15(b).

The bandwidth of the filter is directly proportional to β. Higher values of β cause the bandwidth of the filtered signal to increase.

As β approaches 0 the raised cosine filter converges to an ideal brick wall filter.

For the 8PSK simulations a raised cosine filter with β = 0.5 will be used.

The Matlab Communication Toolbox provides a function to implement the raised cosine filter , called 'rcosine'. This function was used as follows:

```
%obtain the impulse response %filter coefficients
[pulse_filter den] = rcosine(1,18,'default');
% normalize filter for % unit gain
 pulse_filter = pulse_filter./sqrt(sum(pulse_filter));
```

The 'rcosine' function has three parameters. The first '1' specifies the input signal sampling frequency. The second '18' specifies the sampling frequency of the filter. The last parameter specifies if the filter is an FIR (Finite Impulse Response) or IIR (Infinite Impulse Response) filter, or Root Raised Cosine Filter, and 'default' is equivalent to saying it is a FIR filter.

Using these specific parameters a filter with 109 samples long was achieved.
(3 symbols = 36x3)

The second line shown in the code above is the same filter normalization method used before in the Gaussian Filter. This is to ensure unit gain of the overall filter.

The Pre-modulation Filtering Process

The filtering was performed using the '`conv`' function in matlab in a similar way to the filtering performed in the GMSK modulation. (Refer *Appendix C2.1* for source code).

Figure 3.16 shows the result of the filtering process to the real component of the 8PSK signal, when the two different types of filters are used. The Imaginary component of the signal was filtered in a similar way. Only the first 20 symbols are shown in Figure 3.16.

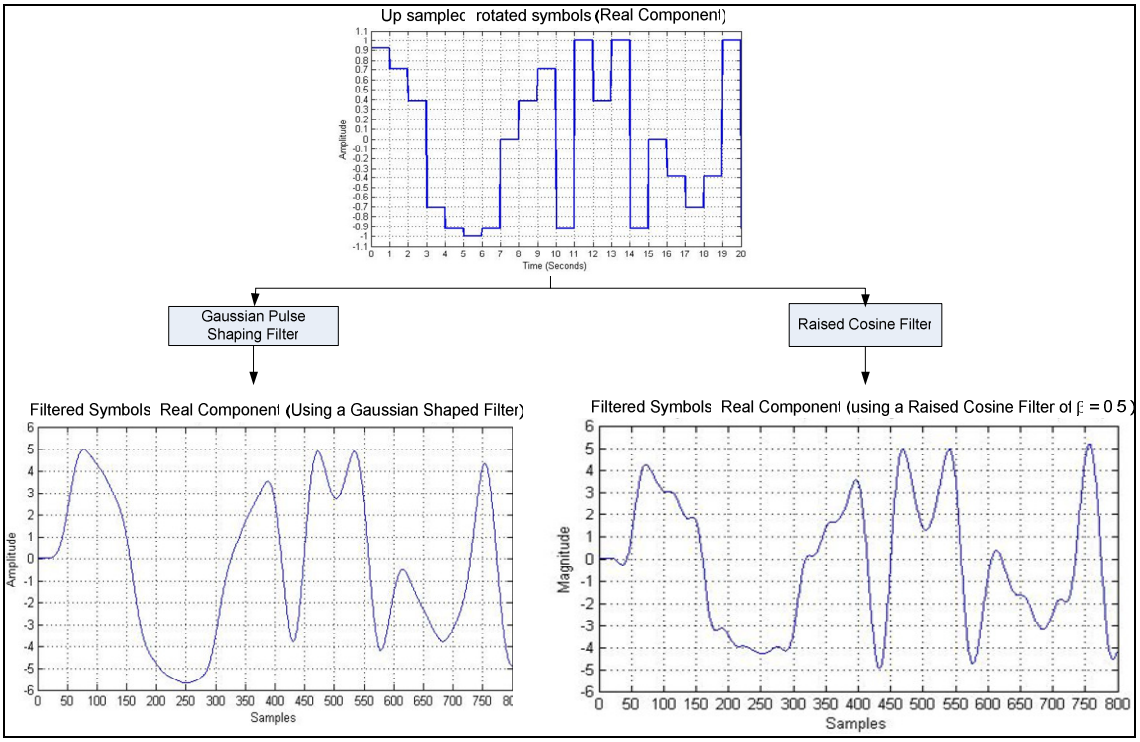

Figure 3.16 – The Filtering process, using a Gaussian Filter and a Raised Cosine Filter.

It is difficult to distinguish separate individual symbols in Figure 3.16 after they have been filtered using a Gaussian Filter. This is one observation that was made, when using filters which does not obey the Nyquist criterion (as seen in the GMSK pre-modulation filtering). The signal ultimately results in severe ISI. Careful sampling at the receiver should be performed to obtain the correct symbols.

When the raised cosine filter was used, the pulse shaping of each individual symbol could still be roughly seen in Figure 3.16. These observations can be verified by Eye diagrams which are shown in Chapter 4.

Figure 3.17(a) and Figure 3.17(b) shows the I/Q diagram of the 8PSK Baseband signal, for both the filtering types (GLP and RC Filter) and for a simulation run using the same 1000 random bits used before.

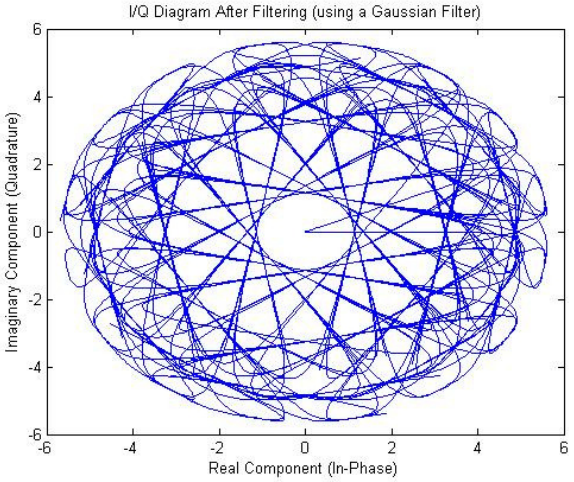

Figure 3.17(a)
I/Q Diagram of the Baseband 3π/8 rotated 8PSK signal using a Gaussian Low Pass Filter

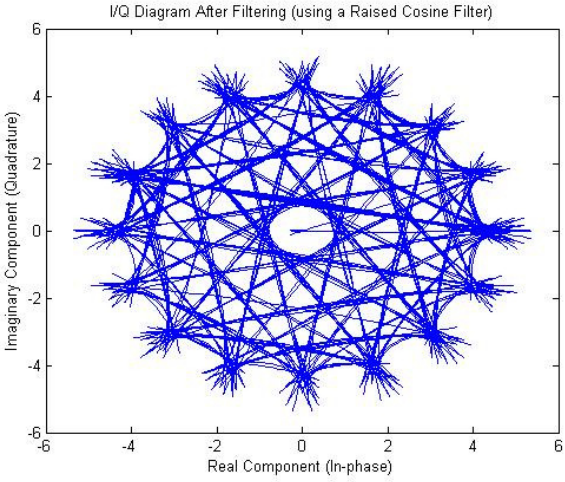

Figure 3.17(b)
I/Q Diagram of the Baseband 3π/8 rotated 8PSK signal using a Raised Cosine Filter

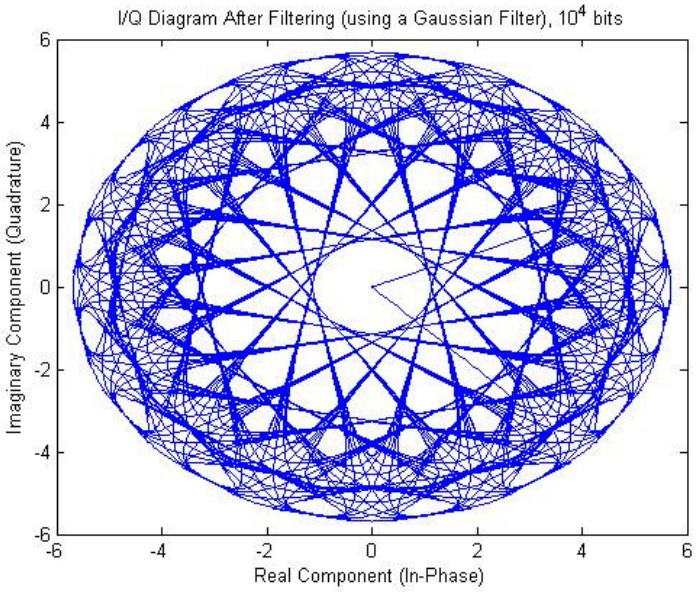

Figure 3.18 – I/Q Diagram of Baseband 3π/8 rotated 8PSK using a GLPF (Simulated for 10000 bits)

The 16 different symbols are almost visible in the Baseband signal using a GLP Filter in Figure 3.17(a), but the symbols are clearly visible when a RC filter was used. 10000 were simulated to investigate the IQ diagram further when a larger number of bits were modulated. Figure 3.18 displays the IQ diagram now shows a much tighter constellation than before, and the 16 different symbols of the 3π/8 rotated 8PSK signal can now be seen clearly.

Pulse shaping is the final step that is performed in the modulator, after which the filtered signal are applied onto the carrier and transmitted.

3.3.2 8PSK Demodulator Implementation

Figure 3.3 in Section 3.2.4 describes the procedure in which the demodulation of the 8PSK signal will take place. And in general terms the process can be broken down as follows:

6. Filter the received signal using a Matched Filter.
7. The filtered signal is in continuous time and therefore it needs to be sampled appropriately to convert into discrete time.
8. The sampled signal contains complex symbols which have to be de-rotated, because symbol rotation was performed at the modulator.
9. The phase of each of these de-rotated complex symbols have to be calculated and each symbol has to be converted into 3 bipolar bit groups (decision checking).

Matched Filtering

As mentioned before, two 8PSK simulations were performed using two types of filters. Accordingly two types of Matched filters were used and investigated. They are as follows:

- A Gaussian shaped Matched filter, equivalent to the Gaussian pulse shaping filter used at the modulator with the following characteristics:
  - Sampling frequency = 7Hz.
  - Length = 36 samples.
- A Raised Cosine Matched filter, equivalent to the filter used as the pulse shaping filter, with the following characteristics:
  - Roll off factor: $\beta = 0.5$
  - Sampling frequency = 10Hz.
  - Length = 61 samples.

Certain issues in the matched filters were dealt with and are described in detail in Section 3.3.3.

Figure 3.19(a) and Figure 3.19(b) show the matched filters that will be used in the simulations.

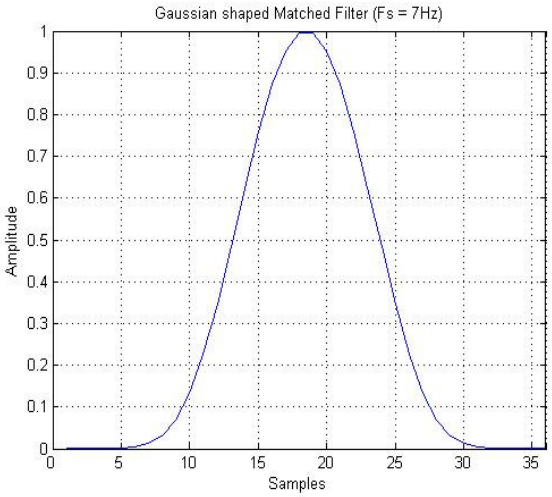

Figure 3.19(a)
The Gaussian Shaped Matched Filter

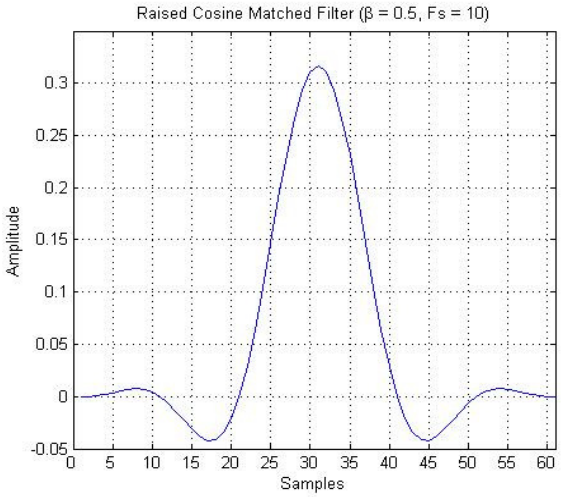

Figure 3.19(b)
The Raised Cosine Matched Filter

Expression (2.6) was used to implement the Gaussian shaped matched filter, but the filter normalization method was not used, because this produced unsatisfactory results.

The raised cosine matched filter was implemented using the same '`rcosine`' function used for the pre-modulation filtering. The sampling frequency was set to 10Hz.

The filtering process was performed using the 'conv' function as before. The results of the matched filtering can be seen in Figure 3.20(a) and 3.20(b), where the Gaussian Matched filter and the Raised Cosine Matched filter was used respectively.

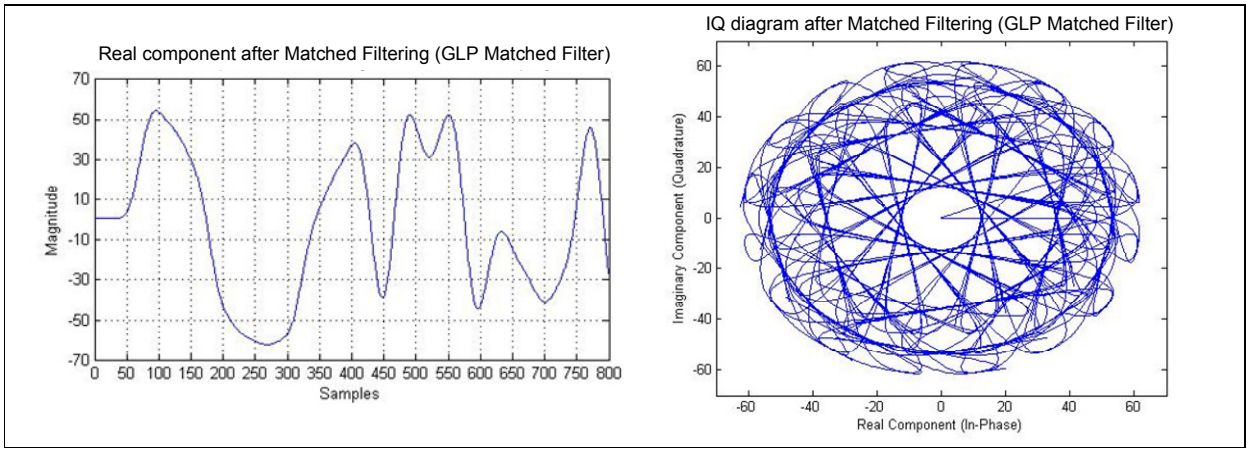

Figure 3.20(a) – Real component and IQ diagram after Matched filtering (using Gaussian Matched Filter)

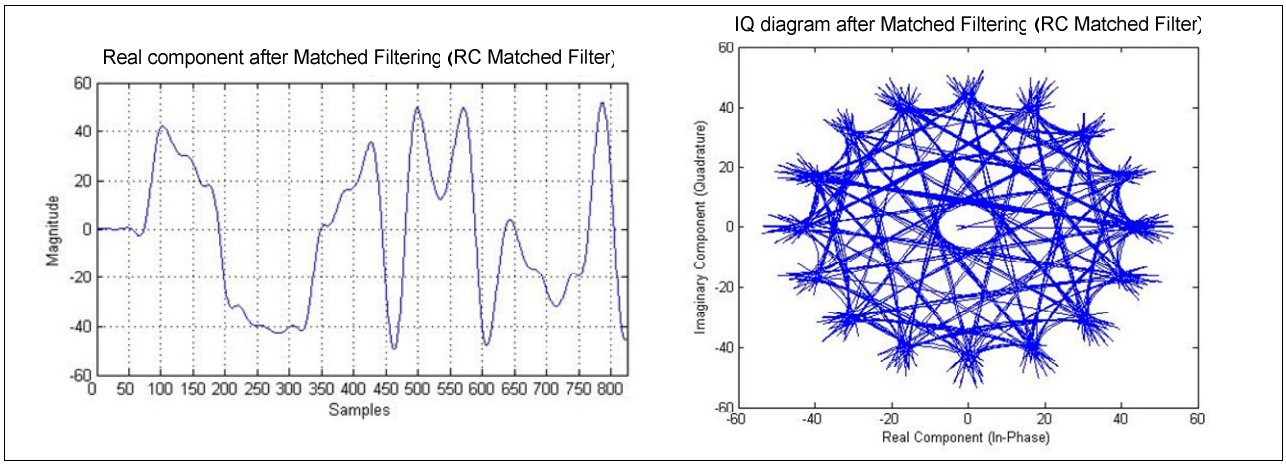

Figure 3.20(b) – Real component and IQ diagram after Raised Cosine Matched Filtering

Notice that the magnitude of the signal has increased after matched filtering when compared with the signal before matched filtering in Figure 3.16 and 3.17(a) and 3.17(b). One indication that the matched filtering was performed successfully is that the transmitted signal has completely been recovered at the receiver (assuming a zero noise condition). The shape of the transmitted signal (in terms of the IQ diagrams) is retained after matched filtering, which means the filter, in frequency domain was wide enough to recover the complete signal (unlike the scenario faced during the GMSK simulation). Therefore further analysis into the frequency domain was considered unnecessary.

Continuous Time to Discrete Time Conversion (Sampling)

The next step in the demodulator is to convert the continuous time complex signal into discrete time. This is done by sampling the signal at the same sampling rate used in the modulation to convert the discrete time symbols into continuous time. The sampling rate used before was 36Hz, meaning every 36 samples contain one symbol and one sample out of each of these 36 samples has to be obtained to form the discrete time signal.

The vector containing the matched filtered received signal is '`s_rx`'. This vector has a length of 12194 samples. The same technique used in the GMSK simulation to sample the GMSK received signal has to be used here. This is to take away the extra samples that were introduced during convolution.

A separate function was made to perform this sampling process. It is identical to the function used in the GMSK demodulator for the sampling process (refer Section 2.6.2 - Sampling). This function was named '`epsk_downsample`' and has 4 parameters. The following is how it is called from the main function:

```
s_rx_dwnsmpled = epsk_downsample(s_rx,samplerate,start,end);
```

The function returns a vector containing the discrete time samples which are then stored in '`s_rx_dwnsmpled`'. This specific details as to how this function operates is similar to the

explanation given in the GMSK demodulator simulation, therefore only the parameters will be explained.

`s_rx` = input vector to be sampled
`samplerate` = sampling rate used for the sampling process.
`start` = which sample to start sampling from.
`end` = how many samples to leave out from the end of the input vector

For the 8PSK simulation which used a Gaussian Pulse shaping filter and an equivalent matched filter, the parameters were:

- Sample rate = 36
- Start = 86
- End = 57

For the 8PSK simulation which used a Raised Cosine Pulse shaping filter and an equivalent matched filter, the parameters were:

- Sample rate = 36
- Start = 109
- End = 62

Section 3.3.3 contains an explanation as to how these values were obtained.

Figure 3.21 shows the outcome of the Sampling process in the simulation where the Gaussian shaped filter and matched filter was used. The first 20 symbols in the real component are shown. The simulation which used the Raised cosine filters also gave a similar result, in the sense that the obtained samples were different in value but similar in terms of what the samples represented (symbols).

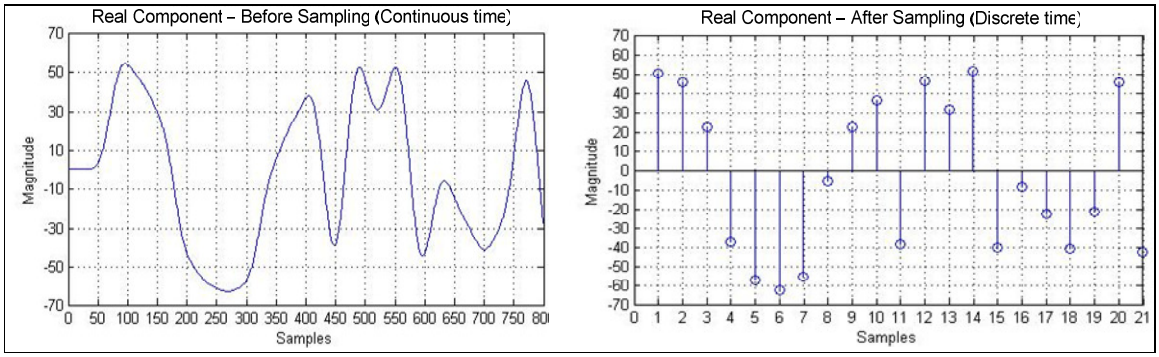

Figure 3.21 – Continuous time to Discrete Time conversion of the Real Component of the 8PSK signal

Figure 3.22 shows the outcome of the I/Q diagrams of the signal after the sampling process. Results are shown for the simulation which uses Gaussian shaped filtering and for the simulation which uses Raised cosine filtering. One observation that can be made at this point is that it is much easier to make out the 16 symbols after the sampling process, when a Raised cosine filter is used for the pulse shaping and matched filtering. The 16 symbols are not

evident when Gaussian filtering is used. This can directly affect the decision making process (ADC), when AWGN and Fading is introduced. This indicates the 8PSK modulator which uses the Gaussian filtering is more vulnerable to noise. The decision making process will output less erroneous bits if the symbols in the I/Q diagram are easily distinguishable.

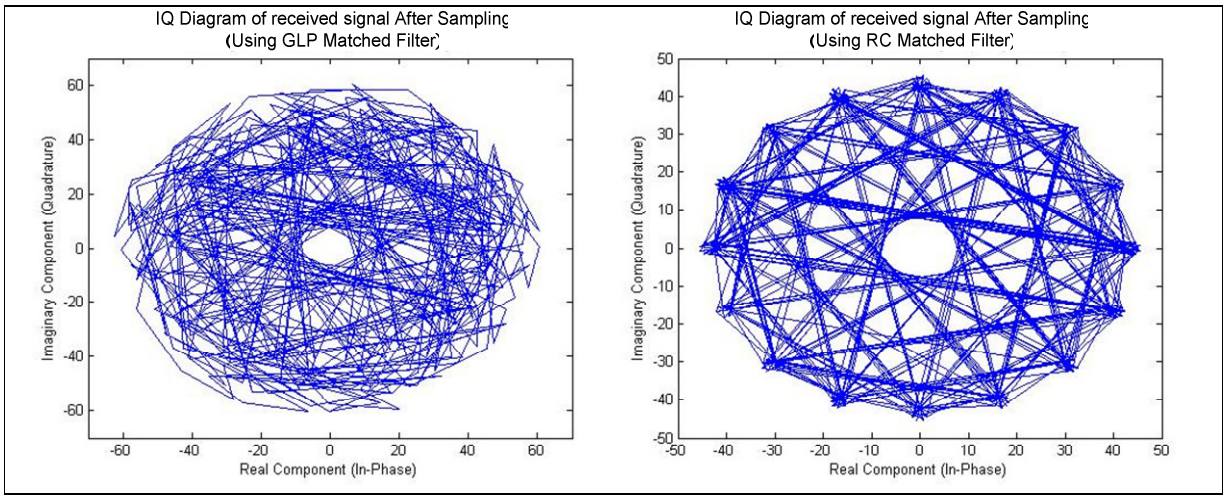

Figure 3.22 – The I/Q diagrams after sampling

Symbol De-Rotation

The sampled I/Q diagram of the 8PSK modulation which uses RC filtering can be seen in Figure 3.21. This sampled I/Q diagram shows 16 different distinguishable symbols or amplitude levels. This is because the 8 symbols in the 8PSK constellation diagram had been rotated by 3π/8 in the modulator. It is only logical that this process has to be reversed to obtain the original 8 symbols in the I/Q diagram.
The symbol rotation at the modulator was performed by multiplying each i^{th} symbol by $e^{\frac{ji3\pi}{8}}$. Therefore to reverse this process the sampled symbols have to be divided by $e^{\frac{ji3\pi}{8}}$, according to the expression (2.7).

This process was implemented exactly the same way the symbol rotation was implemented, except instead of the multiplication (⊗) operation division (÷) was performed and the Flow chart in Figure 3.9 was carried out accordingly (refer *Appendix C2.1* for source code).

A separate function named '`epsk_derotate`' was written to perform the de-rotation. This function was a common function for both the GLP filtered 8PSK and RC filtered 8PSK simulation. This function's input is the vector containing the sampled signal. It performs the above mentioned operation on each element of the vector and outputs a de-rotated vector, which will be stored in a new vector called '`s_rx_derotate`'. A line plot of this vector can be seen in Figure 3.23.

The result of the symbol de-rotation process can be seen by inspecting the I/Q diagrams. The de-rotated I/Q diagrams are shown for both types of filtering in Figure 3.22. Once again, the 8

distinct symbols can be clearly seen in the IQ diagram of the 8PSK modem which uses RC filtering. The 8 symbols are not visible in the IQ diagram of the 8PSK modem which uses Gaussian filtering.

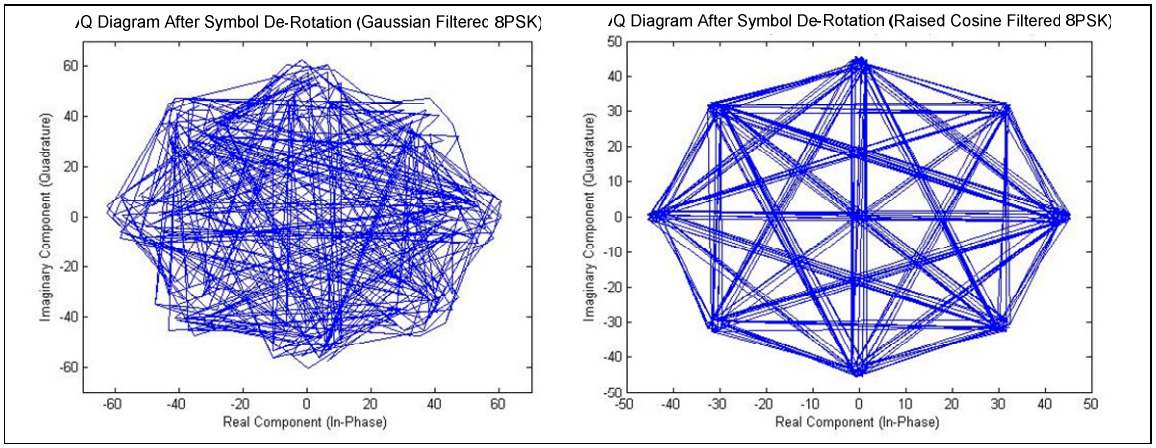

Figure 3.23 – Result of the symbol de-rotation performed for 8PSK using both types of Filtering

Decision Making Procedure – Analog to Digital Converter

The final stage of the demodulator is to convert each element of '`s_rx_derotate`' (the de-rotated symbols) into respective 3-bipolar bit groups. This task is similar to the Digital to Analog process mentioned in Section 3.3.1, but worked backwards.

To implement this task, the angle of each complex element in vector '`s_rx_derotate`' has to be calculated. The same built in matlab function used in the GMSK demodulator called '`angle`' was used. The '`unwrap`' function was not used here, because the decision checking routine takes into account of the sudden phase jumps from $-\pi$ to π, by performing a initial check to see if the angle is positive or negative. Figure 3.24 shows the output of the ADC process of the first 63 bipolar bits (corresponding to the first 21 symbols shown in Figure 3.21, i.e. 21x3 = 63)

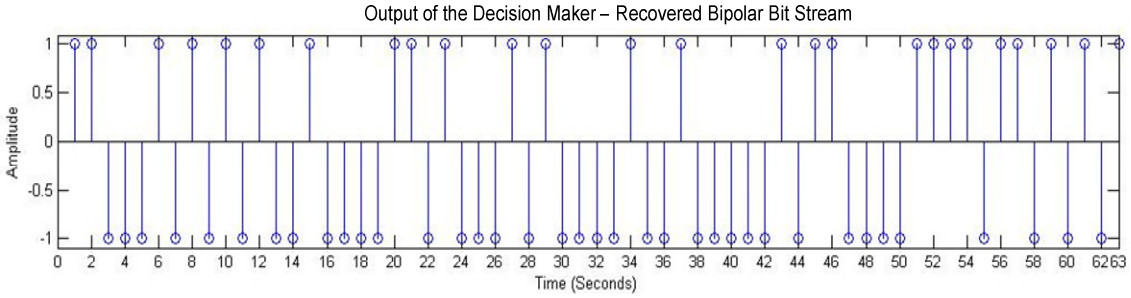

Figure 3.24 - Recovered Data bits, output of the Analog to Digital Converter

Figure 3.25 shows a flow chart as to how the ADC task was performed. A separate function named '`epsk_ADC`' was written, and the source code can be found in *Appendix C2.1*. This

function was a common function, used by both the GLP filtered 8PSK and RC filtered 8PSK simulation.

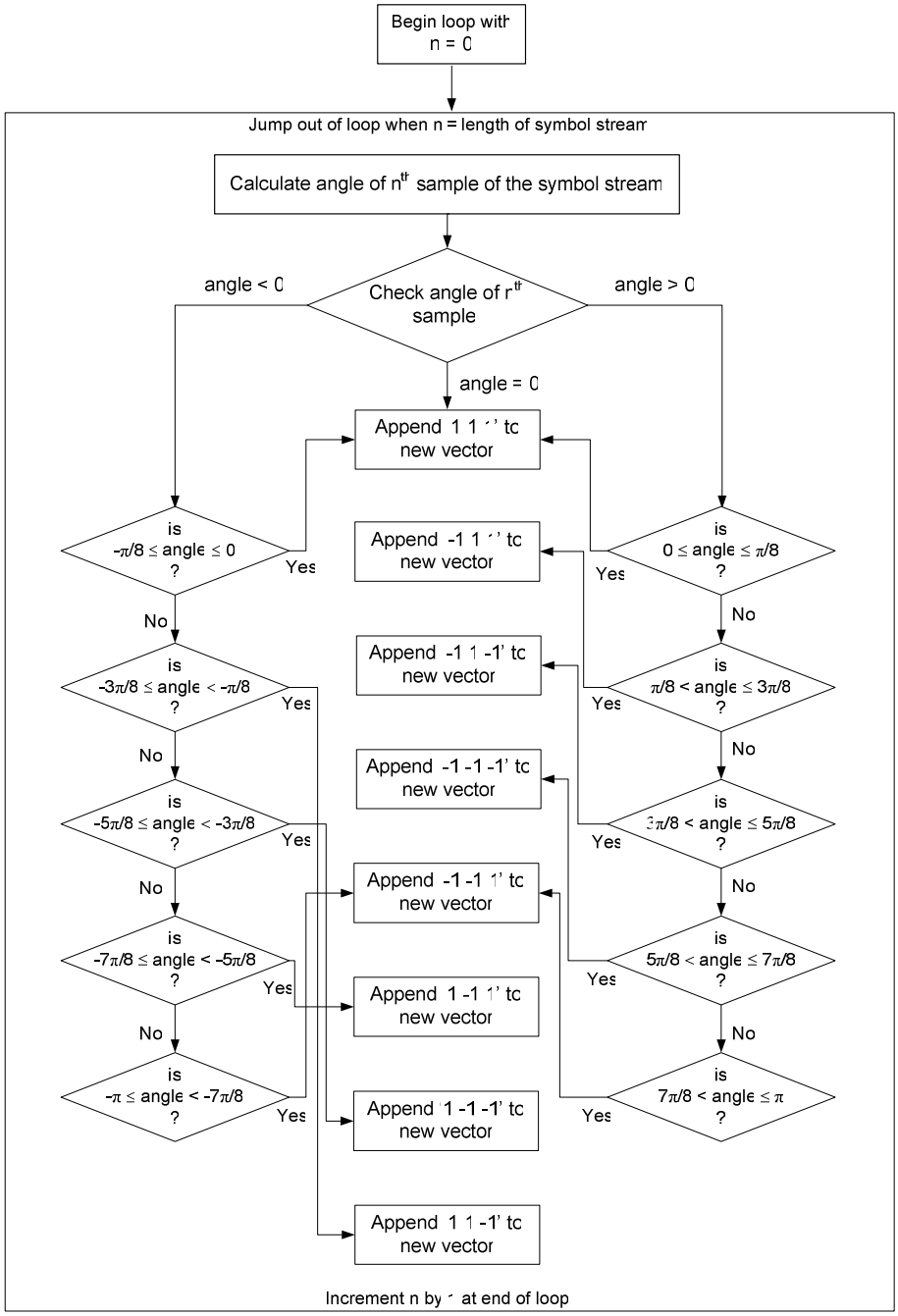

Figure 3.25 – Flow chart of the decision making process.

3.3.3 Demodulator issues

- **Deciding on a suitable sampling rate for the matched filters.**

  The original design was to use the same sampling rate used for the pulse shaping filters. But this proved unsuccessful, because the resulting I/Q diagrams after matched filtering were very much different to the I/Q diagrams produced at the modulator, before matched filtering. Figure 3.26 shows the different I/Q diagrams obtained for two different Matched filters with two different sampling rates.

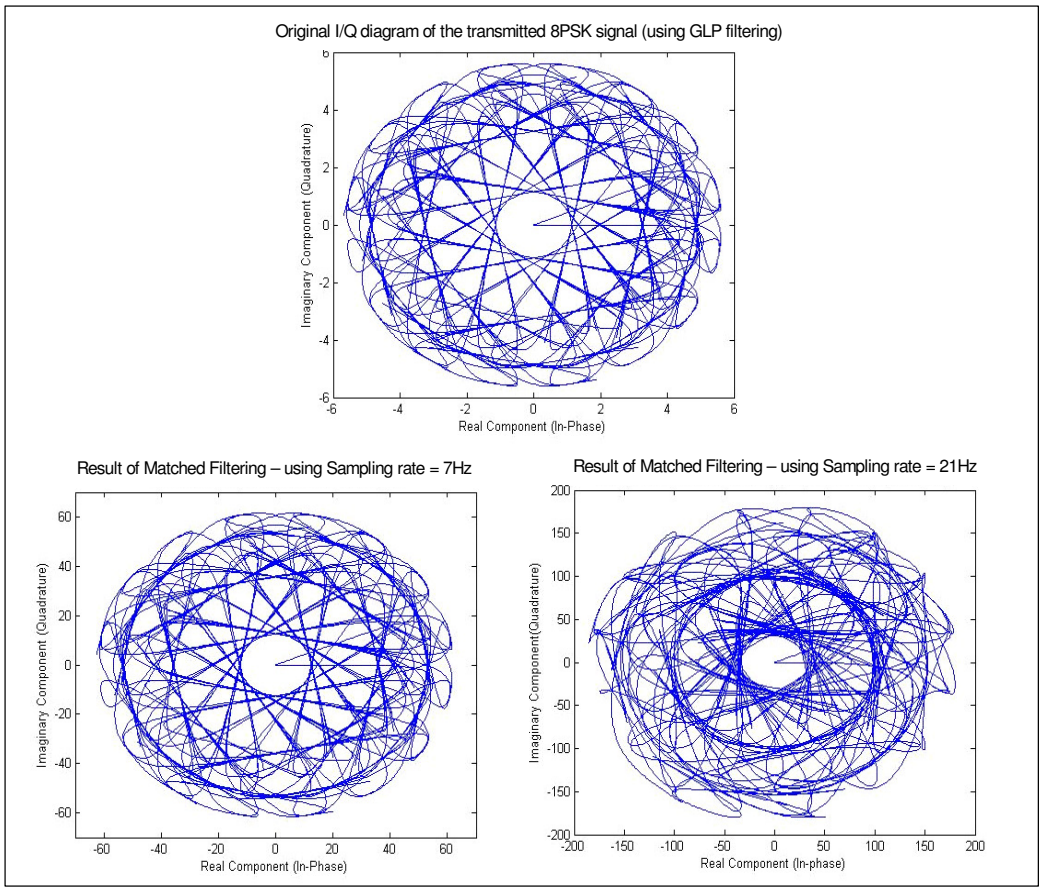

Figure 3.26 – Different Sampling rates produce different I/Q diagrams after matched filtering.

The topmost I/Q plot is the original I/Q diagram before matched filtering. The bottom left plot shows the result obtained when a sampling rate of 7Hz was used, and the bottom right plot shows the result obtained when a sampling rate of 21Hz (same as the pulse shaping filter) was used.

It is clear from Figure 3.26 that the choice of sampling rate for the gaussian matched filter should be 7Hz, as it has completely recovered the original transmitted signal.

A similar approach was taken when deciding on a sampling rate for the raised cosine matched filter. Different I/Q plots were obtained and compared, until a close match was made.

- **The vector containing the 8PSK Matched Filtered signal at the demodulator had a length much longer than the original up sampled message at the modulator.**

  Conclusion: The Pulse shaping and matched filtering which were performed in the Modem used convolution. As mentioned before convolution results in extra samples than the original vector.

  Solution: The pulse shaping filter was 106 samples long when the Gaussian filter was used and 109 samples long when the Raised cosine filter was used. The Gaussian shaped matched filter was 36 samples long, and the Raised cosine matched filter was 61 samples long.

  Therefore in the 8PSK simulation where the gaussian filtering was used, (106+36 = 142) samples needed to be taken out before sampling, to obtain the original 334 symbols. And where the raised cosine filtering was used, (109+61= 170) samples needed to be taken out before sampling, to obtain the original 334 symbols

  For the 8PSK simulation which used the gaussian filtering, 85 samples were taken out form the front and 57 samples were taken out from the end, before sampling. Initial parameters were similar to the approach taken in the GMSK demodulator sampling process (i.e. 71 and 71 and gradually incremented). In the end these values were obtained experimentally, by working with only a few amounts of bits initially (for example 12 or 15 bits, and then gradually increased the number to 102 and 150 bits). Working with a small number of bits, made it easier to understand what happens to each individual element in a vector.

  For the 8PSK simulation which used the raised cosine filtering, 109 samples were taken out form the front and 62 samples were taken out from the end, before sampling. These were the initial parameters used, and proved successful.

- **Output of the decision checking routine was producing incorrect bit groups.**

  Conclusion: The initial implementation of the decision checking routine did not take into account positive and negative angles the '`angle`' function outputs. Also the ranges of the conditional statements (refer *Appendix C2.1* for source code) used in the code were incorrect, in the sense that some of the boundaries overlapped each other.

  Solution: the diagram in Figure 3.4 was drawn and this helped to implement the correct logic needed for the decision making routine. Once again the decision checking routine was initially tested using a few bits and then gradually increased the amount to verify that there were no erroneous bits at the output.

3.3.4 Preliminary Tests without Additive White Gaussian Noise

The preliminary tests on the two types of (RC filtered and GLP filtered) 8PSK simulations were carried out without the introduction of Additive White Gaussian Noise.

The matlab function '`symerr`' was used. This function was explained in detail in Section 2.6.4, and will be used likewise.

Simulation runs were performed for a small number of bits first and this helped to correct the bugs that were described in Section 3.3.3. The simulation bits were gradually increased to 100 and then 1000 and finally checked the simulation for 100,000 bits. As explained in Section 2.6.4, the maximum number of bits which were able to simulate were 100,000.

Multiple simulation runs were produced up to 100,000 bits with zero BER (without introducing noise). This implied that further simulations regarding scenarios with AWGN and Multipath Fading can now be carried out.

CHAPTER 4
MODEM SIMULATION WITH ADDITIVE WHITE GAUSSIAN NOISE

To estimate the performance of a digital communication system, using Monte Carlo Simulation, N bits are passed through the computer simulation of the communication system and the amount of transmission errors, N_e is calculated.

To investigate these errors in the simplest form, an Additive White Gaussian Noise (AWGN) channel is assumed. Figure 4.1 shows the simulation setup for a simple communication system.

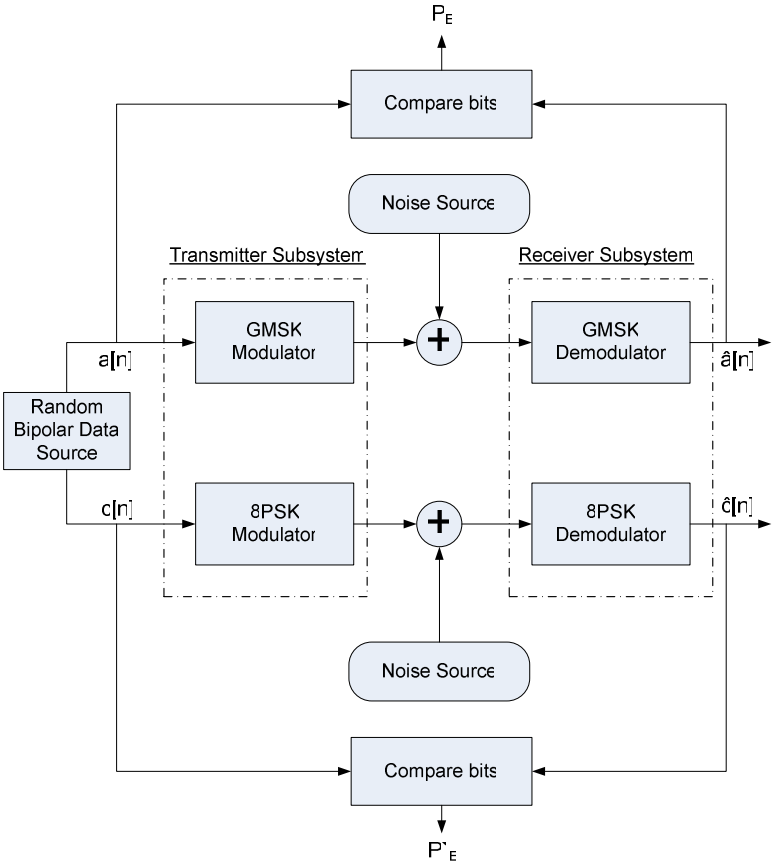

Figure 4.1 – Simulation setup for a simple communication system.

The main objective in this section of the project is to calculate different values of P_E and $P`_E$ for different amounts of noise and critically analyse the results.

4.1 INTRODUCTION TO WHITE GAUSSIAN NOISE AND BIT ERROR PROBABILITY

White Gaussian Noise

Noise refers to unwanted electrical signals that are always present in electrical systems, due to many sources such as thermally induced noise, electromagnetic radiation, celestial sources such as the sun, etc. The presence of noise superimposed on the signal causes different amplitude variations and thus distorts the signal.

According to [16]:

> *"We can describe thermal noise as a zero-mean Gaussian random process."*
>
> *"The primary spectral characteristics of thermal noise is that it's power spectral density is the same for all frequencies of interest in most communication systems. Therefore for a simple model for thermal noise assumes that its power spectral density $G_n(f)$ is flat for all frequencies"*

White noise is another term given for Thermal noise, and its Power Spectrum Density (PSD) is given by,

$$G_n(f) = \frac{N_0}{2} \text{ W/Hz} \tag{2.9}$$

Where, N_0 is the noise PSD, and the factor of 2 is included to indicate that $G_n(f)$ is a two sided power spectral density. Expression (2.9) implies that the PSD of white noise is a constant/flat for all frequencies.

[16] Also gives a derivation for the variance of white noise, which results in,

$$\sigma_n^{\,2} = \frac{N_0}{2} \tag{3.0}$$

And thus the standard deviation becomes,

$$\sigma_n = \sqrt{\frac{N_0}{2}} \tag{3.1}$$

In [18] they introduce a slight modification to expression (3.1) which was useful for simulation purposes; it is as follows,

$$\sigma_n = \sqrt{\frac{N_0 f_s}{2}} \tag{3.2}$$

Where, f_s is the sampling rate of the system.

Bit Energy to Noise Power spectral density Ratio

In communication systems, *'Average signal power to Average noise power ratio'* (S/N or SNR) is an important figure of merit. In Digital communication systems E_b/N_0 is more often used, and is the ratio between the Bit Energy to Noise Power Spectral Density.

One main objective of the project to be able to determine and plot the Bit Error Rate (BER) as a function of E_b/N_0 for the system illustrated in Figure 4.1.

Theoretical Bit Error Probability

It is shown in [17] that the theoretical Bit Error rates for different modulation schemes can be obtained by the Q-functions. There are several definitions of the Q-functions, expression (1.6) gives one definition, and expression (3.3) is another definition of the Q-function using the *erfc (*error function complementary) function.

$$Q(x) = \frac{1}{2} erfc\left(\frac{x}{\sqrt{2}}\right) \tag{3.3}$$

According to [17] the Bit error probability (P_b) for a M-ary PSK system is as follows,

$$P_b = \frac{1}{3} erfc\left(\frac{\left(\sqrt{\frac{2E_b}{N_o}\log_2 M}\,\sin\frac{\pi}{M}\right)}{\sqrt{2}}\right) \tag{3.4}$$

Where M=8 for 8-PSK, and M=16 for 16-PSK etc.

According to [15] the Bit error probability (P_b) for GMSK is as follows,

$$P_b = \frac{1}{2} erfc\left(\frac{\sqrt{\gamma \frac{E_b}{N_0}}}{\sqrt{2}}\right) \tag{3.5}$$

Where the term γ= 1.78 for BT= 0.3, γ= 1.93 for BT=0.5 and γ=2.0 for BT = ∞ (MSK)

A plot of the theoretical Bit Error Rates can be seen in Figure 4.2. It shows BER for BPSK, GMSK (BT=0.3) and 8PSK. The objective of the initial simulations is to arrive at close approximations to these curves through Monte Carlo simulations.

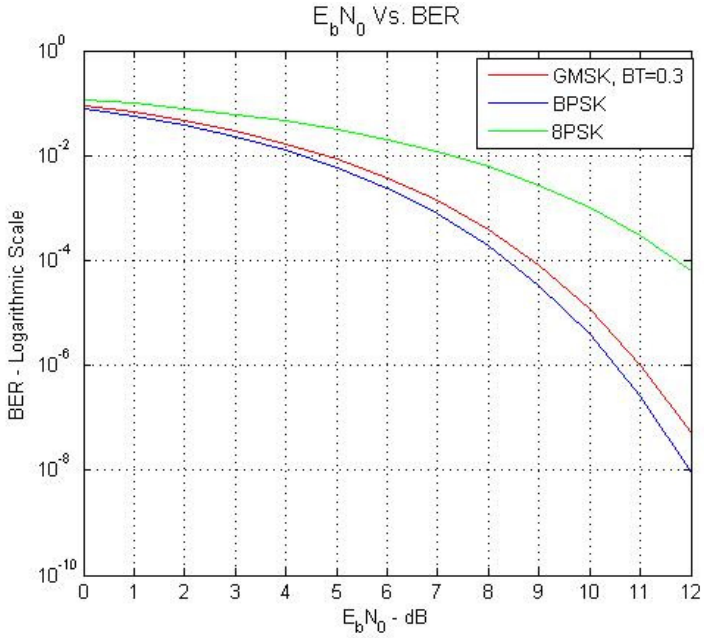

Figure 4.2 – Theoretical Bit Error Rates

4.2 IMPLEMENTING THE ADDITIVE WHITE GAUSSIAN NOISE CHANNEL

The code that was written for the AWGN channel is a direct implementation of equation (3.2) and is as follows:

```
function result = AWGN_channel(signal, EbNo_db)
  Eb = 1;                          % define bit energy (normalized to 1)
  No = (36*Eb)/(10^(EbNo_db/10)); % calculate No in terms of Eb
  sigma = sqrt(No);                % calculate standard deviation

  % generate and add noise to signal
  result = signal + (sigma * randn(1,length(signal)));
end

%introduce AWGN noise into the system.
tx_noisy_real = AWGN_channel(real(s_tx),EbNo); % add noise to I-channel
tx_noisy_imag = AWGN_channel(imag(s_tx),EbNo); % add noise to Q-channel
```

The first two lines of code were used to call the sub-function '`AWGN_channel`'. The noise component is added to the In-phase and Quadrature Channels separately. The parameters that are passed to the function are the real/imaginary component of the transmitted signal, and the E_b/N_0 value which is in the logarithmic domain (dB).

Inside the function, the factor 36 represents the sampling rate of the system. Also, the E_b/N_0 value is converted into linear domain before being processed. A one sided noise power spectrum density was assumed for the noise source.

Finally a Matlab built in function called '`randn`' is used to generate a vector (of equal length to the input signal vector) containing uniformly distributed random numbers with zero mean. The scalar '`sigma`' contains the standard deviation of the required gaussian noise source. By multiplying the output of the '`randn`' function with the standard deviation, the intensity of the noise source can be changed.

The simulation setup in Figure 4.1 was implemented by making the modulation and demodulation process *loop* itself for a range of E_b/N_0 values. For each run of the loop the E_b/N_0 value was incremented by 1. The ratio of bits in error to the total number of bits were calculated using the '`symerr`' function (as described in chapter 2 and 3) and appended to a temporary vector at the end of each cycle of the loop.(*Refer Appendix C1, C2.1 and C2.2 for the source code.)*

The Bit Energy in the GMSK simulation was not normalized to 1, therefore the following line of code was used to obtain the Bit energy. It calculates the average value of all the samples in the vector containing the transmitted signal. Simulation results were favourable when the following line was used, and resulted in an incorrect BER plot when $E_b = 1$ was used.

```
Eb = sum(abs(signal.^2))/length(signal);
```

4.3 PRELIMINARY ANALYSIS OF GMSK AND 8PSK

The following sections will critically analyse the GMSK and 8PSK modulation schemes, using the following analysis methods:

- Bit Error Rate plots.
- Eye diagrams.
- Power Spectrum Density plots.

Simulations were performed for GMSK and 8PSK with AWGN introduced, using 100,000 random bits. Therefore statistically it is sufficient to obtain an accurate result up to a BER of 10^{-4}.

4.3.1 Bit Error Rate Analysis

Figure 4.3 shows the resulting E_b/N_0 vs. BER plot for the simulation of *8PSK using Raised Cosine filtering* over an AWGN channel.

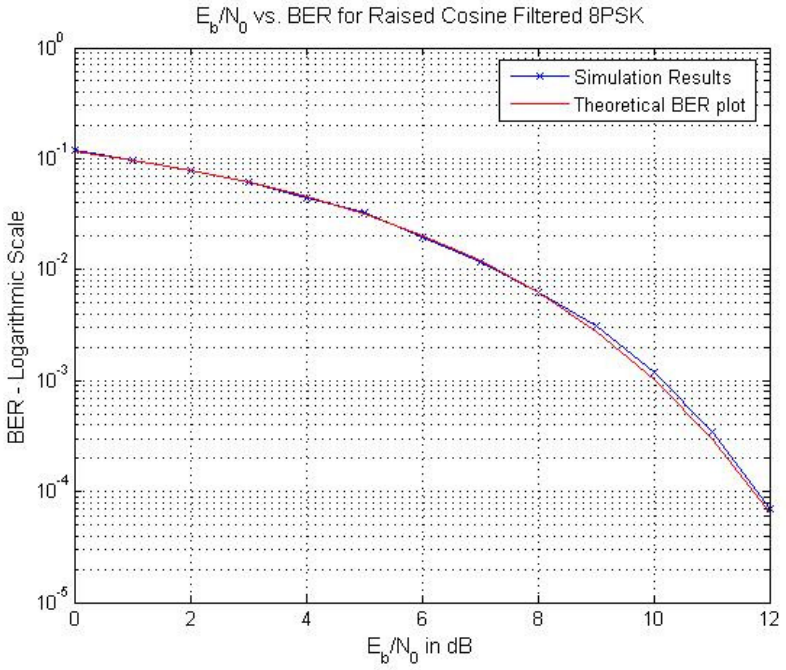

Figure 4.3 – Simulation result for Raised Cosine Filtered 8PSK over an AWGN channel

The resulting BER plot for the RC filtered 8PSK simulation (over AWGN) was verified by comparing it with the theoretical BER plot given by expression (3.4). The result for the RC filtered 8PSK, closely matched the theoretical BER plot, which implied that the simulation was successful.

Due to the memory issues in Matlab mentioned earlier, it was only possible to get statistically correct results of up to a BER of 10^{-4}.

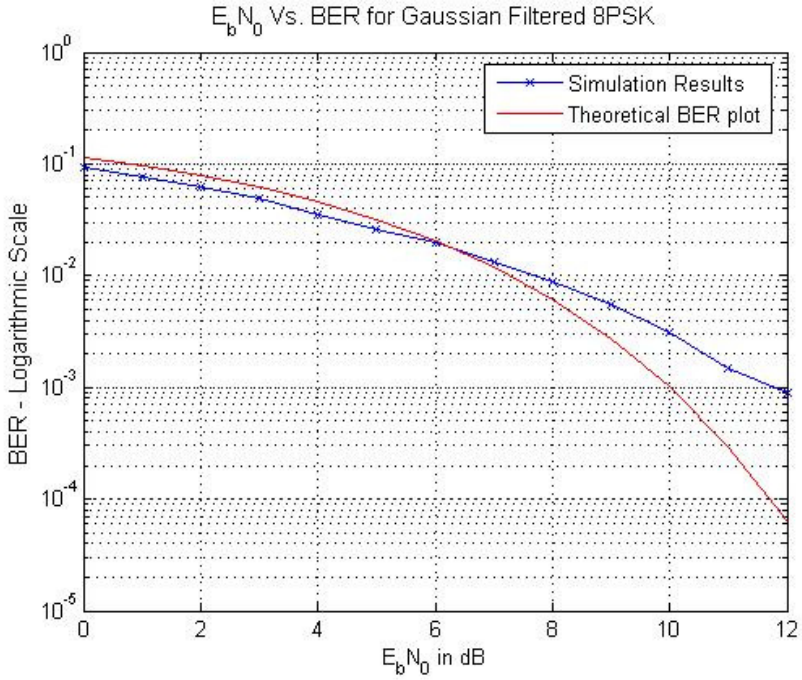

Figure 4.4 - Simulation result for Gaussian Filtered 8PSK over an AWGN channel

A theoretical plot for Gaussian Filtered 8PSK was not found in any of the digital/wireless communication text books, nor in any of the 'IEEE Explore' archives. The only possible validation available for the Gaussian filtered 8PSK was the theoretical BER plot given in expression (3.4).

It can be seen that up to an E_b/N_0 value of 6dB, the simulation results matched the theoretical plot but after E_b/N_0 = 6dB, the BER obtained was much higher than the theoretical. For an E_b/N_0 of 12dB, the results were just under 10^{-3} bits in error.

Figure 4.5 shows the resulting BER performance plot for the GMSK simulation, up to an E_b/N_0 of 10dB. This is because for over an E_b/N_0 value of 12dB, the theoretical GMSK plot showed a BER of approximately 10^{-7} and this was difficult to obtain because the maximum number of bits that can be simulated is 10^5, (due to the memory problem mentioned earlier in the report). The BER obtained from the simulation converged to zero after an E_b/N_0 of 10dB.

The GMSK simulation was validated using expression (3.5), with $\gamma = 1.78$. The results obtained closely matched the theoretical BER curve until an E_b/N_0 value of 4dB and then gradually separated from the theoretical curve. At E_b/N_0 = 10dB, the simulation produced about 6 in 100,000 bits in error, while the theoretical plot showed 1 in 100,000 bits in error. This means the simulation was 5 bits off target for an E_b/N_0 of 10dB.

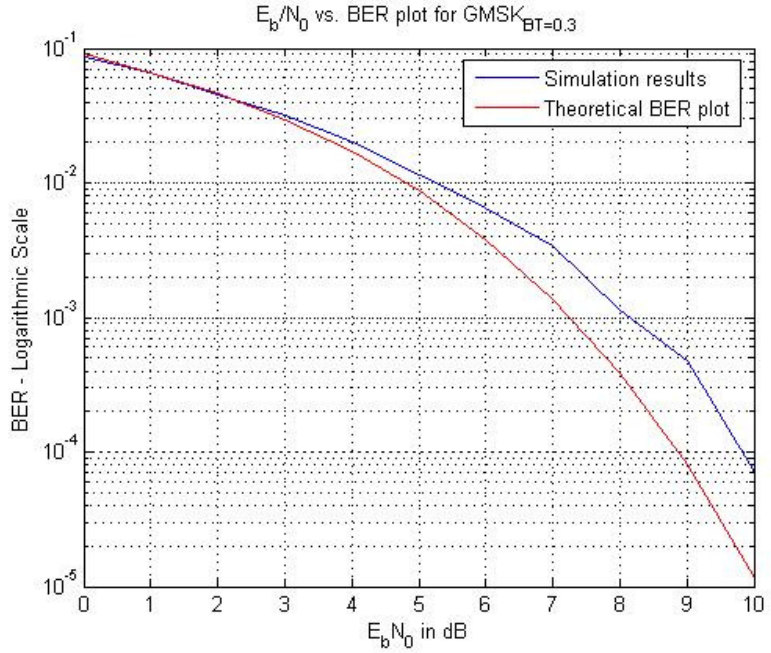

Figure 4.5 – Simulation result for GMSK over an AWGN channel

It is easier to view the difference between the 3 different modulation types, when the BER plots obtained are superimposed in one graph. Figure 4.6 shows a similar plot, where the GMSK and 8PSK simulation results were obtained in one graph, up to an E_b/N_0 = 10dB.

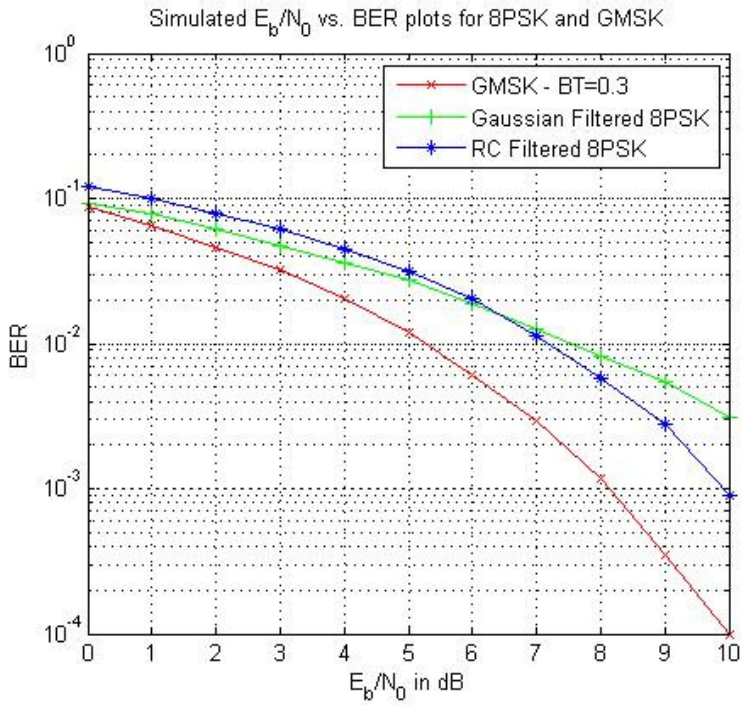

Figure 4.6 – Comparison of Bit Error Rates for GMSK and 8PSK over an AWGN channel

From Figure 4.6 is it clear that GMSK performs better in an AWGN channel, when compared with 8PSK. The GMSK BER curve has a much steeper fall than the 8PSK BER plots.

Higher order modulation schemes have many bits mapped on to one symbol, thus the decision maker at the demodulator is more complicated than lower order modulation schemes. Therefore the decision maker of a higher order modulation scheme is statistically capable of making more errors in the presence of AWGN. Similarly it is correct to say that modulation schemes with symbols in their IQ constellation that are closer to one another are less tolerant to noise. 8PSK is one such modulation scheme where the symbol spacing (on the IQ diagram) is far less than GMSK (only 2 symbols in IQ diagram). Thus a reduction in BER performance is seen in the presence of AWGN.

According to the GSM and EDGE specification [1], the maximum allowable BER is 10^{-3} for a voice channel. GMSK is able to obtain this BER with an E_b/N_0 of about 8dB, while RC filtered 8PSK just manages to reach the target with a E_b/N_0 of 10dB, and Gaussian filtered 8PSK manages to obtain the target with an E_b/N_0 of 12dB (Figure 4.4). This is a 4dB increase in signal to noise ratio, which the receiver has to cope with, when using EDGE (gaussian filtered 8PSK).

It can also be seen that the Gaussian filtered 8PSK scheme achieves the BER performance which is just under 0.5dB less than the RC filtered 8PSK scheme. However the results shown in this report are Raw BER results, because channel coding and other error detection methods have not been employed.

4.3.2 Eye Diagrams

In digital communication systems Eye diagrams are used to quickly and qualitatively measure a systems' performance. In terms of signal processing, an 'eye' is obtained by overlapping many random bits over each other. When these bits are filtered using a pulse shaping filter, they are no longer rectangular pulses, but shaped according to the used filter, and now when these bits are overlapped, they form an 'eye' shape. Figure 4.7 illustrates the formation of an eye diagram.

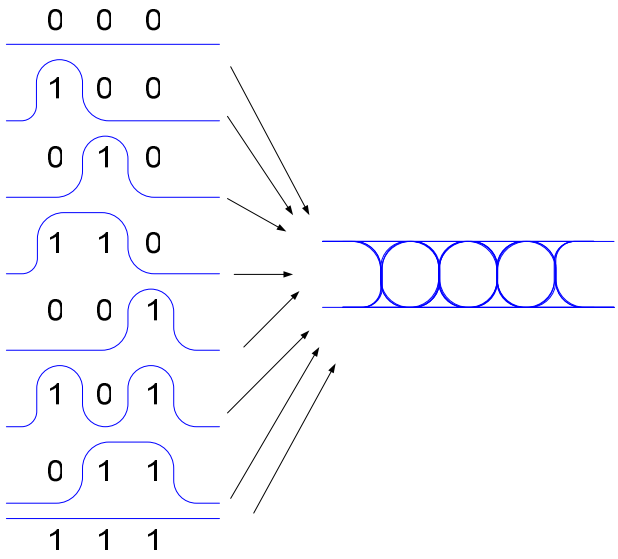

Figure 4.7 – Overlapping of pulse shaped bit sequences to form an Eye diagram

The optimum sampling time is obtained by the maximum opening of the eye. The decision maker at the demodulator should obtain samples at these maximum eye openings to avoid making errors. The more 'open' the eye diagram of a signal, the less susceptible it is to noise caused by the channel.

Eye diagrams are usually obtained at either the transmitter or receiver ends, depending on which sub-system is being examined. Common signal impairments such as clock jitter, poor synchronization of the receiver circuitry, signal attenuation can be examined by observing the eye diagram on an oscilloscope.

To properly understand the generation of an eye diagram, a small Matlab script was written to obtain the eye diagram. The function contains a loop which plots segments of a vector over each segment. In each run of the loop consecutive segments are obtained and plotted over each other, using the '`hold on`' command in Matlab. (*Refer Appendix C4 for source code*)

The eye diagram function written shows the amplitude in the y-axis and samples in the x-axis. Seeing the eye diagram with respect to samples, instead of time (symbol period), helped to visually obtain the optimum sampling rate required at the demodulator. Since the sampling rate used for all the simulations is 36Hz, all the eye diagrams show a maximum opening at the 36^{th} sample as intended.

Eye Diagrams obtained from the GMSK simulation.

Figures 4.8(a – d) shows the eye diagrams of different Gaussian Filters with different BT values.

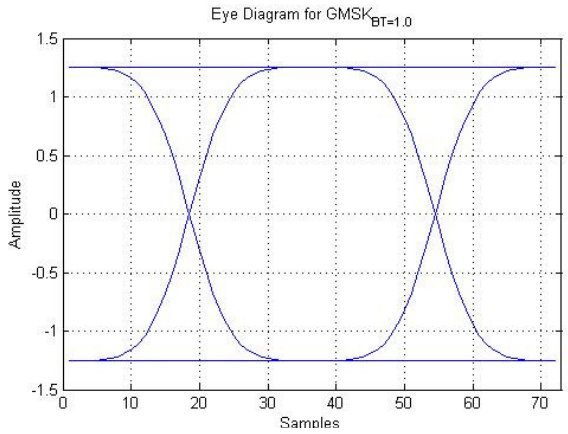

Figure 4.8(a)
Eye Diagram of GMSK, BT=1.0

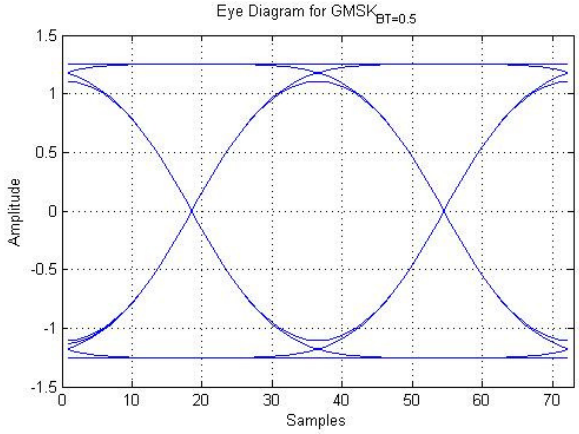

Figure 4.8(b)
Eye Diagram of GMSK, BT=0.5

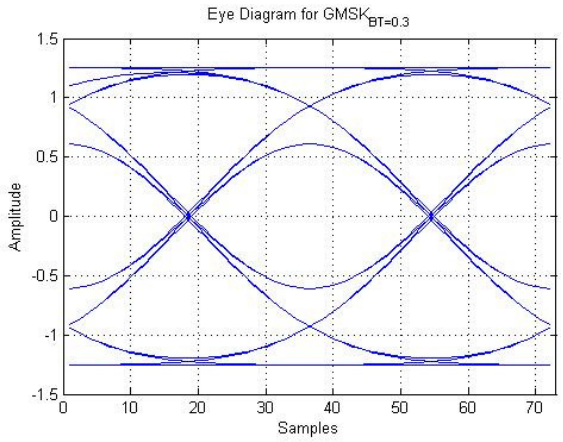

Figure 4.8(c)
Eye Diagram of GMSK, BT=0.3

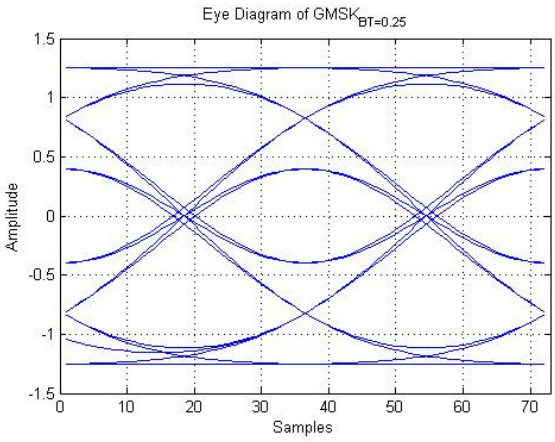

Figure 4.8(d)
Eye Diagram of GMSK, BT=0.25

Figure 4.9 points out the important areas in an eye diagram, such as maximum eye opening, symbol zero crossing, the ISI component etc.

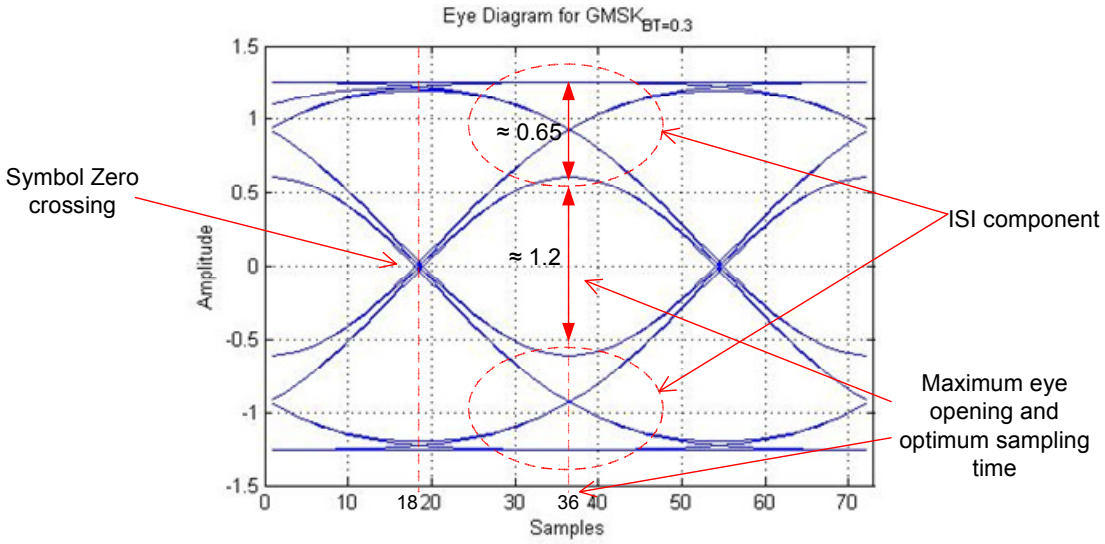

Figure 4.9 – Key Characteristics of an Eye Diagram (for GMSK, BT=0.3)

With reference to Figure 4.8(a) to Figure 4.8(d), it is observed that as the BT value of the gaussian filter increases the amount of ISI caused by the filter to the signal is reduced, and the maximum eye opening is larger for BT values closer to 1.0. For smaller values of the BT product the ISI component is larger. When BT = 1.0, it is approximately equal to MSK, where no ISI is introduced to the signal (no filtering performed).

However reduction of ISI is at the expense of more bandwidth usage, because as Figure 2.10(b) in Chapter 2 showed the bandwidth of the GMSK signal will increase as the BT product of the Gaussian filter is increased. Therefore a balance between amount of ISI introduced to a system, and the amount of bandwidth used has to be made according to a

given system specification. As the ISI component increases the level of Signal to Noise Ratio (SNR) decreases.

In order to investigate the effect of noise to the communication system, the eye diagram in Figure 4.10 was generated for GMSK after AWGN noise (E_b/N_0 = 10dB) had been introduced. The maximum opening of the eye is visible, but is located at the 46^{th} sample, because of the convolution process in the simulation. Distortion to the transmitted signal is clearly visible, but the decision maker still manages to obtain the correct symbols with only a BER of 10^{-4}.

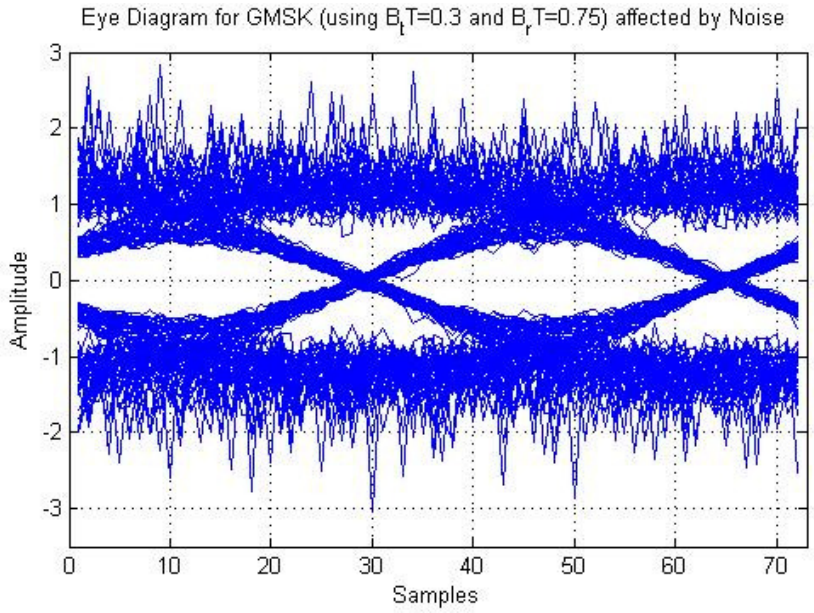

Figure 4.10 – Eye diagram for GMSK corrupted by Noise.

Eye Diagram obtained from the Gaussian Filtered 8PSK simulation

Figure 4.11 shows the eye diagram for the Gaussian filtered 8PSK signal. The eye diagrams were obtained in a similar fashion. The function written to obtain the GMSK eye diagram was modified to obtain separate eye diagrams for the In-phase and Quadrature components.

Once again as with the GMSK eye diagram a large ISI component is present in the Gaussian Filtered 8PSK signal. The gaussian filtering (non Nyquist filter) is the element in the modulator which causes this ISI but is vital to obtain the tight spectrum which is shown in the PSD plots to follow.

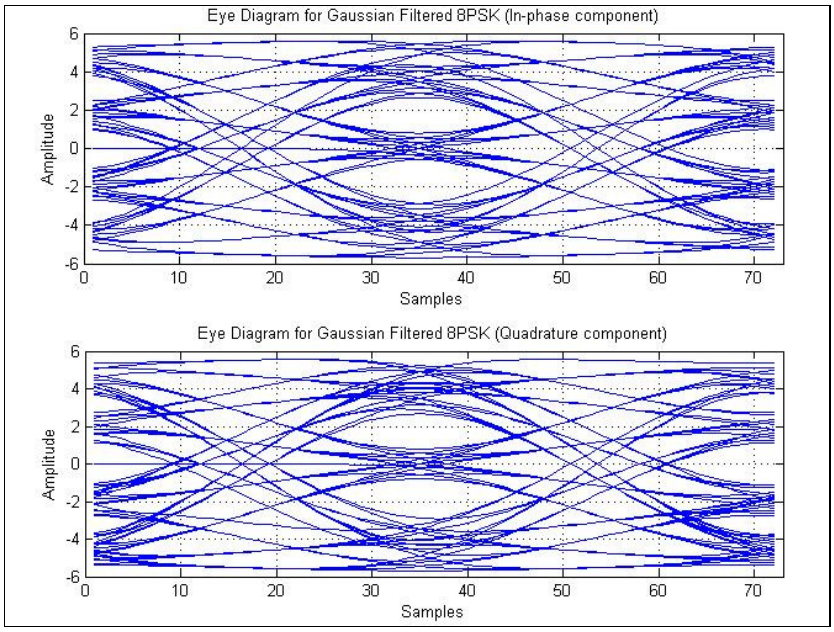

Figure 4.11 – Eye diagram for Gaussian Filtered 8PSK

Maximum eye opening in Figure 4.11 is shown at 36, as expected, but only two clear eye openings are shown, due to the heavy ISI components distorting the other two eye openings. This later adversely affects the BER as in Figure 4.4, because the decision maker at the demodulator is finding it difficult to detect the correct symbols, due to the amount of ISI.

Eye Diagram obtained from the Gaussian Filtered 8PSK simulation

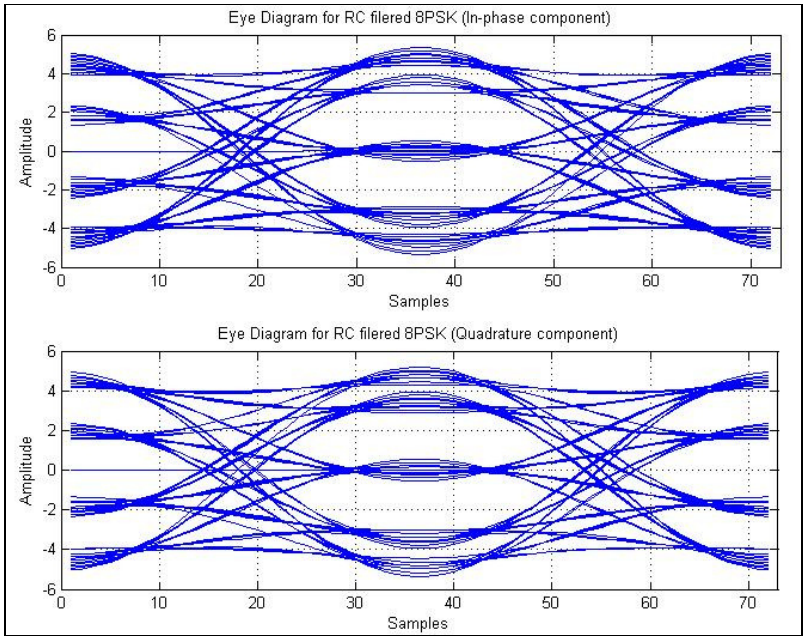

Figure 4.12 – Eye diagram for Raised Cosine Filtered (β =0.5) 8PSK

Figure 4.12 shows the eye diagram obtained from the RC filtered 8PSK simulation. The eye diagram shows four eye openings unlike Figure 4.11 which only showed 2. This is because

8PSK has four different voltage/amplitude levels in each of it's I and Q components after symbol mapping. There is still a certain amount of ISI introduced, because of β = 0.5, but still a great deal less than the ISI introduced in the GMSK and GF-8PSK simulations.

Figures 4.13(a) and 4.13(b) shows the Eye diagrams for different values of the roll-off factor (β). Only the real component is shown, because the imaginary component looks similar.

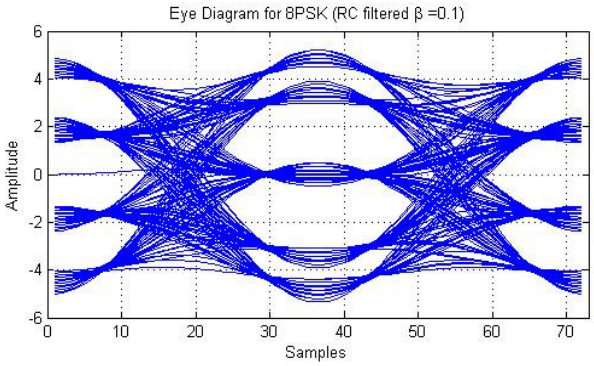

Figure 4.13(a)
In-phase, Eye diagram of RC filtered 8PSK (β = 0.1)

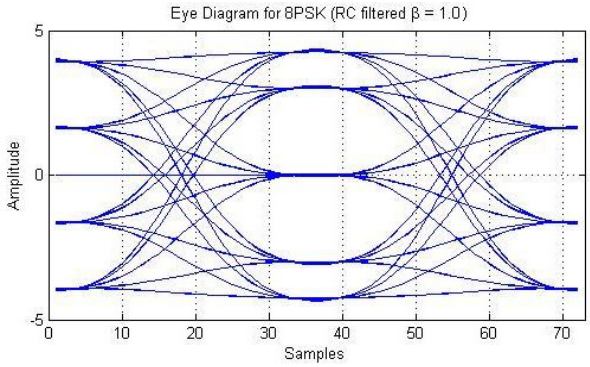

Figure 4.13(b)
In-Phase, Eye diagram of RC filtered 8PSK (β = 1.0)

From Figures 4.13(a) and 4.13(b) it can be observed, that as the roll off factor is increased, the level of ISI introduced o the system decreases. As β is increased the ISI level is decreased, but the bandwidth of the signal will be increased. Once again a suitable value for the roll-off factor should be calculated, to make a trade-off between bandwidth and ISI.

4.3.3 Power Spectral Density

According to Sklar [16],

> *"The spectral density of a signal characterizes the distribution of the signals energy or power in the frequency domain"*

From the Power Spectral Density (PSD) it is possible to determine the channel bandwidth required to transmit the information bearing signal. The Power spectrum *S(f)* can be calculated as follows [13];

$$S(f) = \lim_{T \to \infty} \frac{1}{T} | X_T(f) |^2 \qquad (3.6)$$

Where $X_T(f)$ is the signal in frequency domain, and T is the duration of the signal.
There are a few important points to note before comparing the three modulation schemes in terms of PSD, they are,

- The simulation was performed at Baseband. Therefore the frequencies shown in the PSD are narrowband frequencies.
- Different line coding techniques affect the PSD of the signal. This is not an issue, because all the simulations used Bipolar NRZ signalling.

Implementing the Power Spectral Density plots

The Signal processing toolbox in Matlab has a built in function called '`pwelch`' which uses Welch's averaged modified Periodogram method of spectral estimation. The periodogram is the simplest and fastest and most frequently used PSD estimation algorithm. It uses FFT with *N* points to estimate the spectrum. The 'pwelch' function splits the time series into segments which overlap about 50% with each other, and applies a hamming window to each segment to smooth out the ripples/overshoot. (*Refer Appendix C5 for source code*)

The following are the parameters used for this function to generate the PSD plots :

- Window length = 2^{15} (this is also the same as the FFT size).
- Sampling frequency = 36Hz

Figure 4.14 shows the PSD plots obtained for GF-8PSK, RCF-8PSK and GMSK. It was easier to view the difference between the spectrums when they were superimposed over each other.

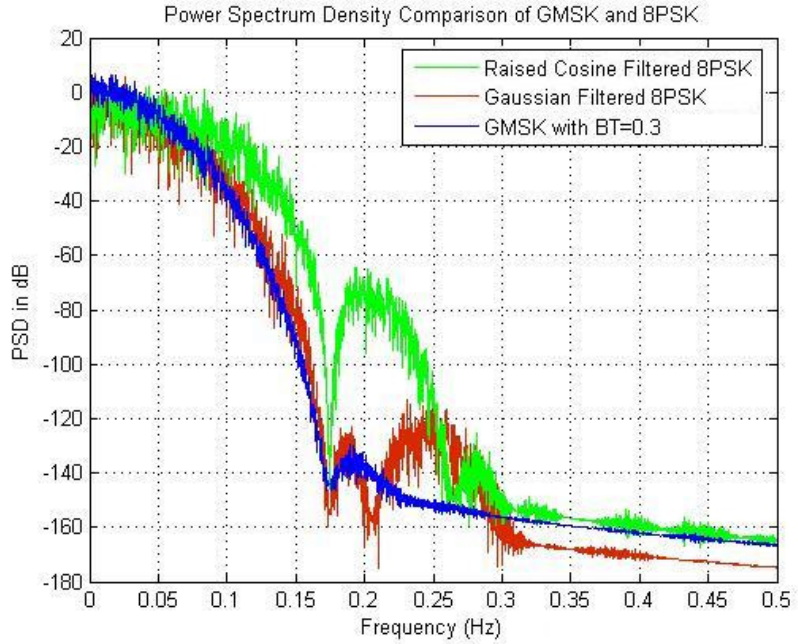

Figure 4.14 – Power Spectrum Comparison of GMSK and 8PSK

The PSD plots shown in Figure 4.14 shows main lobes, accompanied by first and second lobes. An almost zero DC component is shown in by the PSD, due to the bipolar signalling used.

It can be observed that the first main lobe of the Gaussian filtered 8PSK and GMSK spectrum has an equal main lobe of 0 to 0.17Hz in width. This is due to the similar type of pulse shape filtering that was performed in both modulation schemes. More power can be found in the main lobes, and a very steep attenuation in the main lobe can be seen. In reality the EDGE spectrum has to be narrow enough to be able to fit into the GSM spectrum. We can see the filtering that was used at the modulator has helped to enforce this fact.

One difference between the Gaussian filtered 8PSK signal and the GMSK signal is that the GLP filtered 8PSK signal has visible side lobes with peaks at 0.18Hz and 0.25Hz, while the GMSK PSD seems to have only a main lobe.

The RC filtered spectrum however is wider than the other two (from 0Hz to about 18Hz), has a higher first lobe and the level of attenuation in the main lobe is not as steep as the other two modulation types. The majority of the power is concentrated in the main lobe. The first side lobe also contains a considerable amount of power (about 50dB higher than the GMSK side lobe), when compared with the other two signals.

Figure 4.15 shows the PSD for different values of the BT product in the Gaussian filter used in GMSK. As mentioned in Section 2.6.1 in Chapter 2, the BT product is directly proportional to the bandwidth of the Baseband GMSK signal. Higher BT values such as 0.5 and 1.0 show wider power spectrums and higher power values in their main and side lobes. The GSM engineers could have chosen a BT value of 0.2 but it would have server consequences with the bit error rate.

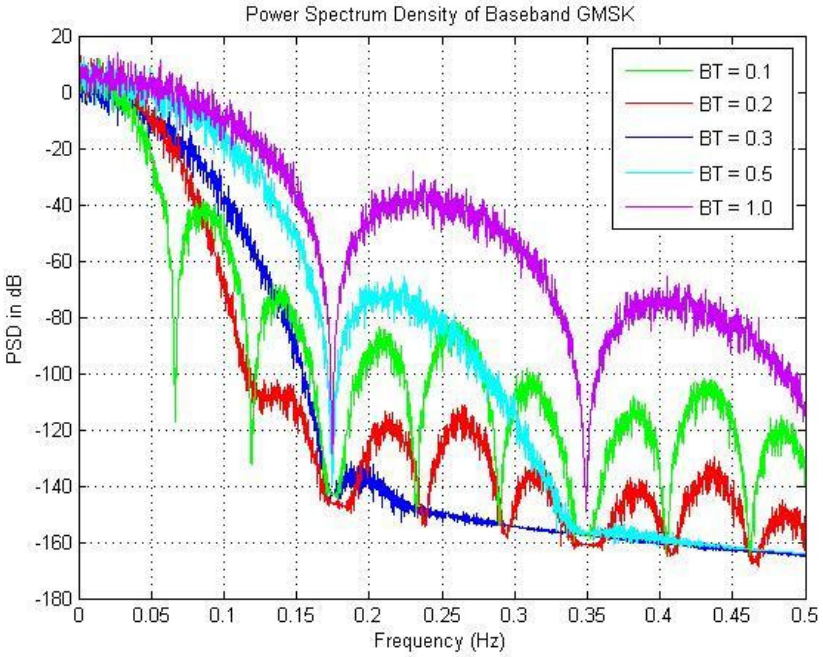

Figure 4.15 – Power Spectral Density of GMSK for different values of the BT product

4.3.4 Summary of the Preliminary Results

Table 4.1 contains a brief summary of the preliminary results spread out in this chapter.

Key Characteristic of Modulation Scheme	**GMSK with BT =0.3 (GSM)**	**Gaussian Filtered 8PSK (EDGE)**	**Raised Cosine Filtered (β=0.5) 8PSK**
Required E_b/N_0 to achieve BER of 10^{-3} *(lower is better)*	8dB	12dB	10dB
Is ISI visible in the Eye diagram? *(the less visible the better)*	Clearly visible ISI component	Severe ISI component. Only 2 symbol levels out of 4 are visible	Least amount of ISI out of the three
Power Spectral Density - bandwidth of main lobe *(lower is better)*	≈ 0.17Hz	≈ 0.17Hz	≈ 0.18Hz
Power Spectral Density - Power of first Side lobe *(lower side lobes are more favourable)*	Negligible first side lobe	≈ 130dB lower than Main lobe	≈ 70dB lower than Main lobe
Raw Bit Rate, according to simulation parameters. *(higher is better)*	1 Symbol = 1 bit ∴ Bit rate = 1bps	Symbol rate = 1sps 1 Symbol = 3 bits ∴ Bit rate = 3bps	Symbol rate = 1sps 1 Symbol = 3 bits ∴ Bit rate = 3bps

Table 4.1 – Summary of the Preliminary Results

The results shown in Table 4.1 can be interpreted as follows:

- In terms of BER, GMSK outperforms 8PSK, obtaining the maximum allowed bit error rate with an E_b/N_0 value 4dB less than the GF-8PSK and 2dB less than RCF-8PSK.
- In terms of Power efficiency, 8PSK produces more bits in error for the same power used. Thus making 8PSK power inefficient.
- In terms of spectral efficiency, the GF-8PSK modem performance is better, because it produces more bits within the same spectrum as GMSK.
- In terms of tolerance to ISI, the RCF-8PSK modem performs better, because it shows a clear Eye diagram with a minimum ISI component. This in turn would be favourable for its performance in a Fading environment.

CHAPTER 5
MODEM SIMULATION OVER FADING CHANNELS

This chapter of the report contains the simulation results of more realistic mobile channels. A brief introduction to fading channels is given, the simulation setup is introduced and results are presented.

5.1 INTRODUCTION TO MULTIPATH FADING CHANNELS

Fading channels can be mainly categorised as follows:

- Large scale fading (due to motion of the signal over large areas)
- Small scale fading (also known as Multipath fading)

Multipath and motion induced fading are two of the most severe performance limiting conditions in a wireless radio channel. In any wireless communication channel, there is more than one path that the signal can travel to get from the transmitter to the receiver. The presence of Multipath may be due to refraction or reflection from objects in the environment.

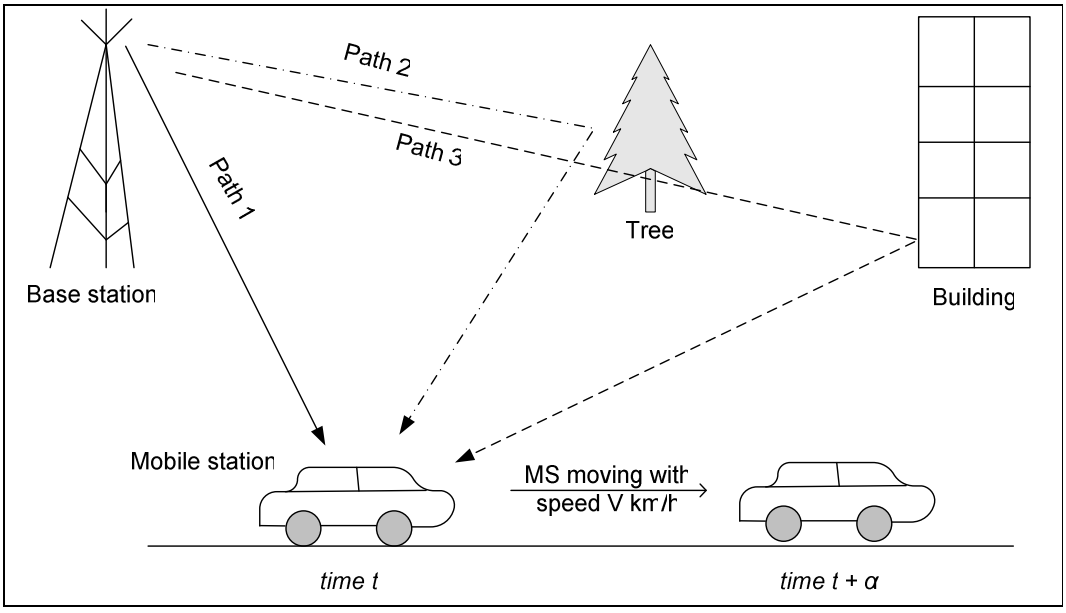

Figure 5.1 – Typical Multipath Fading Scenario

Figure 5.1 illustrates a typical Multipath fading scenario where there may be three distinct paths a radio signal can take to arrive to the mobile station. The total signal that arrives at the receiver is made up of the sum of the three signals.

Rayleigh Fading

Small scale fading is called Rayleigh fading if there are multiple reflective paths that are large in number and if there is no Line of Sight (LOS) component. The envelope of such a received signal can be statistically described by the Rayleigh Probability Distribution Function (PDF) given by,

$$P(r) = \frac{r}{\sigma^2} e^{\left(\frac{-r^2}{2\sigma^2}\right)} \tag{3.7}$$

Where,

r = envelope of fading channel power
σ^2 = average signal power of received signal

Rician Fading

Rician Fading is a type of Small scale fading where there are multiple reflection paths but also a LOS path, which is stronger in power than the rest. A Rician faded signal has an envelope statistically described by the Rician PDF given by,

$$P(r) = \frac{r}{\sigma^2} e^{-\left(\frac{r^2+\beta^2}{2\sigma^2}\right)} I_0\left(\frac{\beta_r}{\sigma^2}\right) \tag{3.8}$$

Where, $I_0(x) \approx \frac{e^x}{\sqrt{2\pi x}}$ is the zero order modified Bessel Function, and

β = amplitude of direct signal.

In certain literature the term 'fading' is used because in television transmission the second attenuated signal received in a Rician/Rayleigh environment may cause a duplicate faded (low intensity) picture next to the original one. Similarly in mobile communications, the multiple signals arriving at the receiving antenna will be delayed duplicates of the original signal. D

Different demodulation techniques are used to cope with this type of distortion and to maximise the signal to noise ratio, but for this project the simple coherent demodulators implemented in Chapter 2 and 3 will be used.

5.2 THE THREE-RAY FADING CHANNEL SIMULATOR

The approach that will be used to simulate Rayleigh and Rician Fading is a simple three ray model, in which only three different paths are assumed to be taken by the transmitted signal. The scenario is much like one illustrated in Figure 5.1. The three rays used in the simulation are characterised by the two properties; delay and gain. The flow chart in Figure 5.2 illustrates the simulation setup that will be used, which is a modified version of the simulation performed in [18].

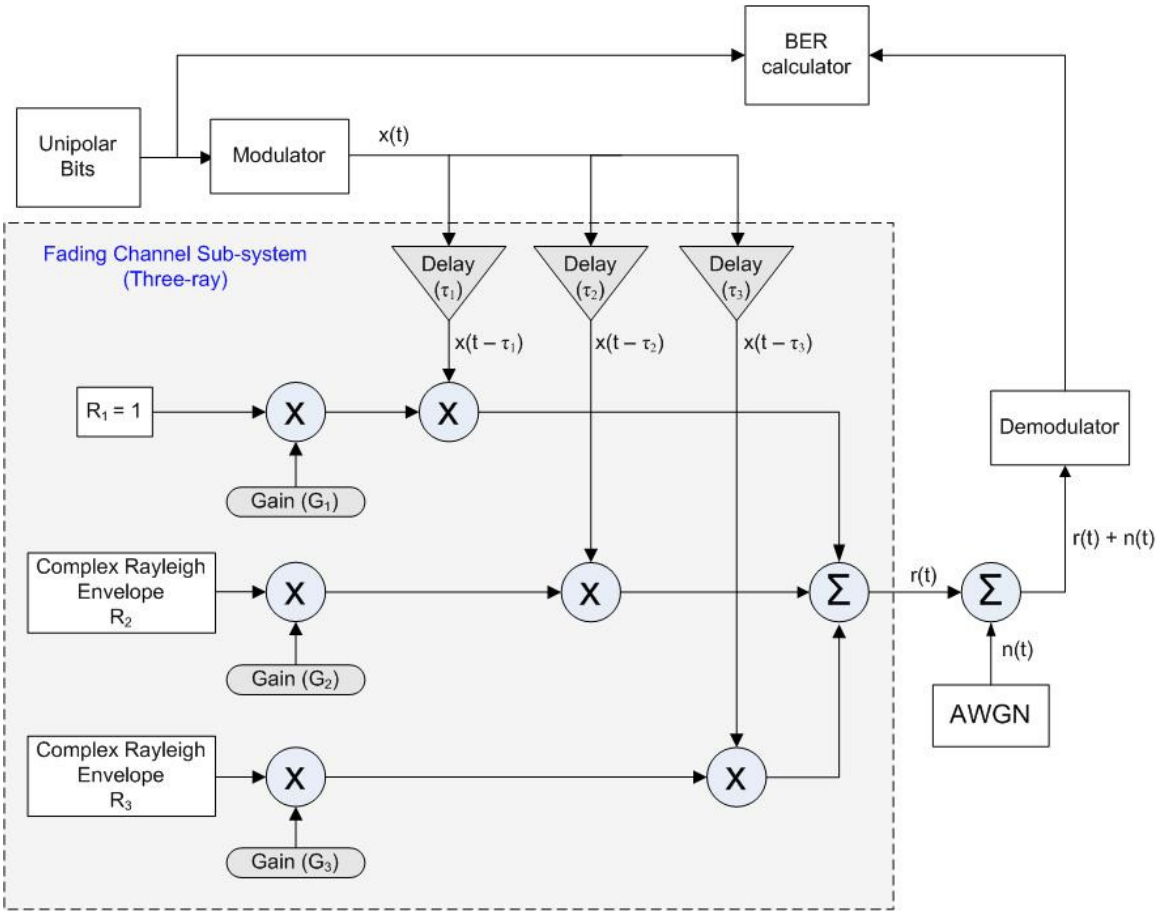

Figure 5.2 – Simulation setup for the Three Ray Fading Channel Simulator

Unlike the AWGN channel, fading channels are considered as multiplicative noise. Thus the multiplication operations are seen in Figure 5.2. The Complex Rayleigh envelopes are used to ensure the multiplicative noise introduced take a Rayleigh distribution, unlike the Gaussian distributed White noise which is added at the receiver. As shown in Figure 5.2 an AWGN source is also incorporated to the model, because in reality a signal undergoes additive and multiplicative disturbances. (fading caused by the mobile environment and thermal noise at the receiver).

The total signal that arrives at the receiver is made up of the sum of a large number of scattered components. As given in [18] by using the Central Limit theorem, it is shown that

these components have random phases and can be modelled as Complex Gaussian Random numbers.

The received signal for this example can be written as:

$$r(t) = G_1R_1x(t-\tau_1) + G_2R_2x(t-\tau_2) + G_3R_3x(t-\tau_3) \quad (3.9)$$

And the first term in expression (3.9) becomes the LOS component when $\tau_1 = 0$, or in other terms the first element can be completely left out if there is no LOS component (i.e. when $G_1=0$). The second and third terms represent the Rayleigh faded signals.

In the simulation implemented,

$\tau_1 = 0$ (constant – no delay),
$R_1 = 1$ (constant – no Rayleigh envelope),

and $G_1 = \begin{cases} 1, & Rician\ Fading \\ 0, & Rayleigh\ Fading \end{cases}$

G_1 term is either made 0 or 1 to make the received signal Rayleigh or Rician respectively.

Several assumptions were made for this simulation, they are:

- The fading in the channel, affects only the amplitude, and not the phase of the transmitted signal.
- The power attenuation of the transmitted signal with respect to the distance travelled is not taken into account.
- No Doppler spreading was taken into account, thus implying the MS is stationary.
- No phase synchronisation and symbol timing recovery is performed in the demodulator.

5.2.1 Implementation of the Three-Ray Fading Channel

Since the R_2 and R_3 components can be modelled as Gaussian Random processes, the Matlab built in functions '`randn`' and '`abs`' functions were used to obtain the Rayleigh envelopes.

E.g.:

```
ray2 = abs(randn(1,length(signal))+ j*randn(1,length(signal)));
```

Figure 5.3 shows the verification as to whether the resulting vector had a correct Rayleigh PDF. The Histogram of the vector '`ray2`' was obtained which gives an approximation to its PDF. As seen from Figure 5.3, the vector's PDF is approximately Rayleigh, and sufficient for the integrity of the simulations.

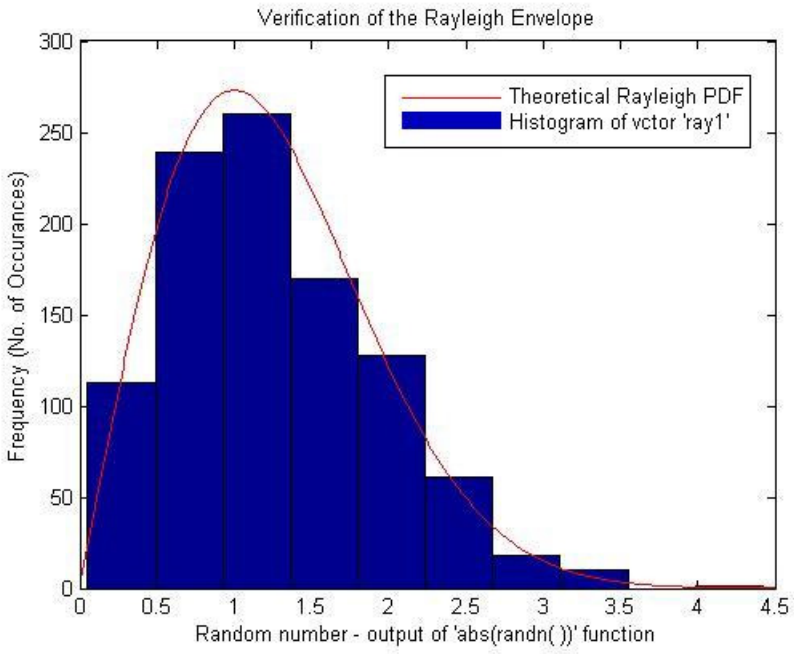

Figure 5.3 – Verification of the Rayleigh Envelope

The delays were introduced to the signal by padding the modulated signal vector with zeros as follows:

```
path1 = [zeros(1,delay1*sps) signal];

% chop off the end to agree with original lengths
path1_chop = path1(1: (length(path1)-(delay1*sps)));
```

The vector '`sps`' = 36 for all simulations, and is the sampling frequency. The delay factor was varied between 0 and 1 to vary the delay of the signal up to a symbol period (36 samples per symbol). The resulting vector had to be cropped at the end, to match with the original signal vector length. *Appendix C6* contains the source code for the function written to implement the three-path fading simulator named '`rayleigh_sim`'.

5.2.2 Fading Channel Simulator Parameters

The previously mentioned implementation procedures were was performed for all three delayed signal. However the parameters were changed slightly, to match the modulation schemes.

Similar parameters were chosen for the Gaussian Filtered 8PSK and the GMSK modulation techniques, but much tougher fading conditions were introduced to the RC filtered 8PSK modulator. This was because, the RC filtered 8PSK modem used a Nyquist filter, which reduced ISI, and therefore it should be able to handle much more severe conditions of fading. The Tables 5.1, 5.2 and 5.3 below give the parameters for each simulation performed. Four Rician channels and two Rayleigh channels were simulated.

Scenario	Gain1	Gain2	Gain3	Delay1	Delay2	Delay3	Description
0	-	**-**	**-**	**-**	**-**	**-**	Theoretical, using Q-Function (Validation Plot)
1	1.00	0.00	0.00	0	0	0	No delay, 1 LOS component
2	1.00	0.10	0.10	0	0.2	0.1	Rician fading
3	1.00	0.10	0.20	0	0.2	0.2	Rician fading
4	1.00	0.25	0.25	0	0.3	0.25	Rician fading
5	0.00	0.20	0.20	0	0.1	0.1	Rayleigh fading
6	0.00	0.20	0.20	0	0.5	0.2	Rayleigh fading

Table 5.1 – Fading Parameters for GMSK (3-path simulator)

Scenario	Gain1	Gain2	Gain3	Delay1	Delay2	Delay3	Description
0	-	**-**	**-**	**-**	**-**	**-**	Theoretical, using Q-Function (Validation Plot)
1	1.00	0.00	0.00	0	0	0	No delay, 1 LOS component
2	1.00	0.10	0.10	0	0.2	0.1	Rician fading
3	1.00	0.10	0.20	0	0.2	0.2	Rician fading
4	1.00	0.10	0.50	0	0.5	0.25	Rician fading
5	0.00	0.20	0.20	0	0.1	0.1	Rayleigh fading
6	0.00	0.20	0.20	0	0.5	0.2	Rayleigh fading

Table 5.2 – Fading Parameters for GLP filtered 8PSK (3-path simulator)

Scenario	Gain1	Gain2	Gain3	Delay1	Delay2	Delay3	Description
0	**-**	**-**	**-**	**-**	**-**	**-**	Theoretical, using Q-Function (Validation Plot)
1	1.00	0.00	0.00	0	0	0	No delay, 1 LOS component
2	1.00	0.10	0.00	0	1	0	Rician fading
3	1.00	0.20	0.20	0	1	0.5	Rician fading
4	1.00	0.20	0.10	0	1	0.75	Rician fading
5	0.00	0.20	0.20	0	0.5	0.5	Rayleigh fading
6	0.00	0.20	0.20	0	0.5	0.5	Rayleigh fading

Table 5.3 – Fading Parameters for RC filtered 8PSK (3-path simulator)

Note: Table 5.2 originally had the same parameters as Table 5.1, but Scenario 4 had to be changed, due to unexpected results, that were obtained. For Gaussian Filtered 8PSK, using the same parameters for Scenario 4 in Table 5.1 gave better results than the Theoretical. This was probably because, the delays used were small and the fading gains were small as well, thereby it increased the original power of the transmitted signal and thus performed better when faced with noise.

5.2.3 Results of the Fading Channel Simulation (Three-Ray Simulation)

Figures 5.4, 5.5 and 5.6 present the Simulations obtained for the different modulations schemes. 100000 bits were simulated for all the scenarios to get more accurate results. As mentioned previously, it was only statistically possible to get results up to E_b/N_0 = 10dB for the GMSK fading simulation.

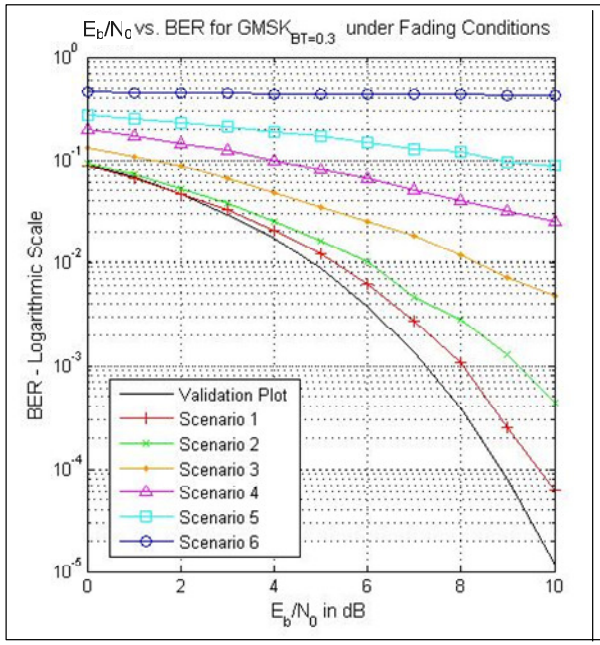

Figure 5.4
Fading Simulation Results for GMSK

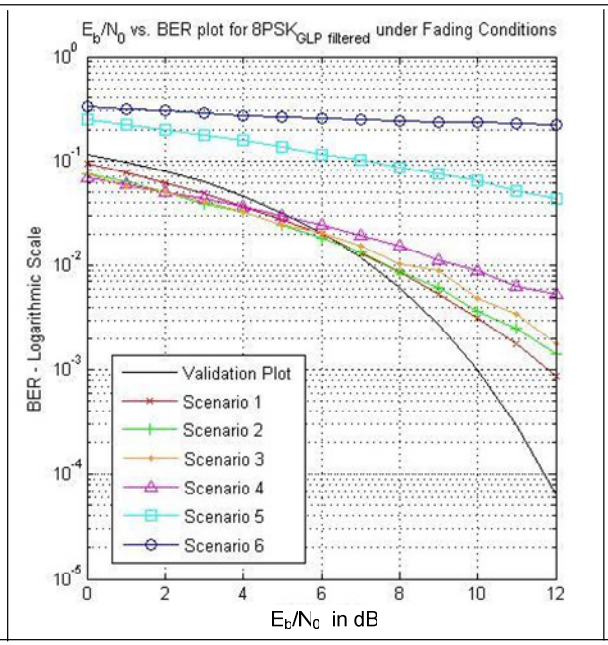

Figure 5.5
Fading Simulation Results for Gaussian Filtered 8PSK

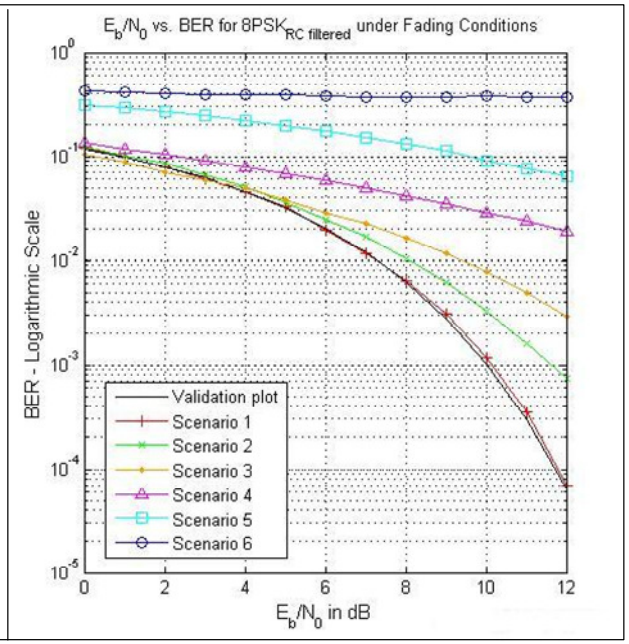

Figure 5.6
Fading Simulation Results for RC Filtered 8PSK

5.2.4 Analysis of the Three Path Simulator Results

From the obtained E_b/N_0 vs. BER plots shown in Figures 5.4 to 5.6, the following observations can be made about the three modulation techniques. (Note: parameters are different for RC filtered 8PSK)

- Comparison between GMSK and Gaussian filtered 8PSK (GF-8PSK):
    - For Scenario 2 at E_b/N_0 = 10dB, GMSK has a BER of under 10^{-3} while GF-8PSK has a BER of just under 10^{-2}.
    - For Scenario 3 both GMSK and GF-8PSK have similar results having a BER of just under 10^{-2} for a E_b/N_0 of 10dB.
    - For Scenario 4 GF-8PSK has a better performance maintaining a BER of 10^{-2} for E_b/N_0 = 10dB, while GMSK has a BER of over 10^{-1}.
    - For Scenario 5, GF-8PSK performs slightly better than GMSK, because it has a BER of below 10^{-1} at E_b/N_0 = 10dB, while GMSK is exactly 10^{-1}.
    - For Scenario 6, the GF-8PSK modulation shows less than 0.5dB improvement than GMSK, but still both are heavily influenced by fading.
    - To summarize, GMSK performs better in Rician fading environments where the delay of a signal is less than half a symbol period, but as the delay gets larger GMSK is severely subjected to ISI. GF-8PSK performs slightly better than GMSK in Rayleigh environments, in the sense that it can obtain the same error rates as GMSK, but with less signal power.

- Analysis of RC filtered 8PSK(RCF-8PSK) simulation results:
    - The simulation parameters used for RCF-8PSK are all higher than the other two simulations. It manages to maintain the same error rates as GMSK and GF-8PSK for delays over one symbol period (according to Scenario 2 and 3). Thus performances in Rician environments are much better than GMSK and GF-8PSK.
    - When faced with a Rayleigh channel, in Scenario 5 where almost a one symbol duration delay (in total) is induced, the RCF-8PSK manages to obtain a BER of about 10^{-1} for an E_b/N_0 of 10dB. This is a lot better than the other two, which cannot cope with more than a half a symbol delay in a Rayleigh environment.

5.3 INTRODUCTION TO THE DOPPLER SPREAD

The simulations performed in Section 5.2 do not take into account the motion of the mobile station. The speed of the mobile determines how fast the channel is fading. Motion of the MS causes a Doppler Shift to the received signal components. The Doppler shift is the apparent shift or broadening of the spectrum of the transmitted signal due to motion.

For a vehicle or mobile station moving at the constant velocity v, the Doppler frequency (shift) f_d according to [19] is given by,

$$f_d = \frac{v \cos\theta}{\lambda} \tag{4.0}$$

Where, θ = the angle between the arriving wave and direction of motion, and
λ = wavelength of the transmitted signal.

The maximum Doppler frequency when $\theta = 0$ is,

$$f_m = \frac{v \cos 0}{\lambda} = \frac{v}{\lambda} \tag{4.1}$$

For the purpose of all the simulations $\lambda = (3 \times 10^8) / (900 \times 10^6) = 0.333$ (assuming GSM 900)

The Doppler Spread (B_d) is the difference between the Maximum and minimum values of the Doppler Frequency f_d.

The term 'Fast fading' is given when $B_s << B_d$ and the term 'Slow fading' is used when $B_s >> B_d$.

5.4 IMPLEMENTING THE RAYLEIGH FADING SIMULATOR

Different methods have been discussed to simulate a fading channel in [19]. All methods introduce the employ the use of the maximum Doppler shift which takes into account the motion of the mobile station. Some of these methods are:

- Filtered Gaussian Noise Method
- Sum of Sinusoids Method (SOS)
- Autoregressive models

Out of the three methods, the easiest to understand and implement was the Sum of Sinusoids method, which is also known as the Jake's fading simulator.

[19] offers a modified version of Jakes SOS method to obtain the faded envelope of the received signal. It is as follows:

$$
\begin{aligned}
g(t) &= gI(t) + jgQ(t) \\
&= \sqrt{2}\left\{\left[2\sum_{n=1}^{M}\cos\beta_n \cos 2\pi f_n t + \sqrt{2}\cos\alpha\cos 2\pi f_m t\right]\right. \\
&\quad \left. + j\left[2\sum_{n=1}^{M}\sin\beta_n \cos 2\pi f_n t + \sqrt{2}\sin\alpha\cos 2\pi f_m t\right]\right\}
\end{aligned}
\tag{4.2}
$$

Where,

$$f_n = f_m \cos(2\pi n / N), \quad n = 1,2,.....M$$

$$and \quad M = \frac{1}{2}\left(\frac{N}{2} - 1\right)$$

N = number of sinusoids, and M = the number of oscillators required. The phases α and β_n had to be chosen so that the phase of *g(t)* would remain uniformly distributed. Figure 5.7 illustrates the structure, (obtained from [19]) that was used to implement expression (4.2).

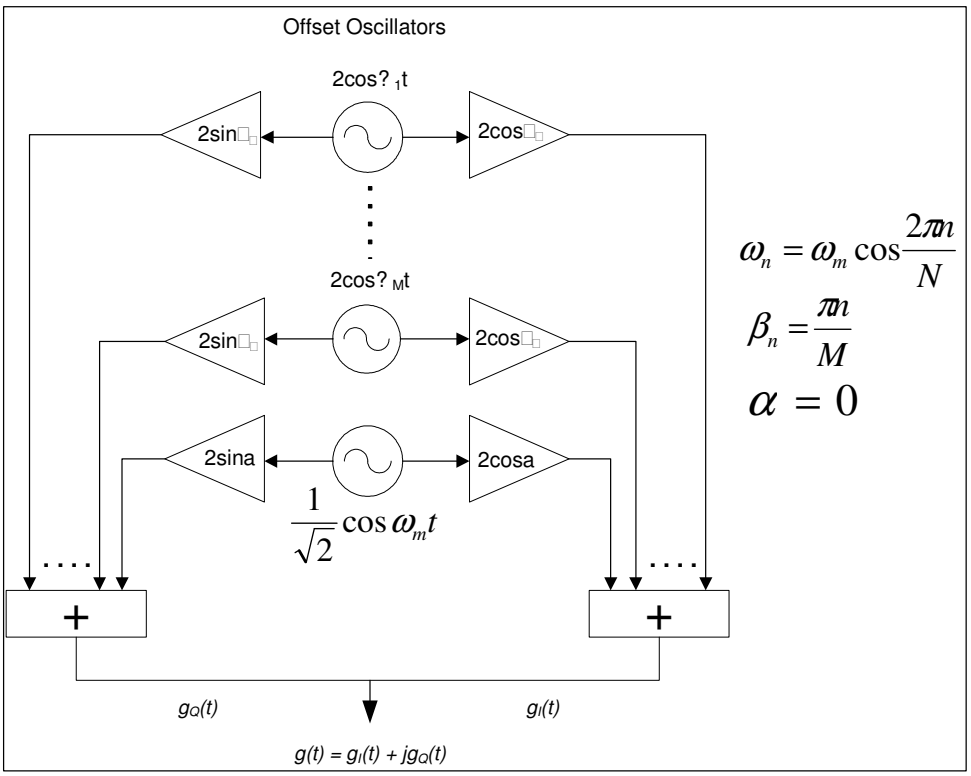

Figure 5.7 – Structure of the Jakes SOS Fading Simulator

The resulting envelope *g(t)* will replace the R_1 R_2 and R_3 parameters in Figure 5.2. The same simulation setup described in Figure 5.2 will be used, but the gains will be produced by the structure given in Figure 5.7.

No special functions were used to implement the Jakes SOS simulator. One '`for`' loop was used to implement the series of oscillators. According to [19], N = 34 (or M = 8) is sufficient to produce a statistically correct with Rayleigh PDF. Since the Maximum Doppler frequency would be in the ranges of 5Hz to 200Hz the sampling frequency for the simulation was chosen to be 1 kHz. (*Refer Appendix C7 for Source Code*).

Figures 5.8 shows different envelopes obtained in terms of power in (dB) for different vehicular speeds. The values obtained for the envelope were turned into the logarithmic domain; the difference betweens the RMS value was plotted against Time (in milliseconds).

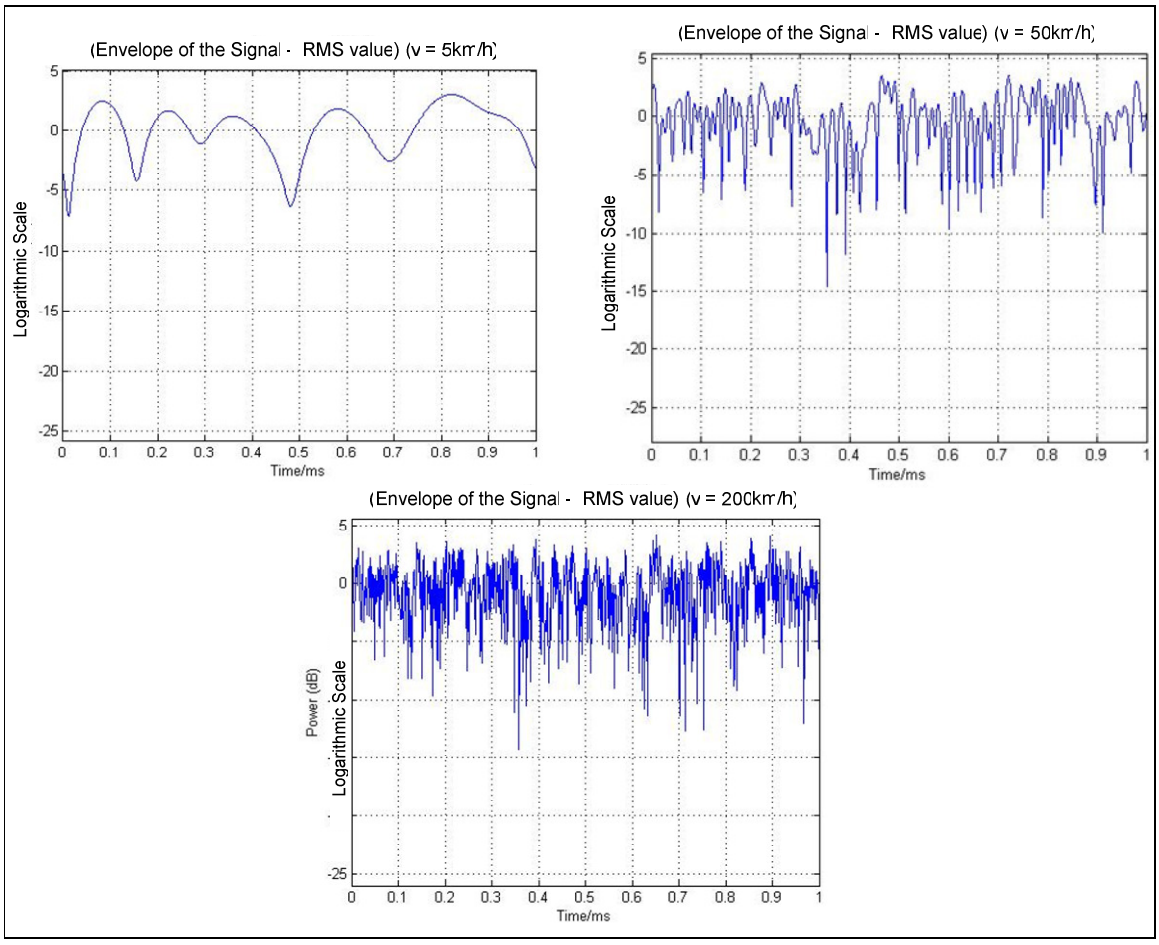

Figure 5.8 – Signal Fading with respect to Motion of the Mobile Station

It is clear from Figure 5.8 that, as the speed of the mobile station increases the envelope of the signal undergoes much more fades, per unit time. This would imply that the BER performance of the system would be worse for scenarios where the mobile station was moving at high velocities.

The implemented Jakes simulator was put to the test by integrating it to the three-path simulator. Table 5.4 shows the parameters used for the simulation.

	Gain1	**Gain2**	**Gain3**	**Delay1**	**Delay2**	**Delay3**	**Speed of MS – *v* (km/h)**
Scenario 1	0.00	0.20	0.20	0.00	0.50	0.50	5
Scenario 2	0.00	0.20	0.20	0.00	0.50	0.50	200

Table 5.4 – Parameters for the Rayleigh Simulator & Three path model

5.4.1 Results of the Rayleigh Fading Simulator

Figure 5.9 shows the results of the simulation for Scenario 1, and Figure 5.10 shows the results for Scenario 2.

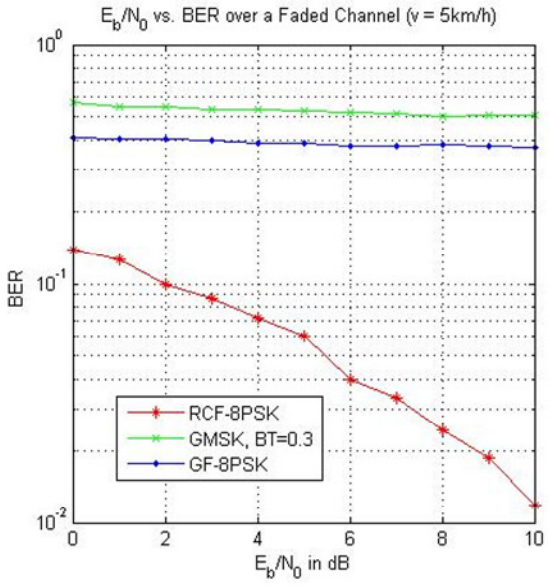

Figure 5.9
BER plot of Scenario 1, v = 5km/h

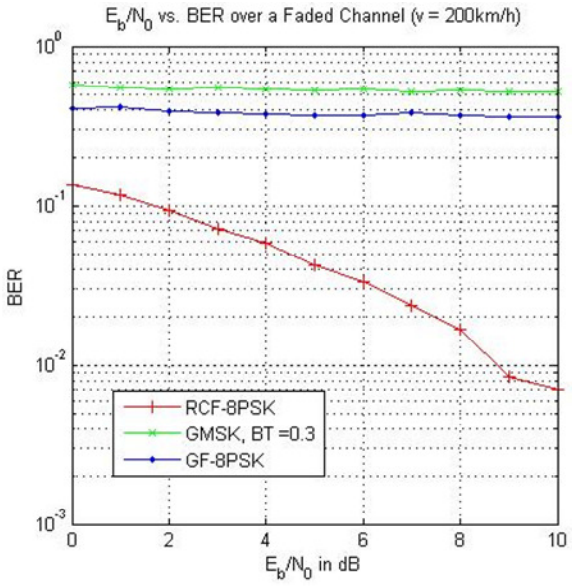

Figure 5.10
BER plot of Scenario 2, v = 200km/h

The results obtained from the final simulations were unsatisfactory, because as Figure 5.10 shows, the increase in crease in vehicular speed had not given an adverse result as expected. The results obtained from both scenarios were almost equal; implying that the simulation results were incorrect therefore further analysis of the simulation results cannot be carious out.

However the Rayleigh fading simulator's integrity can be verified by the plots shown in Figure 5.8. Therefore the fault/bug in the simulation is in the way the Rayleigh envelope was introduced to the three path simulation. Further attempts were made to identify the problem but it could not be resolved.

One hypothesis to this situation could be that, the BER curves obtained could show better results if the simulation was continues for a higher number of bits, and for a higher value of E_b/N_0, but since computational resources were not available for such a task, the project was concluded.

CHAPTER 6
CONCLUSION AND FUTURE WORK

CONCLUSION

The conclusion of this documentation will be divided into two sections. The first section will describe the work done in the project with respect to the project specification. The second section will give a brief summary of the overall results obtained from the simulations that were carried out.

There are eight primary objectives and out of the choice of secondary objectives, the first was chosen; to simulate realistic propagation channels *(refer Appendix A for project specification).* This secondary objective was divided into two sections; the first being simulating a Multipath fading channel (Rician and Rayleigh), which was completed successfully. Figure 1 displays the work plan that was used for this project which spread out over almost one year.

Work done in the project

In this thesis the performance of GMSK and 8PSK modems over an ideal AWGN channel and a Multipath Fading Channel were investigated. Two types of 8PSK modems were simulated; one being the Gaussian Filtered 8PSK (GF-8PSK) used in the EDGE system, as specified by the 3GPP specifications and the other being the traditional Raised Cosine Filtered 8PSK (RCF-8PSK) modem. These were the first and second primary objectives.

Chapters 2 and 3 contain the steps taken to implement the GMSK and 8PSK modem simulations in Matlab. These Chapters also contain the various problems faced during the coding of the simulations, and how solutions were made to overcome these problems. Step by step analysis of the Matlab plots produced in the Modulators and Demodulators gave an in depth understanding of the life of a bit stream inside the physical layer of a communication system. Two of the important subsections in the Modems were the Pre-modulation filtering and Matched Filtering. In Sections 2.6 and 3.3 different matlab plots were produced to show the types of filtering that was used and their affect on the transmitted and received signal was analysed. Constellation diagrams were obtained and compared. This work fulfilled primary objectives three, four and five.

Chapter 4 contains modelling of the AWGN channel and information on the simulations written in Matlab to implement the AWGN channel. A preliminary analysis of the GMSK and 8PSK modems' performance in an AWGN was performed. Bit Error rates were obtained for a range of E_b/N_0values (typically from $E_bN_0 = 0$ to $E_b/N_0= 12$). Further analysis was performed using Power Spectral Density plots and Eye Diagrams. A summary of the preliminary results are included in Section 4.3.4. The final three primary objectives were completed with the completion of the work in chapter 4.

In Chapter 5 Multipath Fading channels and the Three-path model that will be used for simulation purposes were introduced. A brief description was given on the design of the Three-path simulation, by use of flow charts. Five different fading scenarios were implemented, three of which were Rician fading and two were Rayleigh faded channels. The

GMSK and 8PSK modems were put to the test by passing the modulated signals through the Multipath fading channels. BER results were obtained and compared. Also in this section a Rayleigh fading channel simulator was implemented according to the Jakes' Sum of Sinusoids (SOS) method in matlab to investigate the performance of GMSK and 8PSK when introduced to the Doppler Effect and speed of the mobile station. This work completed 90% of the specified secondary goals. The remaining 10% is the unsuccessful integration of the Jakes SOS simulator to the propagation channel.

With respect to the project specification *(found in Appendix A)*, all the primary objectives have been successfully achieved although many hardships were encountered in the process. The latter sections of chapter 2 and 3 briefly describe the various problems faced during the simulations. The most logical approach was taken to resolve these problems, many of which were implementation issues.

Brief Analysis of the Results Obtained

- From Table 4.1 in Section 4.3.4 it is evident that in terms of bandwidth, the GMSK and the GF-8PSK signal have main lobes with equal Baseband bandwidth, while the RCF-8PSK signals' main lobe is ~ 0.1Hz or more wider. The RCF-8PSK signal also has a first side lobe which is ~ 50dB higher than the other two modulated signals. This implies traditional filtering methods cannot be employed to obtain a tight spectrum. The roll-off factor in the raised cosine filter can be lowered to obtain a narrower PSD, but that will in turn affect the ISI tolerance level of the RCF-8PSK signal.

- In terms of spectral efficiency, one 8PSK symbol carries 3 times the amount of bits a GMSK symbol carries. Thus making GF-8PSK spectrally efficient, because it can be used to transmit more information using the same bandwidth.

- By observing the eye diagram, it is clear that using a lower value of BT for the Gaussian filter causes significant amount of ISI to the GMSK signal, due to the fact that it is not a typical Nyquist filter. Similarly by using a Gaussian Filter in the 8PSK modulator, a significant increase of BER was seen. However increasing the BT product of the gaussian filter will result in a broader spectrum, and this is an option the GSM engineers were not willing to make due to the bandwidth limitation issue. They could have increased the amount of bandwidth used, but that in turn would decrease the number of channels that can be multiplexed into the provided radio spectrum range.

- When faced with an ideal AWGN channel the GMSK modem performs better than the 8PSK modem. The BER performance results given in Section 4.3.1 show that if a communication system was using the GF-8PSK modem, then an E_b/N_0 of 12dB was required to obtain a BER of 10^{-3}, while the GMSK simulations show a BER of 10^{-3} achieved with an E_b/N_0 of 8dB. Traditional RCF-8PSK modem simulation gave a BER of 10^{-3} achieved with a E_bN_0 of 10dB. These simulation results help GSM and EDGE engineers to obtain an accurate value for the receiver sensitivity, which is dependent on the E_b/N_0 value.
- In terms of Power efficiency, the GMSK modem overtakes the 8PSK systems, because more power is wasted on erroneous bits. This is always the issue with high order

modulation schemes, and engineers have to make a trade-off between high data rates and power efficiency.

- As seen in Section 5.2.3 in a typical Multipath channel, GF-8PSK performed slightly better than GMSK. GMSK performs better in Rician fading environments where the delay of a signal is less than half a symbol period, but as the delay gets larger and when the fading becomes Rayleigh, GMSK shows a poor BER performance of 0.5dB lower than GF-8PSK. Both GMSK and GF-8PSK modems showed a BER of under 10^{-1} for path delays of over half a symbol period. The RCF-8PSK modem was able to cope with much larger path delays than the gaussian filtered modems. This is an advantage of using a Nyquist filter over a Gaussian filter; its resistance to ISI is confirmed by the results.

- The results obtained cannot exactly decide which type of modem is best. As stated each modem has its weaknesses and strengths. Therefore the choice of a particular modem will vary according to the requirements and specifications of the communication system. For example for use in environments with minimum fading and also if spectral bandwidth is an issue then the GMSK modem would be a probable choice. However if high data rates are required, and if the spectrum and a high BER is not an issue or can be dealt with by other means then a higher order modulation scheme such as 8PSK should be used.

- All the results obtained from the simulations confirm published work in this area of mobile communications. The BER results obtained from the GF-8PSK simulations were unique in the sense that no certified published work was found in the course of my literature research to support my findings. However basic communication principles validate the integrity of the simulation results.

FUTURE WORK

At the time of writing this report all the secondary goals have not been completed in full. As mentioned before, certain problems arose when integrating the Rayleigh simulator (which was successfully implemented using, Jakes SOS method given in [19]) into the three path propagation model. Given more time to completion, this problem would have been resolved however further attempts to rectify this problem are being carried out.

This project was conducted using ideal coherent demodulators. It maybe useful to investigate the affect on BER performance when better receivers or demodulators are used. Receivers such as the Rake receiver and Maximum Likelihood Sequence Estimation can be explored. It was also assumed that frequency, phase and symbol timing were perfectly restored at the receiver. This is however not the case in realistic mobile receivers. Methods of obtaining the symbol timing and carrier frequency at the receiver can be looked into. In order to reduce the ISI caused by the time delay spreading of the wave, different Equalizers at the receiver can be used. It is therefore suggested to investigate different types of Equalizers as part of future work that can be performed.

The modem simulations conducted in this project does not involve channel coding or decoding. Therefore it is suggested that coding methods such as convolution coding, turbo coding and Reed Solomon codes be investigated. And common decoding methods such as Viterbi decoding can be employed.

Even though the GSM mobile system is still the most widely used Mobile communication system, other 3G technologies such as multiple carrier technologies (e.g. CDMA and OFDM) can be explored, and compared with GMSK and 8PSK modulation methods. It would be interesting to see if at all how they improve the Bit error rate when compared with 2G and 2.5G mobile systems.

REFERENCES

[1] 3GPP (3rd Generation Partnership Project) - *"GSM/EDGE, Radio Access Network: Modulation (Release 7)"*, Technical Specification 45.004, 2007-2008.

[2] LINK Collaborative Reasearch, Personal Communications Programme (PCP) – *"Electromagnetic Compatibility Aspects of Radio-based Mobile Telecommunications Systems, Final Report"* , Appendix D (University of Hull, "Digital Modulation and GMSK"), 1999

[3] Agilent Technologies - *"EGPRS Test, Meeting the Challenge of 8PSK Modulation"*, Application Note, 2005

[4] T.Turletti – *"GMSK in a Nutshell"*, Telemedia Networks and Systems Group, Massachussets Institute of Technology, April 1996.

[5] S. V. Schell - *"Implementation Effects on GSM's EDGE Modulation"* ,Tropian Inc., 2000

[6] K. Kuchi, V.K. Prabhu - *"Spectral Occupancy and error rate considerations of Coherent GMSK"*, IEEE Wireless Communications and Networking Conference, 1999

[7] K. Murota and K. Hirade - *"GMSK Modulation for Digital Mobile Radio Telephony"* , IEEE Transactions on Communications, Vol.Com-29,No7, July 1981

[8] J.D Laster – *"Robust GMSK Demodulation using Demodulator Diversity and BER Estimation"*, Virginia Polytechnic Institute and State University, Department of Electrical Engineering, March 1997.

[9] P. Berthelmy – *"A Model-Based Receiver for CPM Signals in a Cochannel Interference Limited Environment"*, Virginia Polytechnic Institute and State University, Department of Electrical Engineering, May 2002.

[10] M. Mouly and M. B. Pautet – *"The GSM System for Mobile Communications"*, Telecom Publishing, 1992.

[11] T.S. Rappaport - *"Wireless Communications: Principles and Practice"*, Prentice Hall, Second Edition, 2002

[12] S. Redl, M.K. Weber, M. Oliphant - *"An Introduction to GSM"*, The Artech House Publications, 1993

[13] W.C.Y. Lee - *"Mobile Communication Engineering: Theory and Applications"*, McGraw-Hill, Second Edition, 1998

[14] J. D Gibson - *"The Mobile Communications Handbook"*, IEE Press, 1996.

[15] N. Benvenuto, G. Cherubini - *"Algorithms for Communications Systems and their Applications"*, Wiley, 2002

[16] B. Sklar - *"Digital Comunications - Fundamentals and Applications"*, Prentice Hall, Second Edition, 2001.

[17] J.G. Proakis - *"Digital Communications"*, McGraw-Hill, Fourth Edition, 2001

[18] W. H. Tranter, K. S. Shanmugan, T. S. Rappaport and K. L. Kosbar – *"Principles of Communication Systems Simulation, with Wireless Applications"*, Pearson Education Inc, 2004.

[19] G. L. Stüber – *"Principles of Mobile Communication"*, Second Edition, Kluwer Academic Publishers, 2001.

[20] Abhijit Patait, HelloSoft Inc, *"Efficient GMSK Modulator Targets GSM Designs"*, CommsDesign, accessed via Opera Web Browser, on 20th December 2007 at http://www.commsdesign.com/design_corner/OEG20020708S0008

BIBLIOGRAPHY

- X. Yan, T. Cai, M. Jin, K.S. Kwak - *"Design of Trellis-Coded GMSK Modulation for AWGN channel"*, Proceedings of the IEEE Region 10 Conference, Volume 2, 1999

- Shahzada Basharat Rasool - *"Simulation of GMSK over AWGN & Two-path AWGN with delay"* , King Fahd University of Petroleum and Minerals, Dhahran

- S. Isoard - *"Implementation of an Analysing and Modelling Tool of the GSM/GPRS/EDGE, L1 Layer & Design of a Demodulator Adapted to the Specificity of a Network Simulator"*, Royal Institute of Technology, Department of Signals, Sensors & Systems Signal Processing S-100 44 Stockholm, March 8th 2004.

- Y. Ning, - *"Mobile Speed Estimation for Hierarchical Wireless Network",* University of Missouri, Columbia, July 2005.

- F.G Stremler – *"Introduction to Communication Systems"*, Addison-Wesley Publishing Inc, Third Edition, 1992

- C. B. Rorabaugh - *"Simulating Wireless Communication Systems ,Practical Models in C++"*, Prentice Hall, 2004

- D. C. Hanselman, B. L. Littlefield - *"Mastering Matlab 7"*, Prentice Hall, 2004

- J. G. Proakis, M. Salehi, G. Bauch – *"Contemporary Communication Systems using Matlab and Simulink"*, Thomson-Brooks/Cole, 2004.

- K. Christofylaki-Fines , lecture notes - *"Mobile Radio Systems" and "Communication Systems"*, Department for Electronic Systems, University of Westminster, 2006-2008

- T. Jiang, lecture notes - *"Advanced Digital Communications"* - School of Electrical Engineering, Royal Institute of Technology (KTH) ,Stockholm, 2005.

APPENDIX

The Appendix Section contains the following Items:

- **Appendix A -** Project Specification Form
-
- **Appendix B -** List of Figures and Tables

    - Appendix (B1): List of Figures
    - Appendix (B2): List of Tables

- **Appendix C** – Matlab Source code used for the project

    - Appendix (C1): Code for the GMSK Modem.
    - Appendix (C2.1): Code for the GF-8PSK Modem
    - Appendix (C2.2): Code for the RCF-8PSK Modem
    - Appendix (C3): Code for the AWGN channel.
    - Appendix (C4): Code for the Eye diagrams.
    - Appendix (C5): Code for the Power Spectrum Density plot.
    - Appendix (C6): Code for the Three-path Fading Simulator.
    - Appendix (C7): Code for the Rayleigh Fading Simulator (Jakes SOS).

- **Appendix D** – Contents of the attached Software CD.

Appendix A – Project Specification Form

UNIVERSITY OF WESTMINSTER
DEPARTMENT OF ELECTRONICS
PROJECT SPECIFICATION FORM

Project Title: GPRS and 8PSK Modem software implementation

Student Name: Hashan Roshantha Mendis (10030637) Year: 2007/8

Supervisor(s): Katerina Christofylaki-Fines

Degree: BSc. Network and Communication Engineering

AIMS and DESCRIPTION

Global System for Mobile communications (GSM) technology is the foundation for many of the wireless communication systems throughout Europe and the rest of the world. General Packet Radio Service (GPRS) is a Mobile Data Service available to users of the GSM system. EDGE (Enhanced Data Rates for GSM Evolution) is a mobile technology which was introduced to 2G networks to enable high speed data transfer. EDGE is also known as Enhanced GPRS (EGPRS), an improvement to to GPRS.

This project aims to study in detail the Physical layer of the GSM and EDGE system, mainly focusing on the modulation and the demodulation of bit streams. It will involve a study of the Gaussian Minimum-Shift Keying (GMSK) and 8-Phase Shift Keying (8PSK) modems, specified for GSM and EDGE systems respectively.

The study will also include analysing the Bit Error Rate of the two types of modems, comparing power spectra and distortion of the signal due to Intersymbol Interference (ISI).

The model should be tested under additive White Gaussian noise.

The model should include ideal Demodulators.

The possibility of moving to more realistic propagation channels (Rayleigh & Rician) would be an advantage.

PRIMARY GOALS

1. Understanding the GSM system, and Enhanced GPRS (EGPRS): Their data rates and functionality.

2. Thorough study of the GMSK modulator and demodulator. Being able to describe the behaviour of the modem sub sections.

3. Implementing a simulation of the GMSK modem using Simulink/Matlab.

4. Detailed study of the 8PSK modulator and demodulator. Being able to describe the behaviour of the modem sub sections.

5. Implementing a simulation of the 8PSK modem using Simulink/Matlab.

6. Introducing Additive White Gaussian Noise (AWGN) into the Simulation.

7. Start measuring BER. Create Matlab plots for the Modulation Constellations, Generate Eye Diagrams to investigate the ISI level. (for both modulation schemes)

8. Validate Theoretical test data with realistic values.

SECONDARY GOALS

1. Analysis of the two modulation methods under Fading conditions. Tests should involve Rayleigh and Rician channel analysis and BER plots.

2. Or Realistic receivers dealing with frequency and time synchronisation.

RESOURCES NEEDED

Matlab and Simulink (essentially the 'Communication Toolbox' and 'Signal Processing Toolbox')

HEALTH and SAFETY ASSESSMENT and ARRANGEMENTS

None

Appendix B1 - List of Figures

Appendix B1 - List of Tables

Appendix C1 – Matlab Code for the GMSK Modem

Note: Individual sub-functions are separated by a horizontal line. Multiple line code has been separated by '…' on the first line, as used in the Matlab Script Editor.

```
% Name : Hashan Roshantha Mendis (10030637)
% Date Updated : 20/04/08
% GMSK - Modulation/Demodulation with AWGN and Fading Channel
% [Main Script]
% -------------------------------------------------------------------------
% -------------------------------------------------------------------------
clear all;%close all
tic % STOPWATCH ON
% =========================================================================
samples = 36;
Tb = 1; % bit duration
SamplePeriod = Tb*(1/samples);

%m = [-1 -1 -1 1 1 -1 1 1 1 1 -1 1 -1 1 -1 -1]; % message bits
Pe =[];

for EbNo = (0:1:10)
    m = randsrc(1,10000); % produces random -1's and 1's

    %t1 = 0:SamplePeriod:(length(m)*Tb); % define timeline
    %t1(:, length(t1)) = []; % get rid of extra column in timeline

    % -------------------------------------------------------------------
    % ---------------------------- GMSK Modulation ----------------------
    % -------------------------------------------------------------------
    % the following converts message into a series of unipolar NRZ data.
    % use the kron function to upsample the bits.
    rect = kron(m,ones(1,samples));

    % create gaussian low pass filter(defined in the gaussian_filter.m)
    gaussfilter = GMSK_gaussian_filter(Tb,samples);

    % pass message signal through Gaussian LPF
    m_filtered = conv(gaussfilter,rect);

    %figure;plot(m_filtered);title('filtered data');
    % add extra sample at end.
    m_filtered = [m_filtered m_filtered(length(m_filtered))];

    m_filtered1 = cumsum(m_filtered); % integrate the data.

    m_filtered2_real = cos(m_filtered1);
    m_filtered2_imag = sin(m_filtered1);

    m_filtered2 = m_filtered2_real + j*m_filtered2_imag;
    % -------------------------------------------------------------------
    % ------------------------- FADING + AWGN Channel -------------------
    % -------------------------------------------------------------------
    % pass through the fading channel

      faded_signal = rayleigh_sim(m_filtered2,samples);
```

```
    % First test : without noise , noise = Nil (uncomment as necessary)
    %noisy_real = m_filtered2_real;
    %noisy_imag = m_filtered2_imag;
    % Test for only AWGN, no fading.(uncomment as necessary)
    %noisy_real = AWGN_channel(real(m_filtered2), EbNo, Tb);
    %noisy_imag = AWGN_channel(imag(m_filtered2), EbNo, Tb);


    % Now introduce AWGN into the signal.
    %EbNo = 10; % EbNo in db
    % apply noise to I-channel
    noisy_real = AWGN_channel(real(faded_signal), EbNo, Tb);
    % apply noise to Q-channel
    noisy_imag = AWGN_channel(imag(faded_signal), EbNo, Tb);
    % ------------------------------------------------------------------
    % ------------------------ GMSK De-Modulation ----------------------
    % ------------------------------------------------------------------

    % pass I and Q through Matched Filter(defined in the matched_filter.m)
    mfilt_samples = 7;
    matchfilter = GMSK_matched_filter(Tb,mfilt_samples);

    % Test without noise (uncomment as necessary)
    %filt_noisy_imag = noisy_imag;
    %filt_noisy_real = noisy_real;

    % convolve the filter with the signal
    filt_noisy_real = conv(matchfilter,noisy_real);
    filt_noisy_imag = conv(matchfilter,noisy_imag);

   % and add an extra sample to the end.
    filt_noisy_real = [filt_noisy_real ...
filt_noisy_real(length(filt_noisy_real))];
    filt_noisy_imag = [filt_noisy_imag ...
filt_noisy_imag(length(filt_noisy_imag))];

    % obtaining the phase of the analog signal
    phase = unwrap(angle(filt_noisy_real+filt_noisy_imag*j));

    % obtain the derivative of the signal.
    derivative = diff(phase);
    derivative = [phase(1) derivative];

    % downsample the signal
    rx_dwnsmpled = GMSK_downsample(70,71,samples,derivative);

    % GMSK_ADC
    rx_digital = GMSK_ADC(rx_dwnsmpled);

    %figure;stem(derivative);figure;stem(rx_dwnsmpled);
    [num,rat] = symerr(m,rx_digital);

    %store Pe values
    Pe = [Pe rat];

    fprintf('end of %d \n',EbNo);

end
```

```
% ---------------- Generate EbNo vs. BER plot -----------------

EbNo_temp = 0:1:10;
semilogy(EbNo_temp,Pe);title('GMSK (BT=0.3) - EbNo vs. BER');hold on

total_time = toc; % STOPWATCH OFF
```

```
% [Gaussian Filter script]
% ========================================================================

% Definition of the Premodulation - Gaussian Filter

function result = GMSK_gaussian_filter(T,sps)
    t = (-1.5*T:T/sps:1.5*T);
    BT = 0.3; % T=1

    h = (BT.*sqrt((2*pi)/log(2))).*exp((- ...
1*(((2*pi^2)*(BT.^2))).*t.^2)./log(2));
    % need to scale the filter, so that there is a phase change of pi/2 for
    % every bit change.

    K = pi/2/sum(h);
    gfilter = K*h;

    %normalize filter gain.
    gfilter = gfilter./sqrt(sum(gfilter));

    %figure;plot(gfilter);title('gaussian filter');
    result = gfilter;

end
```

```
% [Matched Filter Script]
% ========================================================================
 % Definition of the Matched-Filter
function result = GMSK_matched_filter(T,sps)
    t = (-1.5*T:T/sps:1.5*T);
    BT = 0.75;

    h = (BT.*sqrt((2*pi)/log(2))).*exp((- ...
1*(((2*pi^2)*(BT.^2))).*t.^2)./log(2));
    % need to scale the filter, so that there is a phase change of pi/2 for
    % every bit change.

    K = pi/2/sum(h);
    Mfil = h*K;

    %normalize filter
    Mfil = Mfil./sqrt(sum(Mfil));

    result = Mfil;

end
```

```
% [Downsample/Continuous to Discrete time converter Script]
% ========================================================================

% Downsamples the received GMSK signal.

% Function parameters :
% ---------------------
% start - begin sampling from this sample.(i.e. leave out(start-1) samples)
% en    - number of samples to leave out at the end
% sps   - downsampling rate
% derivative - input signal to be sampled.

function result = GMSK_downsample(start,en,sps,derivative)
    i = start:sps:length(derivative)-en;
    temp = derivative(i);
    result = temp;
end
```

```
% [Analog - Digital Converter Script]
% ========================================================================

function result = GMSK_ADC(rx_down)
    temp=[];   % temporary vector to store result

    for n = 1:1:length(rx_down)
        if(rx_down(n)<0)        % if rx_down < 0 convert to -1
            temp(n) = -1;
        elseif(rx_down(n)>0)    % if rx_down > 0 convert to 1
            temp(n) = 1;
        elseif(rx_down(n)==0)   % if rx_down = 0 convert to -1
            temp(n) = -1;
        end
    end
     result = temp;

 end
```

Appendix C2.1 – Matlab Code for the GF-8PSK Modem

Note: individual sub-functions are separated by a horizontal line

```
% Name : Hashan Roshantha Mendis (10030637)
% Date Updated : 20/04/08
% 8PSK - Modulation/Demodulation with AWGN and Fading
% [GF-8PSK Main Script]
% -------------------------------------------------------------------------
% -------------------------------------------------------------------------

clear all;%close all
tic
% ------------------------ 8PSK Transmitter ---------------------------
% ---------------------------------------------------------------------
samples = 36;
Tb = 1; % bit duration
SamplePeriod = Tb*(1/samples);
%m = [1 0 1 1]; % message bits

Pe =[];
  for EbNo = (0:1:12)

    m = randsrc(1,10000); % produces random 1's and -1's.
    m = divisibleby3(m); % make array divisible by 3 (zero padding)

    t1 = 0:SamplePeriod:(length(m)*Tb); % define timeline
    t1(:, length(t1)) = []; % get rid of extra column in timeline

    % group message into 3 bit groups and convert D/A
    % here we group the discrete data bits. NOT the continuous ones

    s = epsk_DAC(m);
    % rotation of symbols.
    s2 = epsk_symbol_rotate(s);

    % upsample the complex signal
    s3 = epsk_upsample(s2,samples);

    % generating the pulse shaping filter and convolve with signal
    pulse_filter = epsk_shaping_filter(Tb, 21);
    s3_filtered = conv(pulse_filter,s3);
    s3_filtered = [s3_filtered 0]; % add an extra sample to the end.


    % ------------------------ Applying AWGN ----------------------------
    % -------------------------------------------------------------------

    s_tx = s3_filtered;
    faded_signal = rayleigh_sim(s_tx,samples);  % pass thru fading channel

    % introduce AWGN noise into the faded system.
    tx_noisy_real = AWGN_channel(real(faded_signal),EbNo,Tb);
    tx_noisy_imag = AWGN_channel(imag(faded_signal),EbNo,Tb);
```

```
    % only AWGN
    %tx_noisy_real = AWGN_channel(real(s_tx),EbNo,Tb);
    %tx_noisy_imag = AWGN_channel(imag(s_tx),EbNo,Tb);

    s_tx_noisy = tx_noisy_real + (j*tx_noisy_imag);

    % ------------------------- 8PSK Receiver ---------------------------
    % -------------------------------------------------------------------

    % generate matched filter, convolve with Rx. signal
    m_filt = epsk_matched_filter(Tb,7);
    %s_rx=[];
    s_rx = conv(m_filt,s_tx_noisy);
    s_rx = [s_rx 0]; % add an extra sample to the end.
    %plot(s_rx);title('8psk filtered Rx. constellation');

    % downsample the filtered signal
    s_rx_dwnsmpled = epsk_downsample(s_rx,samples,86,57);

    % de-rotation of symbols.
    s_rx_derotate = epsk_derotate(s_rx_dwnsmpled);

    % convert analog signal to digital.
    s_rx_digital = epsk_ADC(s_rx_derotate);

    % error rate
    [num,rat] = symerr(m,s_rx_digital);

      %store Pe values
       Pe = [Pe rat];

      fprintf('end of iteration %d \n',EbNo);
  end

% ---------------- Generate EbNo vs. BER plot -----------------

EbNo_temp = 0:1:12;
semilogy(EbNo_temp,Pe);hold on;
total_time = toc;
```

```
% [8PSK Divisible by 3 Script]
% -------------------------------------------------------------------
% -------------------------------------------------------------------

% divisible3.m - this function makes the data array divisible by 3
% if remainder after dividing by three is 1 -> 2 zero paddings.
% if remainder after dividing by three is 2 -> 1 zero padding.

function ans = divisibleby3(data)
    R = rem(length(data),3);

    if(R == 0)
        data = data;
    elseif (R == 1)
        data = [data -1 -1];
```

```
    elseif (R == 2)
        data = [data -1];
    end

    ans = data;
end
```

```
% [8PSK DAC Script]
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% This function groups message into 3 bit groups according to the
%constellation diagram
% and converts digital to analog signal.

 function ans = epsk_DAC(data)

    signal = [];
    for n = 1:3:length(data)
        if ((data(n))==1) && ((data(n+1))==1) && ((data(n+2))==1)
            l = 0;
            temp_sig = exp((j*2*pi*l)/8);
        elseif ((data(n))==-1) && ((data(n+1))==1) && ((data(n+2))==1)
            l = 1;
            temp_sig = exp((j*2*pi*l)/8);
        elseif ((data(n))==-1) && ((data(n+1))==1) && ((data(n+2))==-1)
            l = 2;
            temp_sig = exp((j*2*pi*l)/8);
        elseif ((data(n))==-1) && ((data(n+1))==-1) && ((data(n+2))==-1)
            l = 3;
            temp_sig = exp((j*2*pi*l)/8);
        elseif ((data(n))==-1) && ((data(n+1))==-1) && ((data(n+2))==1)
            l = 4;
            temp_sig = exp((j*2*pi*l)/8);
        elseif ((data(n))==1) && ((data(n+1))==-1) && ((data(n+2))==1)
            l = 5;
            temp_sig = exp((j*2*pi*l)/8);
        elseif ((data(n))==1) && ((data(n+1))==-1) && ((data(n+2))==-1)
            l = 6;
            temp_sig = exp((j*2*pi*l)/8);
        elseif ((data(n))==1) && ((data(n+1))==1) && ((data(n+2))==-1)
            l = 7;
            temp_sig = exp((j*2*pi*l)/8);
        end

        signal = [signal temp_sig];
    end

  ans = signal;
end
```

```
% [8PSK Symbol Rotation Script]
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------

% This function rotates the current 8psk symbols by (3*pi/8)

function result = epsk_symbol_rotate(signal)
    temp = [];
    x = 1:length(signal);
    temp = signal.*exp((j*x*3*pi)/8);
    result = temp;
end
```

```
% [8PSK Upsample Script]
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------

% upsamples the input signal by sps

function result = epsk_upsample(signal,sps)
    temp = kron(signal,ones(1,sps));
    result = temp;
end
```

```
% [8PSK Pulse Shaping Filter Script]
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% Definition of the Premodulation - Linearised Gaussian Filter for 8psk

function result = epsk_shaping_filter(T,sps)
    t = (-5*T/2:T/sps:5*T/2);
    approx_filt = exp((-1.045*(t/T).^2) - (0.218*(t/T).^4));

    %normalize filter gain.
    approx_filt = approx_filt/sqrt(sum(approx_filt));

    result = approx_filt;
end
% [8PSK Symbol Rotation Script]
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------

% This function rotates the current 8psk symbols by (3*pi/8)

function result = epsk_symbol_rotate(signal)
    temp = [];
    x = 1:length(signal);
    temp = signal.*exp((j*x*3*pi)/8);
    result = temp;
end
```

```
% [8PSK Pulse Shaping Filter Script]
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------

% Definition of the Premodulation - Linearised Gaussian Filter for 8psk

function result = epsk_shaping_filter(T,sps)
    t = (-5*T/2:T/sps:5*T/2);
    approx_filt = exp((-1.045*(t/T).^2) - (0.218*(t/T).^4));

    %normalize filter gain.
    approx_filt = approx_filt/sqrt(sum(approx_filt));

    result = approx_filt;

end
```

```
% [8PSK Matched Filter Script]
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% Definition of the Matched Filter - Linearised Gaussian Filter for 8psk

function result = epsk_matched_filter(T,sps)
    t = (-5*T/2:T/sps:5*T/2);
    approx_filt = exp((-1.045*(t/T).^2) - (0.218*(t/T).^4));

    %normalize filter gain
    %approx_filt = approx_filt/sqrt(sum(approx_filt));

    %figure;plot(t,approx_filt);title('approximate matched filter');figure

    result = approx_filt;

end
```

```
% [8PSK De-rotate Script]
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% derotates the signal by (3*pi/8) to fit the original 8psk constellation

function result = epsk_derotate(signal)
    x = 1:length(signal);
    temp = signal./(exp((j*x*3*pi)/8));
    result = temp;
end
```

```
% [8PSK Downsample Script]
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% Downsamples the signal, the head and tail are the start and end samples.
% sps = sampling rate

function result = epsk_downsample(signal,sps,head,tail)
    n = head:sps:(length(signal)-tail);
```

```
    temp = signal(n);
    result = temp;
end
```

```
% [8PSK ADC Script]
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% This function converts the analog signal into 3 bit
% groups according to the 8psk constellation at receiver end.

function result = epsk_ADC(sig)

s1=[1 1 1]; s2=[-1 1 1]; s3=[-1 1 -1]; s4=[-1 -1 -1];
s5=[-1 -1 1]; s6=[1 -1 1]; s7=[1 -1 -1]; s8=[1 1 -1];

rx_angle = angle(sig);
rx_d = [];

for n = 1:length(sig)
    if(rx_angle(n) < 0)
        if ((rx_angle(n) <= 0) && (rx_angle(n) >= -pi/8))
            rx_d = [rx_d s1]; % group = [1 1 1]
        elseif ((rx_angle(n) < -pi/8) && (rx_angle(n) >= -3*pi/8))
            rx_d = [rx_d s8]; % group = [1 1 -1]
        elseif ((rx_angle(n) < -3*pi/8) && (rx_angle(n) >= -5*pi/8))
            rx_d = [rx_d s7]; % group = [1 -1 -1]
        elseif ((rx_angle(n) < -5*pi/8) && (rx_angle(n) >= -7*pi/8))
            rx_d = [rx_d s6]; % group = [1 -1 1]
        elseif ((rx_angle(n) < -7*pi/8) && (rx_angle(n) >= -pi))
            rx_d = [rx_d s5]; % group = [-1 -1 1]
        end
    end

    if(rx_angle(n)>0)
        if((rx_angle(n) >= 0) && (rx_angle(n) <= pi/8))
            rx_d = [rx_d s1]; % group = [1 1 1]
        elseif((rx_angle(n) > pi/8) && (rx_angle(n) <= 3*pi/8))
            rx_d = [rx_d s2]; % group = [-1 1 1]
        elseif((rx_angle(n) > 3*pi/8) && (rx_angle(n) <= 5*pi/8))
            rx_d = [rx_d s3]; % group = [-1 1 -1]
        elseif((rx_angle(n) > 5*pi/8) && (rx_angle(n) <= 7*pi/8))
            rx_d = [rx_d s4]; % group = [-1 -1 -1]
        elseif((rx_angle(n) >  7*pi/8) && (rx_angle(n) <= pi))
            rx_d = [rx_d s5]; % group = [-1 -1 1]
        end
    end

    if(rx_angle(n) == 0)
        rx_d = [rx_d s1];
    end


end

result = rx_d;

end
```

Appendix C2.2 – Matlab Code for the RCF-8PSK Modem

Note: individual functions are separated by a horizontal line. This main script shares all of the sub functions with the GF-8PSK Modem code, therefore those function have not been included to avoid repetition.

```
% Name : Hashan Roshantha Mendis (10030637)
% Date Updated : 20/04/08
% 8PSK - Modulation/Demodulation with AWGN
% [RCF-8PSK Main Script]
% -------------------------------------------------------------------------
% -------------------------------------------------------------------------

clear all;%close all
tic
% ------------------------ 8PSK Transmitter -----------------------------
% -------------------------------------------------------------------------
samples = 36;
Tb = 1; % bit duration
SamplePeriod = Tb*(1/samples);
%m = [1 0 1 1]; % message bits

Pe =[];
for EbNo = 0:1:12

    m = randsrc(1,10000); % produces random 1's and -1's.
    m = divisibleby3(m); % make array divisible by 3 (zero padding)

    t1 = 0:SamplePeriod:(length(m)*Tb); % define timeline
    t1(:, length(t1)) = []; % get rid of extra column in timeline

    % the following code, upsamples the datastream.
    rect = epsk_upsample(m,samples);

    % group message into 3 bit groups and convert D/A
    s = epsk_DAC(m);

    % rotation of symbols.
    s2 = epsk_symbol_rotate(s);
    %s2=s;

    % upsample the complex signal
    s3 = epsk_upsample(s2,samples);

    % generating the pulse shaping filter and convolve with signal
    %s3_filtered = rcosflt(s3,1,36,'fir/Fs');

    %pulse_filter = epsk_shaping_filter(Tb, 18);
    [pulse_filter den] = rcosine(1,18,'default');
    pulse_filter = pulse_filter./sqrt(sum(pulse_filter));
    s3_filtered = conv(pulse_filter,s3);
    s3_filtered = [s3_filtered 0]; % add an extra sample to the end.

    % -------------- Pass Through Fading Channel and Add AWGN  ------------
    % -------------------------------------------------------------------
    % No noise applied
    s_tx = s3_filtered;
```

```
    % apply Rayleigh multipath fading.
    faded_signal = rayleigh_sim(s_tx,samples);

    %EbNo = 1;
    %introduce AWGN noise into the system.
    tx_noisy_real = AWGN_channel(real(faded_signal),EbNo);
    tx_noisy_imag = AWGN_channel(imag(faded_signal),EbNo);

    % Following two lines used for NON-FADING scenario (only AWGN)
    %tx_noisy_real = AWGN_channel(real(s_tx),EbNo);
    %tx_noisy_imag = AWGN_channel(imag(s_tx),EbNo);

    s_tx_noisy = tx_noisy_real + (j*tx_noisy_imag);


    %s_tx_noisy = s_tx;

    %s_rx = s_tx_noisy;
    % ------------------------- 8PSK Receiver -----------------------------
    % ---------------------------------------------------------------------

    % generate matched filter, convolve with Rx. signal

    %m_filt = epsk_matched_filter(Tb,18);
    [m_filt den] = rcosine(1,10,'default');
    m_filt = m_filt./sqrt(sum(m_filt));
    s_rx = conv(m_filt,s_tx_noisy);
    s_rx = [s_rx 0]; % add an extra sample to the end.


    % downsample the filtered signal
    s_rx_dwnsmpled = epsk_downsample(s_rx,samples,109,62);

    % de-rotation of symbols.
    s_rx_derotate = epsk_derotate(s_rx_dwnsmpled);

    % convert analog signal to digital.
    s_rx_digital = epsk_ADC(s_rx_derotate);

    % error rate
    [num,rat] = symerr(m,s_rx_digital);

    %store Pe values
    Pe = [Pe rat];

    fprintf('end of %d \n',EbNo);
end

% ---------------- Generate EbNo vs. BER plot -----------------

EbNo_temp = 0:1:12;
semilogy(EbNo_temp,Pe);title('8PSK - EbNo vs. BER');
hold on;
total_time = toc;
```

Appendix C3 – Matlab Code for the AWGN channel

```
% Name : Hashan Roshantha Mendis (10030637)
% Date Updated : 20/04/08
% 8PSK - Modulation/Demodulation with AWGN
% [AWGN Script]
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% Creates WGN and adds it to the signal.
% result = noisy signal.

function result = AWGN_channel(signal, EbNo_db)

    Eb = 1;                             % define bit energy (normalized to 1)
    No = (36*Eb)/(10^(EbNo_db/10)); % calculate No in terms of Eb
    sigma = sqrt(No);                % calculate standard deviation

    % generate and add noise to signal
    result = signal + (sigma * randn(1,length(signal)));

end
```

Appendix C4 – Matlab Code for the Eye Diagrams

```
% Name : Hashan Roshantha Mendis (10030637)
% Date updated : 20/04/08
% ------ Plots Eye Diagram for the GMSK signal -------
N = 200;
n=72;
figure;

for i = 1:N

    % plot eye diagram
    temp1 = real(m_filtered((i-1)*n+1:(i-1)*n+n));
    plot(temp1)
    title('eye diagram for mfilter')
    hold on

end
```

```
% ---------- Plots Eye Diagram for the 8PSK Signal ----------
% Real and imaginary component ploted seperately

function eye_plot(signal)
    N = 100;
    n=72;

    for i = 1:N
        temp_real = real(signal((i-1)*n+1:(i-1)*n+n));
        temp_imag = imag(signal((i-1)*n+1:(i-1)*n+n));
        subplot(2,1,1);plot(temp_real);
```

```
        title('Eye diagram for 8psk - I-phase');
        hold on
        subplot(2,1,2);plot(temp_imag);
        title('Eye diagram for 8psk - Q-phase');
        hold on
    end
end
```

Appendix C5 – Matlab Code for the Power Spectral Density Plot

```
%Name : Hashan Roshantha Mendis (100306370
%Date Updated : 20/04/08
%-----------------------------------------
% Script used to plot the PSD of the signals

fs =36; % sampling rate
window = 2^15; % FFT window size
[pxx,f] = pwelch(real(s_tx),window,fs);
pxx = pxx/sum(pxx);
n = length(f)/2;
pxxdB = 10*log(pxx/pxx(1)); % convert into dB
plot(f(1:n),pxxdB(1:n));
```

Appendix C6- Matlab code for the Three-path Fading Simulator.

```
%Name : Hashan Roshantha Mendis (10030637)
%Date Updated : 20/04/08
%-----------------------------------------------------------------
% Three Path Rayleigh Fading Simulator
% simulates Multipath Rayleigh fading (for 3 paths
% simulate with 4 paths
%-----------------------------------------------------------------
function rt = rayleigh_sim(signal,sps)

   % Setup delay parameters
    delay1 = 0.0; delay2 = 0.5; delay3 = 0.5;

   % setup gain parameters
     gain1 = 0; gain2=0.2;  gain3=0.2;
% ------------------------------------------------
    path1 = [zeros(1,round(delay1*sps)) signal];
    path2 = [zeros(1,round(delay2*sps)) signal];
    path3 = [zeros(1,round(delay3*sps)) signal];

    % chop off the end to agree with original lengths
    path1_chop = path1(1: (length(path1)- round(delay1*sps)));
    path2_chop = path2(1: (length(path2)- round(delay2*sps)));
    path3_chop = path3(1: (length(path3)- round(delay3*sps)));

    % setup rayleigh envelope weights
    wavlen = 0.333; %wavelength for GSM
    mobspeed = (200*1000)/(60*60); % mobile speed
    %initially test without doppler shift.

    ray1 = 1; % Direct LOS wave.
    ray2 = abs(randn(1,length(signal))+ j*randn(1,length(signal))); %path2
    ray3 = abs(randn(1,length(signal))+ j*randn(1,length(signal))); %path3


    % test with doppler filter.
    %ray2 = ray_sos(wavlen,mobspeed,10^3,length(path2_chop)); % envelope 2
    %ray3 = ray_sos(wavlen,mobspeed,10^3,length(path3_chop)); % envelope 3


    % apply gain to rayleigh weights
    ray1_g = ray1*gain1;
    ray2_g = ray2*gain2;
    ray3_g = ray3*gain3;

    % multiply signal with rayleigh envelopes.
    sig1 = ray1_g.*path1_chop;
    sig2 = ray2_g.*path2_chop;
    sig3 = ray3_g.*path3_chop;

    % combine multipath faded signals.
    rt = sig1 + sig2 + sig3;

end
```

Appendix C7- Matlab Code for the Rayleigh Fading Simulator (using Jakes SOS method)

```
%Name : Hashan Roshantha Mendis
%Date updated : 20/04/08
%-------------------------------------------------------------------------
%The following implements Jakes SOS method to generate a Rayleigh
%faded envelope.

function rt_env_out = ray_sos(lamda,v,fs,samples)
    %----- Static function parameters (for testing) -------
        %lamda = 0.3333;
        %v = (5*1000)/(60*60);
        %N = 66;
        %fs = 36;
        %samples = 360000;
        %M = 0.5*(N/2 -1);
    % -----------------------------------------------------
    fm = v/lamda;
    M = 16;    N = 2*(2*M +1);
    ts = 1/fs;

    % have to stretch the time axis to avoid incorrect samples at the front
    extra = M*(1/fm);
    tlen = 2*extra + samples*ts;

    t = 0 : ts : tlen;
    a = 0;
    I_x =  sqrt(2)*cos(a)*cos(2*pi*fm*t); % results in cos(2*pi*fm*t)
    Q_x =  sqrt(2)*sin(a)*cos(2*pi*fm*t); % results in zero

    I = zeros(1,length(I_x));
    Q = zeros(1,length(Q_x));

    for nn = 1:M
      bn = (pi*nn)/M;

      fn = fm*cos((2*pi*nn)/N);

        Itemp = (2*cos(bn)*cos(2*pi*fn*t));
        Qtemp = (2*sin(bn)*cos(2*pi*fn*t));

      I = I+Itemp;
      Q = Q+Qtemp;
    end

    I_final = I + I_x;
    Q_final = Q + Q_x;

    ss = extra + 0.9*rand(1)*extra;
    ss = round(ss/ts);
    es = ss + samples-1;

    rt_complex = (1/sqrt(2*M+1))*(I_final+j*Q_final);
    rt_complex = rt_complex(ss:1:es);

    rt_env = sqrt((real(rt_complex).^2)+(imag(rt_complex).^2));
```

```
    rt_env_out = abs(rt_complex);

        %r_dB = 10*log10(rt_env);
        %r_rms_dB = 10*log10(sqrt(mean(rt_env)^2));
        %r_rms_dB = 10*log10(r_rms);
        %plot([0:samples-1]*ts,r_dB-r_rms_dB);title('envelope r_d_B')
        %axis([0 1 min(r_dB-r_rms_dB) max(r_dB-r_rms_dB)]);
end
```

Appendix D

This section contains a brief explanation of the contents of the Software CD that is attached to this report.

- Folder named "Report Softcopy" which contains a softcopy of the project report in two file formats: Microsoft Word (.doc) and Adobes' Portable Document Format (.pdf).

- Folder named "GMSK_MODEM" which contains the Matlab scripts written to simulate the GMSK modem and its respective sub functions. All Matlab code needs the Matlab interpreter, Communication Toolbox (v3.2 or above) and Signal Processing Toolbox (v6.4 or above). Below are the file listing for this folder:

  - AWGN_channel.m
  - eye1.m
  - GMSK_ADC.m
  - GMSK_downsample.m
  - GMSK_gaussian_filter.m
  - GMSK_main.m
  - GMSK_main_fading.m
  - GMSK_matched_filter.m
  - GMSK_theoretical_BER.m
  - integral.m
  - psd_pwelch.m
  - rayleigh_sim.m
  - ray_sos.m

- Folder named "8PSK_MODEM" which contains the Matlab scripts written to simulate the GF-8PSK and RCF8PSK modems and their respective sub functions. All Matlab code needs the Matlab interpreter, Communication Toolbox (v3.2 or above) and Signal Processing Toolbox (v6.4 or above). Below are the file listing for this folder:

  - AWGN_channel.m
  - BER_theoretical.m
  - divisibleby3.m
  - epsk_ADC.m
  - epsk_ADC_backup.m
  - epsk_DAC.m
  - epsk_derotate.m
  - epsk_downsample.m
  - epsk_GLPF_main_fading.m
  - epsk_GLPF_main_fading_4report.m
  - epsk_matched_filter.m
  - epsk_RC_main_fading.m
  - epsk_shaping_filter.m
  - epsk_symbol_rotate.m
  - EPSK_theoretical_BER.m
  - epsk_upsample.m
  - eye_plot.m
  - rayleigh_sim.m
  - ray_sos.m